C H A P T E R 5

Coprocessor

Topical Cross-reference for Coprocessor Instructions

Arithmetic

FABS	FADD/FIADD	FADDP
FCHS	FDIV/FIDIV	FDIVP
FDIVR/FIDIVR	FDIVRP	FMUL/FIMUL
FMULP	FPREM	FPREM1§
FRNDINT	FSCALE	FSQRT
FSUB/FISUB	FSUBP	FSUBR/FISUBR
FSUBRP	FXTRACT	

Compare

FCOM/FICOM	FCOMP/FICOMP	FCOMPP
FSTSW/FNSTSW	FTST	FUCOM§
FUCOMP§	FUCOMPP§	FXAM

Load

FLD/FILD/FBLD	FLDCW	FLDENV
FRSTOR	FXCH	

Load Constant

FLD1	FLDL2E	FLDL2T
FLDLG2	FLDLN2	FLDPI
FLDZ		

Processor Control

FCLEX/FNCLEX	FDECSTP	FDISI/FNDISI*
FENI/FNENI*	FFREE	FINCSTP
FINIT/FNINIT	FLDCW	FNOP
FRSTOR	FSAVE/FNSAVE	FSETPM–
FSTCW/FNSTCW	FSTENV/FNSTENV	FSTSW/FNSTSW
FWAIT		

Store Data

FSAVE/FNSAVE	FST/FIST	FSTCW/FNSTCW
FSTENV/FNSTENV	FSTP/FISTP/FBSTP	FSTSW/FNSTSW

Transcendental

F2XM1	FCOS§	FPATAN
FPREM	FPREM1§	FPTAN
FSIN§	FSINCOS§	FYL2P1
FYL2X		

* 8087 only † 80287 only. § 80387–80486 only.

Interpreting Coprocessor Instructions

This section provides an alphabetical reference to instructions of the 8087, 80287, and 80387 coprocessors. The format is the same as the processor instructions except that encodings are not provided. Differences are noted in the following.

The 80486 has the coprocessor built in. This one chip executes all the instructions listed in the previous section and this section.

Syntax

Syntaxes in Column 1 use the following abbreviations for operand types:

Syntax	Operand
reg	A coprocessor stack register
memreal	A direct or indirect memory operand storing a real number
memint	A direct or indirect memory operand storing a binary integer
membcd	A direct or indirect memory operand storing a BCD number

Examples

The position of the examples in Column 2 is not related to the clock speeds in Column 3.

Clock Speeds

Column 3 shows the clock speeds for each processor. Sometimes an instruction may have more than one possible clock speed. The following abbreviations are used to specify variations:

Abbreviation	Description
EA	Effective address. This applies only to the 8087. See the Processor Section, “Timings on the 8088 and 8086 Processors,” for an explanation of effective address timings.
s,l,t	Short real, long real, and 10-byte temporary real.
w,d,q	Word, doubleword, and quadword binary integer.
to, fr	To or from stack top. On the 80387 and 80486, the *to* clocks represent timings when ST is the destination. The *fr* clocks represent timings when ST is the source.

Instruction Size

The instruction size is always 2 bytes for instructions that do not access memory. For instructions that do access memory, the size is 4 bytes on the 8087 and 80287. On the 80387 and 80486, the size for instructions that access memory is 4 bytes in 16-bit mode, or 6 bytes in 32-bit mode.

On the 8087, each instruction must be preceded by the **WAIT** (also called **FWAIT**) instruction, thereby increasing the instruction's size by 1 byte. The assembler inserts **WAIT** automatically by default, or with the **.8087** directive.

Architecture

The 8087, 80287, and 80387 coprocessors, along with the 80486, have several common elements of architecture. All have a register stack made up of eight 80-bit data registers. These can contain floating-point numbers in the temporary real format. The coprocessors also have 14 bytes of control registers. Figure 5.1 shows the format of registers.

