HW-ENT

FasMath™

83D87 User's Manual

High Performance CMOS Math Processor

Cyrix™

Advancing the Standards

$9.95

Cyrix

Table of Contents

1. About This Manual

This document defines the electrical and functional characteristics of the Cyrix CX-83D87 device. The device is described in sufficient detail to provide both the system designer and the programmer the information needed to evaluate and use the device. Detailed descriptions of the internal architecture of the device are not contained in this document.

2. References

The following documents are frequently cited throughout the body of this specification. The reader should be familiar with their contents and have them available for reference when using this manual:

1. **"Microprocessor and Peripheral Handbook Volume I Microprocessor"**, 1988, Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051.

2. **"80386 Hardware Reference Manual"**, 1986, Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051.

3. **"80387 Programmer's Reference Manual"**, 1987, Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051.

4. **"IEEE 754 Standard for Binary Floating Point Arithmetic"**, 1985, Institute for Electrical and Electronic Engineers, Inc., 345 East 47th Street, New York, NY 10017.

3. General Introduction

3.1 Overview

The Cyrix CX-83D87 is a CMOS VLSI integrated circuit which is pin compatible and software compatible with the Intel 80387 math coprocessor yet offers substantially improved performance. The CX-83D87 achieves 4 to 10 times greater performance than the Intel 80387 by implementing its floating point primitive operations in hardware rather than in a microprogrammed sequencer as the 80387 does. This approach allows the CX-83D87 to perform simple floating point operations at the same speed as the 80386 processor can perform integer additions; square root, elementary functions and transcendental functions are evaluated correspondingly faster.

3.2 Circuit

The CX-83D87 device is fabricated using a 1.0 micron, double layer metal CMOS process. This permits the CX-83D87 to operate at clock rates of 20, 25 and 33 Mhz. State of the art ESD protection and latch-up prevention circuits are incorporated into the CX-83D87 design. The CX-83D87 is packaged in a 68-pin ceramic pin grid array.

3.3 Standards

The CX-83D87 implements a full extended double precision IEEE-754-1985 architecture using 80-bit internal format for data storage and computation. This format assures maximum accuracy and operand/result compatibility with the original 80387.

3.4 Architecture

The CX-83D87 architecture is partitioned into three functional blocks: The Interface Unit, the Execution Unit, and the Control Unit. The Interface Unit manages the CX-83D87 interface to the host 80386 device. The Execution Unit performs all floating point primitive operations including operand type conversions, normalizations, additions, multiplications, divisions, and rounding of results. The Control Unit supervises the execution of primitives, sequences primitives to realize complex operations, and controls traffic to/from the Interface Unit.

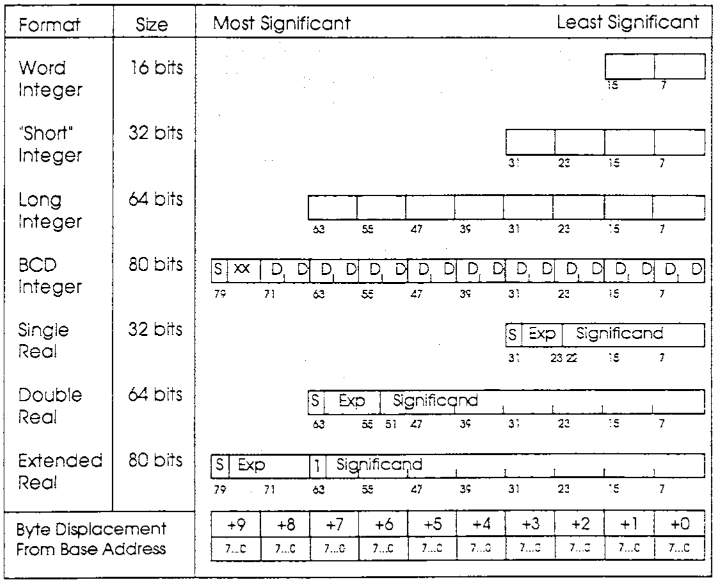
3.5 Programming

The CX-83D87 processor provides the user 8 data registers accessed in a stack-like manner, a control register, and a status register. The CX-83D87 also provides a data register tag word which improves context switching and stack performance by maintaining empty/non-empty status for each of the eight data registers. In addition, registers in the 80386 contain pointers to (a) the memory location containing the current CX-83D87 instruction word and (b) the memory location containing the operand associated with that instruction (if any).

The instructions used to program the CX-83D87 are binary and function compatible with those defined for the Intel 80387 Numeric Processor Extension (cf 2.1). Instructions are provided which load/store data and constants, perform arithmetic, elementary, and transcendental functions, manipulate fraction and exponent fields of operands, and control the status and operating mode of the CX-83D87.

3.6 Data Types

The Cyrix CX-83D87 supports IEEE-754 32-bit single precision, 64-bit double precision, and 80-bit extended double precision real data formats. The CX-83D87 also supports 18 digit BCD integer data format and 16-bit, 32-bit, and 64-bit binary integer data formats. The CX-83D87 data formats are schematically depicted in the following figure.



Memory Data Formats

Internally, all data are converted to 80-bit extended precision format for storage and manipulation. The range and precision provided by 80-bit extended precision format allows the *exact* representation of 16, 32, & 64 bit binary integers, 18 digit BCD integers, and 32 & 64 bit real numbers. Note that the reverse is not necessarily true: the results of 80-bit calculations may require rounding to be represented in the other formats. The following figure shows the range and precision characteristics of each CX-83D87 data type:

Format	Size	Dynamic Range	Resolution
Word Integer	16 bits	-32768 +32767	1 in 2^{16}
"Short" Integer	32 bits	-2,147,483,648 +2,147,483,647	1 in 2^{32}
Long Integer	64 bits	-9,223,382,027,854,875,808 +9,223,382,027,854,875,807	1 in 2^{64}
BCD Integer	80 bits	-999,999,999,999,999,999 +999,999,999,999,999,999	1 in 10^{18}
Single Real	32 bits	$5.8775 \times 10^{-39} \leq \mid N \mid \leq$ $3.4028 \times 10^{+38}$	1 in 2^{24}
Double Real	64 bits	$1.1125 \times 10^{-308} \leq \mid N \mid \leq$ $1.7977 \times 10^{+308}$	1 in 2^{53}
Extended Real	80 bits	$1.68105 \times 10^{-4932} \leq \mid N \mid \leq$ $1.1897 \times 10^{+4932}$	1 in 2^{64}

CX-83D87 Data Type Numerical Properties

3.7 Computational Accuracy

The representation of real numbers in any number system of finite precision is inherently approximate. A simple fraction such as 1/3 cannot be precisely represented in a finite number of digits. This simple observation raises the possibility that different computing systems may choose different methods of approximation and produce different results given the same inputs and algorithms.

The IEEE-754 Standard for Binary Floating Point Arithmetic attempts to solve this problem by specifying the error bounds for the calculation of binary floating point values. These error bounds are controlled in two ways: (1) by directly specifying the error allowable in a primitive calculation and (2) by specifying the exact rounding algorithms to be used. The result of this standardization is that given the same set of input values and rounding instructions, all conformal machines will produce the same result to within a defined margin of error.

Computations internal to the CX-83D87 are performed using 80-bit extended precision operands. The 15-bit exponent and the 64-bit significand data are operated on using independent 15-bit & 67-bit integer arithmetic units. The significand ALU includes provisions for handling extra Guard, Round, & Indicator (sticky) bits to assist in maintaining precision when rounding.

These extra bits are used in IEEE specified rounding modes and help maintain accuracy when the precision of a result exceeds that available in the significand. The programmer can select any of the four IEEE specified rounding modes for use in computation: round to nearest or even, round down, round up, and truncate.

In addition, for the sake of compatibility with earlier generations of floating point processors, a "precision control" function is provided to the programmer. This is used to specify that internal results be maintained in single, double, or extended precision range and resolution.

4. Instruction Set

4.1 Instruction Set Summary

The following table summarizes the operation and allowed forms of every member of the CX-83D87 instruction set:

Mnemonic	Result	Operation
F2XM1	TOS ←	$2^{TOS}-1$
FABS	TOS ←	|TOS|
FADD	ST(I) ←	ST(I)+TOS
FADD	TOS ←	TOS+ST(I)
FADD	TOS ←	TOS+M.DR
FADD	TOS ←	TOS+M.SR
FADDP	ST(I-1) ←	ST(I)+TOS; SP←SP+1
FIADD	TOS ←	TOS+M.SI
FIADD	TOS ←	TOS+M.WI
FCHS	TOS ←	-TOS
FCLEX	—	Clear Exceptions
FCOM	CC ←	TOS-ST(I)
FCOM	CC ←	TOS-M.DR
FCOM	CC ←	TOS-M.SR
FCOMP	CC ←	TOS-ST(I); SP←SP+1
FCOMP	CC ←	TOS-M.DR; SP←SP+1
FCOMP	CC ←	TOS-M.SR; SP←SP+1
FCOMPP	CC ←	TOS-ST(1); SP←SP+2
FICOM	CC ←	TOS-M.SI
FICOM	CC ←	TOS-M.WI
FICOMP	CC ←	TOS-M.SI; SP←SP+1
FICOMP	CC ←	TOS-M.WI; SP←SP+1
FCOS	TOS ←	COS(TOS)
FDECSTP	SP ←	SP-1
FDIV	ST(I) ←	ST(I)/TOS
FDIV	TOS ←	TOS/ST(I)
FDIV	TOS ←	TOS/M.DR
FDIV	TOS ←	TOS/M.SR
FDIVP	ST(I-1) ←	ST(I)/TOS; SP←SP+1
FDIVR	TOS ←	ST(I)/TOS
FDIVR	ST(I) ←	TOS/ST(I)
FDIVR	TOS ←	M.DR/TOS
FDIVR	TOS ←	M.SR/TOS
FDIVRP	ST(I-1) ←	TOS/ST(I); SP←SP+1
FIDIV	TOS ←	TOS/M.SI
FIDIV	TOS ←	TOS/M.WI
FIDIVR	TOS ←	M.SI/TOS
FIDIVR	TOS ←	M.WI/TOS
FFREE	TAG(I) ←	Empty
FINCSTP	SP ←	SP+1
FINIT	—	Initialize

FLD	TOS ←	ST(I); SP←SP-1
FLD	TOS ←	M.DR; SP←SP-1
FLD	TOS ←	M.LI; SP←SP-1
FLD	TOS ←	M.SR; SP←SP-1
FLD	TOS ←	M.XR; SP←SP-1
FBLD	TOS ←	M.BCD; SP←SP-1
FILD	TOS ←	M.SI; SP←SP-1
FILD	TOS ←	M.WI; SP←SP-1
FLD1	TOS ←	1.0; SP←SP-1
FLDCW	Ctl Word ←	Memory
FLDENV	Env Regs ←	Memory
FLDL2E	TOS ←	Log$_2$(e); SP←SP-1
FLDL2T	TOS ←	Log$_2$(10); SP←SP-1
FLDLG2	TOS ←	Log$_{10}$(2); SP←SP-1
FLDLN2	TOS ←	Log$_e$(2); SP←SP-1
FLDPI	TOS ←	π; SP←SP-1
FLDZ	TOS ←	0.0; SP←SP-1
FMUL	ST(I) ←	ST(I)*TOS
FMUL	TOS ←	TOS*ST(I)
FMULP	ST(I-1) ←	ST(I)*TOS; SP←SP+1
FMUL	TOS ←	TOS*M.DR
FMUL	TOS ←	TOS*M.SR
FIMUL	TOS ←	TOS*M.SI
FIMUL	TOS ←	TOS*M.WI
FNOP	—	No Operation
FNOP(FENI)	—	No Operation
FNOP(FDISI)	—	No Operation
FNOP(FSETPM)	—	No Operation
FPATAN	TOS ←	ATAN($\frac{ST(1)}{TOS}$); SP←SP+1
FPREM	TOS ←	Rem($\frac{TOS}{ST(1)}$)
FPREM1	TOS ←	Rem($\frac{TOS}{ST(1)}$)
FPTAN	TOS;ST(1) ←	1; TAN(TOS); SP←SP-1
FRNDINT	TOS ←	Round(TOS)
FRSTOR	—	Restore state.
FSAVE	—	Save state.
FSCALE	TOS ←	TOS*2$^{(ST(1))}$
FSIN	TOS ←	SIN(TOS)
FSINCOS	TOS;ST(1) ←	COS;SIN(TOS); SP←SP+1
FSQRT	TOS ←	$\sqrt{TOS}$
FST	ST(i) ←	TOS
FST	M.DR ←	TOS
FST	M.SR ←	TOS
FSTP	ST(i-1) ←	TOS; SP←SP+1
FSTP	M.DR ←	TOS; SP←SP+1
FSTP	M.LI ←	TOS; SP←SP+1
FSTP	M.SR ←	TOS; SP←SP+1
FSTP	M.XR ←	TOS; SP←SP+1
FBSTP	M.BCD ←	TOS; SP←SP+1

FIST	M.SI ←	TOS
FIST	M.WI ←	TOS
FISTP	M.SI ←	TOS; SP←SP+1
FISTP	M.WI ←	TOS; SP←SP+1
FSTCW	Memory ←	Control word.
FSTENV	Memory ←	Ctl,Status,IP,DP.
FSTSW	Memory ←	Status
FSTSWAX	AX ←	Status
FSUB	ST(I) ←	ST(I)-TOS
FSUB	TOS ←	TOS-ST(I)
FSUBP	ST(I-1) ←	ST(I)-TOS; SP←SP+1
FSUB	TOS ←	TOS-M.DR
FSUB	TOS ←	TOS-M.SR
FISUB	TOS ←	TOS-M.WI
FISUB	TOS ←	TOS-M.SI
FSUBR	TOS ←	ST(I)-TOS
FSUBR	ST(I) ←	TOS-ST(I)
FSUBRP	ST(I-1) ←	TOS-ST(I); SP←SP+1
FSUBR	TOS ←	M.DR-TOS
FSUBR	TOS ←	M.SR-TOS
FISUBR	TOS ←	M.WI-TOS
FISUBR	TOS ←	M.SI-TOS
FTST	CC ←	TOS-0.0
FUCOM	CC ←	TOS-ST(I)
FUCOMP	CC ←	TOS-ST(I); SP←SP+1
FUCOMPP	CC ←	TOS-ST(1); SP←SP+2
FXAM	CC ←	Class of TOS.
FXCH	TOS ↔	ST(I) Exchange
FXTRACT	TOS;ST(1) ←	Signif; Exp; SP←SP-1
FYL2X	TOS ←	ST(1)*Log$_2$(TOS); SP←SP+1
FYL2XP1	TOS ←	ST(1)*Log$_2$(1+TOS); SP←SP+1

The abbreviations and conventions used in in the CX-83D87 instruction summary are:

1. TOS == Top of stack register pointed to by SSS in Control Register. (cf. Sec 4.3).

2. ST(1)== Next to Top of stack register (pointed to by SSS+1).

3. Memory operands are referred to as "M.XX" where "XX"=:

"WI"	-->	16-bit integer;
"SI"	-->	32-bit integer;
"LI"	-->	64-bit integer;
"SR"	-->	32-bit real;
"DR"	-->	64-bit real;
"XR"	-->	80-bit real;
"BCD"	-->	18-digit BCD integer.

4. The "Operation" column refers to stack layout *before* instruction execution.

5. The "Result" column refers stack layout *after* instruction execution.

6. Operands separated by a semicolon indicate that the leftmost destination receives the leftmost source, e.g., TOS<--COS(TOS) in TSINCOS; ST(1)<--SIN(TOS).

7. SP== CX-83D87 Stack Pointer (Contents of SSS).

8. IP== CX-83D87 Instruction Pointer.

9. DP== CX-83D87 Data Pointer.

10. ST(I)== Randomly addressed CX-83D87 data register I.

11. CC== Condition codes in CX-83D87 Status Register. (cf. Sec 4.3)

12. Env Regs== SP, Status, Tags, IP, & DP.

4.2 Data Registers

The Cyrix CX-83D87 provides a set of eight 80-bit data registers for use in computations. These registers are accessed in a stack-like fashion for programming purposes. An explicit data load instruction (FLD) must be used to store a datum into the CX-83D87 "Top of Stack" prior to performing arithmetic operations on it. The load instruction causes a "push" operation by decrementing the "stack pointer" (SSS field of the Status Register) by one modulo 8 and loading the datum into the physical register newly addressed by SSS.

The CX-83D87 provides a complete set of dual operand instructions to ease programming of its pseudo-stack register set. The instructions FADD, FCOM, FDIV, FMUL, and FSUB perform arithmetic on the Top of Stack and either memory or explicitly addressed register operands. The result is placed in the Top of Stack. Note that conversion of the memory operand to 80-bit internal format is performed automatically.

All other CX-83D87 arithmetic instructions operate only on data contained in CX-83D87 registers.

The explicit register addressing feature operates by specifying the displacement (I) of the target register from the current Top of Stack. This displacement is added to the SSS field modulo 8 (prior to any increment or decrement operations specified by the instruction) to calculate the physical register number. Syntactically, ST(I) is used to specify displacement (I) from the Top of Stack. Thus, ST(0) is Top of Stack, ST(1) is next to Top of Stack, and so on. Logical register references of the form ST(I) are always relative to the current Top of Stack.

The limited number of physical registers and the use of "wrapping" modulo 8 arithmetic to address them leads to the possibility of a data register overwrite error. To prevent this and to simplify error detection, the CX-83D87 maintains a register Tag Word comprised of two bits for each physical data register. Tag Word fields assume one of four values depending on the contents of their associated data registers, Valid("00"), Zero("01"), Special("10"), and Empty("11"). Tag values are maintained transparently by the CX-83D87 and are only available to the programmer indirectly through the FSTENV and FSAVE instructions. The Tag Word with tag fields for each associated physical register, tag(R), is schematically depicted below:

15			8	7			0
Tag(7)	Tag(6)	Tag(5)	Tag(4)	Tag(3)	Tag(2)	Tag(1)	Tag(0)

The CX-83D87 detects two kinds of register operation errors: Source Register Empty and Destination Register Full . The Source Register Empty error occurs when an instruction attempts to read a source operand from a data register which is Empty, i.e. Tag(R)="11". The Data Register Full error occurs when an attempt is made to store data into a register which in not Empty, i.e. Tag(R)≠"11". In the context of load ("push") and store ("pop") operations these errors are interpreted as stack wraparound/overflow (Dest. Full) and "stack underflow" (Source Empty) conditions. Note that in other contexts, such as "FST ST(4),ST(0)" with ST(4) not empty, or "FLD ST(0),ST(6)" with ST(6) empty, these errors also occur. Register errors generate the Invalid Exception and are reported through the SF bit of the Status Register.

The results of CX-83D87 operations are converted to the desired data format and stored in memory using the register store instruction, FST. The store instruction always uses the Top of Stack as its source operand. All forms of the FST instruction allow a "pop" of the Top of Stack upon completion.

"Pop" operations are effected by marking the physical register addressed by the SSS field of the Status Register Empty and incrementing the SSS field by 1 modulo 8. "Pop" operations are provided as options for the dual operand instructions FADD, FDIV, FMUL, FSUB, and FUCOM *when using a data register source operand.* In addition, FCOM provides a "pop" option when used with either memory or a data register as a source operand to facilitate conditional testing. Finally, both FCOM and FUCOM provide a double "pop" form which compares the Top of Stack to the next to Top of Stack and "pops" both on completion.

For special programming situations, the FFREE instruction can be used to mark an explicitly addressed register Empty. FINCSTP & FDECSTP can also be used to increment or decrement the SSS field of the Status Register.

