UMBC

CMPE - 310

Lab 06 – Floating Point Coprocessor (80x87)

Outline

Floating Point Coprocessor Basics

Floating Point Coprocessor Status And Control Registers

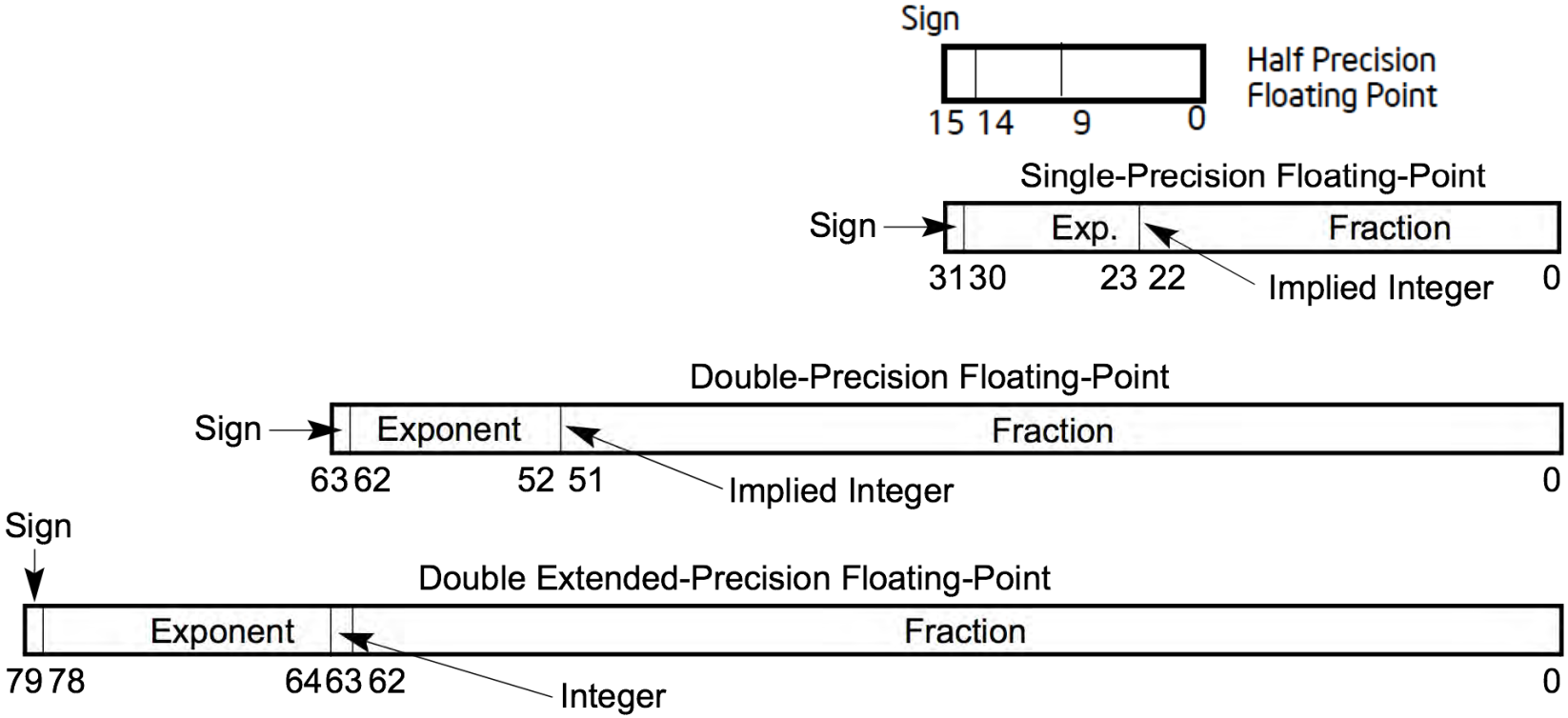
Floating Point Coprocessor Instructions

Coprocessor Basics

The 80x87 is able to multiply, divide, add, subtract, find the sqrt and calculate transcendental functions and logarithms.

Data types include 16-, 32- and 64-bit signed integers; 18-digit BCD data; and 32-, 64- and 80-bit (extended precision) floating-point numbers.

The directives ***dw***, ***dd*** and ***dq*** are used for declaring signed integer storage while ***dd***, ***dq*** and ***dt*** are used for floating-point.