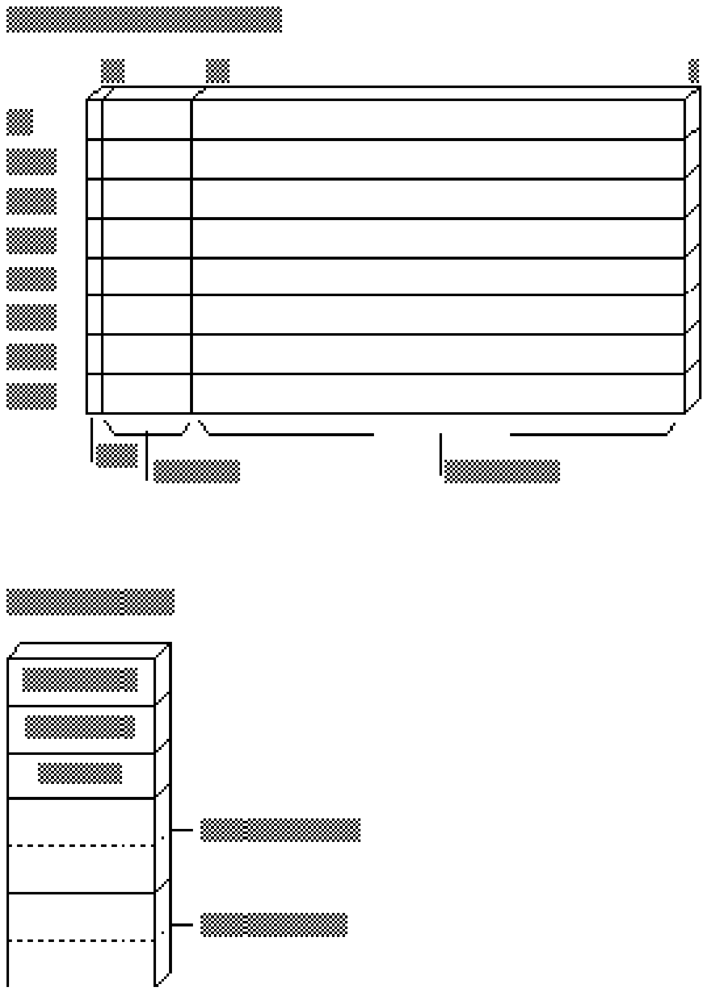

Fig. 5.1 Coprocessor Registers

The most important control registers are the control word and the status word. Figure 5.2 shows the format of these registers.

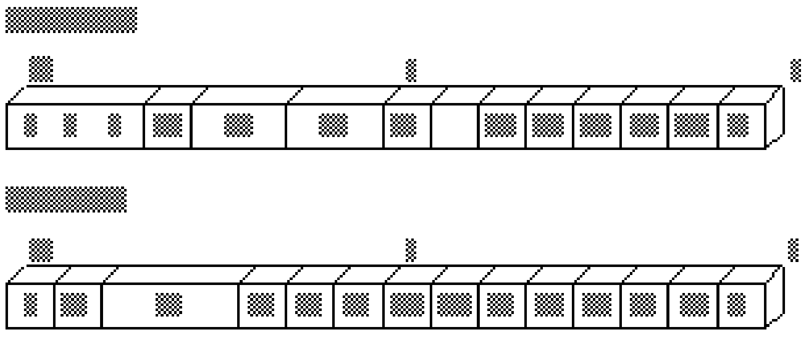

Fig. 5.2 Control Word and Status Word

F2XM1 $2^X - 1$

Calculates $Y = 2^X - 1$. X is taken from ST. The result, Y, is returned in ST. X must be in the range $0 \leq X \leq 0.5$ on the 8087/287, or in the range $-1.0 \leq X \leq +1.0$ on the 80387–80486.

Syntax	Examples	CPU	Clock Cycles
F2XM1	`f2xm1`	87	310–630
		287	310–630
		387	211–476
		486	140–279

FABS Absolute Value

Converts the element in ST to its absolute value.

Syntax	Examples	CPU	Clock Cycles
FABS	`fabs`	87	10–17
		287	10–17
		387	22
		486	3

FADD/FADDP/FIADD Add

Adds the source to the destination and returns the sum in the destination. If two register operands are specified, one must be ST. If a memory operand is specified, the sum replaces the value in ST. Memory operands can be 32- or 64-bit real numbers or 16- or 32-bit integers. If no operand is specified, ST is added to ST(1) and the stack is popped, returning the sum in ST. For **FADDP**, the source must be ST; the sum is returned in the destination and ST is popped.

Syntax	Examples	CPU	Clock Cycles
FADD ⟦*reg*,*reg*⟧	`fadd st,st(2)`	87	70–100
	`fadd st(5),st`	287	70–100
	`fadd`	387	*to*=23–31, *fr*=26–34
		486	8–20
FADDP *reg*,**ST**	`faddp st(6),st`	87	75–105
		287	75–105
		387	23–31
		486	8–20
FADD *memreal*	`fadd QWORD PTR [bx]` `fadd shortreal`	87	(*s*=90–120,*s*=95–125)+*EA*
		287	*s*=90–120,*l*=95–125
		387	*s*=24–32,*l*=29–37
		486	8–20
FIADD *memint*	`fiadd int16` `fiadd warray[di]`	87	(*w*=102–137,*d*=108–143)+*EA*
	`fiadd double`	287	*w*=102–137,*d*=108–143
		387	*w*=71–85,*d*=57–72
		486	*w*=20–35,*d*=19–32

FBLD Load BCD

See **FLD**.

FBSTP Store BCD and Pop

See **FST**.

FCHS Change Sign

Reverses the sign of the value in ST.

Syntax	Examples	CPU	Clock Cycles
FCHS	`fchs`	87	10–17
		287	10–17
		387	24–25
		486	6

FCLEX/FNCLEX Clear Exceptions

Clears all exception flags, the busy flag, and bit 7 in the status word. Bit 7 is the interrupt-request flag on the 8087, and the error-status flag on the 80287, 80387, and 80486. The instruction has wait and no-wait versions.

Syntax	Examples	CPU	Clock Cycles*
FCLEX	`fclex`	87	2–8
FNCLEX		287	2–8
		387	11
		486	7

* These timings reflect the no-wait version of the instruction. The wait version may take additional clock cycles.

FCOM/FCOMP/FCOMPP/FICOM/FICOMP Compare

Compares the specified source operand to ST and sets the condition codes of the status word according to the result. The instruction subtracts the source operand from ST without changing either operand. Memory operands can be 32- or 64-bit real numbers or 16- or 32-bit integers. If no operand is specified or if two pops are specified, ST is compared to ST(1) and the stack is popped. If one pop is specified with an operand, the operand is compared to ST. If one of the operands is a NAN, an invalid-operation exception occurs (see **FUCOM** for an alternative method of comparing on the 80387–80486).