4.3 Control & Status Registers

The Cyrix CX-83D87 communicates information about its status and the results of operations to the host 80386 via the Status Register. The CX-83D87 Status Register is comprised of bit fields that reflect exception status, operation execution status, register status, operand class, and comparison results. This register is continuously accessible to the 80386 CPU regardless of the state of the Control or Execution Units.

The CX-83D87 Mode Control Register (MCR) is used by the 80386 uP to specify the operating mode of the CX-83D87 processor. The MCR contains bit fields which specify the rounding mode to be used, the precision with which to calculate results, and the exception conditions which should be reported to the host via traps. The user controls precision, rounding, and exception reporting by setting or clearing appropriate bits in the MCR.

The following figure details the CX-83D87 Status Register and Mode Control Register as they appear to the programmer:

Register	15			12	11			8	7			4	3			0
Status	B	C3	S	S	S	C2	C1	C0	ES	SF	P	U	O	Z	D	I
Mode Ctl	*	*	*	*	RC	RC	PC	PC	*	*	P	U	O	Z	D	I

Exceptions/Masks

Table 3: CX-83D87 Status Register & Mode Control Register

The bit fields used in Table 3 are defined as follows:

B	–	Copy of ES bit.	
C3,C2,C1,C0	–	Condition code bits.	
SSS	--	Top of Stack Register number.	
ES	--	Error Indicator. Set to 1 if an unmasked exception is detected.	
SF	--	Stack Fault or invalid register operation bit.	
RC	--	Rounding Control bits:	
		00 -	Round to nearest or even.
		01 -	Round toward minus infinity.
		10 -	Round toward plus infinity.
		11 -	Truncate.
PC	–	Precision Control bits:	
		00 -	24-bit mantissa.
		01 -	Reserved.
		10 -	53-bit mantissa.
		11 -	64-bit mantissa.
Exception Status & Mask Bits	--	"1" in Status Register indicates exception; "1" in MCR masks exception trap.	
		P -	Precision error.
		U -	Underflow error.
		O -	Overflow error.
		Z -	Divide by zero error.
		D -	Denormalized operand error.
		I -	Invalid operation.

4.4 Exception Handling

The CX-83D87 performs extensive data checking during the data input and type conversion process and during the process of performing calculations. The detection of an inconsistency or an error is considered an exception condition and is reported in the CX-83D87 status word. Optionally, exceptions can initiate an exception trap sequence in the host processor.

There are seven types of exceptions that the CX-83D87 can detect and report: Precision Exception, Underflow Exception, Overflow Exception, Divide By Zero Exception, Denormal

Operand Exception, Invalid Operation Exception, and Register Error. These seven exceptions map into six separately enabled exception traps -- Invalid operation and register errors are reported on the same trap. Exception traps are unmasked (enabled) by writing a "0" into the appropriate bit of the CX-83D87 Mode Control Register.

When the CX-83D87 arithmetic unit detects an exception, it sets the corresponding exception status bit in the CX-83D87 Status Word in all cases except Underflow when Underflow is masked. In this case, the Underflow bit is only set if Precision Exception is also set. The CX-83D87 Interface Unit compares the unmasked exceptions in the CX-83D87 Control Word against the exception status bits maintained in the Status Word. When an unmasked exception is detected, a trap sequence is initiated by the Interface Unit by asserting the CX-83D87 ERROR# output signal. Note that unmasking an exception will cause an immediate trap if the exception status bit is set.

The detection of an exception during the execution of an instruction results in one of two possible outcomes. If the exception is masked, the CX-83D87 generates the IEEE-754-1985 specified result, stores the result in the destination location, and proceeds with program execution. If the exception is unmasked and the exception is Precision, Underflow, or Overflow, the IEEE-754-1985 specified result is stored in the destination and an Error trap sequence is initiated. If the exception is unmasked Zero Divide, Denormal, or Invalid, the offending instruction is aborted, no result is written, and the Error trap sequence is begun.

4.5 Precision Exception

This exception is caused by the production of a result which is not exactly representable in the destination location of the instruction. For instance, the number 1/10 is not exactly representable as a binary floating point number. Similarly, all transcendental and elementary functions return approximations to the infinitely precise, irrational value. Both situations will cause the Precision Exception to be set.

4.6 Underflow Exception

This exception is caused by the production of a result which is tiny in magnitude and requires either a denormal or zero result to represent the value. The significance of the Underflow Exception depends on the state of the Underflow Mask bit. When Underflow is masked the exception status is set only when the result is both denormal/zero and precision has been lost .

When Underflow is unmasked, the Underflow Exception occurs whenever the result would be denormal/zero. In this case, if the destination is a stack register, the true result is multiplied by $2^{24,576}$, rounded, and stored in the destination register. This behavior occurs for all underflow cases except the extreme underflow produced by the FSCALE instruction. FSCALE extreme underflow will produce +/- 0 just as if Underflow was masked. If the destination is memory, the instruction is aborted and no result is written.

4.7 Overflow Exception

This exception is caused by the production of a result which is larger in magnitude than the destination format can accomodate. When Overflow is masked the exception status is set and the result is rounded in accordance with the RC mode bits.

When Overflow is unmasked, the Overflow Exception causes the results to be scaled to fit in the destination format. In this case, if the destination is a stack register, the true result is divided by $2^{24,576}$, rounded, and stored in the destination register. This behavior occurs for all overflow cases except the extreme overflow produced by the FSCALE instruction. FSCALE extreme overflow will produce +/- ∞ just as if Overflow was masked. If the destination is memory, the instruction is aborted and no result is written.

4.8 Denormal Exception

This exception is caused by the use of a denormal operand as input to an instruction. When Denormal is masked the instruction proceeds to completion using the denormal operand.

When Denormal is unmasked the exception status is set and an exception trap sequence is initiated without writing any result to the the destination of the instruction.

4.9 Divide By Zero Exception

This exception is caused by an attempt to divide a finite non-zero operand by zero. In addition to the divide instruction, the FXTRACT and FYL2X instructions may generate this exception. The masked response to this exception is to return ∞ with its sign set to the exclusive-or of the operands' signs. For FXTRACT, ST(1) is set to -∞ and TOS is set to zero with its sign set to the sign of the initial operand.

When Divide By Zero is unmasked the exception status is set and an exception trap sequence is initiated without writing any result to the the destination of the instruction.

4.10 Stack Error Exception

Register errors (cf. sec 2.4.2) produce the stack error exception. When masked the QNaN indefinite overwrites the destination. When unmasked, an exception trap sequence is initiated and the destination is left undisturbed. This exception is indicated by simultaneously setting the Invalid and Stack Fault bits in the Status Word.

When a Stack Error is signaled, the C1 status bit indicates whether the error was Destination Register Full (C1="1") or Source Register Empty (C1="0").

4.11 Invalid Operation Exception

The IEEE-754 Standard provides for the detection and reporting of attempts to perform arithmetic on operands that cannot provide meaningful results. Such attempts are called Invalid Operations. When Invalid Exception is unmasked, the occurence of such an event causes the instruction to be aborted, a trap sequence to be initiated, and the destination to be left undisturbed. When Invalid is masked, the appropriate IEEE specified response is generated and written into the destination.

The following table defines invalid operations and their default (masked) results:

Operand(s)	Invalid Operation	Default Result
Unsupported	Any Arithmetic	QNaN Indefinite.
*NaN or (NaN,Valid)	Any Arithmetic	Quietized Nan.
QNaN & SNaN	Any Arithmetic	The QNaN.
SNaN & SNaN	Any Arithmetic	Larger SNaN Quietized.
*QNaN & QNaN	Any Arithmetic	Larger QNaN.
NaN	FCOM(P)(P) & FTST	CC= Unordered.
$(+\infty,-\infty)$ or $(-\infty,+\infty)$	Addition	QNaN Indefinite.
$(+\infty,+\infty)$ or $(-\infty,-\infty)$	Subtraction	QNaN Indefinite.
$0,\infty$	Multiplication	QNaN Indefinite.
(∞,∞) or $(0,0)$	Division	QNaN Indefinite.
Zero Divisor	Partial Remainder	QNaN Indefinite.
∞ Dividend	Partial Remainder	QNaN Indefinite.
∞	Forward Trigonometric	QNaN Indefinite.
Negative Number	Square Root, Log_2	QNaN Indefinite.
Empty Reg, NaN, ∞	Integer & BCD Store	Integer/BCD Indefinite.
Exceeds Integer Range	Integer & BCD Store	Integer/BCD Indefinite.
Empty Register	Exchange Registers	Set Empty to QNan Indefinite & exchange.

*Note: Generally the use of a QNaN as an operand in these contexts does <u>not</u> generate the Invalid Exception. All other Invalid Operations in the table produce the Invalid Exception.

Certain other combinations of valid numbers, zeroes, and infinities may give rise to the Invalid Exception for FSCALE, FYL2X, & FYL2XP1. Please consult section 4.12, Detailed Instruction Descriptions, for details.

4.12 Detailed Instruction Descriptions

The detailed instruction descriptions provide a complete reference on the use of each CX-83D87 instruction. The instruction mnemonic, its syntactical variants, allowed operands, encoding, operation, effect on status, possible exceptions, and special operands/results are presented for each instruction type.

Please note that integer variants which cause insertion of the letter "I" and BCD variants which cause insertion of the letter "B" into their mnemonics are included in the listings under their basic type, i.e. FIADD is described under FADD. With this single exception, all the CX-83D87 instructions are catalogued alphabetically by mnemonic in this section.

The integer variants presented under their basic type are: FIADD, FICOM, FIDIV, FILD, FIMUL, FIST, & FISUB. Similarly, the BCD integer forms are grouped with FLD (FBLD) and FST (FBSTP).

Detailed instruction descriptions are presented in the following standard format:

Syntax:	A BNF-like description of the assembler syntax of the instruction.
Forms:	An complete list of the permitted operand source & destination combinations for the instruction.
Operands:	A detailed table showing all permitted operand types, the instruction encoding for each type, and the execution time in clock cycles.
Operation:	A textual description of the action caused by the execution of the instruction.
Status:	A table which summarizes the effect on CX-83D87 condition codes of each instruction.
Exceptions:	A table which summarizes the possible exceptions an instruction can produce and the resulting exception status bits in the Status Register.
Zero/Inf:	A summary of the results produced when executed with extraordinary valued operands.
Notes:	A brief statement of points of special interest not covered elsewhere.

The following symbols and abbreviations are used throughout the detailed instruction descriptions:

Syntax: Square brackets ("[]") indicate an optional argument. Nesting is permitted.

Angle brackets ("<>") indicate an argument of type "<*type* >" must be supplied. Allowed types are:

DST	Destination operand.
SRC	Source operand.

Literals are: "P" (pop), "R" (reverse), and ",".

Forms: <TOS> refers to the Top of Stack register, denoted by "ST(0)" to the Intel assembler.

<Reg> refers to an explicitly specified CX-83D87 data register denoted by "ST(I)" where 0≤I≤7.

<Memory> refers to a legal 80386 memory operand address, e.g. "DWORD PTR 0100".

Operands: Source Operand specifications are given for each form of the instruction. The key to the specifications is:

16-bit Integer	Memory location "WORD PTR"
32-bit Integer	Memory location "DWORD PTR"
64-bit Integer	Memory location "QWORD PTR"
32-bit Real	Memory Location "DWORD PTR"
64-bit Real	Memory Location "QWORD PTR"
80-bit Real	Memory Location "TBYTE PTR"
18 dig BCD Int.	Memory location "TBYTE PTR"
80-bit Register	Explicit register "ST(i)"
Top of Stack	Register "ST(0)"

Encoding fields may have the following values:

SXX	Hexadecimal value of 1st byte.
MD	2-bit 80386 "MOD" field.(of MOD R/M.)
R/M	3-bit 80386 "Reg/Mem" field.(MOD R/M.)
p	1-bit specifying "pop"=1/no "pop"=0.
REG	3-bits specifying CX-83D87 data register.
SIB	80386 Stack Index Base field.
DISP	80386 Displacement field.

Cycle counts are given in 80386 clock cycles and assume no wait states and no DMA overhead in the host system.

Status: Status bits are indicated as:

U	Undefined after instruction execution.
1	Set to "1" as a result of instruction.
0	Set to "0" as a result of instruction.
M	Set to the value loaded from memory.

Exceptions: Exceptions are listed by type and specify the result stored in the destination based on the trap mode: masked or unmasked. Exceptions which are not possible for a given instruction are shown as "N/A". Exception status bits are indicated as:

-	Unaffected by instruction execution.
1	Set to "1" given specified exception.
0	Set to "0" unconditionally.
M	Set to the value loaded from memory.

Zero/Inf: Special symbols used have the following definitions:

"->"	"Converted to".
sgn0	Arithmetic sign of argument 0.
"X"	A positive, finite, nonzero integer.
"Y"	A positive, finite, nonzero integer.
NaN	Not a Number.
R(0)	Zero as produced by current RC mode.

F2XM1 Function Evaluation: 2^{x}-1. F2XM1

Syntax: **F2XM1**

Forms: **F2XM1**

Operands:

Inst	Source Operand	Encoding			Cycles
F2XM1	Top of Stack	$D9	11 110 000		14-67

Operation: The Top of Stack contains the source operand (x). The function y= 2^{x}-1 is evaluated and and the result is normalized and rounded according to the RC mode in effect. The result (y) replaces (x) in the Top of Stack. The source operand (x) must be in the range $-1 \le x \le 1$. To obtain the value 2^{x} (antilog$_2$(x)), add 1 to the result.

Arguments $|x| \ge 1.0$ produce undefined results without signaling any exceptions.

Status:

Result of Instruction		C3	C2	C1[1]	C0
Normal Execution:		U	U	0	U
Register Error:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Operand Used	-	-	-	-	-	1	-
	Unmasked	Trap/Unaltered	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Trap/Unaltered	-	-	-	-	-	-	1
Reg Error	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Trap/Abort	1	-	-	-	-	-	1

Zero/Infinity:

X	Result
$+0$	$+0$
-0	-0
$-\infty$	-1
$+\infty$	$+\infty$

Notes: 1. After a Precision Exception the "C1" status bit indicates whether rounding was away from zero.

FABS Floating Absolute Value FABS

Syntax: **FABS**

Forms: **FABS**

Operands:

Inst	Source Operand	Encoding			Cycles
FABS	Top of Stack	$D9	11 100 001		4

Operation: The Top of Stack is always the source operand. The sign of the Top of Stack is set to zero (positive).

Status:

Result of instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Reg Error	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Trap/Abort	1	-	-	-	-	-	1

Zero/Infinity:

OP1	Result
+0	+0
−0	−0
−∞	+∞
+∞	+∞

Notes: None.

FADD Floating Point Add FADD

Syntax: **FADD** **([<DST>,]<SRC>)**

Forms:

FADD	**<TOS>,<Memory>**
FADD	**<TOS>,<Register>**
FADDP	**<Register>,<TOS>**

Operands:

Inst	Source Operand	Encoding			Cycles
FIADD	16-bit Integer	$DE	MD 000 R/M	SIB,DISP	13
FIADD	32-bit Integer	$DA	MD 000 R/M	SIB,DISP	13
FADD	32-bit Real	$D8	MD 000 R/M	SIB,DISP	13
FADD	64-bit Real	$DC	MD 000 R/M	SIB,DISP	15
FADD	80-bit Register	$D8	11 000 REG		6
FADD	Top of Stack	$DC	11 000 REG		6
FADDP	Top of Stack	$DE	11 000 REG		6

Operation: The source and destination operands are fetched. The source is converted to extended precision format if necessary. The operands are added and the result is normalized and rounded according to the RC mode in effect at the precision specified by the PC mode bits. The result is stored in the destination register. When the "pop" form is used, the top of stack is popped.

Status:

Result of Instruction		C3	C2	C1[1]	C0
Normal Execution:		U	U	0	U
Register Error:	Dest. Reg. Full	U	U	1	U
	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	Masked	R(∞)	-	-	-	1	-	-	-
	Unmasked	Round & Scale	-	-	-	1	-	-	-
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

FADD Floating Point Add FADD

Zero/Infinity:

OP1	OP2	Result	OP1	OP2	Result
+0	+0	+0	+∞	+∞	+∞
-0	-0	-0	-∞	-∞	-∞
+0	-0	R(0)	+∞	-∞	Inv Op
-0	+0	R(0)	-∞	+∞	Inv Op
-X	+X	R(0)	+∞	X	+∞
+X	-X	R(0)	-∞	X	-∞

Notes: 1. After a Precision Exception the "C1" status bit indicates whether rounding was away from zero.

FCHS Floating Change Sign FCHS

Syntax: **FCHS**

Forms: **FCHS**

Operands:

Inst	Source Operand	Encoding			Cycles
FCHS	Top of Stack	$D9	11 100 000		4

Operation: The Top of Stack is always the source operand. The sign of the Top of Stack is complemented.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Reg Error	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Trap/Abort	1	-	-	-	-	-	1

Zero/Infinity:

OP1	Result	OP1	Result
+0	-0	+∞	-∞
-0	+0	-∞	+∞

Notes: None.

FCLEX Clear Exceptions FCLEX

Syntax: **FCLEX**

Forms: **FCLEX**

Operands:

Inst	Source Operand		Encoding		Cycles
FCLEX	80-bit Register	$DB	11 100 010		4

Operation: All CX-83D87 exception flags are reset to zero. The busy flag is reset to zero. ERROR# goes inactive.

Status:

Result of Instruction	C3	C2	C1	C0
Normal Execution:	U	U	U	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
None	N/A	All Cleared	0	0	0	0	0	0	0

Zero/Infinity: None.

Notes: None.

FCOM Floating Point Compare FCOM

Syntax: **FCOM(P) ((<DST>,)<SRC>)**

Forms: **FCOM(P) <TOS>,<Memory>**
FCOM(P) <TOS>,<Reg>

Operands:

Inst	Source Operand	Encoding			Cycles
FICOM(P)	16-bit Integer	$DE	MD 01p R/M	SIB,DISP	11
FICOM(P)	32-bit Integer	$DA	MD 01p R/M	SIB,DISP	11
FCOM(P)	32-bit Real	$D8	MD 01p R/M	SIB,DISP	11
FCOM.D	64-bit Real	$DC	MD 01p R/M	SIB,DISP	13
FCOM(P)	80-bit Reg	$D8	11 01p REG		4
FCOMPP	80-bit Reg	$DE	11 011 001		4

Operation: The source operand is fetched and converted to extended precision format if necessary. The source operand is subtracted from the destination (Top of Stack) and the condition codes are set according to the result. When the "P" form is used, the Top of Stack is popped. The "PP" form causes two "pop" operations to take place.

The result "unordered" is produced when the operands are NaNs, unsupported or when Stack Fault occurs. These operands also cause the Invalid Exception.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:	DST > SRC	0	0	0	0
	DST < SRC	0	0	0	1
	DST = SRC	1	0	0	0
	Unordered	1	1	0	1
Register Error:	Source Reg Empty	1	1	0	1

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	QNaN	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	"Unordered"	-	-	-	-	-	-	1
	Unmasked	Trap/Abort^2	-	-	-	-	-	-	1
Register Error	Masked	"Unordered"	1	-	-	-	-	-	1
	Unmasked	Trap/Abort^2	1	-	-	-	-	-	1

FCOM Floating Point Compare FCOM

Zero/Infinity:

DST -	SRC	Result	DST -	SRC	Result
+0	+0	=	+∞	+∞	=
-0	-0	=	-∞	-∞	=
+0	-0	=	+∞	-∞	DST>SRC
-0	+0	=	-∞	+∞	DST<SRC
+0	+X	DST<SRC	+∞	X	DST>SRC
-0	+X	DST<SRC	-∞	X	DST<SRC
+0	-X	DST>SRC	X	-∞	DST>SRC
-0	-X	DST>SRC	X	+∞	DST<SRC
NaN	X	Unordered	X	NaN	Unordered

Notes:

1. QNAN operands produce the invalid exception for this instruction.
2. CC bits are set to unordered and any 'pop' operations are inhibited.

