Computer Architecture and System Programming Laboratory

TA Session 8

X87 Float Point Unit - math coprocessor

Float Point representation

```
int x;
for (x = 0; x != 10; x += 1)
       printf("%d\n", x);
```

this loop prints all integer numbers from 0 to 9 including

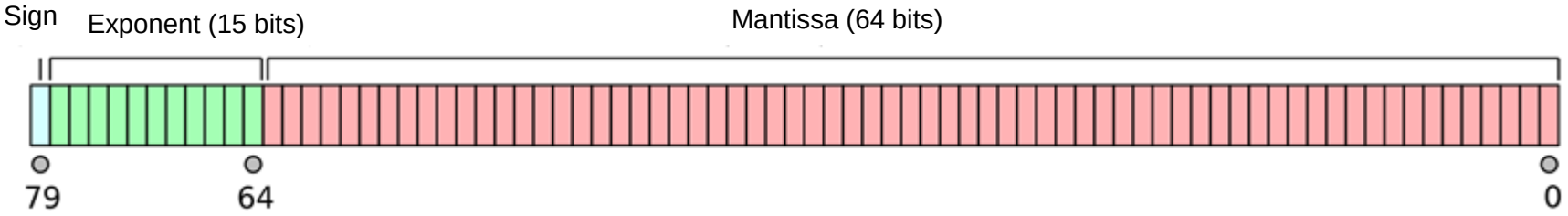
```
float x;
for (x = 0; x != 0.1; x += 0.01)
       printf("%f\n", x);
```

This is actually infinite loop !

Most real numbers only APPROXIMATED by any FP format, requires care by programmers!

Float Point representation

IEEE 754 standard, 80-bit double extended precision



2.625

0.625 × 2 =1.25	1.25-1=0.25	*generate **1** and continue with reminder*
0.25 × 2 = 0.5		*generate **0** and continue*
0.5 × 2 = 1.0	1-1=0	*generate **1** and nothing remains*

integer part: $2_{10} = 10_2$

fraction part: $0.625_{10} = 0.101_2$ and $2.625_{10} = 10.101_2$

normalize: $10.101_2 = 1.0101_2 \times 2^1 = (-1)^0 \times 1.0101_2 \times 2^1$

Mantissa: 10101

Exponent: $1 + \text{bias} = 1 + 16{,}383 = 16{,}384 = 1000000000000000_2$

Sign bit is 0

according to IEEE, float point must be after leftmost '1' bit

Exponent is biased – the value stored is offset from the actual value by the exponent bias.
The reason is that we store exponent as unsigned integer, but exponent may be negative. For this, we add $2^{|exponent|-1}$ to the original exponent value.

bias = $2^{|exponent|-1} - 1$, in our case $2^{15-1} - 1 = 16{,}383$

x87 Floating-Point Unit (FPU) provides high-performance floating-point processing capabilities

- IEEE Standard 754
- floating-point processing algorithms
- exception handling

"For now the 10-byte Extended format is a tolerable compromise between the value of extra-precise arithmetic and the price of implementing it to run fast"

intel

CHAPTER 8
PROGRAMMING WITH THE X87 FPU

http://galaxy.agh.edu.pl/~amrozek/AK/x87.pdf

x87 basic example

```
n:  dd 9
x: dt 0

fild   dword [n]
fstp  tbyte [x]
```

load dword integer to ST(0)

copy ST(0) to m80fp

$$9_d = 1001_b = 1001.0_b = (-1)^0 \cdot 1.001 \cdot 2^{11}{}_b$$

Sign bit = 0

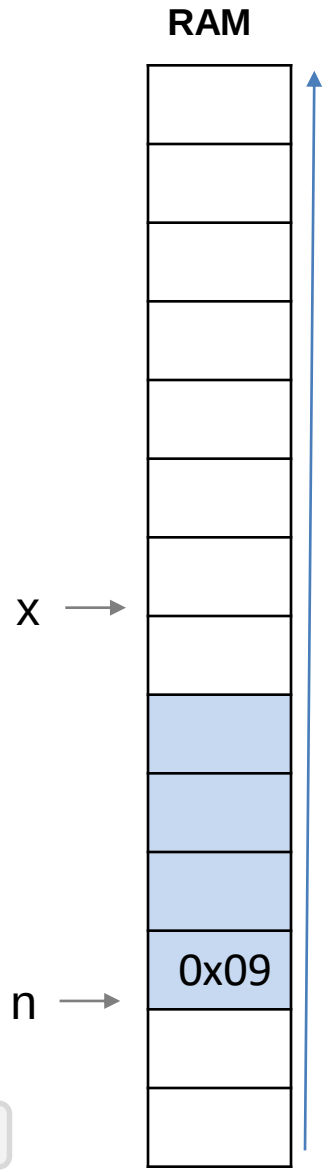
Exponent = 11 b (+ 16383 d bias) = 100 0000 0000 0010 b