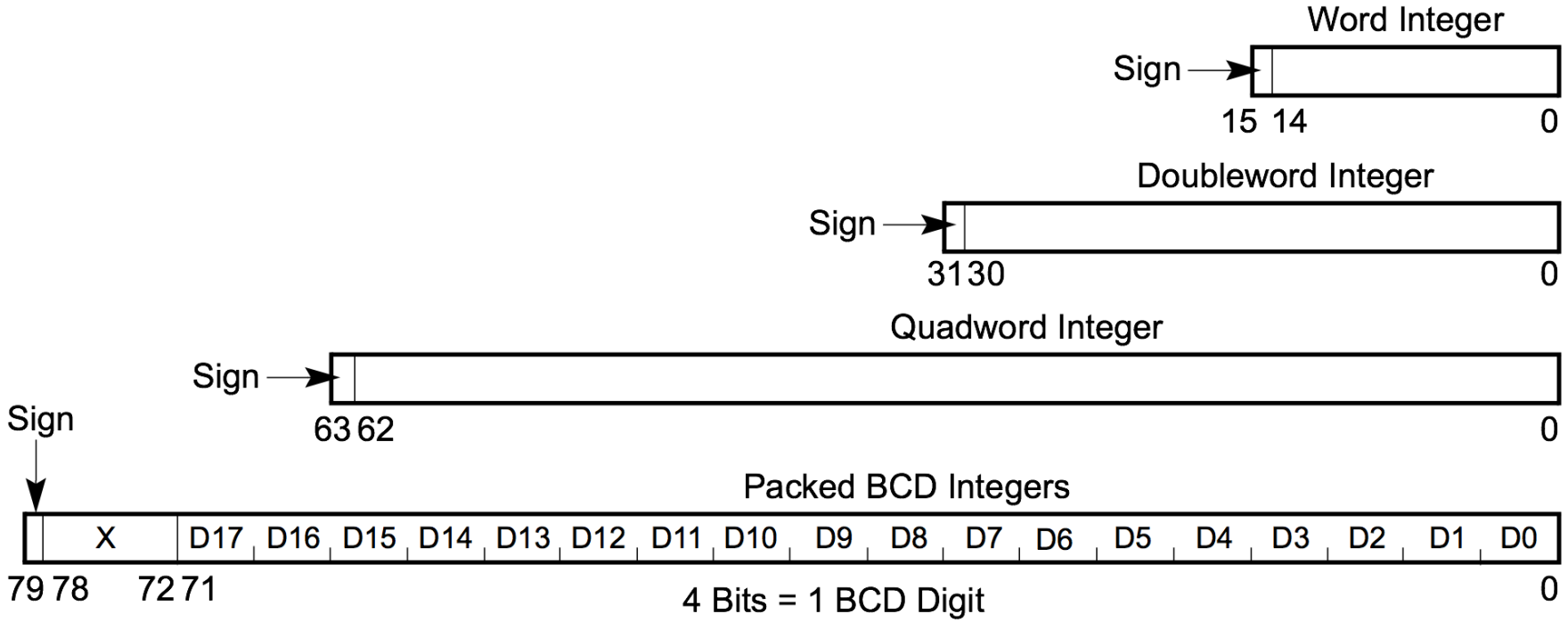


Coprocessor Basics

The 80x87 is able to multiply, divide, add, subtract, find the sqrt and calculate transcendental functions and logarithms.

Data types include 16-, 32- and 64-bit signed integers; 18-digit BCD data; and 32-, 64- and 80-bit (extended precision) floating-point numbers.

The directives ***dw***, ***dd*** and ***dq*** are used for declaring signed integer storage while ***dd***, ***dq*** and ***dt*** are used for floating-point.



Coprocessor Basics

The 80x87 is able to multiply, divide, add, subtract, find the sqrt and calculate transcendental functions and logarithms.

Data types include 16-, 32- and 64-bit signed integers; 18-digit BCD data; and 32-, 64- and 80-bit (extended precision) floating-point numbers.

The directives ***dw***, ***dd*** and ***dq*** are used for declaring signed integer storage while ***dd***, ***dq*** and ***dt*** are used for floating-point.

Instruction	Operand	Comment
dd	0x12345678	; 0x78 0x56 0x34 0x12
dd	1.234567E+20	; floating-point constant
dq	0x123456789abcdef0	; eight byte constant
dq	1.234567E+20	; double-precision float
dt	1.234567E+20	; extended-precision float

Coprocessor Basics

Converting from *decimal* to *floating-point* is accomplished:

- Convert the decimal number into binary.
- Normalize the binary number.
- Calculate the biased exponent.
- Store the number in the floating-point format.

```
100.25 = 1100100.01
1100100.01 = 1.10010001 x 2^6
110 + 01111111 = 10000101
Sign = 0; Exponent = 10000101;
Significand = 10010001000000000000000
```

Bias is 0x7F, 0x3FF and 0x3FFF for the 3 floating-point number types.