FCOS — Function Evaluation: Cos(x). — FCOS

Syntax: **FCOS**

Forms: **FCOS**

Operands:

Inst	Source Operand	Encoding			Cycles
FCOS	Top of Stack	SD9	11 111 111		5-97

Operation: The Top of Stack contains the source operand (x). The instruction requires (x) in radians and returns (y) such that y= cos(x). The result (y) is normalized and rounded according to the RC mode in effect. The value (y) replaces the contents of the Top of Stack. The source operand (x) must be in the range $|x| \leq 2^{63}$.

Status:

Result of Instruction		C3	C2	C1[1]	C0
Incomplete Reduction of (x):		U	1	0	U
Normal completion:		U	0	0	U
Register Error:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Operand Used	-	-	-	-	-	1	-
	Unmasked	Trap/Unaltered	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Trap/Unaltered	-	-	-	-	-	-	1
Register Error	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

x	Result
+0	+1
-0	+1
+∞	Invalid Op.
-∞	Invalid Op.

Notes: 1. After a Precision Exception the "C1" status bit indicates whether rounding was away from zero.

FDECSTP Decrement CX-83D87 Stack Pointer FDECSTP

Syntax: **FDECSTP**

Forms: **FDECSTP**

Operands:

Inst	Source Operand		Encoding		Cycles
FDECSTP	None	$D9	1111 0110		5

Operation: The CX-83D87 data register stack pointer is decremented modulo 8. The result of decrementing SP=0 is SP=7.

Status:

Result of Instruction	C3	C2	C1	C0
Unconditional:	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
None	N/A	N/A	-	-	-	-	-	-	-

Zero/Infinity: None.

Notes: This instruction has no effect on any data register's contents or the Tag Word.

FDIV Floating Point Divide FDIV

Syntax: **FDIV(R) ((<DST>,)<SRC>)**

Forms:

FDIV(R) <TOS>,<Memory>
FDIV(R) <TOS>,<Reg>
FDIV(R) <Reg>,<TOS>

Operands:

Inst	Source Operand	Encoding			Cycles
FIDIV(R)	16-bit Integer	$DE	MD 11r R/M	SIB,DISP	20-31*
FIDIV(R)	32-bit Integer	$DA	MD 11r R/M	SIB,DISP	20-31*
FDIV(R)	32-bit Real	$D8	MD 11r R/M	SIB,DISP	20-31*
FDIV(R)	64-bit Real	$DC	MD 11r R/M	SIB,DISP	21-32*
FDIV(R)	80-bit Reg	$D8	11 11r REG		13-24*
FDIV	Top of Stack	$DC	11 111.REG		14-25
FDIVR	Top of Stack	$DC	11 110 REG		13-24
FDIVP	Top of Stack	$DE	11 111 REG		14-25
FDIVRP	Top of Stack	$DE	11 110 REG		13-24

*Add one clock cycle to these instruction execution times for reversed operands.

Operation: The source operand is fetched and converted to extended precision format if necessary. The destination is divided by the source and the quotient is normalized and rounded according to the RC mode in effect at the precision specified by the PC mode bits. The quotient replaces the original contents of the destination register. When the "pop" form is used, the top of stack is popped.

The "reverse" form causes the source to be divided by the destination. The quotient is placed in the destination.

Status:

Result of instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Source Reg Empty	U	U	0	U

FDIV Floating Point Divide FDIV

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	Masked	R(∞)	-	-	-	1	-	-	-
	Unmasked	Round & Scale	-	-	-	1	-	-	-
Div by Zero	Masked	∞ (Note 2)	-	-	-	-	1	-	-
	Unmasked	Trap/Unaltered	-	-	-	-	1	-	-
Denormal	Masked	Operand Used	-	-	-	-	-	1	-
	Unmasked	Trap/Unaltered	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Trap/Unaltered	-	-	-	-	-	-	1
Register Error	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

OP1 ÷ OP2		Result	OP1 ÷ OP2		Result
+/-0	+/-0	Invalid	+/-∞	+/-∞	Invalid
+0	+X	+0	+∞	+X	+∞
-0	-X	+0	+∞	-X	-∞
+0	-X	-0	-∞	+X	-∞
-0	+X	-0	-∞	-X	+∞
+X	+Y	+0[3]	+∞	+0	+∞
-X	-Y	+0[3]	+∞	-0	-∞
+X	-Y	-0[3]	-∞	+0	-∞
-X	+Y	-0[3]	-∞	-0	+∞

Notes:

1. After a Precision Exception the "C1" status bit indicates whether rounding was away from zero.

2. Division by zero produces an infinity result with a sign equal to the exclusive-or of the signs of the operands when Divide By Zero Exception is masked.

3. Applies to division of X by Y producing extreme denormalization or underflow when Underflow Exception is masked..

FFREE — Free Floating Point Register — FFREE

Syntax: **FFREE (<DST>)**

Forms: **FFREE <Register>**

Operands:

Inst	Source Operand		Encoding		Cycles
FFREE	80-bit Register	SDD	11 000 REG		5

Operation: The destination register tag is changed to empty. The contents of the register are unaffected.

Status:

Result of Instruction	C3	C2	C1	C0
Normal Execution:	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
None	N/A	N/A	-	-	-	-	-	-	-

Zero/Infinity: None.

Notes: None.

FINCSTP Increment CX-83D87 Stack Pointer FINCSTP

Syntax: **FINCSTP**

Forms: **FINCSTP**

Operands:

Inst	Source Operand	Encoding			Cycles
FINCSTP	None	$D9	1111 0111		5

Operation: The CX-83D87 data register stack pointer is incremented modulo 8. The result of incrementing SP=7 is SP=0.

Status:

Result of Instruction	C3	C2	C1	C0
Unconditional:	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
None	N/A	N/A	-	-	-	-	-	-	-

Zero/Infinity: None.

Notes: This instruction has no effect on any data register's contents or the Tag Word.

FINIT CX-83D87 Initialize

Syntax: **FINIT**

Forms: **FINIT**

Operands:

Inst	Source Operand	Encoding			Cycles
FINIT	None	$DB	11 100 011		5

Operation: The CX-83D87 is set to its reset condition. The CX-83D87 Mode Control Register is set to $037F$_{16}$, the CX-83D87 Status Register is reset to $0000, and all data registers are marked empty (Tag word=$FFFF).

This action sets Rounding control to "round to nearest or even" (RC=00), Precision control to 64-bit extended precision (PC=11), and the Top of Stack register number to zero (SSS=000). All exceptions are cleared (=0), all condition codes are cleared (C_0-C_3=0), and all exceptions are masked (=1).

This instruction differs from a hardware reset by setting MCR bit 0 (Invalid Exception Mask) to 1 and Status Register bits 7 & 0 (Error and Invalid Exception) to 0.

Status:

Result of Instruction	C3	C2	C1	C0
Unconditional:	0	0	0	0

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
None	N/A	N/A	0	0	0	0	0	0	0

Zero/Infinity: None.

Notes: None.

FLD — Load CX-83D87 Register — FLD

Syntax: **FLD** **((<DST>,)<SRC>)**

Forms:

FLD **<TOS>,<Memory>**
FLD **<TOS>,<Register>**

Operands:

Inst	Source Operand	Encoding			Cycles
FILD	16-bit Integer	$DF	MD 000 R/M	SIB,DISP	7
FILD	32-bit Integer	$DB	MD 000 R/M	SIB,DISP	7
FILD	64-bit Integer	$DF	MD 101 R/M	SIB,DISP	9
FBLD	18 dig BCD Int.	$DF	MD 100 R/M	SIB,DISP	32
FLD	32-bit Real	$D9	MD 000 R/M	SIB,DISP	10
FLD	64-bit Real	$DD	MD 000 R/M	SIB,DISP	12
FLD	80-bit Real	$DB	MD 101 R/M	SIB,DISP	11
FLD	Top of Stack	$D9	11 000 REG		4

Operation: The source operand is fetched and converted to extended precision format if necessary. The operand is stored in the destination register. If TOS is the destination, the SSS field of the Control Register is pre-decremented modulo 8.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Dest. Reg Full	U	U	1	U
	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal[1]	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op[1]	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

OP1	Result	OP1	Result
+0	+0	+∞	+∞
-0	-0	-∞	-∞

Notes: 1. Denormal and Invalid exceptions cannot occur as a result of FLDEP or FLD <reg> instructions..

FLD1 Load Floating Constant= 1.0 FLD1

Syntax: **FLD1**

Forms: **FLD1**

Operands:

Inst	Source Operand		Encoding		Cycles
FLD1	80-bit Internal	\$D9	11 101 000		4

Operation: The 80-bit extended precision constant one (1.0) is pushed onto the Stack.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Dest. Reg Full	U	U	1	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	N/A	N/A							
Invalid Op	N/A	N/A							
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Trap/Abort	1	-	-	-	-	-	1

Zero/Infinity: None.

Notes: None.

FLDCW Load CX-83D87 Mode Control Register FLDCW

Syntax: **FLDCW** **<SRC>**

Forms: **FLDCW** **<Memory>**

Operands:

Inst	Source Operand	Encoding			Cycles
FLDCW	2 Bytes	$D9	MD 101 R/M	SIB,DISP	4

Operation: The CX-83D87 Mode Control Register is loaded with the contents of the specified memory location. Note that exceptions indicated in the Status Register due to previous operations may cause an ERROR# trap sequence if the corresponding mask bit in the MCR is set to 0 by this instruction.

Status:

Result of Instruction	C3	C2	C1	C0
Unconditional:	U	U	U	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
None	N/A	N/A	-	-	-	-	-	-	-

Zero/Infinity: None.

Notes: None.

FLDENV Load CX-83D87 Environment FLDENV

Syntax: **FLDENV** **<SRC>**

Forms: **FLDENV** **<Memory>**

Operands:

Inst	Source Operand	Encoding			Cycles
FLDENV	14 or 28 Bytes	SD9	MD 100 R/M	SIB,DISP	22

Operation: The CX-83D87 "Environment" is loaded from the memory location specified. The "Environment" consists of the Mode Control Word, the Status Register, and the Tag Word which are loaded into the CX-83D87. The "Environment" also includes the CX-83D87 Instruction Pointer and the CX-83D87 Data Pointer which are loaded into 80386 CPU registers during execution of this instruction.

The format of the "Environment" data structure is dependent on the operating mode of the 80386 CPU and the operand size in effect.

Loading an "Environment" that contains a Status Register field with an exception indicated and an MCR with that exception enabled causes an ERROR# trap sequence when the next WAIT or exception checking CX-83D87 instruction is executed.

The ERROR# signal is unconditionally de-asserted while the "Environment" data is loaded. If the newly loaded "Environment" calls for an exception trap, ERROR# will be asserted upon completion of the "Environment" data transfer. A subsequent WAIT or exception checking instruction will execute a trap sequence.

32-bit Protected Mode:

31		15 0
Reserved		Mode Control Word
Reserved		Status Word
Reserved		Tag Word
Instruction Pointer Offset		
00000	Opcode(10:0)	Code Segment Selector
Data Operand Offset		
Reserved		Operand Seg Selector

32-bit Real Mode:

31		15		0
Reserved		Mode Control Word		
Reserved		Status Word		
Reserved		Tag Word		
Reserved		Instruction Ptr(15:0)		
0000	Instruction Ptr(31:16)	0	Opcode(10:0)	
Reserved		Operand Ptr(15:0)		
0000	Operand Ptr(31:16)	0000	0000	0000

FLDENV Load CX-83D87 Environment FLDENV

16-bit Protected Mode:

15 0
Mode Control Word
Status Word
Tag Word
Instruction Ptr Offset
Code Segment Selector
Data Operand Offset
Operand Seg Selector

Status:

Result of Instruction	C3	C2	C1	C0
Loaded from Memory:	M	M	M	M

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Loaded	N/A	N/A	M	M	M	M	M	M	M

Zero/Infinity: None.

Notes: None.

FLDL2E Load Floating Constant= $Log_2(e)$ FLDL2E

Syntax: **FLDL2E**

Forms: **FLDL2E**

Operands:

Inst	Source Operand	Encoding			Cycles
FLDL2E	80-bit Const.	SD9	11 101 010		7

Operation: The 80-bit extended precision constant approximating $Log_2(e)$ is pushed onto the Stack. The constant is rounded according to the RC mode in effect.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Dest. Reg Full	U	U	1	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	N/A	N/A							
Invalid Op	N/A	N/A							
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Trap/Abort	1	-	-	-	-	-	1

Zero/Infinity: None.

Notes: None.

FLDL2T Load Floating Constant= $Log_2(10)$ FLDL2T

Syntax: **FLDL2T**

Forms: **FLDL2T**

Operands:

Inst	Source Operand	Encoding			Cycles
FLDL2T	80-bit Const.	SD9	11 101 001		8

Operation: The 80-bit extended precision constant approximating $Log_2(10)$ is pushed onto the Stack. The constant is rounded according to the RC mode in effect.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Dest. Reg Full	U	U	1	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	N/A	N/A							
Invalid Op	N/A	N/A							
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Trap/Abort	1	-	-	-	-	-	1

Zero/Infinity: None.

Notes: None.

FLDLG2 Load Floating Constant= $\text{Log}_{10}(2)$ FLDLG2

Syntax: **FLDLG2**

Forms: **FLDLG2**

Operands:

Inst	Source Operand	Encoding			Cycles
FLDLG2	80-bit Const.	$D9	11 101 100		8

Operation: The 80-bit extended precision constant approximating $\text{Log}_{10}(2)$ is pushed onto the Stack. The constant is rounded according to the RC mode in effect.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Dest. Reg Full	U	U	1	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	N/A	N/A							
Invalid Op	N/A	N/A							
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Trap/Abort	1	-	-	-	-	-	1

Zero/Infinity: None.

Notes: None.

FLDLN2 Load Floating Constant= Ln(2) FLDLN2

Syntax: **FLDLN2**

Forms: **FLDLN2**

Operands:

Inst	Source Operand	Encoding			Cycles
FLDLN2	80-bit Const.	SD9	11 101 101		6

Operation: The 80-bit extended precision constant approximating Ln(2) is pushed onto the Stack. The constant is rounded according to the RC mode in effect.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Dest. Reg Full	U	U	1	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	N/A	N/A							
Invalid Op	N/A	N/A							
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Trap/Abort	1	-	-	-	-	-	1

Zero/Infinity: None.

Notes: None.

FLDPI — Load Floating Constant= π. — FLDPI

Syntax: **FLDPI**

Forms: **FLDPI**

Operands:

Inst	Source Operand	Encoding			Cycles
FLDPI	80-bit Const.	SD9	11 101 011		6

Operation: The 80-bit extended precision constant approximating π (Pi) is pushed onto the Stack. The constant is rounded according to the RC mode in effect.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Dest. Reg Full	U	U	1	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	N/A	N/A							
Invalid Op	N/A	N/A							
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Trap/Abort	1	-	-	-	-	-	1

Zero/Infinity: None.

Notes: None.

FLDZ Load Floating Constant= 0.0. FLDZ

Syntax: **FLDZ**

Forms: **FLDZ**

Operands:

Inst	Source Operand		Encoding		Cycles
FLDZ	80-bit Const.	SD9	11 101 110		4

Operation: The 80-bit extended precision constant zero (0.0) is pushed onto the Stack.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Dest. Reg Full	U	U	1	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	N/A	N/A							
Invalid Op	N/A	N/A							
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Trap/Abort	1	-	-	-	-	-	1

Zero/Infinity: None.

Notes: None.

FMUL Floating Point Multiply FMUL

Syntax: **FMUL ((<DST>,)<SRC>)**

Forms:
FMUL <TOS>,<Memory>
FMUL <TOS>,<Register>
FMULP <Register>,<TOS>

Operands:

Inst	Source Operand	Encoding			Cycles
FIMUL	16-bit Integer	$DE	MD 001 R/M	SIB,DISP	16
FIMUL	32-bit Integer	$DA	MD 001 R/M	SIB,DISP	16
FMUL	32-bit Real	$D8	MD 001 R/M	SIB,DISP	16
FMUL	64-bit Real	$DC	MD 001 R/M	SIB,DISP	18
FMUL	80-bit Register	$D8	11 001 REG		10
FMUL	Top of Stack	$DC	11 001 REG		10
FMULP	Top of Stack	$DE	11 001 REG		10

Operation: The source and destination operands are fetched. The source is converted to extended precision format if necessary. The operands are multiplied and the result is normalized and rounded according to the RC mode in effect at the precision specified by the PC mode bits. The result is stored in the destination register. When the "pop" form is used, the top of stack is popped.

Status:

Result of Instruction		C3	C2	C1[1]	C0
Normal Execution:		U	U	0	U
Register Error:	Dest. Reg Full	U	U	1	U
	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	Masked	R(∞)	-	-	-	1	-	-	-
	Unmasked	Round & Scale	-	-	-	1	-	-	-
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

FMUL Floating Point Multiply FMUL

Zero/Infinity:

OP1	OP2	Result	OP1	OP2	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[2]	+∞	+0	Inv. Op.
+X	-Y	-0[2]	+∞	-0	Inv. Op.
-X	+Y	-0[2]	-∞	+0	Inv. Op.
-X	-Y	+0[2]	-∞	-0	Inv. Op.

Notes:

1. After a Precision Exception the "C1" status bit indicates whether rounding was away from zero.

2. For cases in which X times Y produces extreme underflow and Underflow Exception is masked, the result is denormalized to zero.

FNOP No Operation FNOP

Syntax: **FNOP**

Forms: **FNOP**

Operands:

Inst	Source Operand		Encoding		Cycles
FNOP	None	SD9	MD 010 000		4

Operation: No operation is performed in the CX-83D87.

Status:

Result of Instruction	C3	C2	C1	C0
Unconditional:	U	U	U	U

Exceptions:

Type	Mode	Result	P	U	O	Z	D	I	S
None	N/A	N/A	-	-	-	-	-	-	-

Zero/Infinity: None.

Notes: None.

FPATAN Function Eval: $\tan^{-1}(\frac{y}{x})$ FPATAN

Syntax: **FPATAN**

Forms: **FPATAN**

Operands:

Inst	Source Operand		Encoding		Cycles
FPATAN	Top of Stack	\$D9	11 110 011		89-125

Operation: The Top of Stack contains the first source operand (x). The next to Top of Stack contains the second source operand (y). The instruction computes (z) in radians such that $z = \tan^{-1}(\frac{y}{x})$. The result (z) is normalized and rounded according to the RC mode in effect. The Top of Stack (x) is popped. The value (z) replaces the contents of the new Top of Stack (y). The source operands (x) and (y) are unrestricted in range.