Mantissa = 1001 b

Sign | Exponent | Mantissa

79	78	…	65	64	63	62	61	60	…	2	1	0
0	1	0…	1	0	1	0	0	1	0…	0	0	0

float-point number in x87 data registers stack



x87 basic example

```
n:  dd 9
x: dt 0

fild   dword [n]
fstp  tbyte [x]
```

load dword integer to ST(0)

copy ST(0) to m80fp

bits 63-56=10010000b=0x90
bits 71-64=00000010b=0x02
bits 79-72=01000000b=0x40

$$9_d = 1001_b = 1001.0_b = (-1)^0 \cdot 1.001 \cdot 2^{11}{}_b$$

Sign bit = 0

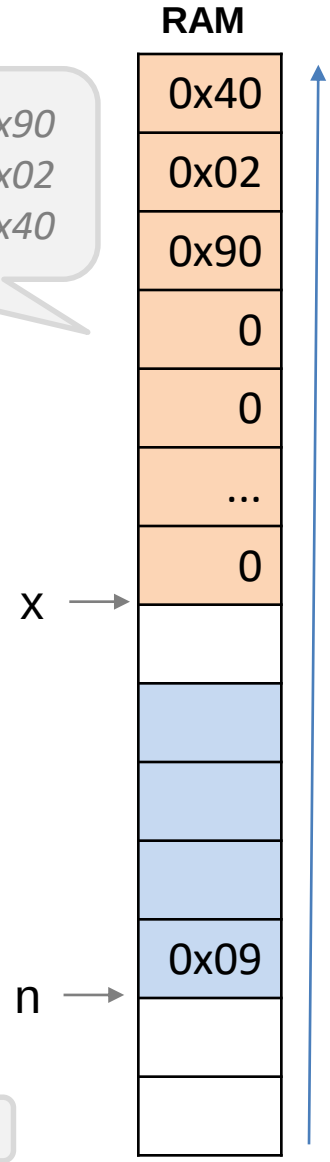
Exponent = 11 b (+ 16383 d bias) = 100 0000 0000 0010 b

