Assignment 2

IA32 Architecture Floating point unit

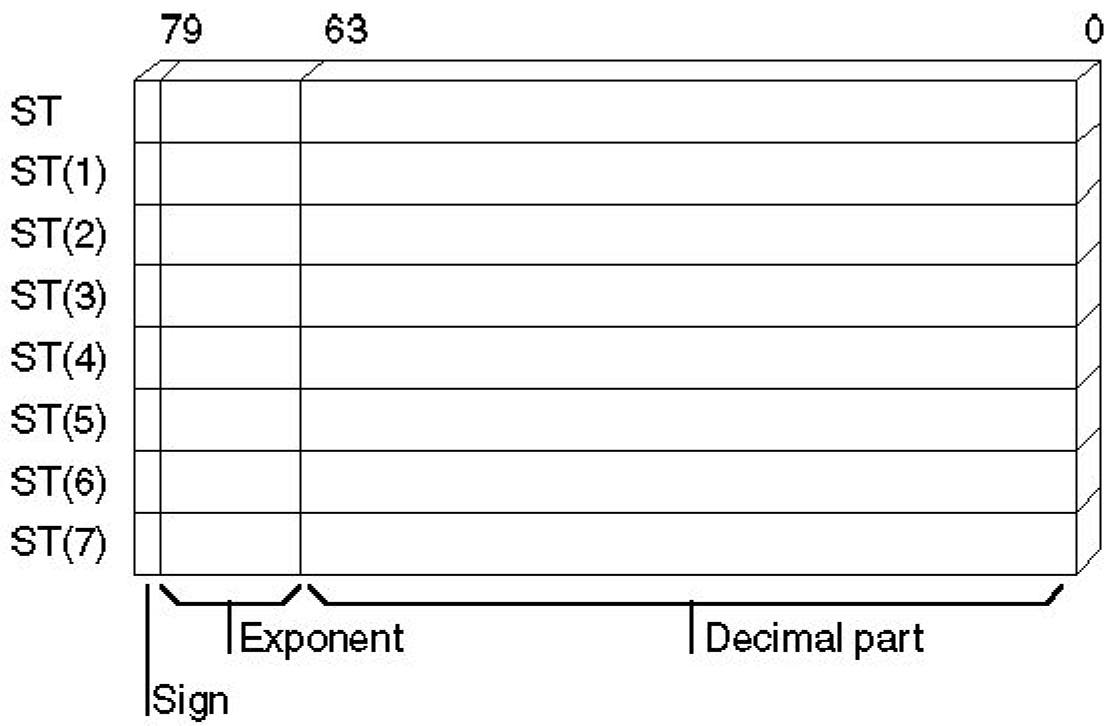
The Floating-Point Unit (FPU) provides high-performance floating-point processing capabilities for use in graphics processing, scientific, engineering, and business applications.

The FPU represents a separate execution environment within the IA-32 architecture.This execution environment consist of 8 data registers and following special purpose registers.

- The status register
- The counter register
- The tag word register

The FPU data register consist of eight 80-bits registers.Values are stored in these register in the double extended-precision floating point format.When floating-point, integer, or packed BCD integer values are loaded from memory into any of the FPU data registers, the values are automatically converted into double extended-precision floating-point format (if they are not already in that format). When computation results are subsequently transferred back into memory from any of the FPU registers, the results can be left in the double extended-precision floating-point format or converted back into a shorter floating-point format, an integer format, or the packed BCD integer format.

The FPU instructions treat the eight FPU data register as a register stack.All addressing of the data registers is relative to the register on the top of the stack.

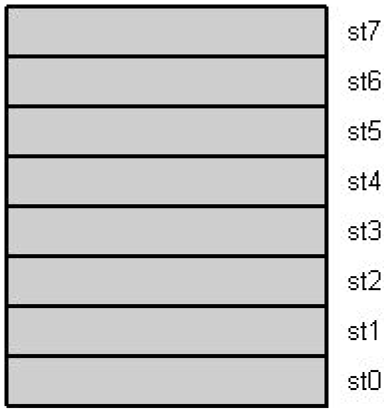


The register number of the current top-of-stack register is stored in the TOP (stack TOP) field in the FPU status word. Load operations decrement TOP by one and load a value into the new top-of-stack register, and store operations store the value from the current TOP register in memory and then increment TOP by one. (For the FPU, a load operation is equivalent to a push and a store operation is equivalent to a pop.) Note that load and store operations are also available that do not push and pop the stack.

if you use constants as operands, you cannot load them directly into FPU registers. You must allocate memory and initialize a variable to a constant value. That variable can then be loaded by using one of the load instructions.

The math FPU offers a few special instructions for loading certain constants. You can load 0, 1, pi, and several common logarithmic values directly. Using these instructions is faster and often more precise than loading the values from initialized variables.