The result (z) falls in the range $-\pi \le z \le +\pi$ according to the following table:

Sgn(y)	Sgn(x)	$\left\lvert\frac{y}{x}\right\rvert$	Result
+	+	<1	$0 < z < \frac{\pi}{4}$
+	+	>1	$\frac{\pi}{4} < z < \frac{\pi}{2}$
+	-	>1	$\frac{\pi}{2} < z < \frac{3\pi}{4}$
+	-	<1	$\frac{3\pi}{4} < z < \pi$
-	+	<1	$0 > z > -\frac{\pi}{4}$
-	+	>1	$-\frac{\pi}{4} > z > -\frac{\pi}{2}$
-	-	>1	$-\frac{\pi}{2} > z > -\frac{3\pi}{4}$
-	-	<1	$-\frac{3\pi}{4} > z > -\pi$

Status:

Result of Instruction		C3	C2	C1[1]	C0
Normal completion:		U	U	0	U
Register Error:	Source Reg Empty	U	U	0	U

FPATAN Function Eval: $\mathrm{Tan}^{-1}(\frac{y}{x})$ FPATAN

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

y	x	Result
$y = +0$	$+\infty \geq x \geq +0$	$z = +0$
$y = -0$	$+\infty \geq x \geq +0$	$z = -0$
$y = +0$	$-\infty \leq x \leq -0$	$z = +\pi$
$y = -0$	$-\infty \leq x \leq -0$	$z = -\pi$
$y > +0$	$x = 0$	$z = +\frac{\pi}{2}$
$y > +0$	$x = +\infty$	$z = +0$
$y > +0$	$x = -\infty$	$z = +\pi$
$y < -0$	$x = 0$	$z = -\frac{\pi}{2}$
$y < -0$	$x = +\infty$	$z = -0$
$y < -0$	$x = -\infty$	$z = -\pi$
$y = +\infty$	$-\infty < x < +\infty$	$z = +\frac{\pi}{2}$
$y = +\infty$	$x = +\infty$	$z = +\frac{\pi}{4}$
$y = +\infty$	$x = -\infty$	$z = +\frac{3\pi}{4}$
$y = -\infty$	$-\infty < x < +\infty$	$z = -\frac{\pi}{2}$
$y = -\infty$	$x = +\infty$	$z = -\frac{\pi}{4}$
$y = -\infty$	$x = -\infty$	$z = -\frac{3\pi}{4}$

Notes: 1. After a Precision Exception the "C1" status bit indicates whether rounding was away from zero.

FPREM Floating Point Remainder (NON-IEEE) FPREM

Syntax: **FPREM**

Forms: **FPREM**

Operands:

Inst	Source Operand	Encoding			Cycles
FPREM	Top of Stack	$D9	11 111 000		49

Operation: The Top of Stack contains the source operand (x). The next to Top of Stack contains the operand (y). The remainder (z) is calculated such that z= x - y*q where (q) is the quotient $q=\frac{x}{y}$ obtained by chopping the exact value toward zero. The result (z) is replaces the contents of the Top of Stack.

FPREM will reduce the exponent of the argument (x) by 63 or more in each pass. The C2 status bit indicates whether the result has been reduced completely, i.e., whether $|z|<|y|$. When completely reduced, the quotient (q) modulo 8 may be read from the condition register.

Status:

Result of Instruction		C3	C2	C1	C0
Reduction Incomplete:		0	1	0	0
Reduction Complete:(q)mod8=	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
Register Error:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	Masked	Denormal	-	-	-	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

FPREM Floating Point Remainder (NON-IEEE) FPREM

Zero/Infinity:

x	y	Result
$x = 0$	$y = 0$	Invalid Op.
$x \neq 0$	$y = 0$	Invalid Op.
$x = -0$	$y \neq 0$	-0
$x = +0$	$y \neq 0$	+0
$x = \infty$	–	Invalid Op.
$-\infty < x < +\infty$	$y = \infty$	x; $q = 0$

Notes: The sign of the remainder is the same as the sign of (x). If (y) evenly divides (x) the remainder is zero.

FPREM1 Floating Point Remainder (IEEE) FPREM1

Syntax: **FPREM1**

Forms: **FPREM1**

Operands:

Inst	Source Operand	Encoding			Cycles
FPREM1	Top of Stack	SD9	11 110 101		50

Operation: The Top of Stack contains the source operand (x). The next to Top of Stack contains the operand (y). The remainder (z) is calculated such that $z = x - y{\cdot}q$ where (q) is the quotient $q = \frac{x}{y}$ obtained by rounding the exact value to nearest/even. The result (z) replaces the contents of the Top of Stack. This instruction is compatible with the IEEE 754-1985 specification.

FPREM1 will reduce the exponent of the argument (x) by 63 or more in each pass. The C2 status bit indicates whether the result has been reduced completely, i.e., whether $|z| \le |\frac{y}{2}|$. When completely reduced, the quotient (q) modulo 8 may be read from the condition register.

Status:

Result of instruction		C3	C2	C1	C0
Reduction Incomplete:		0	1	0	0
Reduction Complete:(q)mod8=	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
Register Error:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	Masked	Denormal	-	-	-	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

FPREM1 Floating Point Remainder (IEEE) FPREM1

Zero/Infinity:

x	y	Result
$x = 0$	$y = 0$	Invalid Op.
$x \neq 0$	$y = 0$	Invalid Op.
$x = -0$	$y \neq 0$	-0
$x = +0$	$y \neq 0$	+0
$x = \infty$	–	Invalid Op.
$-\infty < x < +\infty$	$y = \infty$	x; q = 0

Notes: The sign of the remainder is not necessarily the same as the sign of (x).
If (y) evenly divides (x) the remainder is zero.

FPTAN Function Eval: Tan(x) FPTAN

Syntax: **FPTAN**

Forms: **FPTAN**

Operands:

Inst	Source Operand	Encoding			Cycles
FPTAN	Top of Stack	$D9	11 110 010		5-83

Operation: The Top of Stack contains the source operand (x). The instruction requires (x) in radians and returns (y) such that y= tan(x). The result (y) is normalized and rounded according to the RC mode in effect. The value (y) replaces the contents of the Top of Stack. The value 1 is pushed onto the stack into a new Top of Stack register. The source operand (x) must be in the range $|x| \leq 2^{63}$.

Status:

Result of Instruction		C3	C2	C1[1]	C0
Incomplete Reduction of (x):		U	1	0	U
Normal completion:		U	0	0	U
Register Error:	Dest Reg Full	U	U	1	U
	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN[2]	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN[2]	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

x	Result
+0	y=+0
-0	y=-0
+∞	Invalid Op.
-∞	Invalid Op.

Notes: 1. After a Precision Exception the 'C1' status bit indicates whether rounding was away from zero.

2. The QNaN replaces both (x) and (y).

FRNDINT Round to Integer FRNDINT

Syntax: **FRNDINT**

Forms: **FRNDINT**

Operands:

Inst	Source Operand	Encoding			Cycles
FRNDINT	Top of Stack	$D9	11 111 100		6

Operation: The contents of the Top of Stack are rounded to an integer. The rounding operation to be performed is specified by the RC mode bits of the control word.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:	Round Down	U	U	0	U
	Round Up	U	U	1	U
Register Error:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op¹	Masked	QNaN	0	-	-	-	-	-	1
	Unmasked	Unaltered	0	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

TOS	Result
+0	+0
-0	-0
+∞	+∞
-∞	-∞

Notes: 1. If TOS contains an unsupported, the Invalid Exception is caused.

FRSTOR Load CX-83D87 Environment & Registers FRSTOR

Syntax: **FRSTOR** **<DST>**

Forms: **FRSTOR** **<Memory>**

Operands:

Inst	Source Operand	Encoding			Cycles
FRSTOR	94 or 108 Bytes	$DD	MD 100 R/M	SIB,DISP	95

Operation: The CX-83D87 "Environment" and the CX-83D87 registers are loaded from the memory location specified. The "Environment" consists of the Mode Control Word, the Status Register, and the Tag Word which are loaded into the CX-83D87. The "Environment" also includes the CX-83D87 Instruction Pointer and the CX-83D87 Data Pointer which are loaded into 80386 CPU registers during execution of this instruction.

The format of the data structure saved in memory is dependent on the operating mode of the 80386 CPU and the operand size in effect. See FSAVE for complete details.

Restoring an "Environment" that contains a Status Register field with an exception indicated and an MCR with that exception enabled causes an ERROR# trap sequence when the next WAIT or exception checking CX-83D87 instruction is executed.

The ERROR# signal is unconditionally de-asserted while the "Environment" and register data is loaded. If the newly loaded "Environment" calls for an exception trap, ERROR# will be asserted upon completion of all data transfers. A subsequent WAIT or exception checking instruction will execute a trap sequence.

The CX-83D87 registers are loaded from the 80 consecutive byte memory locations following the "Environment" regardless of the 80386 mode or data size in effect.

Status:

Result of Instruction	C3	C2	C1	C0
Loaded from Memory:	M	M	M	M

Exceptions:

Type	Mode	Result	P	U	O	Z	D	I	S
Loaded	N/A	N/A	M	M	M	M	M	M	M

Zero/Infinity: None.

Notes: None.

FSAVE Save CX-83D87 Environment & Registers FSAVE

Syntax: **FSAVE** **<DST>**

Forms: **FSAVE** **<Memory>**

Operands:

Inst	Dest Operand	Encoding			Cycles
FSAVE	94 or 108 Bytes	$DD	MD 110 R/M	SIB,DISP	102

Operation: The CX-83D87 "Environment" and the CX-83D87 registers are saved to the memory location specified. The "Environment" consists of the Mode Control Word, the Status Register, and the Tag Word from the CX-83D87 and the CX-83D87 Instruction Pointer and CX-83D87 Data Pointer from the 80386 CPU internal registers.

The CX-83D87 is set to its condition following an FINIT. The CX-83D87 Mode Control Register is set to $037F$_{16}$, the CX-83D87 Status Register is reset to $0000, and all data registers are marked empty (Tag word=$FFFF).

This action sets Rounding control to "round to nearest or even" (RC=00), Precision control to 64-bit extended precision (PC=11), and the Top of Stack register number to zero (SSS=000). All exceptions are cleared (=0), all condition codes are cleared (C_0-C_3=0), and all exceptions are masked (=1).

This instruction differs from a hardware reset by setting MCR bit 0 (Invalid Exception Mask) to 1 and Status Register bits 7 & 0 (Error and Invalid Exception) to 0.

The format of the "Environment" data structure is dependent on the operating mode of the 80386 CPU and the operand size in effect. See below for complete details. The CX-83D87 data registers are saved to consecutive memory locations following the "Environment" block regardless of the 80386 mode or data size in effect.

Status:

Result of Instruction	C3	C2	C1	C0
Unchanged	0	0	0	0

Exceptions: None.

Zero/Infinity: None.

Notes: None.

FSAVE Save CX-83D87 Environment & Registers FSAVE

32-bit Protected Mode:

31			15	0
	Reserved		Mode Control Word	00
	Reserved		Status Word	04
	Reserved		Tag Word	08
	Instruction Pointer Offset			0C
00000	Opcode(10:0)		Code Segment Selector	10
	Data Operand Offset			14
	Reserved		Operand Seg Selector	18
	R0 Significand(31:0)			1C
	R0 Significand(63:32)			20
	R1 Significand(15:0)	S	R0 Exponent	24
	R1 Significand(47:16)			28
S	R1 Exponent		R1 Significand(63:48)	2C
	R2 Significand(31:0)			30
	R2 Significand(63:32)			34
	R3 Significand(15:0)	S	R2 Exponent	38
	R3 Significand(47:16)			3C
S	R3 Exponent		R3 Significand(63:48)	40
	R4 Significand(31:0)			44
	R4 Significand(63:32)			48
	R5 Significand(15:0)	S	R4 Exponent	4C
	R5 Significand(47:16)			50
S	R5 Exponent		R5 Significand(63:48)	54
	R6 Significand(31:0)			58
	R6 Significand(63:32)			5C
	R7 Significand(15:0)	S	R6 Exponent	60
	R7 Significand(47:16)			64
S	R7 Exponent		R7 Significand(63:48)	68

FSAVE Save CX-83D87 Environment & Registers FSAVE

32-bit Real Mode:

31		15		0	
	Reserved		Mode Control Word		00
	Reserved		Status Word		04
	Reserved		Tag Word		08
	Reserved		Instruction Ptr(15:0)		0C
0000	Instruction Ptr(31:16)	0	Opcode(10:0)		10
	Reserved		Operand Ptr(15:0)		14
0000	Operand Ptr(31:16)	0000	0000	0000	18
	R0 Significand(31:0)				1C
	R0 Significand(63:32)				20
	R1 Significand(15:0)	S	R0 Exponent		24
	R1 Significand(47:16)				28
S	R1 Exponent		R1 Significand(63:48)		2C
	R2 Significand(31:0)				30
	R2 Significand(63:32)				34
	R3 Significand(15:0)	S	R2 Exponent		38
	R3 Significand(47:16)				3C
S	R3 Exponent		R3 Significand(63:48)		40
	R4 Significand(31:0)				44
	R4 Significand(63:32)				48
	R5 Significand(15:0)	S	R4 Exponent		4C
	R5 Significand(47:16)				50
S	R5 Exponent		R5 Significand(63:48)		54
	R6 Significand(31:0)				58
	R6 Significand(63:32)				5C
	R7 Significand(15:0)	S	R6 Exponent		60
	R7 Significand(47:16)				64
S	R7 Exponent		R7 Significand(63:48)		68

FSAVE Save CX-83D87 Environment & Registers FSAVE

16-bit Protected Mode:

15	0	
	Mode Control Word	00
	Status Word	02
	Tag Word	04
	Instruction Ptr Offset	06
	Code Segment Selector	08
	Data Operand Offset	0A
	Operand Seg Selector	0C
	R0 Significand(15:0)	0E
	R0 Significand(31:16)	10
	R0 Significand(47:32)	12
	R0 Significand(63:48)	14
S	R0 Exponent	16
	Repeat for R1	18
	Repeat for R2	22
	Repeat for R3	2C
	Repeat for R4	36
	Repeat for R5	40
	Repeat for R6	4A
	R7 Significand(15:0)	54
	R7 Significand(31:16)	56
	R7 Significand(47:32)	58
	R7 Significand(63:48)	5A
S	R7 Exponent	5C

FSCALE Floating Multiply by 2^n FSCALE

Syntax: **FSCALE**

Forms: **FSCALE**

Operands:

Inst	Source Operand	Encoding			Cycles
FSCALE	Top of Stack	$D9	11 111 101		8

Operation: The contents of the Top of Stack (x) are multiplied by 2^n where (n) is the contents of the next to Top of Stack. The result $y= x^* 2^n$ is normalized and rounded according to the RC mode in effect. The result (y) replaces the contents of the Top of Stack. The value (n) is an integer determined by chopping actual contents of the next to Top of Stack toward zero.

Status:

Result of Instruction		C3	C2	C1[1]	C0
Normal Execution:		U	U	0	U
Register Error:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale[2]	-	-	1	-	-	-	-
Overflow	Masked	R(∞)	-	-	-	1	-	-	-
	Unmasked	Round & Scale[2]	-	-	-	1	-	-	-
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

x	n	Result
+0	-∞	+0
-0	-∞	-0
0	+∞	Inval. Op.
+∞	-∞	Inval. Op.
-∞	-∞	Inval. Op.
+∞	$-\infty<n\leq+\infty$	+∞
-∞	$-\infty<n\leq+\infty$	-∞
$x\neq0$	+∞	sgn(x)*∞
$x\neq0$	-∞	sgn(x)*0

FSCALE Floating Multiply by 2^n FSCALE

Notes:

1. After a Precision Exception the "C1" status bit indicates whether rounding was away from zero.

2. In the event of "massive" over/underflow in which exponent adjustment by -/+ 24,576 does not yield a normal result, the instruction returns a signed ∞ or zero respectively.

FSIN — Function Evaluation: Sin(x). — FSIN

Syntax: **FSIN**

Forms: **FSIN**

Operands:

Inst	Source Operand	Encoding			Cycles
FSIN	Top of Stack	SD9	11 111 110		5-63

Operation: The Top of Stack contains the source operand (x). The instruction requires (x) in radians and returns (y) such that y= sin(x). The result (y) is normalized and rounded according to the RC mode in effect. The value (y) replaces the contents of the Top of Stack. The source operand (x) must be in the range $|x| \le 2^{63}$.

Status:

Result of Instruction		C3	C2	C1[1]	C0
Incomplete Reduction of (x):		U	1	0	U
Normal completion:		U	0	0	U
Register Error:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

x	Result
+0	+0
-0	-0
+∞	Invalid Op.
-∞	Invalid Op.

Notes: 1. After a Precision Exception the "C1" status bit indicates whether rounding was away from zero.

FSINCOS Function Eval: Sin(x) & Cos(x) FSINCOS

Syntax: **FSINCOS**

Forms: **FSINCOS**

Operands:

Inst	Source Operand	Encoding			Cycles
FSINCOS	Top of Stack	SD9	11 111 011		5-104

Operation: The Top of Stack contains the source operand (x). The instruction requires (x) in radians and returns (y) and (z) such that y= sin(x) and z= cos(x). The results (y) & (z) are normalized and rounded according to the RC mode in effect. The value (y) replaces the contents of the Top of Stack. The value (z) is pushed onto the stack into a new Top of Stack register. The source operand (x) must be in the range $|x| \le 2^{63}$.

Status:

Result of Instruction		C3	C2	C1[1]	C0
Incomplete Reduction of (x):		U	1	0	U
Normal completion:		U	0	0	U
Register Error:	Dest Reg Full	U	U	1	U
	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

x	Result
+0	y=+0;z=+1
-0	y=-0;z=+1
+∞	Invalid Op.
-∞	Invalid Op.

Notes: 1. After a Precision Exception the 'C1' status bit indicates whether rounding was away from zero.

FSQRT Floating Point Square Root FSQRT

Syntax: **FSQRT**

Forms: **FSQRT**

Operands:

Inst	Source Operand	Encoding			Cycles
FSQRT	Top of Stack	$D9	11 111 010		26

Operation: The contents of the Top of Stack (x) are replaced by $\sqrt{x}$. The result is normalized and rounded according to the RC mode in effect at the precision specified by the PC mode bits.

Status:

Result of Instruction		C3	C2	C1[1]	C0
Normal Execution:		U	U	0	U
Register Error:	Dest Reg Full	U	U	1	U
	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

x	Result
+0	+0
-0	-0
$-\infty \le x < -0$	Inval. Op.
$+\infty$	$+\infty$

Notes: 1. After a Precision Exception the 'C1' status bit indicates whether rounding was away from zero.

FST Store CX-83D87 Register FST

Syntax: **FST(P) (<DST>)**

Forms: **FST(P) <Memory>**
FST(P) <Register>

Operands:

Inst	Dest Operand	Encoding			Cycles
FIST(P)	16-bit Integer	SDF	MD 01p R/M	SIB,DISP	10
FIST(P)	32-bit Integer	SDB	MD 01p R/M	SIB,DISP	10
FISTP	64-bit Integer	SDF	MD 111 R/M	SIB,DISP	11
FBSTP	18 dig BCD Int.	SDF	MD 110 R/M	SIB,DISP	61
FST(P)	32-bit Real	SD9	MD 01p R/M	SIB,DISP	9
FST(P)	64-bit Real	SDD	MD 01p R/M	SIB,DISP	11
FSTP	80-bit Real	SDB	MD 111 R/M	SIB,DISP	13
FST(P)	80-bit Register	SDD	11 01p REG		4

Operation: The source operand (x) is fetched from the Top of Stack and, if necessary, converted to the destination data format and rounded according to the RC mode in effect. The result is stored in the destination. When the "pop" form is used, the Top of Stack is popped upon completion.

The operand is rounded to the width of the destination according to the RC mode specified. If the destination type is 32-bit or 64-bit real and the operand is zero, ∞, or NaN, the significand and the exponent are chopped and transferred as-is.

Status:

Result of Instruction		C3	C2	C1[1]	C0
Normal Execution:		U	U	0	U
Register Error:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow[2]	Masked	Rounded	-	-	1	-	-	-	-
	Unmasked	Trap/Abort	-	-	1	-	-	-	-
Overflow[2]	Masked	Rounded	-	-	-	1	-	-	-
	Unmasked	Trap/Abort	-	-	-	1	-	-	-
Div by Zero	N/A	N/A							
Denormal	N/A	N/A							
Invalid Op[3]	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Trap/Abort	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Trap/Abort	1	-	-	-	-	-	1

FST Store CX-83D87 Register FST

Zero/Infinity:

x	Result
Empty	Invalid Op.
NaN->Integer	Invalid Op.
∞->Integer	Invalid Op.
|x|>Int Range	Invalid Op.