Mantissa = 1001 b

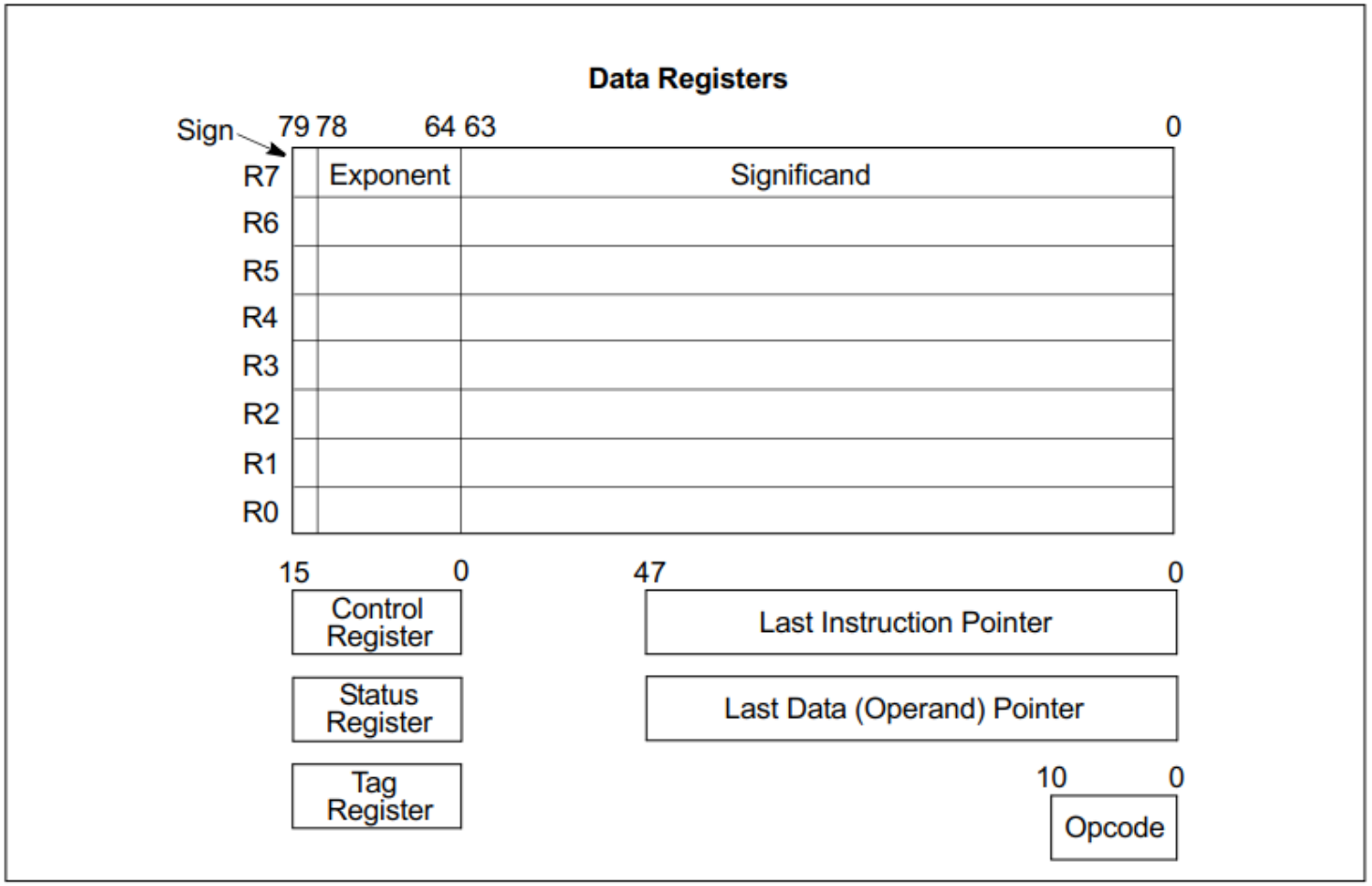
Sign | Exponent | Mantissa

79	78	…	65	64	63	62	61	60	…	2	1	0
0	1	0…	1	0	1	0	0	1	0…	0	0	0

float-point number in x87 data registers stack



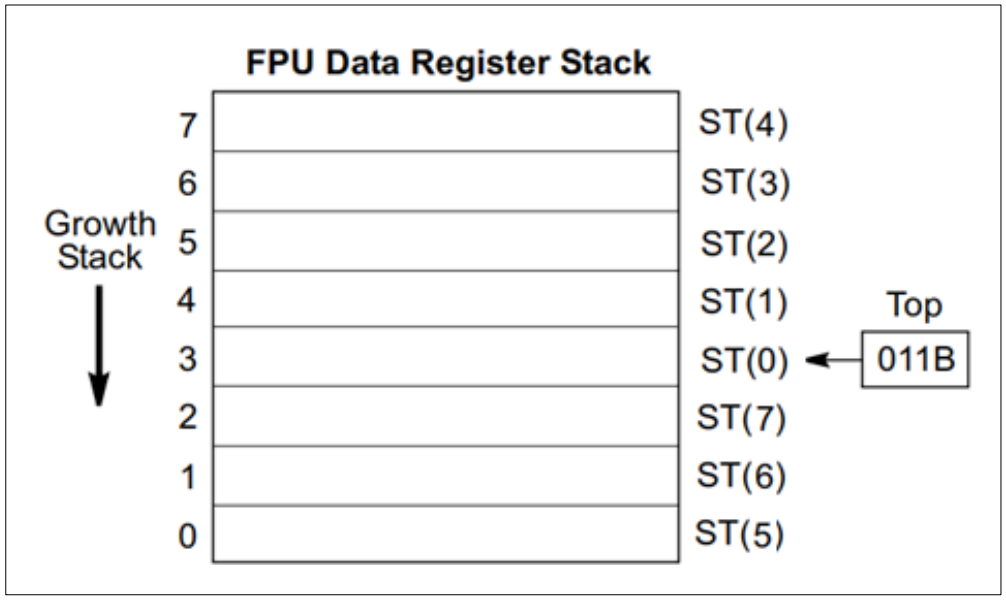
x87 FPU represents a separate execution environment, consists of 8 data registers and special-purpose registers



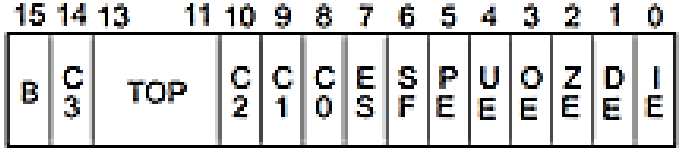
x87 FPU instructions treat the eight x87 FPU data registers as a register stack

The register number of the current top-of-stack register is stored in the TOP (stack TOP) field in the x87 FPU status word.