Syntax	Examples	CPU	Clock Cycles
FCOM ⟦*reg*⟧	`fcom st(2)`	87	40–50
	`fcom`	287	40–50
		387	24
		486	4
FCOMP ⟦*reg*⟧	`fcomp st(7)`	87	42–52
	`fcomp`	287	42–52
		387	26
		486	4
FCOMPP	`fcompp`	87	45–55
		287	45–55
		387	26
		486	5
FCOM *memreal*	`fcom shortreals[di]`	87	(*s*=60–70,*l*=65–75)+*EA*
	`fcom longreal`	287	*s*=60–70,*l*=65–75
		387	*s*=26,*l*=31
		486	4
FCOMP *memreal*	`fcomp longreal`	87	(*s*=63–73,*l*=67–77)+*EA*
	`fcomp shorts[di]`	287	*s*=63–73,*l*=67–77
		387	*s*=26,*l*=31
		486	4
FICOM *memint*	`ficom double`	87	(*w*=72–86,*d*=78–91)+*EA*
	`ficom warray[di]`		*w*=72–86,*d*=78–91
		287	*w*=71–75,*d*=56–63
		387	*w*=16–20,*d*=15–17
		486	
FICOMP *memint*	`ficomp WORD PTR`	87	(*w*=74–88,*d*=80–93)+*EA*
	`[bp+6]`		*w*=74–88,*d*=80–93
	`ficomp darray[di]`	287	*w*=71–75,*d*=56–63
		387	*w*=16–20,*d*=15–17
		486	

Condition Codes for FCOM

C3	C2	C1	C0	Meaning
0	0	?	0	ST > source
0	0	?	1	ST < source
1	0	?	0	ST = source
1	1	?	1	ST is not comparable to source

FCOS Cosine

80387–80486 Only Replaces a value in radians in ST with its cosine. If |ST| < 2^{63}, the C2 bit of the status word is cleared and the cosine is calculated. Otherwise, C2 is set and no calculation is performed. ST can be reduced to the required range with **FPREM** or **FPREM1**.

Syntax	Examples	CPU	Clock Cycles
FCOS	`fcos`	87	—
		287	—
		387	123–772*
		486	257–354†

* For operands with an absolute value greater than π/4, up to 76 additional clocks may be required.

† For operands with an absolute value greater than π/4, add *n* clocks where *n* = *operand*/(π/4).

FDECSTP Decrement Stack Pointer

Decrements the stack-top pointer in the status word. No tags or registers are changed, and no data is transferred. If the stack pointer is 0, **FDECSTP** changes it to 7.

Syntax	Examples	CPU	Clock Cycles
FDECSTP	`fdecstp`	87	6–12
		287	6–12
		387	22
		486	3

FDISI/FNDISI Disable Interrupts

8087 Only Disables interrupts by setting the interrupt-enable mask in the control word. This instruction has wait and no-wait versions. Since the 80287, 80387, and 80486 do not have an interrupt-enable mask, the instruction is recognized but ignored on these coprocessors.

Syntax	Examples	CPU	Clock Cycles*
FDISI	`fdisi`	87	2–8
FNDISI		287	2
		387	2
		486	3

* These timings reflect the no-wait version of the instruction. The wait version may take additional clock cycles.

FDIV/FDIVP/FIDIV Divide

Divides the destination by the source and returns the quotient in the destination. If two register operands are specified, one must be ST. If a memory operand is specified, the quotient replaces the value in ST. Memory operands can be 32- or 64-bit real numbers or 16- or 32-bit integers. If no operand is specified, ST(1) is divided by ST and the stack is popped, returning the result in ST. For **FDIVP**, the source must be ST; the quotient is returned in the destination register and ST
is popped.

Syntax	Examples	CPU	Clock Cycles
FDIV ⟦*reg*,*reg*⟧	`fdiv   st,st(2)`	87	193–203
	`fdiv   st(5),st`	287	193–203
		387	*to*=88, *fr*=91
		486	73
FDIVP *reg*,**ST**	`fdivp   st(6),st`	87	197–207
		287	197–207
		387	91
		486	73
FDIV *memreal*	`fdiv   DWORD PTR [bx]`	87	(*s*=215–225,*l*=220–230)+*EA*
	`fdiv   shortreal[di]`		
	`fdiv   longreal`	287	*s*=215–225,*l*=220–230
		387	*s*=89,*l*=94
		486	73
FIDIV *memint*	`fidiv  int16`	87	(*w*=224–238,*d*=230–243)+*EA*
	`fidiv  warray[di]`		
	`fidiv  double`	287	*w*=224–238,*d*=230–243
		387	*w*=136–140,*d*=120–127
		486	*w*=85–89,*d*=84–86

FDIVR/FDIVRP/FIDIVR Divide Reversed

Divides the source by the destination and returns the quotient in the destination. If two register operands are specified, one must be ST. If a memory operand is specified, the quotient replaces the value in ST. Memory operands can be 32- or 64-bit real numbers or 16- or 32-bit integers. If no operand is specified, ST is divided by ST(1) and the stack is popped, returning the result in ST. For **FDIVRP**, the source must be ST; the quotient is returned in the destination register and ST is popped.