This is how FPU works.



Status word

The Status Word 16-bit register indicates the general condition of the FPU. Its content may change after each instruction is completed. Part of it cannot be changed directly by the programmer. It can, however, be accessed indirectly at any time to inspect its content.

The Status Word is divided into several bit fields as depicted in the following Fig.

Fig.1.3 Status Word Fields

When the FPU is initialized, all the bits are reset to 0.

Control Word

The Control Word 16-bit register is used by the programmer to select between the various modes of computation available from the FPU, and to define which exceptions should be handled by the FPU or by an exception handler written by the programmer.

The Control Word is divided into several bit fields as depicted in the following Fig.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			IC	RC		PC		IEM		PM	UM	OM	ZM	DM	IM

Fig.1.2 Control Word Fields

Tag Word

The Tag Word 16-bit register is managed by the FPU to maintain some information on the content of each of its 80-bit registers.

The Tag Word is divided into 8 fields of 2 bits each as depicted in the following Fig.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Tag(7)		Tag(6)		Tag(5)		Tag(4)		Tag(3)		Tag(2)		Tag(1)		Tag(0)	

Fig.1.4 Tag Word Fields

The above Tag numbers correspond to the FPU's internal numbering system for the 80-bit registers (the BC numbers). The meaning of each pair of bits is as follows:

00 = The register contains a valid non-zero value
01 = The register contains a value equal to 0
10 = The register contains a special value (NAN, infinity, or denormal)
11 = The register is empty

Instruction Set

Move

fld x	; push real4, real8, tbyte, convert to tbyte
fild x	; push word, dword, qword, convert to tbyte
fst x	; convert ST and copy to real4, real8, tbyte
fist x	; convert ST and copy to word, dword, qword
fstp x	; convert to real and pop
fistp x	; convert to integer and pop
fxch st(n)	; swap with st(0)

Arithmetic

FADD	Adds the source and destination
FSUB	Subtracts the source from the destination
FSUBR	Subtracts the destination from the source
FMUL	Multiplies the source and the destination
FDIV	Divides the destination by the source
FDIVR	Divides the source by the destination
FABS	Sets the sign of ST to positive
FCHS	Reverses the sign of ST
FRNDINT	Rounds ST to an integer
FSQRT	Replaces the contents of STwith its square root
FSCALE	Multiplies the stack-top value by 2 to the power contained in ST(1)
FPREM	Calculates the remainder of ST divided by ST(1)

Data type of floating point

Masm	C++	Sign Bits	Exponent	Mantissa	Range	Significant digits
real4	float	1	8	23	+- 1.7e38	6
real8	double	1	11	52	+- 1e308	14
real10	long double	1	15	64	+- 1e4932	18

Example-1:

```
.DATA
down REAL4 10.35 ; Sides of a rectangle
across REAL4 13.07
status WORD ?

.CODE

Main Proc
; Get area of rectangle
fld across                      ; Load one side
fmul down                       ; Multiply by the other
fstp status             ; store it
exit
Main endp
End Main
```

Example-2:

```
.DATA

diamtr REAL4 12.93 ; Diameter of a circle
status WORD ?

.CODE

Main proc
; Get area of circle: Area = PI * (D/2)2

fld1                    ; Load one and
fadd st, st             ; double it to get constant 2
fdivr diamtr            ; Divide diameter to get radius
fmul st, st             ; Square radius
fldpi                   ; Load pi
fmul                    ; Multiply i
fstp status             ; store it

exit
Main endp
End Main
```

Example-3:

```
.DATA
a REAL4 3.0
b REAL4 7.0
cc REAL4 2.0
posx REAL4 0.0
negx REAL4 0.0

.CODE

Main proc
; Solve quadratic equation - no error checking

; The formula is: -b +/- square root(b2 - 4ac) / (2a)

fld1            ; Get constants 2 and 4
fadd st,st      ; 2 at bottom
fld st          ; Copy it
fmul a          ; = 2a

fmul st(1),st   ; = 4a
fxch            ; Exchange
fmul cc         ; = 4ac

fld b           ; Load b
fmul st,st      ; = b2
fsubr           ; = b2 - 4ac

; Negative value here produces error
```

```
fsqrt               ; = square root(b2 - 4ac)
fld b               ; Load b
fchs                ; Make it negative
fxch                ; Exchange

fld st              ; Copy square root
fadd st,st(2)       ; Plus version = -b + root(b2 - 4ac)
fxch                ; Exchange
fsubp st(2),st      ; Minus version = -b - root(b2 - 4ac)

fdiv st,st(2)       ; Divide plus version
fstp posx           ; Store it
fdivr               ; Divide minus version
fstp negx           ; Store i

exit
Main endp
End Main
```