Load operations decrement TOP by one and load a value into the new top-of-stack register. Store operations store the value from the current TOP register in memory and then increment TOP by one.



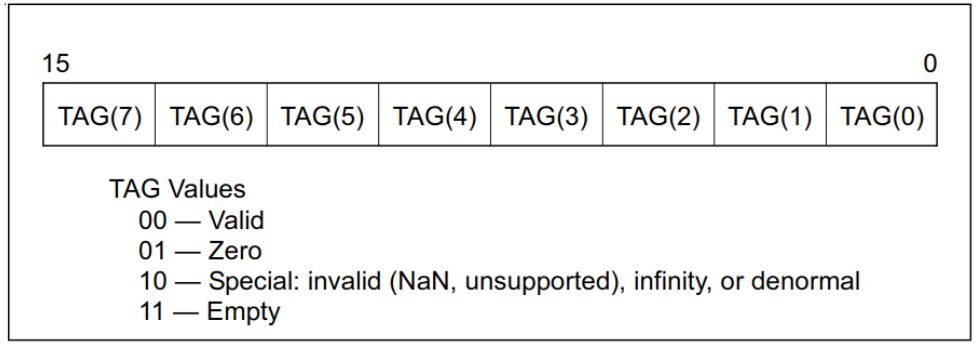
status register

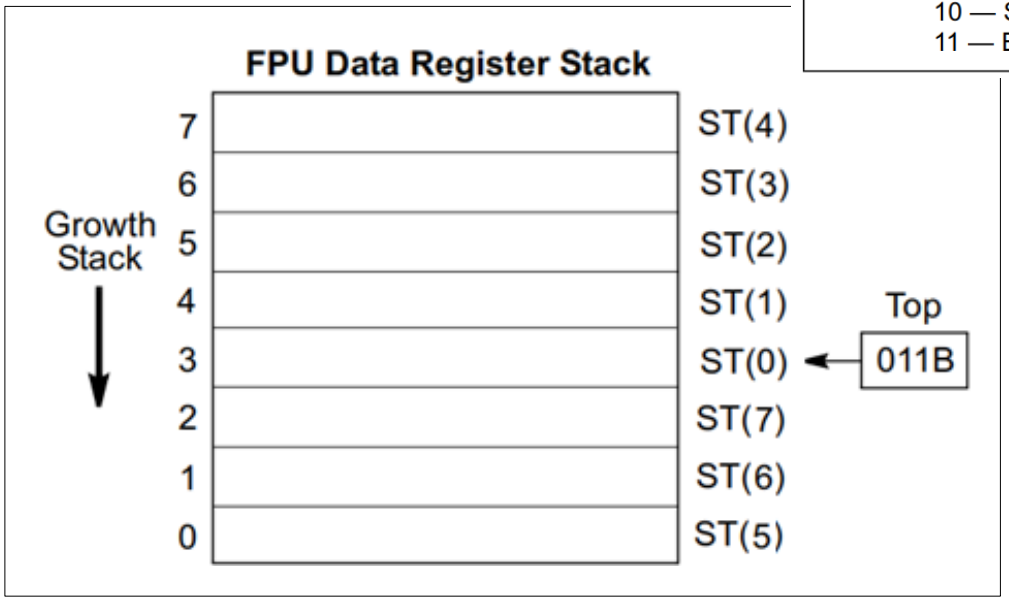


16-bit x87 FPU status register indicates the current state of x87 FPU

16-bit tag word indicates the contents of each the 8 registers in x87 FPU data-register stack (one 2-bit tag per register).

Each tag in tag word corresponds to a physical register. TOP pointer is used to associate tags with registers relative to ST(0).