Syntax	Examples	CPU	Clock Cycles
FDIVR ⟦*reg*,*reg*⟧	`fdivr  st,st(2)`	87	194–204
	`fdivr  st(5),st`	287	194–204
	`fdivr`	387	*to*=88, *fr*=91
		486	73
FDIVRP *reg*,**ST**	`fdivrp  st(6),st`	87	198–208
		287	198–208
		387	91
		486	73
FDIVR *memreal*	`fdivr  longreal`	87	(*s*=216–226,*l*=221 –231)+*EA*
	`fdivr  shortreal[di]`		
		287	*s*=216–226,*l*=221–231
		387	*s*=89,*l*=94
		486	73
FIDIVR *memint*	`fidivr  double`	87	(*w*=225–239,*d*=231 –245)+*EA*
	`fidivr  warray[di]`		
		287	*w*=225–239,*d*=231 –245
		387	*w*=135–141,*d*=121–128
		486	*w*=85–89,*d*=84–86

FENI/FNENI Enable Interrupts

8087 Only Enables interrupts by clearing the interrupt-enable mask in the control word. This instruction has wait and no-wait versions. Since the 80287, 80387, and 80486 do not have interrupt-enable masks, the instruction is recognized but ignored on these coprocessors.

Syntax	Examples	CPU	Clock Cycles*
FENI	`feni`	87	2–8
FNENI		287	2
		387	2
		486	3

* These timings reflect the no-wait version of the instruction. The wait version may take additional clock cycles.

FFREE Free Register

Changes the specified register’s tag to empty without changing the contents of the register.

Syntax	Examples	CPU	Clock Cycles
FFREE ST(*i*)	`ffree st(3)`	87	9–16
		287	9–16
		387	18
		486	3

FIADD/FISUB/FISUBR/ FIMUL/FIDIV/FIDIVR Integer Arithmetic

See **FADD**, **FSUB**, **FSUBR**, **FMUL**, **FDIV**, and **FDIVR**.

FICOM/FICOMP Compare Integer

See **FCOM**.

FILD Load Integer

See **FLD**.

FINCSTP Increment Stack Pointer

Increments the stack-top pointer in the status word. No tags or registers are changed, and no data is transferred. If the stack pointer is 7, **FINCSTP** changes it
to 0.

Syntax	Examples	CPU	Clock Cycles
FINCSTP	`fincstp`	87	6–12
		287	6–12
		387	21
		486	3

FINIT/FNINIT Initialize Coprocessor

Initializes the coprocessor and resets all the registers and flags to their default values. The instruction has wait and no-wait versions. On the 80387–80486, the condition codes of the status word are cleared. On the 8087/287, they are unchanged.

Syntax	Examples	CPU	Clock Cycles*
FINIT	`finit`	87	2–8
FNINIT		287	2–8
		387	33
		486	17

* These timings reflect the no-wait version of the instruction. The wait version may take additional clock cycles.

FIST/FISTP Store Integer

See **FST**.

FLD/FILD/FBLD Load

Pushes the specified operand onto the stack. All memory operands are automatically converted to temporary-real numbers before being loaded. Memory operands can be 32-, 64-, or 80-bit real numbers or 16-, 32-, or 64-bit integers.

Syntax	Examples	CPU	Clock Cycles
FLD *reg*	`fld st(3)`	87	17–22
		287	17–22
		387	14
		486	4
FLD *memreal*	`fld longreal`	87	(*s*=38–56,*l*=40–60,*t*=53–65)+*EA*
	`fld shortarray[bx+di]`	287	*s*=38–56,*l*=40–60,*t*=53–65
	`fld tempreal`	387	*s*=20,*1*=25,*t*=44
		486	*s*=3,*l*=3,*t*=6
FILD *memint*	`fild mem16`	87	(*w*=46–54,*d*=52–60,*q*=60–68)+*EA*
	`fild DWORD PTR [bx]`		
	`fild quads[si]`	287	*w*=46-54,*d*=52-60,*q*=60-68
		387	*w*=61–65,*d*=45–52,*q*=56–67
		486	*w*=13–16,*d*=9–12,*q*=10–18
FBLD *membcd*	`fbld packbcd`	87	(290–310)+*EA*
		287	290–310
		387	266–275
		486	70–103

FLD1/FLDZ/FLDPI/FLDL2E/FLDL2T/FLDLG2/FLDLN2 Load Constant

Pushes a constant onto the stack. The following constants can be loaded:

Instruction	Constant
FLD1	+1.0
FLDZ	+0.0
FLDPI	π

Instruction	Constant
FLDL2E	$\mathrm{Log}_2(e)$
FLDL2T	$\mathrm{Log}_2(10)$
FLDLG2	$\mathrm{Log}_{10}(2)$
FLDLN2	$\mathrm{Log}_e(2)$

Syntax	Examples	CPU	Clock Cycles
FLD1	`fld1`	87	15–21
		287	15–21
		387	24
		486	4
FLDZ	`fldz`	87	11–17
		287	11–17
		387	20
		486	4
FLDPI	`fldpi`	87	16–22
		287	16–22
		387	40
		486	8
FLDL2E	`fldl2e`	87	15–21
		287	15–21
		387	40
		486	8
FLDL2T	`fldl2t`	87	16–22
		287	16–22
		387	40
		486	8
FLDLG2	`fldlg2`	87	18–24
		287	18–24
		387	41
		486	8
FLDLN2	`fldln2`	87	17–23
		287	17–23
		387	41
		486	8

FLDCW Load Control Word

Loads the specified word into the coprocessor control word. The format of the control word is shown in the “Interpreting Coprocessor Instructions” section.

Syntax	Examples	CPU	Clock Cycles
FLDCW *mem16*	`fldcw ctrlword`	87	(7–14)+*EA*
		287	7–14
		387	19
		486	4

FLDENV/FLDENVW/FLDENVD Load Environment State

Loads the 14-byte coprocessor environment state from a specified memory location. The environment includes the control word, status word, tag word, instruction pointer, and operand pointer. On the 80387–80486 in 32-bit mode, the environment state is 28 bytes.