Notes:

1. After a Precision Exception the "C1" status bit indicates whether rounding was away from zero.

2. Exception can only occur when the destination operand is 32 or 64 bit real format.

3. Storing into an integer or BCD destination when the operand is of greater magnitude than the destination format supports produces this exception.

FSTCW Store CX-83D87 Mode Control Register FSTCW

Syntax: **FSTCW** **<DST>**

Forms: **FSTCW** **<Memory>**

Operands:

Inst	Dest Operand	Encoding			Cycles
FSTCW	2 Bytes	SD9	MD 111 R/M	SIB,DISP	5

Operation: The contents of the CX-83D87 Mode Control Register are stored into the specified memory location.

Status:

Result of Instruction	C3	C2	C1	C0
Unconditional:	U	U	U	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
None	N/A	N/A	-	-	-	-	-	-	-

Zero/Infinity: None.

Notes: None.

FSTENV Load CX-83D87 Environment FSTENV

Syntax: **FSTENV** **<DST>**

Forms: **FSTENV** **<Memory>**

Operands:

Inst	Source Operand	Encoding				Cycles
FSTENV	14 or 28 Bytes	SD9	MD 110 R/M	SIB,DISP		17

Operation: The CX-83D87 "Environment" is saved to the memory location specified. The "Environment" consists of the Mode Control Word, the Status Register, and the Tag Word which are saved from the CX-83D87. The "Environment" also includes the CX-83D87 Instruction Pointer and the CX-83D87 Data Pointer which are saved from 80386 CPU registers during execution of this instruction.

The FSTENV instruction sets all the exception mask bits of the MCR to 1 thereby masking all exceptions. This causes the ERROR# signal to be de-asserted.

The FSTENV instruction is designed for use in exception handlers to help analyze the exception condition. The format of the "Environment" data structure is dependent on the operating mode of the 80386 CPU and the operand size in effect.

32-bit Protected Mode:

31		15	0
Reserved		Mode Control Word	
Reserved		Status Word	
Reserved		Tag Word	
Instruction Pointer Offset			
00000	Opcode(10:0)	Code Segment Selector	
Data Operand Offset			
Reserved		Operand Seg Selector	

32-bit Real Mode:

31		15			0
Reserved		Mode Control Word			
Reserved		Status Word			
Reserved		Tag Word			
Reserved		Instruction Ptr(15:0)			
0000	Instruction Ptr(31:16)	0	Opcode(10:0)		
Reserved		Operand Ptr(15:0)			
0000	Operand Ptr(31:16)	0000	0000	0000	

FSTENV Store CX-83D87 Environment FSTENV

16-bit Protected Mode:

15 0
Mode Control Word
Status Word
Tag Word
Instruction Ptr Offset
Code Segment Selector
Data Operand Offset
Operand Seg Selector

Status:

Result of Instruction	C3	C2	C1	C0
Unconditional:	U	U	U	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
None	N/A	N/A	-	-	-	-	-	-	-

Zero/Infinity: None.

Notes: None.

FSTSW Store CX-83D87 Status Register FSTSW

Syntax: **FSTSW** **<DST>**

Forms: **FSTSW** **<Memory>**

Operands:

Inst	Dest Operand	Encoding			Cycles
FSTSW	2 Bytes	$DD	MD 111 R/M	SIB,DISP	5

Operation: The contents of the CX-83D87 Status Register are stored into the specified memory location.

Status:

Result of Instruction	C3	C2	C1	C0
Unconditional:	U	U	U	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
None	N/A	N/A	-	-	-	-	-	-	-

Zero/Infinity: None.

Notes: None.

FSTSWAX — Store CX-83D87 Status to AX — FSTSWAX

Syntax: **FSTSWAX**

Forms: **FSTSWAX**

Operands:

Inst	Dest Operand	Encoding			Cycles
FSTSWAX	80386 AX Reg	SDF	11 100 000		5

Operation: The contents of the CX-83D87 Status Register are stored into the 80386 AX register. The contents of the AX register may then be transferred to the 80386 flags with the SAHF instruction. The following table shows how the 80386 conditional branch instructions can be used to decode CX-83D87 status reflecting the results of FCOM execution:

80386 Branch	Result of FCOM	C3	C2	C1	C0
JA	DST > SRC	0	0	0	0
JB	DST < SRC	0	0	0	1
JE	DST = SRC	1	0	0	0
JP	Unordered	1	1	0	1

Status:

Result of Instruction	C3	C2	C1	C0
Unconditional:	U	U	U	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
None	N/A	N/A	-	-	-	-	-	-	-

Zero/Infinity: None.

Notes: This instruction transfers the contents of the CX-83D87 Status Register to the AX register before 80386 instruction execution may proceed..

FSUB Floating Point Subtract FSUB

Syntax: **FSUB(R) ((<DST>,)<SRC>)**

Forms:

FSUB(R)	**<TOS>,<Memory>**
FSUB(R)	**<TOS>,<Reg>**
FSUB(R)(P)	**<Reg>,<TOS>**

Operands:

Inst	Source Operand	Encoding			Cycles
FISUB	16-bit Integer	$DE	MD 10r R/M	SIB,DISP	13
FISUB	32-bit Integer	$DA	MD 10r R/M	SIB,DISP	13
FSUB	32-bit Real	$D8	MD 10r R/M	SIB,DISP	13
FSUB	64-bit Real	$DC	MD 10r R/M	SIB,DISP	15
FSUB	80-bit Register	$D8	11 10r REG		6
FSUB	Top of Stack	$DC	11 101 REG		6
FSUBR	Top of Stack	$DC	11 100.REG		6
FSUBP	80-bit Register	$DE	11 101 REG		6
FSUBRP	80-bit Register	$DE	11 100 REG		6

Operation: The source and destination operands are fetched. The source is converted to extended precision format if necessary. The source operand is subtracted from the destination and the result is normalized and rounded according to the RC mode in effect at the precision specified by the PC mode bits. The result is stored in the destination register. When the "pop" form is used, the top of stack is popped.

The "reverse" form causes the destination operand to be subtracted from the source operand

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Invalid Ex:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	Masked	R(∞)	-	-	-	1	-	-	-
	Unmasked	Round & Scale	-	-	-	1	-	-	-
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

FSUB Floating Point Subtract FSUB

Zero/Infinity:

OP1 -	OP2	Result	OP1 -	OP2	Result
+0	+0	R(0)	+∞	+∞	Inv. Op.
-0	-0	R(0)	-∞	-∞	Inv. Op.
+0	-0	+0	+∞	-∞	+∞
-0	+0	-0	-∞	+∞	-∞
+X	+X	R(0)	+∞	X	+∞
-X	-X	R(0)	-∞	X	-∞
			X	-∞	+∞
			X	+∞	-∞

Notes: After a Precision Exception the "C1" status bit indicates whether rounding was away from zero.

FTST Test Top of Stack FTST

Syntax: **FTST**

Forms: **FTST**

Operands:

Inst	Dest Operand	Encoding			Cycles
FTST	Top of Stack	SD9	11 100 100		6

Operation: The contents of the Top of Stack are compared to zero. The condition code results are the same as those produced by the FCOM instruction with the exception of detecting equality to -0. The Top of Stack is the destination and the constant zero is the source.

The result "unordered" is produced when the operand is NaN, unsupported or when Stack Fault occurs.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:	DST > 0	0	0	0	0
	DST < 0	0	0	0	1
	DST = +0	1	0	0	0
	DST = -0	1	0	1	0
	Unordered	1	1	0	1
Register Error:	Source Reg Empty	U	U	0	U

Exceptions: Type Mode Result P U O Z D I S

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Denorm Used	-	-	-	-	-	1	-
Invalid Op¹	Masked	Unaltered	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	Unaltered	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

DST	Result
+0	=0
-0	=0
+∞	>0
-∞	<0

Notes: 1. QNAN operands produce the Invalid Exception in this instruction.

FUCOM Unordered Compare FUCOM

Syntax: **FUCOM(P)** **[[<DST>,]<SRC>]**

Forms: **FUCOM(P)(P)** **<TOS>,<Reg>**

Operands:

Inst	Source Operand	Encoding			Cycles
FUCOM	80-bit Register	\$DD	11 100 REG		4
FUCOMP	80-bit Register	\$DD	11 101 REG		4
FUCOMPP	80-bit Register	\$DE	11 011 001		15

Operation: The source operand is fetched and subtracted from the destination (Top of Stack) and the condition codes are set according to the result. When the "P" form is used, the Top of Stack is popped. The "PP" form compares the Top of Stack and the next to Top of Stack and causes two "pop" operations upon completion.

This instruction has the same effect as the FCOM instruction except that it does not cause the Invalid Exception when one of the operands is a QNaN.

The result "unordered" is produced when the operands are NaNs, unsupported or when Stack Fault occurs.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:	DST > SRC	0	0	0	0
	DST < SRC	0	0	0	1
	DST = SRC	1	0	0	0
	Unordered	1	1	0	1
Register Error:	Source Reg Empty	1	1	0	1

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Denorm Used	-	-	-	-	-	1	-
Invalid Op	Masked	Unordered	-	-	-	-	-	1	-
	Unmasked	Unaltered	-	-	-	-	-	1	-
Register Error:	Masked	Unordered	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

FUCOM Unordered Compare FUCOM

Zero/Infinity:

DST -	SRC	Result	DST -	SRC	Result
+0	+0	=	+∞	+∞	=
-0	-0	=	-∞	-∞	=
+0	-0	=	+∞	-∞	DST>SRC
-0	+0	=	-∞	+∞	DST<SRC
+0	+X	DST<SRC	+∞	X	DST>SRC
-0	+X	DST<SRC	-∞	X	DST<SRC
+0	-X	DST>SRC	X	-∞	DST>SRC
-0	-X	DST>SRC	X	+∞	DST<SRC

Notes: None.

FXAM Report Class of Operand FXAM

Syntax: **FXAM**

Forms: **FXAM**

Operands:

Inst	Source Operand		Encoding		Cycles
FXAM	80-bit Register	$D9	11 100 101		3

Operation: The Top of Stack is always the source operand. The Top of Stack is examined and condition codes are set according to its class as specified below.

Status: The "C1" Status bit indicates the sign of the Top of Stack operand: "0"=positive; "1"=negative.

Contents of TOS	C3	C2	C0
Unsupported	0	0	0
NaN	0	0	1
Normal	0	1	0
Infinity	0	1	1
Zero	1	0	0
Empty[1]	1	0	1
Denormal	1	1	0

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
None	N/A	N/A	-	-	-	-	-	-	-

Zero/Infinity: None.

Notes: 1. If the stack is empty, the result is 'Empty' and the sign is undefined. No exception is generated.

FXCH Exchange Register with TOS FXCH

yntax: **FXCH** **(SRC)**

orms: **FXCH** **<Reg>**

perands:

Inst	Source Operand	Encoding			Cycles
FXCH	80-bit Register	SD9	11 001 REG		4

peration: The contents of the Top of Stack and the source register are exchanged.

tatus:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Invalid Ex:	Source Reg Empty	1	1	0	1

xceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	N/A	N/A							
Invalid Op	N/A	N/A							
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

ero/Infinity:

Operand	Result
Empty	Invalid Ex.

otes: None.

FXTRACT Extract Exponent FXTRACT

Syntax: **FXTRACT**

Forms: **FXTRACT**

Operands:

Inst	Source Operand	Encoding			Cycles
FXTRACT	Top of Stack	SD9	11 110 100		8

Operation: The Top of Stack contains the source operand (x). The exponent field of (x) is converted to an 80-bit extended precision real number (y) and pushed on the stack, i.e. y=INT(Log$_2$(x)) is pushed. The original operand (x) is modified by having its exponent set to zero, i.e. after execution $1.0 \leq |x| < 2.0$. The sign of (x) is preserved.

Status:

Result of Instruction		C3	C2	C1	C0
Normal Execution:		U	U	0	U
Register Error:	Dest Reg Full	U	U	1	U
	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	N/A	N/A							
Underflow	N/A	N/A							
Overflow	N/A	N/A							
Div by Zero	Masked	TOS=0,ST1=-∞	-	-	-	-	1	-	-
	Unmasked	Trap/Abort	-	-	-	-	1	-	-
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

Zero/Infinity:

x	Result
+0	y = +0; x = -∞; Zero Div. Ex.
-0	y = -0; x = -∞; Zero Div. Ex.
-∞	y = -∞; x = +∞
+∞	y = +∞; x = +∞

Notes: None.

YL2X Function Evaluation: y*Log2(x). FYL2X

yntax: **FYL2X**

orms: **FYL2X**

perands:

Inst	Source Operand		Encoding		Cycles
FYL2X	Top of Stack	$D9	11 110 001		6-93

peration: The Top of Stack contains the source operand (x). The next to Top of Stack contains the source operand (y). The function $z=y*\log_2(x)$ is evaluated and the result is normalized and rounded according to the RC mode in effect. The stack is popped and the result (z) is replaces (y) as the new Top of Stack. The source operand (x) must be in the range $0 < x \le +\infty$.

tatus:

Result of Instruction		C3	C2	C1[1]	C0
Normal Execution:		U	U	0	U
Register Error:	Source Reg Empty	U	U	0	U

xceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	Masked	R(∞)	-	-	-	1	-	-	-
	Unmasked	Round & Scale	-	-	-	1	-	-	-
Div by Zero	Masked	TOS=-∞	-	-	-	-	1	-	-
	Unmasked	Trap/Abort	-	-	-	-	1	-	-
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

ero/Infinity:

y	x	Result
--	$x < 0$	Invalid Op.
$y \neq 0$	$x = 0$	Zero Div Ex
$y = 0$	$x = 0$	Invalid Op.
$y = \infty$	$x = 1$	Invalid Op.
$y = \infty$	$x > 1$	y
$y = \infty$	$0 < x < 1$	-y
$y > +0$	$x = \infty$	$+\infty$
$y < -0$	$x = \infty$	$-\infty$
$y = 0$	$x = \infty$	Invalid Op.

FYL2X Function Evaluation: y*Log2(x). FYL2X

Notes: 1. After a Precision Exception the "C1" status bit indicates whether rounding was away from zero.

FYL2XP1 Function Eval: y*Log2(x+1). FYL2XP1

Syntax: **FYL2XP1**

Forms: **FYL2XP1**

Operands:

Inst	Source Operand		Encoding		Cycles
FYL2XP1	Top of Stack	$D9	11 111 001		6-90

Operation: The Top of Stack contains the source operand (x). The next to Top of Stack contains the source operand (y). The function $z=y^*\log_2(x+1)$ is evaluated and the result is normalized and rounded according to the RC mode in effect. The stack is popped and the result (z) replaces (y) as the new Top of Stack. The operand (x) must be in the range

$$\frac{\sqrt{2}}{2}-1 < x < 1-\frac{\sqrt{2}}{2}.$$

Use FYL2XP1 to calculate log2(x) when | x | is close to 1. FYL2XP1 provides greater accuracy than FYL2X in this case. The magnitude of the input argument must be near zero, so be sure to subtract one from (x) before using FYL2XP1.

Status:

Result of Instruction		C3	C2	C1[1]	C0
Normal Execution:		U	U	0	U
Register Error:	Source Reg Empty	U	U	0	U

Exceptions:

Type	Mode	Result	S	P	U	O	Z	D	I
Precision	Masked	Rounded	-	1	-	-	-	-	-
	Unmasked	Rounded	-	1	-	-	-	-	-
Underflow	Masked	Denorm/Zero	-	1	1	-	-	-	-
	Unmasked	Round & Scale	-	-	1	-	-	-	-
Overflow	N/A	N/A							
Div by Zero	N/A	N/A							
Denormal	Masked	Denorm Used	-	-	-	-	-	1	-
	Unmasked	Trap/Abort	-	-	-	-	-	1	-
Invalid Op	Masked	QNaN	-	-	-	-	-	-	1
	Unmasked	Unaltered	-	-	-	-	-	-	1
Register Error:	Masked	QNaN	1	-	-	-	-	-	1
	Unmasked	Unaltered	1	-	-	-	-	-	1

FYL2XP1 Function Eval: $y*\text{Log}_2(x+1)$. FYL2XP1

Zero/Infinity:

y	x	Result
$y \geq +0$	$x = -0$	-0
$y \geq +0$	$x = +0$	$+0$
$y \leq -0$	$x = -0$	$+0$
$y < -0$	$x = +0$	-0
$y = \infty$	$x = 0$	Invalid Op.
$y = \infty$	$x > 0$	y
$y = \infty$	$-1 < x < 0$	$-y$
$y > +0$	$x = \infty$	$+\infty$
$y < -0$	$x = \infty$	$-\infty$
$y = 0$	$x = \infty$	Invalid Op.

Notes: 1. After a Precision Exception the 'C1' status bit indicates whether rounding was away from zero.

4.13 Instruction Execution Times

his section presents two values for CX-83D87 instruction execution times. The Basic :xecution Time (BET) is the number of clock cycles that the CX-83D87 requires to ıccomplish the execution of an instruction and is given in the individual instruction lescriptions. The Intel System Time (IST) is the time required by an CX-83D87/Intel 80386 :ombination which includes the 80386 overhead of instruction setup and operand ransfer. The difference between the BET and the IST is due entirely to 80386 protocol ›verhead. In both cases, no wait states are included and no allowance for DMA ›verhead is included.

he CX-83D87 device can be used with an interface that simultaneously supports both 0386 coprocessor accesses and memory mapped I/O accesses. Using the memory napped scheme, instruction setup, including operand transfer, can usually be ıccomplished by the 80386 in parallel with the previous instruction execution in the CX-3D87. A smart interface can deliver sustained execution rates equal to the BET values.

ST values are obtained by submitting instruction streams consisting of several repetitions ›f a given instruction to the 80386/CX-83D87 pair and measuring the instruction-to-ıstruction delay time. This technique provides performance figures based upon ›bserved in-system throughput. The IST observed in a stream of dissimilar instructions will ary substantially based on the prior and following instructions. This occurs because of ıe different activities in the 80386 during the various instructions and because the ›xecution times of some CX-83D87 instructions have a dependency on the value of the ıput data. Some examples of this are the FSIN, FCOS, FSINCOS instructions. These ıstructions perform argument reduction if the input data magnitude is greater than :pproximately $\frac{5\pi}{4}$. These two factors result in variations of up to 5% among IST ıeasurements.