Coprocessor Basics

Converting from *floating-point* to *decimal* is accomplished:

- Separate the sign-bit, biased exponent, and significand.
- Convert the biased exponent into a true exponent by subtracting the bias.
- Write the number as a normalized binary number.
- Convert it to a de-normalized binary number.
- Convert the de-normalized binary number to decimal.

```
Sign = 1; Exponent = 10000011;
Significand = 10010010000000000000000
100 = 10000011 - 01111111
1.1001001 x 2^4
11001.001
-25.125
```

Coprocessor Basics

Special Rules:

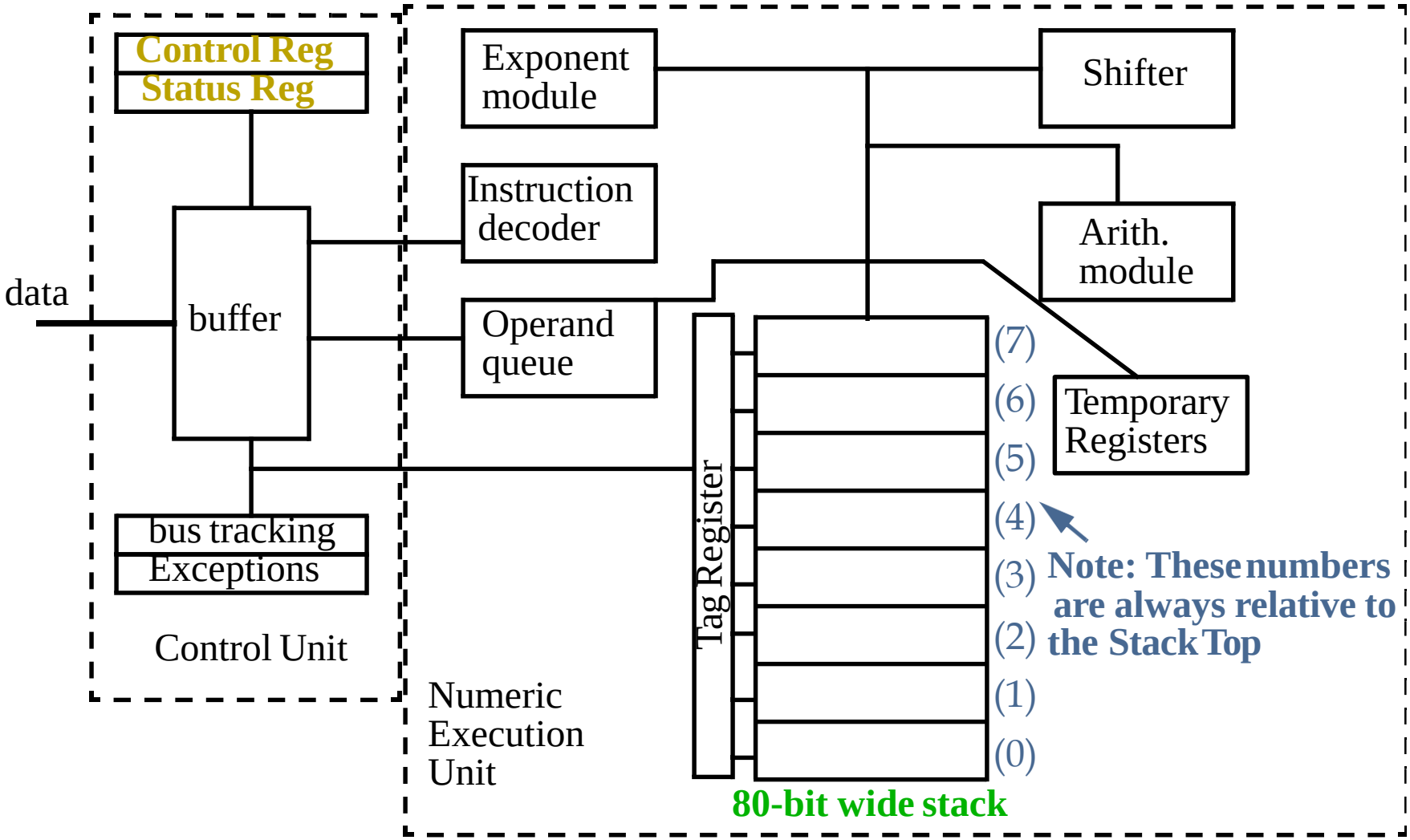
- The number 0 is stored as all 0s (except for the sign bit).
- +/- infinity is stored as logic 1s in the exponent, with a significand of all 0s. Sign bit is used to represent +/- infinity.
- A NAN (not-a-number) is an invalid floating-point result that has all 1s in the exponent with a significand that is NOT all zeros.