Syntax	Examples	CPU	Clock Cycles
FLDENV *mem*	`fldenv [bp+10]`	87	(35–45)+*EA*
FLDENVW *mem**		287	35–45
FLDENVD *mem**		387	71
		486	44,*pm*=34

* 80387–80486 only.

FMUL/FMULP/FIMUL Multiply

Multiplies the source by the destination and returns the product in the destination. If two register operands are specified, one must be ST. If a memory operand is specified, the product replaces the value in ST. Memory operands can be 32- or 64-bit real numbers or 16- or 32-bit integers. If no operand is specified, ST(1) is multiplied by ST and the stack is popped, returning the product in ST. For **FMULP**, the source must be ST; the product is returned in the destination register and ST is popped.

Syntax	Examples	CPU	Clock Cycles
FMUL ⟦*reg*,*reg*⟧	`fmul st,st(2)`	87	130–145 (90–105)*
	`fmul st(5),st`	287	130–145 (90–105)*
	`fmul`	387	*to*=46–54 (49), *fr*= 29–57 (52)†
		486	16
FMULP *reg*,**ST**	`fmulp st(6),st`	87	134–148 (94–108)*
		287	134–148 (94–108)*
		387	29–57 (52)†
		486	16
FMUL *memreal*	`fmul DWORD PTR [bx]`	87	(*s*=110–125,*l*=154–168)+*EA*§
	`fmul shortreal[di+3]`		
	`fmul longreal`	287	*s*=110–125,*l*=154 –168§
		387	*s*=27–35,*l*=32–57
		486	*s*=11,*l*=14
FIMUL *memint*	`fimul int16`	87	(*w*=124–138,*d*=130 –144)+*EA*
	`fimul warray[di]`		
	`fimul double`	287	*w*=124–138,*d*=130 –144
		387	*w*=76–87,*d*=61–82
		486	*w*=23–27,*d*=22–24

* The clocks in parentheses show times for short values—those with 40 trailing zeros in their fraction because they were loaded from a short-real memory operand.

† The clocks in parentheses show typical speeds.

§ If the register operand is a short value—having 40 trailing zeros in its fraction because it was loaded from a short-real memory operand—then the timing is (112–126)+*EA* on the 8087 or 112–126 on the 80287.

FN*instruction* No-Wait Instructions

Instructions that have no-wait versions include **FCLEX**, **FDISI**, **FENI**, **FINIT**, **FSAVE**, **FSTCW**, **FSTENV**, and **FSTSW**. Wait versions of instructions check for unmasked numeric errors; no-wait versions do not. When the **.8087** directive is used, the assembler puts a **WAIT** instruction before the wait versions and a **NOP** instruction before the no-wait versions.

FNOP No Operation

Performs no operation. **FNOP** can be used for timing delays or alignment.

Syntax	Examples	CPU	Clock Cycles
FNOP	`fnop`	87	10–16
		287	10–16
		387	12
		486	3

FPATAN Partial Arctangent

Finds the partial tangent by calculating Z = ARCTAN(Y / X). X is taken from ST and Y from ST(1). On the 8087/287, Y and X must be in the range $0 \leq Y < X < \infty$. On the 80387–80486, there is no restriction on X and Y. X is popped from the stack and Z replaces Y in ST.

Syntax	Examples	CPU	Clock Cycles
FPATAN	`fpatan`	87	250–800
		287	250–800
		387	314–487
		486	218–303

FPREM Partial Remainder

Calculates the remainder of ST divided by ST(1), returning the result in ST. The remainder retains the same sign as the original dividend. The calculation uses the following formula:

```
remainder = ST – ST(1) * quotient
```

The *quotient* is the exact value obtained by chopping ST / ST(1) toward 0. The instruction is normally used in a loop that repeats until the reduction is complete, as indicated by the condition codes of the status word.

Syntax	Examples	CPU	Clock Cycles
FPREM	`fprem`	87	15–190
		287	15–190
		387	74–155
		486	70–138

Condition Codes for FPREM and FPREM1

C3	C2	C1	C0	Meaning
?	1	?	?	Incomplete reduction
0	0	0	0	*quotient* MOD 8 = 0
0	0	0	1	*quotient* MOD 8 = 4
0	0	1	0	*quotient* MOD 8 = 1
0	0	1	1	*quotient* MOD 8 = 5
1	0	0	0	*quotient* MOD 8 = 2
1	0	0	1	*quotient* MOD 8 = 6
1	0	1	0	*quotient* MOD 8 = 3
1	0	1	1	*quotient* MOD 8 = 7

FPREM1 Partial Remainder (IEEE Compatible)

80387–80486 Only Calculates the remainder of ST divided by ST(1), returning the result in ST. The remainder retains the same sign as the original dividend. The calculation uses the following formula:

```
remainder = ST – ST(1) * quotient
```

The *quotient* is the integer nearest to the exact value of ST / ST(1). When two integers are equally close to the given value, the even integer is used. The instruction is normally used in a loop that repeats until the reduction is complete, as indicated by the condition codes of the status word. See **FPREM** for the possible condition codes.

Syntax	Examples	CPU	Clock Cycles
FPREM1	`fprem1`	87	—
		287	—
		387	95–185
		486	72–167

FPTAN Partial Tangent

Finds the partial tangent by calculating Y / X = TAN(Z). Z is taken from ST. Z must be in the range $0 \leq Z \leq \pi / 4$ on the 8087/287. On the 80387–80486, $|Z|$ must be less than 2^{63}. The result is the ratio Y / X. Y replaces Z, and X is pushed into ST. Thus, Y is returned in ST(1) and X in ST.