Mnemonic	Result	Operation	83D87-BET	83D87-IST	'387 IST
F2XM1	TOS ←	$2^{TOS}-1$	14-67	67	344
FABS	TOS ←	|TOS|	4	16	29
FADD	ST(I) ←	ST(I)+TOS	6	16	34
FADD	TOS ←	TOS+ST(I)	6	16	34
FADD	TOS ←	TOS+M.DR	15	21	33
FADD	TOS ←	TOS+M.SR	13	17	29
FADDP	ST(I-1) ←	ST(I)+TOS	6	16	39
FIADD	TOS ←	TOS+M.SI	13	17	53
FIADD	TOS ←	TOS+M.WI	13	17	76
FCHS	TOS ←	TOS*(-1)	4	16	34
FCLEX		Clear Exceptions	4	26	26
FCOM	CC ←	TOS-ST(I)	4	16	29
FCOM	CC ←	TOS-M.DR	13	21	33
FCOM	CC ←	TOS-M.SR	11	17	29
FCOMP	CC ←	TOS-ST(I)	4	16	29
FCOMP	CC ←	TOS-M.DR	13	21	33
FCOMP	CC ←	TOS-M.SR	11	17	29
FCOMPP	CC ←	TOS-ST(1)	4	16	34
FICOM	CC ←	TOS-M.SI	11	17	53
FICOM	CC ←	TOS-M.WI	11	29	70
FICOMP	CC ←	TOS-M.SI	11	17	53
FICOMP	CC ←	TOS-M.WI	11	29	70
FCOS	TOS ←	COS(TOS)	5-97	97	582
FDECSTP	SP ←	SP-1	5	16	35
FDIV	ST(I) ←	ST(I)/TOS	14-25	16-25	74
FDIV	TOS ←	TOS/ST(I)	13-24	16-24	99
FDIV	TOS ←	TOS/M.DR	21-32	21-32	99
FDIV	TOS ←	TOS/M.SR	20-31	20-31	95
FDIVP	ST(I-1) ←	ST(I)/TOS	14-25	16-25	99
FDIVR	TOS ←	ST(I)/TOS	13-24	16-24	94
FDIVR	ST(I) ←	TOS/ST(I)	14-25	16-25	99
FDIVR	TOS ←	M.DR/TOS	22-33	22-33	99
FDIVR	TOS ←	M.SR/TOS	21-32	21-32	95
FDIVRP	ST(I-1) ←	TOS/ST(I)	13-24	16-24	99
FIDIV	TOS ←	TOS/M.SI	20-31	20-31	119
FIDIV	TOS ←	TOS/M.WI	20-31	30-41	136
FIDIVR	TOS ←	M.SI/TOS	21-32	21-32	119
FIDIVR	TOS ←	M.WI/TOS	21-32	30-41	142
FFREE	TAG(I) ←	Empty	5	22	30
FINCSTP	SP ←	SP+1	5	16	35
FINIT	—	Initialize	5	26	46
FLD	TOS ←	ST(I)	4	16	19
FLD	TOS ←	M.DR	12	21	27
FLD	TOS ←	M.SR	10	17	23
FLD	TOS ←	M.XR	11	38	49
FBLD	TOS ←	M.BCD	32	40	271
FILD	TOS ←	M.LI	9	21	51
FILD	TOS ←	M.SI	7	17	41
FILD	TOS ←	M.WI	7	29	58

Mnemonic	Result	Operation	83D87-BET	83D87-IST	'387 IST
FLD1	TOS ←	One	6	16	29
FLDCW	Ctl Word ←	Memory	4	38	42
FLDENV	Env Regs ←	Memory	22	112	123
FLDL2E	TOS ←	$\mathrm{Log}_2(e)$	7-8	16	44
FLDL2T	TOS ←	$\mathrm{Log}_2(10)$	8-9	16	44
FLDLG2	TOS ←	$\mathrm{Log}_{10}(2)$	8-9	16	44
FLDLN2	TOS ←	$\mathrm{Log}_e(2)$	6-7	16	44
FLDPI	TOS ←	π	6-7	16	44
FLDZ	TOS ←	Zero	7	16	29
FMUL	ST(I) ←	ST(I)*TOS	10	16	54
FMUL	TOS ←	TOS*ST(I)	10	16	54
FMULP	ST(I-1) ←	ST(I)*TOS	10	16	59
FMUL	TOS ←	TOS*M.DR	18	21	57
FMUL	TOS ←	TOS*M.SR	16	18	41
FIMUL	TOS ←	TOS*M.SI	16	18	65
FIMUL	TOS ←	TOS*M.WI	16	29	76
FNOP	--	No Operation	4	22	22
FNOP	--	No Operation	4	22	22
FNOP	--	No Operation	4	22	22
FNOP	--	No Operation	4	22	22
FPATAN	TOS ←	$\mathrm{ATAN}(\frac{\mathrm{ST(1)}}{\mathrm{TOS}})$	89-125	125	430
FPREM	TOS ←	$\mathrm{Rem}(\frac{\mathrm{TOS}}{\mathrm{ST(1)}})$	49*	49	134
FPREM1	TOS ←	$\mathrm{Rem}(\frac{\mathrm{TOS}}{\mathrm{ST(1)}})$	50*	50	159
FPTAN	TOS;ST(1) ←	1; TAN(TOS)	5-82	82	394
FRNDINT	TOS ←	Round(TOS)	6	16	74
FRSTOR	--	Restore state	95	362	511
FSAVE	--	Save state	102	444	514
FSCALE	TOS ←	$\mathrm{TOS}*2^{(\mathrm{ST(1)})}$	8	16	79
FSIN	TOS ←	SIN(TOS)	5-63	63	524
FSINCOS	TOS;ST(1) ←	COS;SIN(TOS)	5-104	104	629
FSQRT	TOS ←	$\sqrt{\mathrm{TOS}}$	26	26	134
FST	ST(I) ←	TOS	4	16	19
FST	M.DR ←	TOS	11	38	45
FST	M.SR ←	TOS	9	32	37
FSTP	ST(I-1) ←	TOS	4	16	24
FSTP	M.DR ←	TOS	11	38	45
FSTP	M.SR ←	TOS	9	32	37
FSTP	M.XR ←	TOS	13	44	52
FBSTP	M.BCD ←	TOS	61	61	523
FIST	M.SI ←	TOS	10	32	81
FIST	M.WI ←	TOS	10	34	87
FISTP	M.LI ←	TOS	11	35	83
FISTP	M.SI ←	TOS	10	32	81
FISTP	M.WI ←	TOS	10	34	87

Mnemonic	Result	Operation	83D87-BET	83D87-IST	387 IST
FSTCW	Memory ←	Control word.	5	19	19
FSTENV	Memory ←	Ctl,Status,IP,DP.	17	123	163
FSTSW	Memory ←	Status	5	19	19
FSTSWAX	AX ←	Status	5	16	17
FSUB	ST(I) ←	ST(I)-TOS	6	16	29
FSUB	TOS ←	TOS-ST(I)	6	16	34
FSUBP	ST(I-1) ←	ST(I)-TOS	6	16	32
FSUB	TOS ←	TOS-M.DR	15	21	33
FSUB	TOS ←	TOS-M.SR	13	17	29
FISUB	TOS ←	TOS-M.WI	13	29	70
FISUB	TOS ←	TOS-M.SI	13	17	53
FSUBR	TOS ←	ST(I)-TOS	6	16	29
FSUBR	ST(I) ←	TOS-ST(I)	6	16	34
FSUBRP	ST(I-1) ←	TOS-ST(I)	6	16	34
FSUBR	TOS ←	M.DR-TOS	15	21	33
FSUBR	TOS ←	M.SR-TOS	13	17	29
FISUBR	TOS ←	M.WI-TOS	13	29	76
FISUBR	TOS ←	M.SI-TOS	13	17	53
FTST	CC ←	TOS-0.0	6	16	34
FUCOM	CC ←	TOS-ST(I)	4	16	29
FUCOMP	CC ←	TOS-ST(I)	4	16	49
FUCOMPP	CC ←	TOS-ST(1)	15	16	49
FXAM	CC ←	Class of TOS	3	16	39
FXCH	TOS ↔	ST(I) Exchange	4	16	24
FXTRACT	TOS;ST(1) ←	Signif; Exponent	8	16	74
FYL2X	TOS ←	ST(1)*Log$_2$(TOS)	6-93	93	488
FYL2XP1	TOS ←	ST(1)*Log$_2$(1+TOS)	6-90	90	543

5. Host Processor Interface

The CX-83D87 processor interfaces directly to the Intel 80386 microprocessor bus following the standard numeric processor extension protocol. The interface consists of a 32-bit bidirectional data bus, bus control signals, CX-83D87 status signals, and power connections. The CX-83D87 interface is also easily adapted to other microprocessor buses and communication protocols such as a memory mapped peripheral configuration.

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

5.1 Signal Description

Each paragraph identifies the CX-83D87 signals by name, provides the signal function, the active state, signal direction, and reference signal for each. The signal discussions are arranged in alphabetical order to assist locating the desired topic.

386CLK2 (80386 clock input)

This signal is an input and is used to synchronize the CX-83D87 bus interface to the processor bus. Most of the interface signals are sampled on the rising edge of this clock or driven relative to the rising edge of this clock. This signal also supplies the clock for the internal processor circuitry. The 386CLK2 signal is divided by two to obtain the basic internal clock rate of the CX-83D87. This input accepts either MOS level inputs. The input to this pin must be the same signal that drives the 80386 processor.

387CLK2 (Not Connected)

The CX-83D87 does not use this signal and leaves the pin unconnected to maintain pin compatibility with the 80387.

ADS# (Address Strobe)

The ADS# signal is an input to the CX-83D87 and indicates that the information on NPS1#, NPS2, W/R#, and CMD0# is valid and should be sampled at the next rising edge of 386CLK2. The setup and hold times are referenced to 386CLK2. This signal is normally connected to the 80386 ADS# signal.

BUSY# (Busy Status) output

The BUSY# signal is an output from the CX-83D87 and its active condition indicates that the floating point processor is busy. The signal is referenced to 386CLK2. This signal is normally connected to the BUSY# input of the 80386.

CKM (Not Connected)

The CX-83D87 does not use this signal and leaves the pin unconnected to maintain pin compatibility with the 80387.

CMD0# (Select Command Port)

he CMD0# input indicates that the current CX-83D87 bus cycle is accessing the :ommand port. When inactive the current CX-83D87 bus cycle is accessing the data)ort. This signal is sampled with the rising edge of 386CLK2 when ADS#, NPS1#, NPS2, and .TEN are active. This signal is normally connected to the 80386 A2 output. The setup and)old times are referenced to 386CLK2.

D31-D0 (Data Bus)

hese bidirectional signals are used to transfer information between the host processor ınd the CX-83D87. D31 is the most significant bit of a transfer. These signals are normally :onnected to the D31-D0 pins of the 80386. The timing of these lines is referenced to .86CLK2.

ERROR# (Error Status) Output

he ERROR# output normally reflects the status of the ES bit in the status register. nmediately after reset, it identifies the coprocessor as an 80387 compatible device by)eing active. If ERROR# is going to be asserted during an instruction it will be asserted)rior to BUSY# being made inactive.

NPS1# (Numeric Processor Select) input

he NPS1# input is used as a select signal to the CX-83D87. This signal is sampled .multaneously with ADS# and NPS2 to determine if the current bus cycle is intended for ıe CX-83D87. This signal is normally connected to the M/IO# output of the 80386. Setup ınd hold times are referenced to the rising edge of 386CLK2.

NPS2 (Numeric Processor Select) input

he NPS2 input is used as a select signal to the CX-83D87. This signal is sampled multaneously with ADS# and NPS1# to determine if the current bus cycle is intended for ıe CX-83D87. This signal is normally connected to the A31 output of the 80386. Setup ınd hold times are referenced to the rising edge of 386CLK2.

PEREQ (Processor Extension Request) Output

he PEREQ signal is an output from the CX-83D87 to the 80386 processor. When active, ıis signal indicates that the CX-83D87 is ready for a data transfer with the host processor. Vhen all data transfers have been completed the signal will go inactive. PEREQ will not)e active when BUSY# is inactive. This signal is normally connected to the 80386 PEREQ ıput.

READY# (Bus Ready)

he READY# input is sampled on the rising edge of 386CLK2. The active state of READY# ıdicates that the current bus cycle is being concluded. The CX-83D87 bus interface uses EADY# and ADS# to track the 80386 bus cycles and remain synchronized with the 80386 operation. This signal is normally connected to the same signal that drives the READY# input of the 80386.

READYO# (CX-83D87 Ready)

The READYO# output is activated when the CX-83D87 is ready to conclude a bus cycle. READYO# is referenced to the rising edge of 386CLK2. This signal is normally connected to the READY# input of the 80386 or the READY# generator for the system. READYO# in the CX-83D87 operates independently of BUSY# and PEREQ. Therefore, communication protocols other than the standard 80386 Numeric Processor Extension protocol can be more easily implemented. Memory mapped or I/O mapped protocols can use READYO# as the complete synchronization and handshaking protocol.

RESETIN (System Reset)

The RESETIN input performs a total reset of the CX-83D87. It must remain active for a minimum of TBD input clock periods. The active to inactive transition of RESETIN must be synchronous with 386CLK2 to match internal clock phases with that of the 80386. After RESETIN goes inactive, READYO#, BUSY#, and PEREQ are set to the inactive state and ERROR# is set active. This signal condition indicates to an 80386 that an 80387 compatible numerics coprocessor is connected. At least TBD clock periods must transpire after RESETIN goes inactive before the first CX-83D87 access. This pin is normally connected to the 80386 RESET input.

STEN (Status Enable)

The STEN input enables the functioning of the CX-83D87 when active. All outputs of the CX-83D87 are tri-stated when this signal is inactive. The other input signals are ignored while STEN is inactive. This signal can be used to assist in board level testing by isolating the CX-83D87 from the remainder of the circuit. This signal is normally connected to VCC through a resistor so that it can be pulled inactive during testing.

W/R# (Write or Read)

The W/R# input to the CX-83D87 indicates the direction of the current bus cycle. This signal is sampled simultaneously with ADS#, CMD0#, NPS1# and NPS2 on the rising edge of 386CLK2. This signal is normally connected to the 80386 W/R# output.

5.2 Bus Operation

The CX-83D87 is compatible with the 80386 NPX protocol and can also be operated in a memory mapped or I/O mapped protocol. The Intel NPX coprocessor protocol operation uses the BUSY# and PEREQ signals to ensure that no NPX accesses are generated by the 80386 before the coprocessor is able to complete the transfer. This allows the CX-83D87 to issue READYO# immediately after receiving ADS#. Therefore, the processor bus is available for DMA cycles during the NPX instructions. However, the time required to check these signals contributes to the coprocessor overhead and slows down execution of the NPX instructions.

The mapped protocols increase net system performance by decreasing the protocol overhead. In a mapped protocol configuration the 80386 ignores BUSY# and PEREQ.

nstead, READYO# is used to extend any bus cycles that the CX-83D87 is not ready to complete immediately. The bus is held in a waiting condition during this time and is not available for DMA activity.

he following sections describe the bus activity that occurs for the various categories of numeric instructions. The CX-83D87 is fully synchronous to the 80386 bus operation and upports both pipelined and non-pipelined bus cycles. Examples of coprocessor mode operation and mapped mode operation for the different categories of instructions are provided. The bus cycles run by the 80386 to fetch instructions or to transfer operands to memory are not shown. Also, the 80386 usually does not respond to the BUSY#, ERROR# and PEREQ changes as rapidly as indicated herein. The diagrams shown in this section merely reflect the sequence of occurrences and the method of synchronization; they are not meant to imply actual execution times or bus cycle durations. STEN, NPS1#, and NPS2 are assumed active during this cycle and are not shown. The CMD# and W/R# ignals should be valid during the cycle also.

he following table categorizes the CX-83D87 instructions by their interface characteristics. The *# BUS CYCLES* column includes the instruction opcode transfer and ts operand transfer(s) in the count of bus cycles. The *EARLY WRITE* column indicates that he 80386 may write the opcode even if BUSY# is active. The *USES BUSY#* and *USES PEREQ* columns indicate if the referenced signal is utilized during the instruction execution. A check mark (√) indicates that the signal is active during the instruction, while a dash (-) indicates that the signal is not active during the instruction. The *TYPE* column identifies the category to which the instruction belongs.

Instruction	# Bus Operand	Early Cycles	Uses Write	Uses BUSY#	PEREQ	Type
FCLEX		1	√	–	–	E
FFREE	ST(i)	1	–	√	–	A
Ffunct		1	–	√	–	A
FINIT		1	√	–	–	E
FLD	64-bit real	3	√	√	–	B
FLD	64-bit integer	3	√	√	–	B
FLD	32-bit integer	2	√	√	–	B
FLD	32-bit real	2	√	√	–	B
FLD	80-bit real	4	–	√	√	C
FLD	80-bit BCD	4	–	√	√	C
FLD	16-bit integer	2	–	√	√	C
FLD*const*		1	–	√	–	A
FLDCW	two Bytes	2	–	√	√	C
FLDENV	block pointer	8	–	√	√	C
Fmath	ST(i),ST	1	–	√	–	A
Fmath	ST,ST(i)	1	–	√	–	A
Fmath	64-bit real	3	√	√	–	B
Fmath	64-bit integer	3	√	√	–	B
Fmath	32-bit integer	2	√	√	–	B
Fmath	32-bit real	2	√	√	–	B
Fmath	16-bit integer	2	–	√	√	C
Fops	ST(i)	1	–	√	–	A
FRSTOR	block pointer	28	–	√	√	C
FSAVE	block pointer	28	–	√	√	C
FST	16-bit integer	2	–	√	√	C
FST	32-bit integer	2	–	√	√	C
FST	32-bit real	2	–	√	√	C
FST	64-bit real	3	–	√	√	C
FST	64-bit integer	3	–	√	√	C
FST	80-bit real	4	–	√	√	C
FST	80-bit BCD	4	–	√	√	C
FSTCW	two Bytes	2	–	–	–	D
FSTENV	block pointer	8	–	√	√	C
FSTSW	AX	2	–	–	–	D
FSTSW	two Bytes	2	–	–	–	D

Fmath represents the FADD, FCOM, FCOMP, FDIV, FDIVR, FMUL, FSUB, and FSUBR instructions.
FLD*const* represents the FLD1, FLDL2T, FLDL2E, FLDPI, FLDLG2, FLDLN2, and FLDZ instructions.
Fops represents the FLD ST(i), FXCH ST(i), FNOP, FCHS, FABS, FTST, FXAM, FUCOM, FUCOMP, FUCOMPP, and FCOMPP instructions.
Ffunct represents the F2XM1, FYL2X, FPTAN, FPATAN, FXTRACT, FPREM1, FDECSTP, FINCSTP, FPREM, FYL2XP1, FSQRT, FSINCOS, FRNDINT, FSCALE, FSIN, and FCOS instructions.

5.3 Category A Instructions

The category A instructions are the simplest type of CX-83D87 instruction. These instructions operate on internal registers and have no operand transfers between the 80386 and the CX-83D87. The 80386 coprocessor protocol tests BUSY# and waits for BUSY# to be inactive. The ERROR# signal is then examined. If ERROR# is asserted, the 80386 executes a trap to the coprocessor exception trap routine. If ERROR# is not asserted, the 80386 writes the opcode to the coprocessor and continues to the next instruction. The coprocessor asserts BUSY# during execution of the requested operation. ERROR# will be asserted if an unmasked exception condition occurs during the operation.

Figure 5.2-1a shows the bus signals during two successive category A instructions in the coprocessor interface mode. The first access occurs when the CX-83D87 is idle and starts immediately. The second access is delayed by the 80386 until BUSY# goes inactive.

Figure 5.2-1b shows the bus sequence for three successive category A instructions when the CX-83D87 is memory mapped. The first access occurs when the CX-83D87 is idle and completes immediately. The second access is initiated prior to the CX-83D87 completing the first operation. There are no wait states inserted in the bus cycle since the CX-83D87 can buffer the operation code. The instruction cannot be initiated until the current operation completes. The third access is initiated before the first operation completes. The CX-83D87 holds READYO# inactive until the first operation completes and the second operation starts. The second access shows the capability of the mapped interface to pipeline this category of instruction by completing the operation code transfer while BUSY# is still active.