Data Type	Length	Precision (Bits)	Approximate Normalized Range	
			Binary	Decimal
Half Precision	16	11	2^{-14} to 2^{15}	3.1×10^{-5} to 6.50×10^{4}
Single Precision	32	24	2^{-126} to 2^{127}	1.18×10^{-38} to 3.40×10^{38}
Double Precision	64	53	2^{-1022} to 2^{1023}	2.23×10^{-308} to 1.79×10^{308}
Double Extended Precision	80	64	2^{-16382} to 2^{16383}	3.37×10^{-4932} to 1.18×10^{4932}

Coprocessor Basics

The 80x87 executes 68 different instructions.

Basic structure of the co-processor.

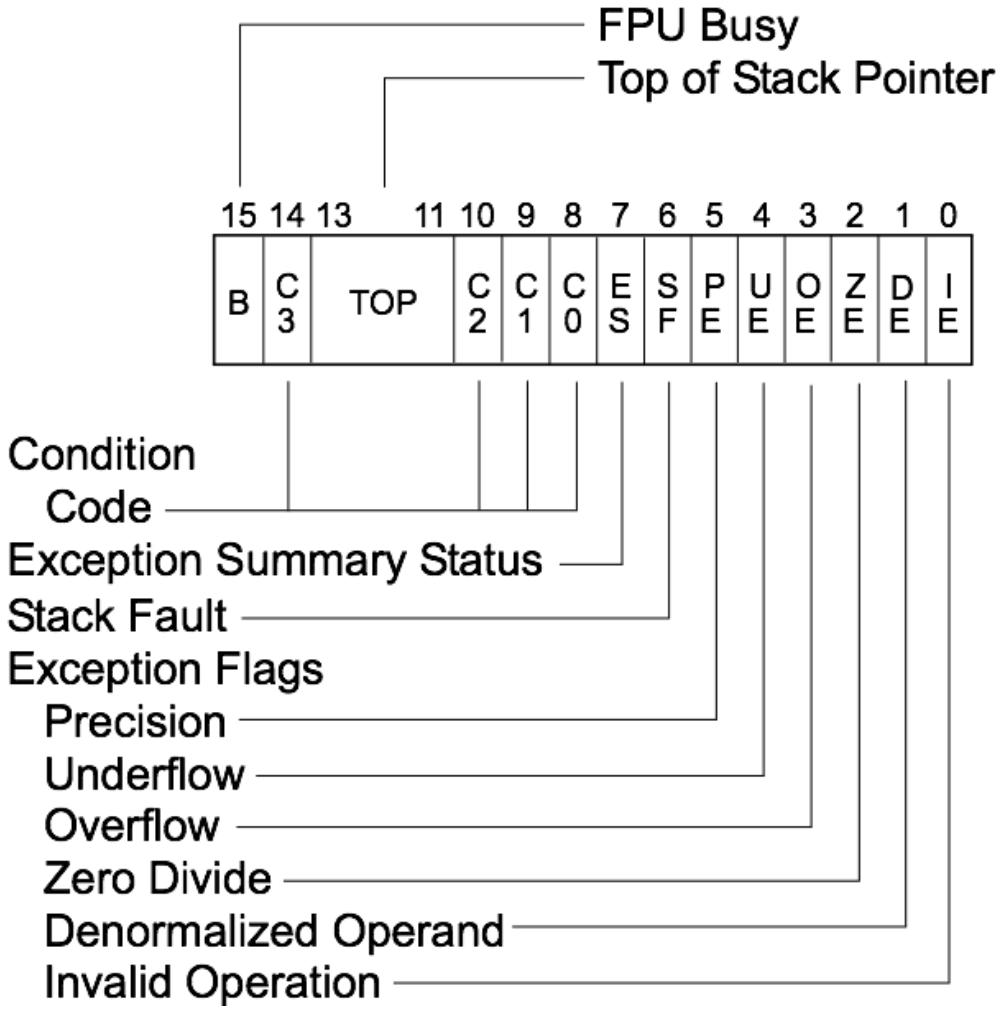


Coprocessor Status Register

The registers in the coprocessor stack always contain 80-bit extended precision data. Memory data, of course, can assume other representations. Therefore, conversions occur during transfers.