Syntax	Examples	CPU	Clock Cycles
FPTAN	`fptan`	87	30–540
		287	30–540
		387	191–497*
		486	200–273†

* For operands with an absolute value greater than π/4, up to 76 additional clocks may be required.

† For operands with an absolute value greater than π/4, add *n* clocks where *n* = *operand*/(π/4).

FRNDINT Round to Integer

Rounds ST from a real number to an integer. The rounding control (RC) field of the control word specifies the rounding method, as shown in the introduction to this section.

Syntax	Examples	CPU	Clock Cycles
FRNDINT	`frndint`	87	16–50
		287	16–50
		387	66–80
		486	21–30

FRSTOR/FRSTORW/FRSTORD Restore Saved State

Restores the 94-byte coprocessor state to the coprocessor from the specified memory location. In 32-bit mode on the 80387–80486, the environment state takes 108 bytes.

Syntax	Examples	CPU	Clock Cycles
FRSTOR *mem*	`frstor [bp-94]`	87	(197–207)+*EA*
FRSTORW *mem**		287	†
FRSTORD *mem**		387	308
		486	131,*pm*=120

* 80387–80486 only.

† Clock counts are not meaningful in determining overall execution time of this instruction. Timing is determined by operand transfers.

FSAVE/FSAVEW/FSAVED/FNSAVE/ FNSAVEW/FNSAVED Save Coprocessor State

Stores the 94-byte coprocessor state to the specified memory location. In 32-bit mode on the 80387–80486, the environment state takes 108 bytes. This instruction has wait and no-wait versions. After the save, the coprocessor is initialized as if **FINIT** had been executed.

Syntax	Examples	CPU	Clock Cycles§
FSAVE *mem*	`fsave [bp-94]`	87	(197–207)+*EA*
FSAVEW *mem**	`fsave cobuffer`	287	†
FSAVED *mem**		387	375–376
FNSAVE *mem*		486	154,*pm*=143
FNSAVEW *mem**			
FNSAVED *mem**			

* 80387–80486 only.

† Clock counts are not meaningful in determining overall execution time of this instruction. Timing is determined by operand transfers.

§ These timings reflect the no-wait version of the instruction. The wait version may take additional clock cycles.

FSCALE Scale

Scales by powers of 2 by calculating the function $Y = Y * 2^X$. X is the scaling factor taken from ST(1), and Y is the value to be scaled from ST. The scaled result replaces the value in ST. The scaling factor remains in ST(1). If the scaling factor is not an integer, it will be truncated toward zero before the scaling.

On the 8087/287, if X is not in the range $-2^{15} \le X < 2^{15}$ or if X is in the range $0 < X < 1$, the result will be undefined. The 80387–80486 have no restrictions on the range of operands.

Syntax	Examples	CPU	Clock Cycles
FSCALE	`fscale`	87	32–38
		287	32–38
		387	67–86
		486	30–32

FSETPM Set Protected Mode

80287 Only Sets the 80287 to protected mode. The instruction and operand pointers are in the protected-mode format after this instruction. On the 80387–80486, **FSETPM** is recognized but interpreted as **FNOP**, since the 80386/486 processors handle addressing identically in real and protected mode.

Syntax	Examples	CPU	Clock Cycles
FSETPM	`fsetpm`	87	—
		287	2–8
		387	12
		486	3

FSIN Sine

80387–80486 Only Replaces a value in radians in ST with its sine. If |ST| < 2^{63}, the C2 bit of the status word is cleared and the sine is calculated. Otherwise, C2 is set and no calculation is performed. ST can be reduced to the required range with **FPREM** or **FPREM1**.

Syntax	Examples	CPU	Clock Cycles
FSIN	`fsin`	87	—
		287	—
		387	122–771*
		486	257–354†

* For operands with an absolute value greater than π/4, up to 76 additional clocks may be required.

† For operands with an absolute value greater than π/4, add *n* clocks where *n* = *operand*/(π/4).

FSINCOS Sine and Cosine

80387–80486 Only Computes the sine and cosine of a radian value in ST. The sine replaces the value in ST, and then the cosine is pushed onto the stack. If |ST| < 2^{63}, the C2 bit of the status word is cleared and the sine and cosine are calculated. Otherwise, C2 is set and no calculation is performed. ST can be reduced to the required range with **FPREM** or **FPREM1**.

Syntax	Examples	CPU	Clock Cycles
FSINCOS	`fsincos`	87	—
		287	—
		387	194–809*
		486	292–365†

* For operands with an absolute value greater than π/4, up to 76 additional clocks may be required.

† For operands with an absolute value greater than π/4, add *n* clocks where *n* = *operand*/(π/4).

FSQRT Square Root

Replaces the value of ST with its square root. (The square root of –0 is –0.)

Syntax	Examples	CPU	Clock Cycles
FSQRT	`fsqrt`	87	180–186
		287	180–186
		387	122–129
		486	83–87

FST/FSTP/FIST/FISTP/FBSTP Store

Stores the value in ST to the specified memory location or register. Temporary-real values in registers are converted to the appropriate integer, BCD, or floating-point format as they are stored. With **FSTP**, **FISTP**, and **FBSTP**, the ST register value is popped off the stack. Memory operands can be 32-, 64-, or 80-bit real numbers for **FSTP** or 16-, 32-, or 64-bit integers for **FISTP**.