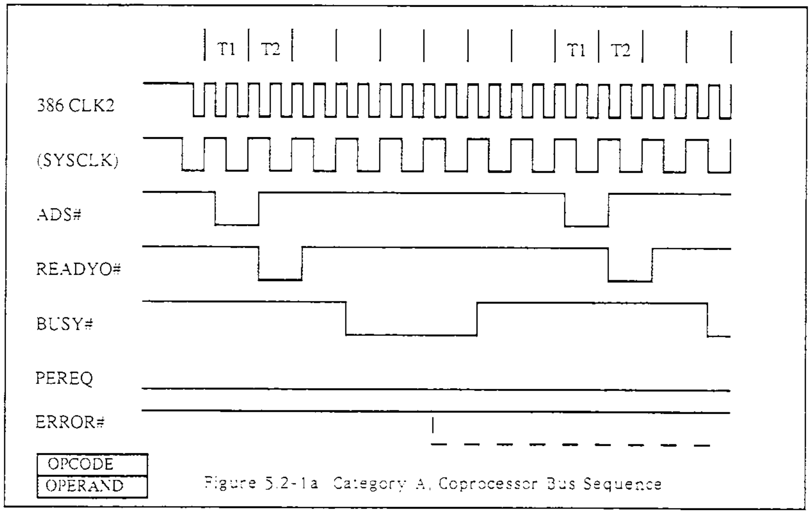


Figure 5.2-1a Category A, Coprocessor Bus Sequence

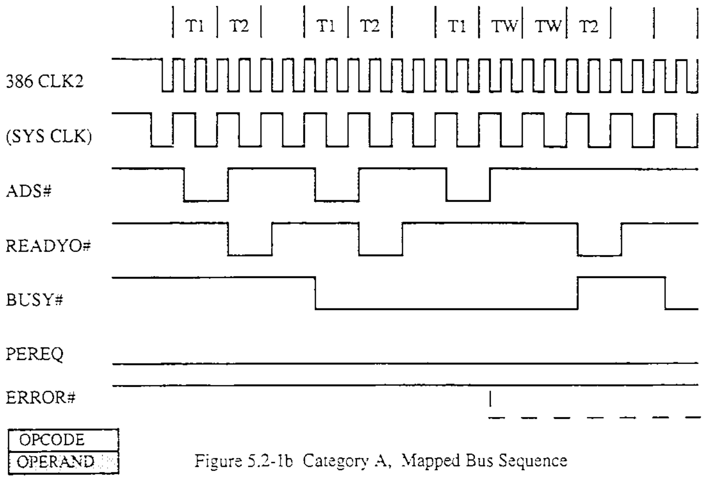


Figure 5.2-1b Category A, Mapped Bus Sequence

5.4 Category B Instructions

The category B instructions require the transfer of one or two 32-bit words from the 80386 to the CX-83D87. The execution of this instruction category is characterized by the 80386 writing the opcode whether or not BUSY# is active. The 80386 then waits for BUSY# to become inactive (if necessary) and examines the ERROR# signal. If ERROR# is asserted, the 80386 executes a trap to the coprocessor exception trap routine. If ERROR# is not asserted, the 80386 then transfers the operand (1 memory cycle for 32-bit operands, 2 memory cycles for 64-bit operands). Since synchronization is accomplished using BUSY# during the operand transfer phase, there is no activity on the PEREQ line. The 80386 then proceeds to the next instruction. The CX-83D87 asserts BUSY# after the operand is accepted and removes BUSY# when the operation is complete. ERROR# is asserted if an unmasked exception condition occurs during the operation.

Figure 5.2-2a shows the bus activity for two successive category B instructions with a 32-bit operand in the coprocessor mode. The initial state of the CX-83D87 is idle allowing the 80386 to immediately transfer both the instruction opcode and the operand. The second instruction is initiated while the CX-83D87 is busy. The opcode transfer is completed but the operand transfer is not initiated by the 80386 until BUSY# goes inactive.

Figure 5.2-2b shows the bus sequence for two successive short integer (32-bit) instructions for an CX-83D87 in the mapped mode that is initially idle. The first instruction and its operand are transferred immediately. The second instruction is initiated while the CX-

3D87 is still busy. The operation code transfers immediately but READYO# is withheld om the operand transfer until the preceding instruction is completed.

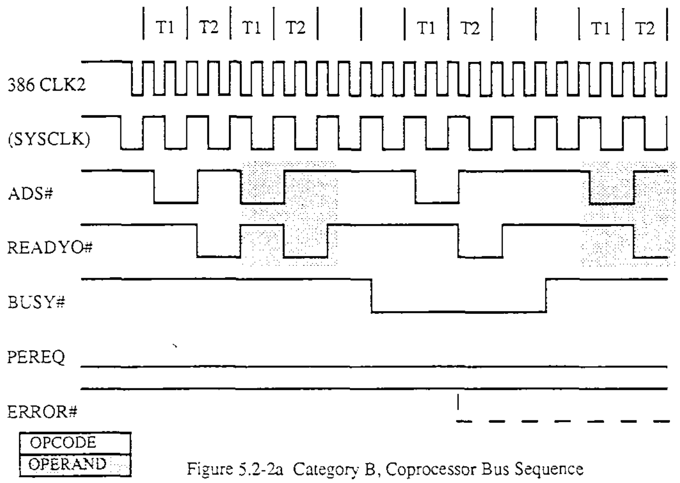


Figure 5.2-2a Category B, Coprocessor Bus Sequence

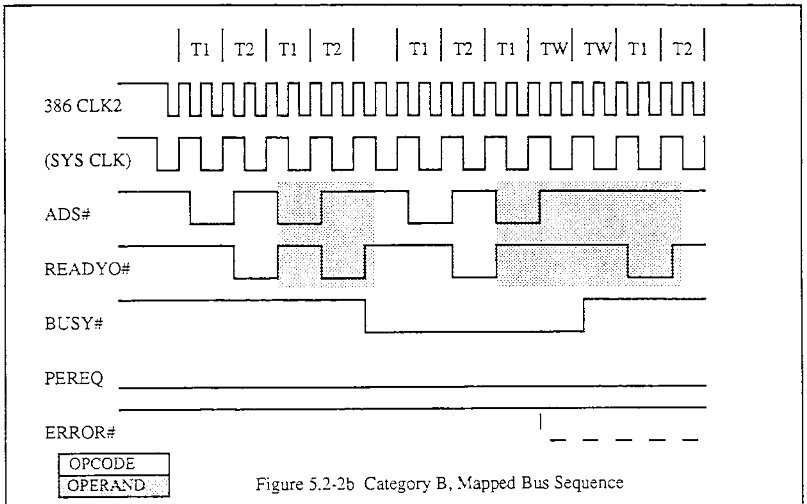


Figure 5.2-2b Category B, Mapped Bus Sequence

5.5 Category C Instructions

The category C instructions are characterized by the use of the PEREQ signal to synchronize the movement of the operand. All store operand instructions and the load operand with 16-bit and 80-bit data values comprise this category. The coprocessor bus activity is characterized by the 80386 waiting for BUSY# to be inactive before writing the opcode to the CX-83D87. The 80386 then waits for BUSY# and PEREQ to be active before transferring the operand.

The operand transfer can take 1, 2, or 3 bus cycles as determined by its size. One bus cycle is required to transfer the 16-bit word integer, 32-bit short integer, and 32-bit single precision real formats. Two bus cycles are used to transfer the 64-bit long integer and 64-bit double precision real formats. Three bus cycles are used to transfer the 80-bit extended real and the 80-bit packed BCD formats.

BUSY# is active and PEREQ is inactive during the time that the CX-83D87 is generating the properly rounded value for the designated format. When the value is ready to be transferred, PEREQ is made active. In the coprocessor operation mode the 80386 will not try to read the value until PEREQ is active. In the mapped operation mode READYO# will be withheld if a read attempt is made before the value is properly formatted.

Figure 5.2-3a shows a coprocessor mode FSTP 64-bit integer with the CX-83D87 initially busy. Figure 5.2-3b shows a mapped mode store of an 80-bit BCD value when the CX-83D87 is initially idle. Figure 5.2-3c shows a mapped mode store of an 80-bit BCD value when the CX-83D87 is initially busy.

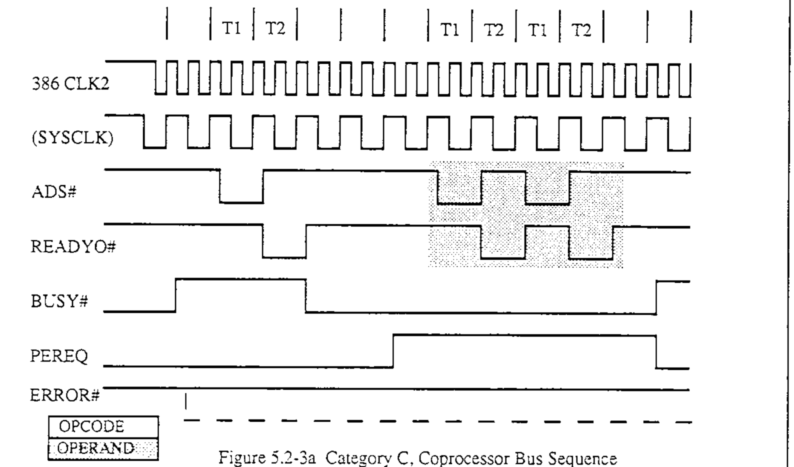


Figure 5.2-3a Category C, Coprocessor Bus Sequence

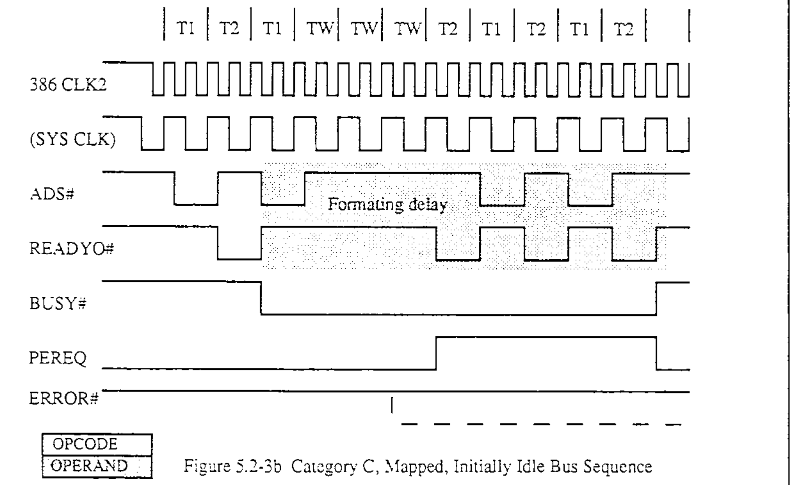


Figure 5.2-3b Category C, Mapped, Initially Idle Bus Sequence

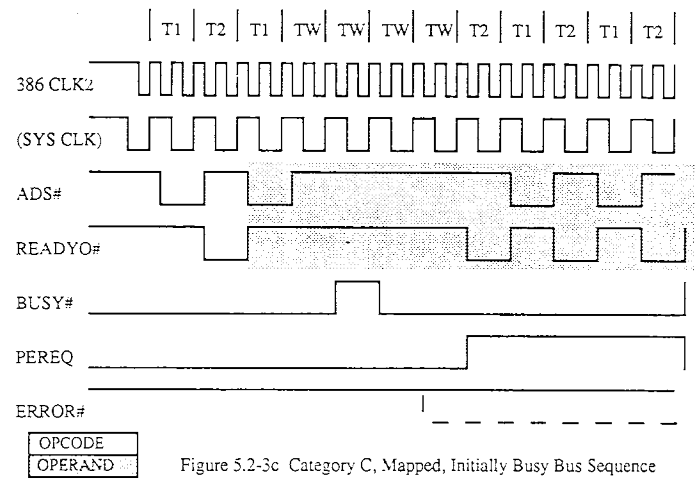


Figure 5.2-3c Category C, Mapped, Initially Busy Bus Sequence

5.6 Category D Instructions

The category D instructions are those operations that wait for BUSY# to be inactive before transferring the instruction opcode and then transfer the operand (if there is one) immediately without using PEREQ. These instructions involve only the store of the status or control registers. The duration of the operation is so short that the BUSY# signal is not asserted. Figure 5.2-4a shows the coprocessor mode synchronization waiting for BUSY# to go inactive. Figure 5.2-4b shows the mapped mode synchronization of READYO# being held inactive until the CX-83D87 is ready to proceed.

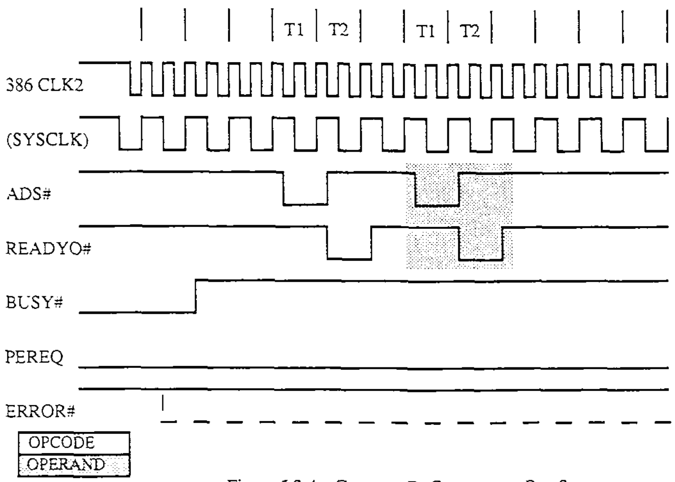


Figure 5.2-4a Category D, Coprocessor Bus Sequence

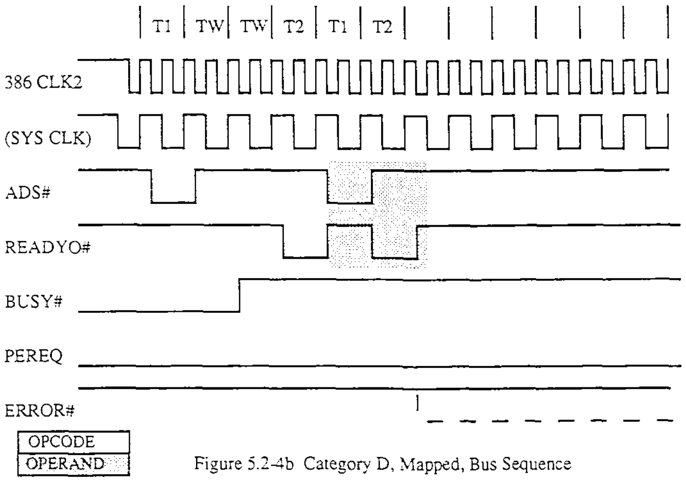


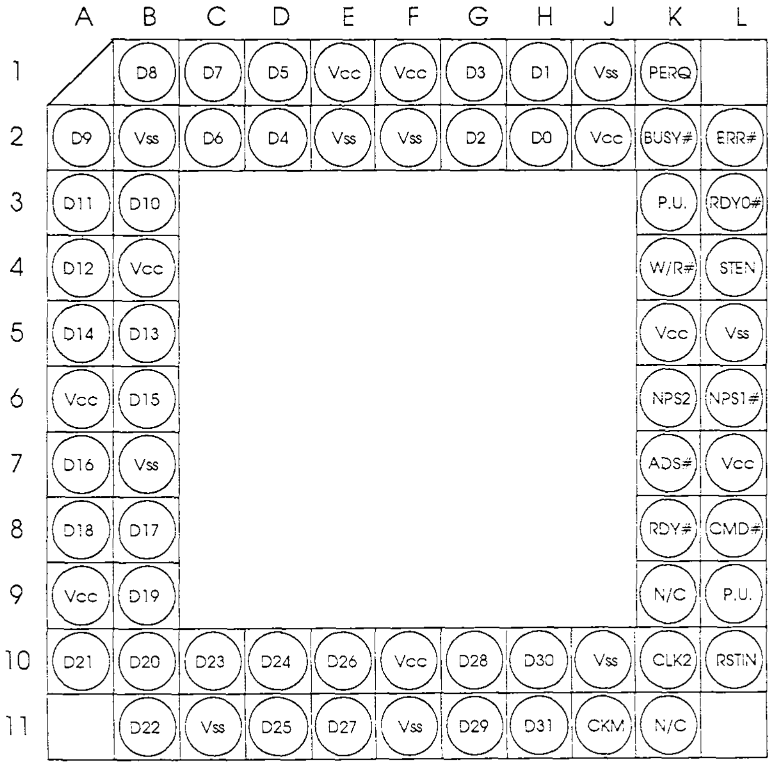
Figure 5.2-4b Category D, Mapped, Bus Sequence

5.7 Category E Instructions

The category E instructions represent those instructions that are executed regardless of the condition of BUSY#. BUSY# will be asserted during the instruction if it is not already asserted. PEREQ is inactive during the execution of these instructions. These instructions operate on the status and control portion of the CX-83D87. The entire instruction consists of writing the operation code to the coprocessor. There are no operands to transfer and no synchronization to be performed.

5. Mechanical Specifications

The CX-83D87 is packaged in a 68 pin pin grid array package. The following diagrams detail the pinout of the CX-83D87 from both the pin side and the top side:



	A	B	C	D	E	F	G	H	J	K	L
1		D8	D7	D5	Vcc	Vcc	D3	D1	Vss	PERQ	
2	D9	Vss	D6	D4	Vss	Vss	D2	D0	Vcc	BUSY#	ERR#
3	D11	D10								P.U.	RDYO#
4	D12	Vcc								W/R#	STEN
5	D14	D13								Vcc	Vss
6	Vcc	D15								NPS2	NPS1#
7	D16	Vss								ADS#	Vcc
8	D18	D17								RDY#	CMD#
9	Vcc	D19								N/C	P.U.
10	D21	D20	D23	D24	D26	Vcc	D28	D30	Vss	CLK2	RSTIN
11		D22	Vss	D25	D27	Vss	D29	D31	CKM	N/C	

Pin Side View

	L	K	J	H	G	F	E	D	C	B	A
1		PERQ	Vss	D1	D3	Vcc	Vcc	D5	D7	D8	
2	ERR#	BUSY#	Vcc	D0	D2	Vss	Vss	D4	D6	Vss	D9
3	RDYO#	P.U.								D10	D11
4	STEN	W/R#								Vcc	D12
5	Vss	Vcc								D13	D14
6	NPS1#	NPS2								D15	Vcc
7	Vcc	ADS#								Vss	D16
8	CMD#	RDY#								D17	D18
9	P.U.	N/C								D19	Vcc
10	RSTIN	CLK2	Vss	D30	D28	Vcc	D26	D24	D23	D20	D21
11		N/C	CKM	D31	D29	Vss	D27	D25	Vss	D22	

Top Side View

he following chart provides a cross reference of signal name to pin coordinates:

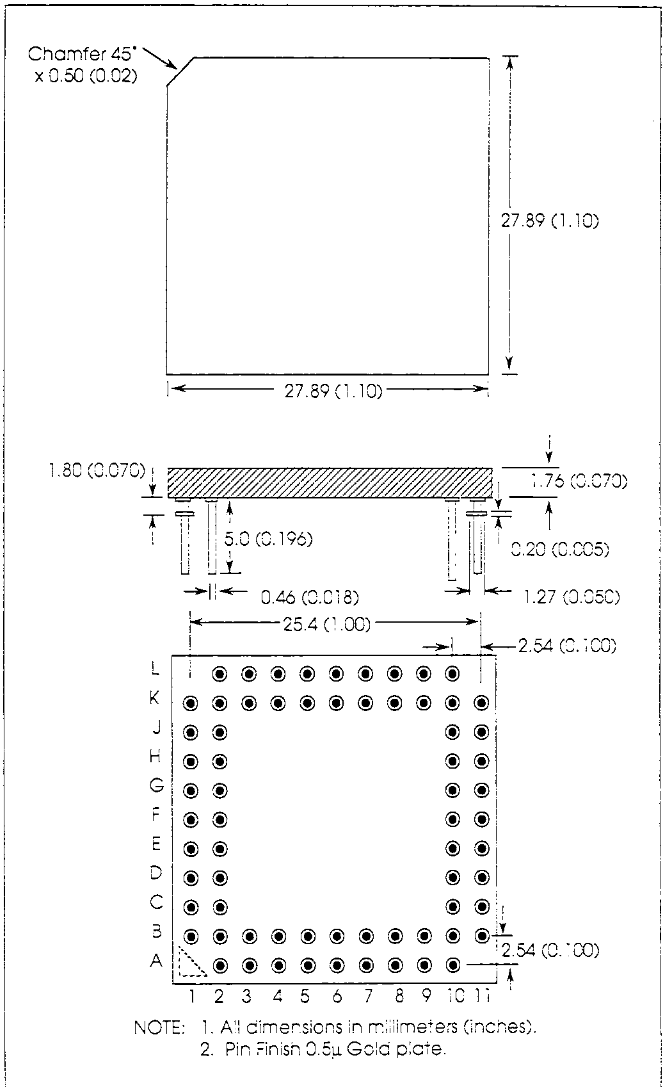
Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
ADS#	K7	D12	A4	D29	G11	Vcc	B4
BUSY#	K2	D13	B5	D30	H10	Vcc	E1
CKM	J11	D14	A5	D31	H11	Vcc	F1
CLK2	K10	D15	B6	ERR#	L2	Vcc	F10
CMD#	L8	D16	A7	NPS1#	L6	Vcc	J2
D0	H2	D17	B8	NPS2	K6	Vcc	K5
D1	H1	D18	A8	N/C	K9	Vcc	L7
D2	G2	D19	B9	N/C	K11	Vss	B2
D3	G1	D20	B10	PERQ	K1	Vss	B7
D4	D2	D21	A10	P.U.	K3	Vss	C11
D5	D1	D22	B11	P.U.	L9	Vss	E2
D6	C2	D23	C10	RSTIN	L10	Vss	F2
D7	C1	D24	D10	RDY#	K8	Vss	F11
D8	B1	D25	D11	RDYO#	L3	Vss	J1
D9	A2	D26	E10	STEN	L4	Vss	J10
D10	B3	D27	E11	Vcc	A6	Vss	L5
D11	A3	D28	G10	Vcc	A9	W/R#	K4

Note: Pins K3 and L9 shown as P.U. ("Pull Up") **must be tied to Vcc** for the device to operate properly.

he Cyrix CX-83D87 is available in both "plastic" and ceramic pin grid array packages. hese packages are described in the following sections.