Status register:

Condition Code	Status Flag
C0	CF
C1	(none)
C2	PF
C3	ZF



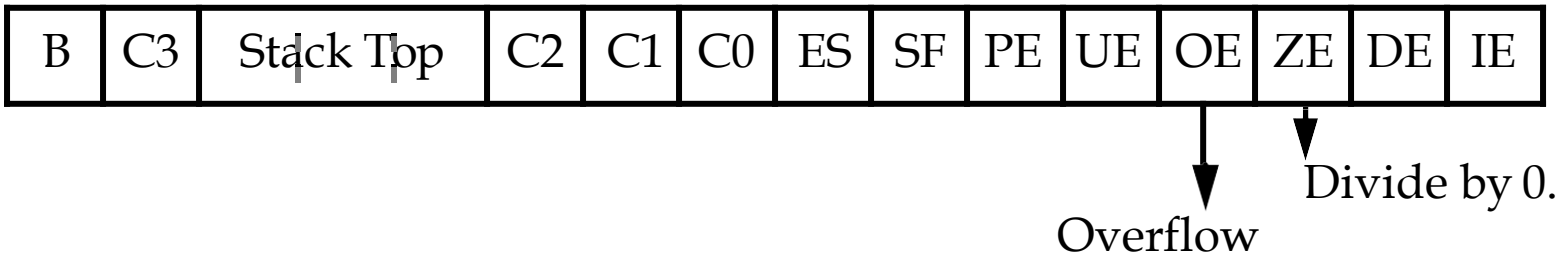
During Store Operations
C1 = 0 (stack underflow)

During Load Operations
C1 = 1 (stack overflow)

Coprocessor Status Register

The registers in the coprocessor stack always contain 80-bit extended precision data. Memory data, of course, can assume other representations. Therefore, conversions occur during transfers.

Status register:



FSTSW AX (Floating-point STore Status Word).

An instruction that transfers data between the coprocessor and the AX register.

Error conditions can be checked in your program by examining bits of this status word. You can use the TEST instruction to test bits or the SAHF instruction to transfer the left-most 8 bits to the EFLAGs register.

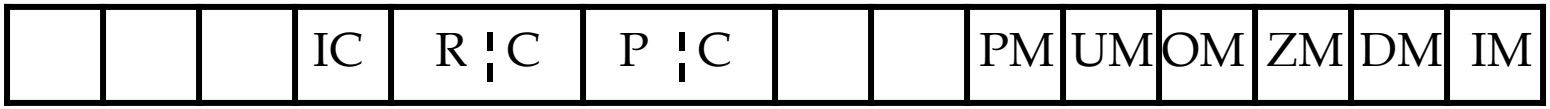
Coprocessor Status/Control Register

For example:

```
fdiv   DATA1
fstsw  ax              ;Copy status reg to AX
test   ax, 4           ;Test bit position 2
jnz    DIVIDE_ERROR

fcom   DATA1           ;Compare DATA1 to ST0 and set status.
fstsw  ax
sahf                   ;Copy status bits to flags.
je     ST_EQUAL
jb     ST_BELOW
ja     ST_ABOVE
```

Control Register:



This register selects precision, rounding control and infinity control.

For example, a value of 00 for P and C sets single precision mode.

R and C control rounding, e.g. round down, up or truncate toward 0.

Coprocessor Instruction Set

Data Transfer Instructions:

- ***FLD*** (Load Real)

  Loads floating-point data to Stack Top (ST). Stack pointer is then decremented by 1.

  Data can be retrieved from memory, or another stack position.

```
fld st2          ;Copies contents of register two to ST.
```

  Note that ST is register 0 after initialization.

- ***FST*** (Store Real), ***FSTP*** (Store Real and Pop)

  Stores a copy of the top of the stack (and pop for FSTP) into memory or another coprocessor register.

  Rounding occurs when the storage operation completes according to the control register.

```
fst dword [eax] ;Pop contents of FP stack to [eax]
```

- ***FXCH*** (Exchange)

  Exchanges register given as operand with ST.