Syntax	Examples	CPU	Clock Cycles
FST *reg*	`fst st(6)`	87	15–22
	`fst st`	287	15–22
		387	11
		486	3
FSTP *reg*	`fstp st`	87	17–24
	`fstp st(3)`	287	17–24
		387	12
		486	3
FST *memreal*	`fst shortreal`	87	(*s*=84–90,*l*=96–104)+*EA*
	`fst longs[bx]`	287	*s*=84–90,*l*=96–104
		387	*s*=44,*l*=45
		486	*s*=7,*l*=8
FSTP *memreal*	`fstp longreal`	87	(*s*=86–92,*l*=98–106, *t*=52–58)+*EA*
	`fstp tempreals[bx]`	287	*s*=86–92,*l*=98–106, *t*=52–58
		387	*s*=44,*l*=45,*t*=53
		486	*s*=7,*l*=8,*t*=6

Syntax	Examples	CPU	Clock Cycles

FIST *memint*	`fist int16`
	`fist doubles[8]`
		87	(*w*=80–90,*d*=82–92)+*EA*

		287	*w*=80–90,*d*=82–92
		387	*w*=82-95,*d*=79-93
		486	*w*=29–34,*d*=28–34

FISTP *memint*	`fistp longint`
	`fistp doubles[bx]`
		87	(*w*=82–92,*d*=84–94, *q*=94–105)+*EA*

		287	*w*=82–92,*d*=84–94, *q*=94–105
		387	*w*=82–95,*d*=79–93, *q*=80–97
		486	29–34
FBSTP *membcd*	`fbstp bcds[bx]`	87	(520–540)+*EA*
		287	520–540
		387	512–534
		486	172–176

FSTCW/FNSTCW Store Control Word

Stores the control word to a specified 16-bit memory operand. This instruction has wait and no-wait versions.

Syntax	Examples	CPU	Clock Cycles*
FSTCW *mem16*	`fstcw ctrlword`	87	12–18
FNSTCW *mem16*		287	12–18
		387	15
		486	3

* These timings reflect the no-wait version of the instruction. The wait version may take additional clock cycles.

FSTENV/FSTENVW/FSTENVD/FNSTENV/FNSTENVW/FNSTENVD Store Environment State

Stores the 14-byte coprocessor environment state to a specified memory location. The environment state includes the control word, status word, tag word, instruction pointer, and operand pointer. On the 80387–80486 in 32-bit mode, the environment state is 28 bytes.

Syntax	Examples	CPU	Clock Cycles†
FSTENV *mem*	`fstenv [bp–14]`	87	(40–50)+*EA*
FSTENVW *mem**		287	40–50
FSTENVD *mem**		387	103–104
FNSTENV *mem*		486	67,*pm*=56
FNSTENVW *mem**			
FNSTENVD *mem**			

* 80387–80486 only.

† These timings reflect the no-wait version of the instruction. The wait version may take additional clock cycles.

FSTSW/FNSTSW Store Status Word

Stores the status word to a specified 16-bit memory operand. On the 80287, 80387, and 80486, the status word can also be stored to the processor's AX register. This instruction has wait and no-wait versions.

Syntax	Examples	CPU	Clock Cycles*
FSTSW *mem16*	`fstsw statword`	87	12–18
FNSTSW *mem16*		287	12–18
		387	15
		486	3
FSTSW AX	`fstsw ax`	87	—
FNSTSW AX		287	10–16
		387	13
		486	3

* These timings reflect the no-wait version of the instruction. The wait version may take additional clock cycles.

FSUB/FSUBP/FISUB Subtract

Subtracts the source operand from the destination operand and returns the difference in the destination operand. If two register operands are specified, one must be ST. If a memory operand is specified, the result replaces the value in ST. Memory operands can be 32- or 64-bit real numbers or 16- or 32-bit integers. If no operand is specified, ST is subtracted from ST(1) and the stack is popped, returning the difference in ST. For **FSUBP**, the source must be ST; the difference (destination minus source) is returned in the destination register and ST is popped.

Syntax	Examples	CPU	Clock Cycles
FSUB ⟦*reg*,*reg*⟧	`fsub   st,st(2)`	87	70–100
	`fsub   st(5),st`	287	70–100
	`fsub`	387	*to*=29–37, *fr*=26–34
		486	8–20
FSUBP *reg*,**ST**	`fsubp  st(6),st`	87	75–105
		287	75–105
		387	26–34
		486	8–20
FSUB *memreal*	`fsub  longreal`	87	(*s*=90–120,*s*=95–125)+*EA*
	`fsub  shortreals[di]`		
		287	*s*=90–120,*l*=95–125
		387	*s*=24–32,*l*=28–36
		486	8–20
FISUB *memint*	`fisub double`	87	(*w*=102–137,*d*=108-143)+*EA*
	`fisub warray[di]`		
		287	*w*=102–137,*d*=108–143
		387	*w*=71–83,*d*=57–82
		486	*w*=20–35,*d*=19–32

FSUBR/FSUBRP/FISUBR Subtract Reversed

Subtracts the destination operand from the source operand and returns the result in the destination operand. If two register operands are specified, one must be ST. If a memory operand is specified, the result replaces the value in ST. Memory operands can be 32- or 64-bit real numbers or 16- or 32-bit integers. If no operand is specified, ST(1) is subtracted from ST and the stack is popped, returning the difference in ST. For **FSUBRP**, the source must be ST; the difference (source minus destination) is returned in the destination register and ST is popped.