6.1 Ceramic Pin Grid Array Package

he dimensions for the ceramic package are detailed on the following page.



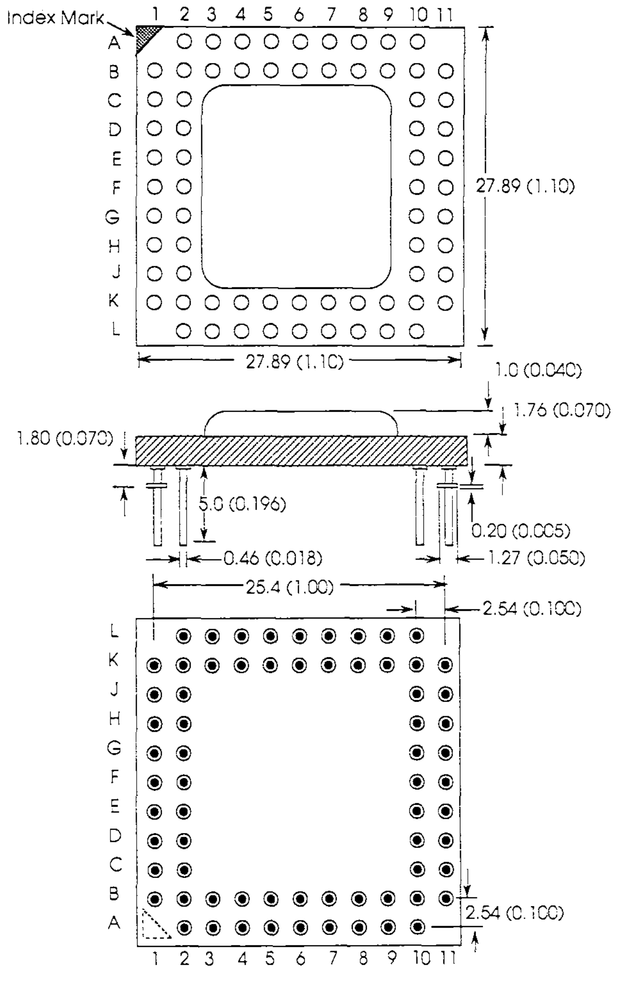
NOTE: 1. All dimensions in millimeters (inches).
2. Pin Finish 0.5µ Gold plate.

6.2 Plastic Pin Grid Array Package

э Cyrix CX-83D87 "plastic" pin grid array package is an alternative to ceramic pin grid ays. It offers superior thermal properties and lower cost although it requires special re in handling when soldered into PCB assemblies. The Cyrix CX-83D87 package uses advanced material called Triazine which combines low moisture absorption aracteristics with high glassivation temperature. Nevertheless, the sudden extreme at of wave soldering or reflow soldering can cause bubbles or fissures in the laminate e to rapid heating of trapped water vapor. This effect can lead to decreased device ability.

-83D87 PPGA devices are shipped in a dessicated, moisture sealed antistatic plastic g. Proper practice for use in soldered assembly dictates storage of the devices in their aled bags until immediately prior to PCB insertion and soldering. If the package seals broken or the devices are exposed to moisture for any reason, they should be baked 24 hours and either moisture sealed or stored in an area of less than 10% RH until used.

ıen not subject to extreme thermal shock (as in socketed assemblies) the PPGA quires no unusual precautions.



NOTE: 1. All dimensions in millimeters (inches).
2. Substrate material Triazine.
3. Pin material Kovar.
4. Pin Finish 1μ 90/10 Solder dip.

Electrical Specifications

7.1 Absolute Maximum Ratings

e following table lists absolute maximum ratings for the CX-83D87 device. Stresses ›yond those listed under "Absolute Maximum Ratings" may cause permanent damage the device. These are stress ratings only and do not imply that operation under any ›nditions other than those listed under "Recommended Operating Conditions" is ›ssible. Exposure to conditions beyond the "Absolute Maximum Ratings" (1) will reduce ›vice reliability and (2) result in premature failure *even when there is no immediately ›parent sign of failure.* Prolonged exposure to conditions at or near "Absolute ›ximum Ratings" may also result in reduced useful life and reliability.

Parameter	Min.	Max.	Units	Notes
›se Temperature	-0°	+100°	°C	Power Applied
›rage Temperature	-65°	+150°	°C	No Bias
›ply Voltage, VCC	-0.5	+6.0	Volts	With respect to VSS
ltage On Any Pin	-0.5	VCC+0.5	Volts	With Respect to VSS
wer Dissipation		1.6	Watts	
›ut Clamp Current, IIK		10	mA	VI<VSS or VI>VCC
›tput Clamp Current, IOK		25	mA	VO<VSS or VO>VCC

7.2 Recommended Operating Conditions

› following table presents the recommended operating conditions for the device:

Parameter	Min.	Max.	Units	Notes
Ambient Temperature	-0°	+70°	°C	Power Applied
c Supply Voltage	+4.75	+5.25	Volts	With respect to VSS
(High Level Input	2.0	VCC	Volts	
Low Level Input	0.0	0.8	Volts	
1 Output Current(High)		-1.0	mA	VOH=VOH(min)
. Output Current (Low)		+4.0	mA	VOL=VOL(max)
Input Clamp Current		±10	mA	VI<VSS or VI>VCC
: Output Clamp Current		±25	mA	VO<VSS or VO>VCC

7.3 DC Electrical Characteristics

Parameter	Min.	Max.	Units	Notes
VCL Clock Input Low	0	0.8	Volts	With respect to VSS
VCH Clock Input High	3.7	Vcc	Volts	
VOL Output Low Voltage		+0.45	Volts	IOL=4.0 mA
VOH Output High Voltage	2.4		Volts	IOH=1.0 mA
ICC Supply Current		300	mA	CLK2=20 Mhz (Typ=200)
ILI Input Leakage		±15	µA	0<VIN<VCC
ILO I/O Leakage		±15	µA	0.45<VO<VCC
CIN Input Capacitance		10	pf	fc=1 Mhz
CO I/O Capacitance		12	pf	fc=1 Mhz
CCLK Clock Capacitance		20	pf	fc=1 Mhz

7.4 20 Mhz Switching Characteristics

he following table summarizes the timing requirements of the CX-83D87-20 device.

Pin	Symbol	Parameter	Min (nsec)	Max (nsec)	Fig.	Notes
LK2	T1	Period	25	TBD	7.1	2.0V
LK2	T2a	High Time	8	TBD		2.0V
LK2	T2b	High Time	5	TBD		3.8V
LK2	T3a	Low Time	8	TBD		2.0V
LK2	T3b	Low Time	6	TBD		0.8V
LK2	T4	Fall Time		8		3.7V To 0.8V
LK2	T5	Rise Time		8		0.8V To 3.7V
EADYO#	T7	Out Delay	3	31	7.2	CL=75 pf
EADYO#	T7	Out Delay	3	27		CL=25 pf
EREQ	T7	Out Delay	5	34		CL=75 pf
USY#	T7	Out Delay	5	29		CL=75 pf
RROR#	T7	Out Delay	5	34		CL=75 pf
31-D0	T8	Out Delay	1	54	7.3	CL=120 pf
31-D0	T10	Setup Time	11			
31-D0	T11	Hold Time	11			
31-D0	T12	Float Time	6	27		CL=120 pf
EREQ	T13	Float Time	1	50		CL=75 pf
USY#	T13	Float Time	1	50		CL=75 pf
RROR#	T13	Float Time	1	50		CL=75 pf
EADYO#	T13	Float Time	1	50		CL=75 pf
DS#, W/R#	T14	Setup Time	21		7.3	
DS#, W/R#	T15	Hold Time	5			
EADY#	T16	Setup Time	12		7.3	
EADY#	T17	Hold Time	4			
MD0#, NPS1#, PS2	T16	Setup Time	19		7.3	
MD0#, NPS1#, PS2	T17	Hold Time	2			
EN	T16	Setup Time	21		7.3	
EN	T17	Hold Time	2			
ESETIN	T18	Setup Time	12		7.4	
ESETIN	T19	Hold Time	4			

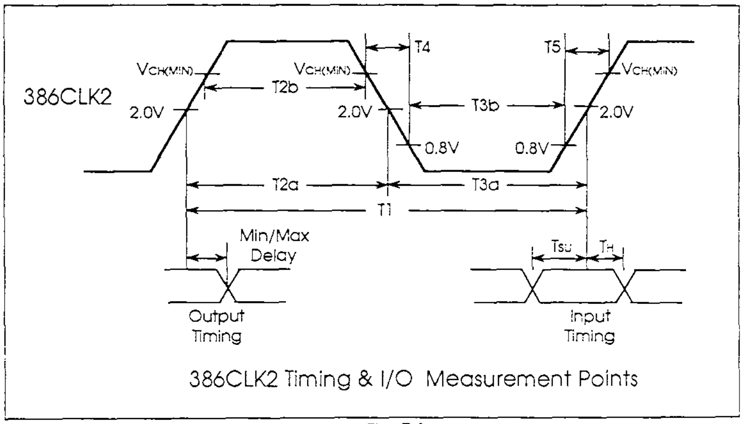

386CLK2
VCH(MIN)
2.0V
T2b
0.8V
T4
T3b
T5
T2a
T3a
T1
Min/Max Delay
Output Timing
TSU
TH
Input Timing
386CLK2 Timing & I/O Measurement Points

Fig. 7.1

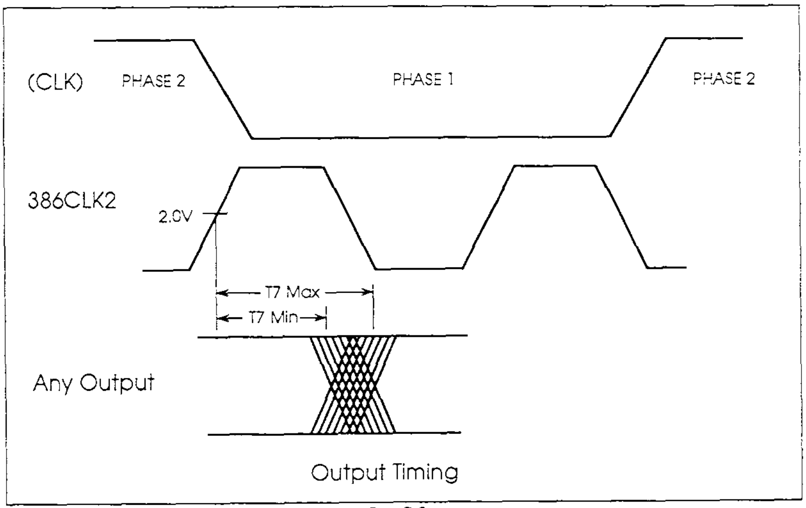
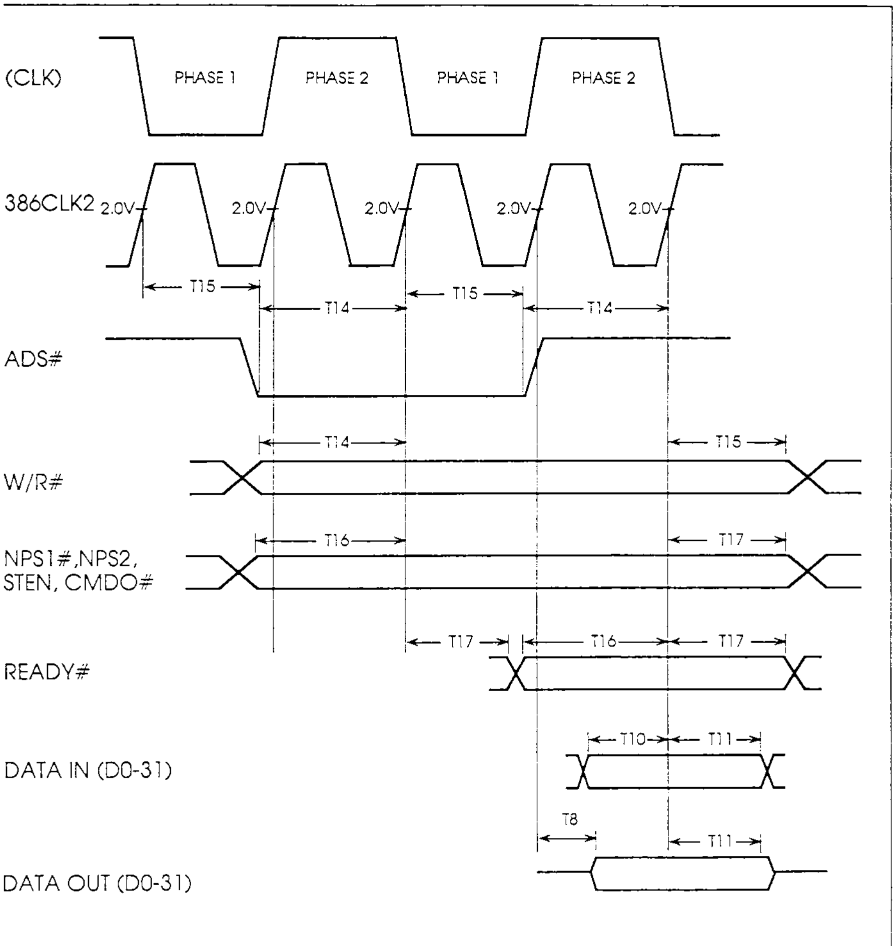

(CLK)
PHASE 2
PHASE 1
PHASE 2
386CLK2
2.0V
T7 Max
T7 Min
Any Output
Output Timing

Fig. 7.2



Data I/O & Control Signal Timing

Fig. 7.3

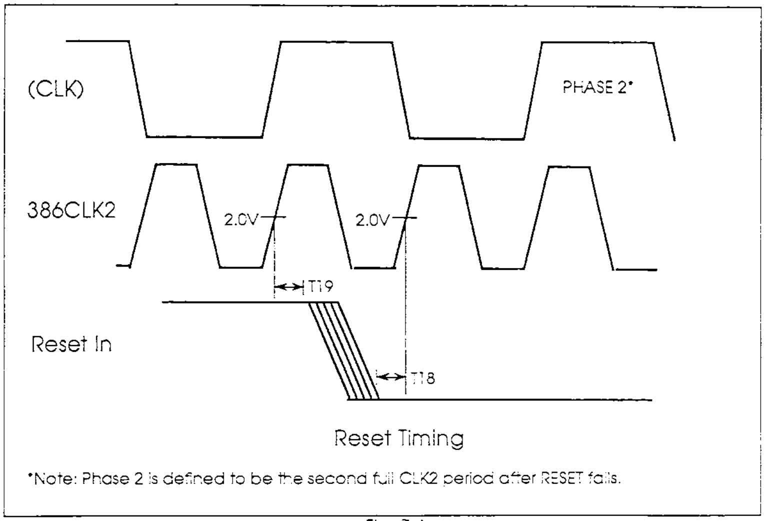
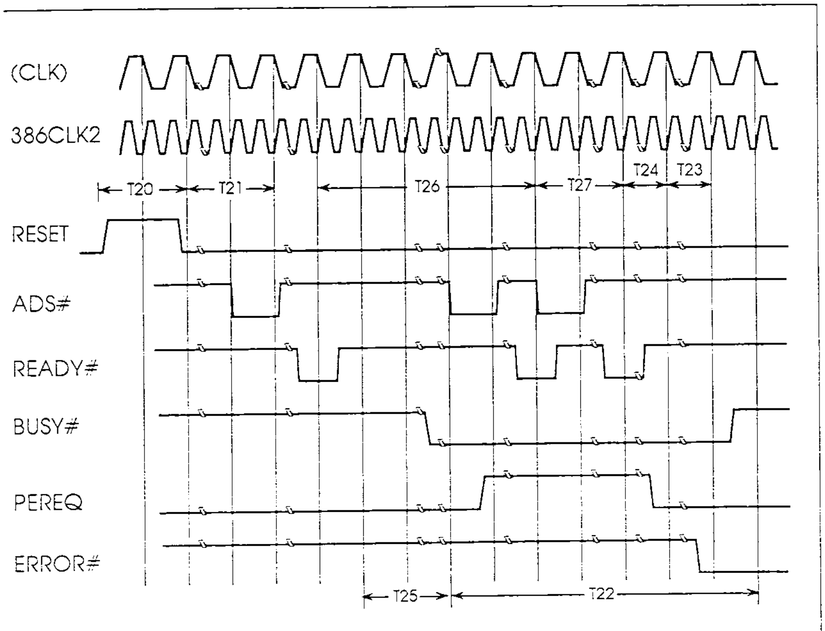

(CLK)
PHASE 2*
386CLK2
2.0V
2.0V
T19
Reset In
T18
Reset Timing
*Note: Phase 2 is defined to be the second full CLK2 period after RESET falls.

Fig. 7.4

7.5 Interface Timing Parameters

The following table sets forth the clock cycle timing requirements for major CX-83D87 interface functions. **Times are specified in 386CLK2** cycle counts. Please refer to figure 7.5 for timing reference information.

Pin	Symbol	Parameter	Min	Max	Notes
RESETIN	T20	Time Active	10		
RESETIN	T21	Time Inactive	10		Before 1st Opcode Write
BUSY#	T22	Time Active	2		
BUSY#	T23	Delay Inactive	6		From ERROR# Inactive
ERROR#	T24	Delay Active	6		From PEREQ Inactive
BUSY#	T25	Delay Active	4	4	From READY# Active
READY#	T26	Delay	0		Opcode Write to Next Cycle
READY#	T27	Delay	0		Operand cycle to next operand cycle



Functional Timing Diagram

Fig. 7.5