- ***FCMOV*** (Conditional floating point MOV)

Coprocessor Instruction Set

Integer Data Transfer Instructions:

- ***FILD*** (load integer)
- ***FIST*** (Store integer)
- ***FISTP*** (Store integer and pop)

Similar to ***FLD***, ***FST*** and ***FSTP*** except they transfer (and convert) integer.

```
fild dword [numpoints]      ;Load and convert integer to FP stack.
```

Arithmetic Instructions:

Addressing modes:

Mode	Form	Example
Stack	*st1, st*	***faddp***
Register	*st, stn*	***fadd*** *st, st2*
	stn, st	***fadd*** *st2, st*
Register Pop	*stn, st*	***faddp*** *st3, st*
Memory	Operand	***fadd*** [DATA2]

Stack addressing mode is restricted to use ST (stack top) and ST1.

The source operand is ST while the destination operand is ST1.

After the operation, the source is **popped**, leaving the dest. at ST.

Coprocessor Instruction Set

Arithmetic Instructions (cont):

Note that ***FSUB*** subtracts ST from ST1, e.g., ST = ST1 - ST.

Use ***FSUBR*** to reverse the order.

For example, to compute the reciprocal (1/X):

```
fld  x              ;Load X.
fld1                ;Load 1.0 to st
fdivr               ;Compute 1/X and save at st.
```

Register addressing mode MUST use ST as one of the operands.

The other operand can be any register, including ST0 which is ST.

Note that the destination can be either ST or STn.

Also, unlike stack addressing, non-popping versions can be used.

Memory addressing mode always uses ST as the destination.

Coprocessor Instruction Set

Arithmetic Instructions (cont):
The following letters are used to additionally qualify the operation:

- P: Perform a register pop after the operation, ***FADD*** and ***FADDP***.
- R: Reverse mode for subtraction and division.
- I: Indicates that the memory operand is an integer. I appears as the second letter in the instruction, e.g., ***FIADD***, ***FISUB***, ***FIMUL***, ***FIDIV***.

Arithmetic Related Instructions:

- ***FSQRT***: Finds the square root of operand at ST. Leave result there. Check IE bit for an invalid result, e.g., the operand was negative using FSTSW AX, and TEST AX, 1.
- ***FSCALE***: Adds contents of ST1 (interpreted as an integer) to the *exponent* of ST.
- ***FPREM1***: Performs modulo division of ST by ST1. The resultant remainder is found at ST.
- ***FRNDINT***: Rounds ST to an integer.

Coprocessor Instruction Set

Arithmetic Related Instructions (cont):

- ***FXTRACT***: Decomposes ST into an unbiased exponent and a significand. Extracted significand is at ST and unbiased exponent at ST1.
- ***FABS***: Change sign of ST to positive.
- ***FCHS***: Invert sign of ST.

Comparison Instructions:

These instructions examine ST in relation to another element and return result of the comparison in the status register bits C3-C0.

- ***FCOM***: Compares ST with an memory or register operand. FCOM by itself compares ST and ST1.
- ***FCOMP***/***FCOMPP***: Compare and pop once or twice.
- ***FICOM***/***FICOMP***: Compare ST with integer memory operand and optionally pop the stack.
- ***FTST***: Compare ST with 0.0.
- ***FXAM***: Exam ST and modify CC bits to indicate whether contents are positive, negative, normalized, etc. (See text).
- ***FCOMI***/***FUCOMI***: Combines FCOM, FNSTSW AX, and SAHF.

Coprocessor Instruction Set

Transcendental Operations: (See text for semantics).

- ***FPTAN***
- ***FPATAN***
- ***F2XM1***: Compute 2^x -1
- ***FSIN***/***FCOS***
- ***FSINCOS***
- ***FYL2X***: Compute $Y\log_2 X$
- ***FYL2XP1***: Compute $Y\log_2(X + 1)$

Constant Returning Operations:

- ***FLDZ***: Store +0.0 to ST.
- ***FLD1***: Store +1.0 to ST.
- ***FLDPI***: Store pi to ST.
- ***FLDL2T***: Store $\log_2 10$ to ST.
- ***FLDL2E***: Store $\log_2 e$ to ST.
- ***FLDLG2***: Store $\log_{10} 2$ to ST.
- ***FLDLN2***: Store $\log_e 2$ to ST.

Coprocessor Instruction Set

Coprocessor Control Instructions:

- ***FINIT***/***FNINIT***: Reset coprocessor with or without waiting afterwards.
- ***FWAIT***: Stops microprocessor until coprocessor has finished an operation. Should be used before the microprocessor accesses memory data that are affected by the coprocessor.

Instruction reference is given in text along with examples.