Syntax	Examples	CPU	Clock Cycles
FSUBR ⟦*reg*,*reg*⟧	`fsubr  st,st(2)`	87	70–100
	`fsubr  st(5),st`	287	70–100
	`fsubr`	387	*to*=29–37, *fr*=26–34
		486	8–20
FSUBRP *reg*,**ST**	`fsubrp st(6),st`	87	75–105
		287	75–105
		387	26–34
		486	8–20

Syntax	Examples	CPU	Clock Cycles

FSUBR *memreal*	`fsubr  QWORD PTR [bx]`
	`fsubr  shortreal[di]`
	`fsubr  longreal`
		87
		
		287
		387
		486
			(*s*=90–120,*s*=95–125)+*EA*
			
			s=90–120,*l*=95–125
			s=25–33,*l*=29–37
			8–20

FISUBR *memint*	`fisubr int16`
	`fisubr warray[di]`
	`fisubr double`
		87
		
		287
		
		387
		486
			(*w*=103–139,*d*=109–144)+*EA*
			
			w=103–139,*d*=109–144
			
			w=72–84,*d*=58–83
			w=20–55,*d*=19–32

FTST Test for Zero

Compares ST with +0.0 and sets the condition of the status word according to the result.

Syntax	Examples	CPU	Clock Cycles
FTST	`ftst`	87 287 387 486	38–48 38–48 28 4

Condition Codes for FTST

C3	C2	C1	C0	Meaning
0	0	?	0	ST is positive
0	0	?	1	ST is negative
1	0	?	0	ST is 0
1	1	?	1	ST is not comparable (NAN or projective infinity)

FUCOM/FUCOMP/FUCOMPP Unordered Compare

80387–80486 Only Compares the specified source to ST and sets the condition codes of the status word according to the result. The instruction subtracts the source operand from ST without changing either operand. Memory operands are not allowed. If no operand is specified or if two pops are specified, ST is compared to ST(1). If one pop is specified with an operand, the given register is compared to ST.

Unlike **FCOM**, **FUCOM** does not cause an invalid-operation exception if one of the operands is NAN. Instead, the condition codes are set to unordered.

Syntax	Examples	CPU	Clock Cycles
FUCOM ⟦*reg*⟧	`fucom   st(2)`	87	—
	`fucom`	287	—
		387	24
		486	4
FUCOMP ⟦*reg*⟧	`fucomp  st(7)`	87	—
	`fucomp`	287	—
		387	26
		486	4
FUCOMPP	`fucompp`	87	—
		287	—
		387	26
		486	5

Condition Codes for FUCOM

C3	C2	C1	C0	Meaning
0	0	?	0	ST > source
0	0	?	1	ST < source
1	0	?	0	ST = source
1	1	?	1	Unordered

FWAIT Wait

Suspends execution of the processor until the coprocessor is finished executing. This is an alternate mnemonic for the processor **WAIT** instruction.

Syntax	Examples	CPU	Clock Cycles
FWAIT	`fwait`	87	4
		287	3
		387	6
		486	1–3

FXAM Examine

Reports the contents of ST in the condition flags of the status word.

Syntax	Examples	CPU	Clock Cycles
FXAM	`fxam`	87	12–23
		287	12–23
		387	30–38
		486	8

Condition Codes for FXAM

C3	C2	C1	C0	Meaning
0	0	0	0	+ Unnormal*
0	0	0	1	+ NAN
0	0	1	0	– Unnormal*
0	0	1	1	– NAN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	– Normal
0	1	1	1	– Infinity
1	0	0	0	+ 0
1	0	0	1	Empty
1	0	1	0	– 0
1	0	1	1	Empty
1	1	0	0	+ Denormal
1	1	0	1	Empty*
1	1	1	0	– Denormal
1	1	1	1	Empty*

* Not used on the 80387–80486. Unnormals are not supported by the 80387–80486. Also, the 80387–80486 use two codes instead of four to identify empty registers.

FXCH Exchange Registers

Exchanges the specified (destination) register and ST. If no operand is specified, ST and ST(1) are exchanged.

Syntax	Examples	CPU	Clock Cycles
FXCH ⟦*reg*⟧	`fxch  st(3)`	87	10–15
	`fxch`	287	10–15
		387	18
		486	4

FXTRACT Extract Exponent and Significand

Extracts the exponent and significand (mantissa) fields of ST. The exponent replaces the value in ST, and then the significand is pushed onto the stack**.**

Syntax	Examples	CPU	Clock Cycles
FXTRACT	`fxtract`	87	27–55
		287	27–55
		387	70–76
		486	16–20

FYL2X Y $\log_2(X)$

Calculates $Z = Y \log_2(X)$. X is taken from ST and Y from ST(1). The stack is popped, and the result, Z, replaces Y in ST. X must be in the range $0 < X < \infty$ and Y in the range $-\infty < Y < \infty$.

Syntax	Examples	CPU	Clock Cycles
FYL2X	`fyl2x`	87	900–1100
		287	900–1100
		387	120–538
		486	196–329

FYL2XP1 Y $\log_2(X+1)$

Calculates $Z = Y \log_2(X + 1)$. X is taken from ST and Y from ST(1). The stack is popped once, and the result, Z, replaces Y in ST. X must be in the range $0 < |X| < (1 - (\sqrt{2} / 2))$. Y must be in the range $-\infty < Y < \infty$.

Syntax	Examples	CPU	Clock Cycles
FYL2XP1	`fyl2xp1`	87	700–1000
		287	700–1000
		387	257–547
		486	171–326