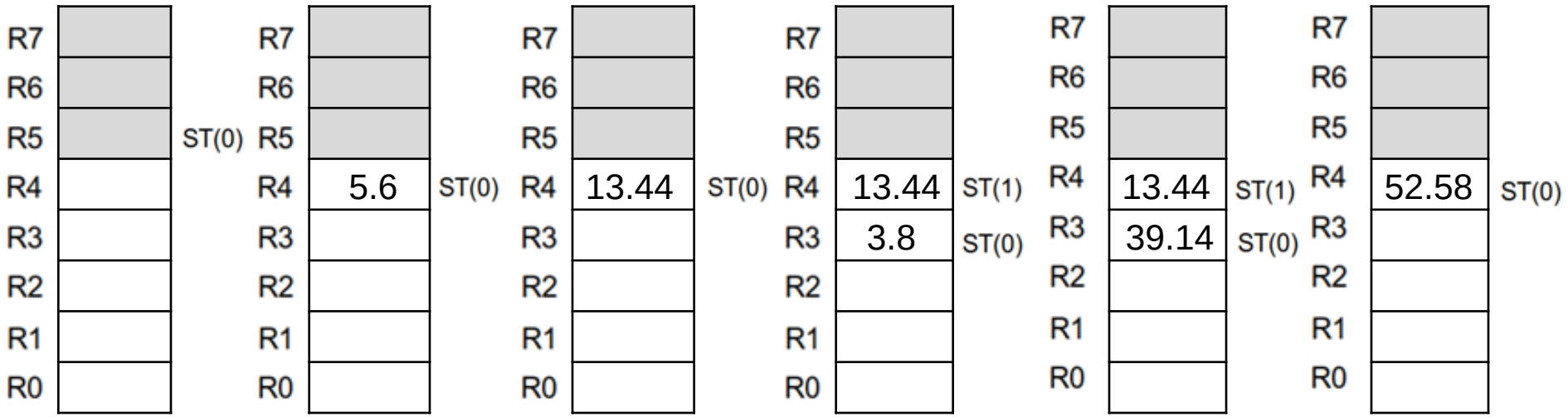
Dot Product = (5.6 x 2.4) + (3.8 x 10.3)

```
var1:    dt 5.6
var2:    dt 2.4
var3:    dt 3.8
var4:    dt 10.3

fld tbyte [var1]    ; st0 = 5.6, TOP=4
fmul tbyte [var2]   ; st0=st0*2.4=13.44, TOP=4
fld tbyte [var3]    ; st0=3.8, st1=13.44, TOP=3
fmul tbyte [var4]   ; st0=st0*10.3=39.14, st1=13.44, TOP=3
faddp               ; st1=st1+st0, pop st0, TOP=4
```

gdb command to see x87 stack data registers:

(gdb) tui reg float



x87 Basic Arithmetic Instructions

FADD/FADDP	Add floating point
FIADD	Add integer to floating point
FSUB/FSUBP	Subtract floating point
FISUB	Subtract integer from floating point
FSUBR/FSUBRP	Reverse subtract floating point
FISUBR	Reverse subtract floating point from integer
FMUL/FMULP	Multiply floating point
FIMUL	Multiply integer by floating point
FDIV/FDIVP	Divide floating point
FIDIV	Divide floating point by integer
FDIVR/FDIVRP	Reverse divide
FIDIVR	Reverse divide integer by floating point
FABS	Absolute value
FCHS	Change sign
FSQRT	Square root
FPREM	Partial remainder
FPREM1	IEEE partial remainder
FRNDINT	Round to integral value
FXTRACT	Extract exponent and significand

Operands in memory can be in single-precision floating-point, double-precision floating-point, word-integer, or doubleword-integer format. They all are converted to double extended-precision floating-point format automatically.

The pop versions of instructions offer the option of popping the x87 FPU register stack following the arithmetic operation. These instructions operate on values in ST(i) and ST(0) registers, store the result in the ST(i) register, and pop the ST(0) register.

x87 Control Instructions

FINIT/FNINIT	Initialize x87 FPU
FFREE	Free x87 FPU register

FINIT/FNINIT instructions initialize x87 FPU and its internal registers to default values

Stack overflow and underflow exceptions

Stack overflow — an instruction attempts to load a non-empty x87 FPU register from memory. A non-empty register is defined as a register containing a zero (tag value of 01), a valid value (tag value of 00), or a special value (tag value of 10).

Stack underflow — an instruction references an empty x87 FPU register as a source operand, including attempting to write the contents of an empty register to memory. An empty register has a tag value of 11.

x87 Constants

constant instruction pushes some commonly used constant onto st0

FLDZ	Load +0.0
FLD1	Load +1.0
FLDPI	Load π
FLDL2T	Load $\log_2 10$
FLDL2E	Load $\log_2 e$
FLDLG2	Load $\log_{10} 2$
FLDLN2	Load $\log_e 2$

x87 Trigonometric instructions

FSIN	Sine
FCOS	Cosine
FSINCOS	Sine and cosine
FPTAN	Tangent
FPATAN	Arctangent

x87 example

Area of a circle

```
section .data
radius:     dd 2.34
area:       dd 0.0
section .text
        ⋮
finit                   ; initialize the x87 subsystem
fld qword [radius]      ; load [radius] into st(0)
fst st1                 ; copy st(0) into st(1)
fmulp                   ; st(0) *= st(1)
fldpi                   ; push pi onto the stack
fmulp                   ; st(0) *= st(1)
fst dword [area]        ; store st(0) into [area]
```