

ECE390

Computer Engineering II

Lecture 15



Dr. Zbigniew Kalbarczyk
University of Illinois at Urbana- Champaign

Outline



- Floating point arithmetic
- 80x87 Floating Point Unit

Floating-Point Numbers

- Components of a floating point number

Sign: 0=pos(+); 1=neg(-)

Exponent: (with Constant bias added)

Mantissa: $1 \leq M < 2$

- Number = (1.Mantissa) * $2^{(\text{exponent}-\text{bias})}$

- Example: 10.25 (decimal)

= 1010.01 (binary)

= + 1.01001 * 2^3 (normalized)



Floating-point Format

Precision	Size (bits)	Exp. (bits)	Bias	Mant. (bits)	Mant. 1st Digit
Single	32	8	127 (7Fh)	23	Implied
Double	64	11	1023 (3FFh)	52	Implied
Extended	80	15	16383 (3FFFh)	63	Explicit

Converting to Floating-Point Form

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

1. $100.25 = 1100100.01$
2. $1100100.01 = 1.10010001 \times 2^6$
3. $110 + 01111111(7Fh) = 10000101(85h)$
4. **Sign** = 0

Exponent = 10000101

Mantissa = 100 1000 1000 0000 0000 0000

Note
mantissa of a number
1.XXXX is the XXXX
The 1. is an implied one-bit
that is only stored in the
extended precision form
of the floating-point number
as an explicit-one bit

Converting to Floating-Point Form

Format	Sign	Exponent	Mantissa
Single	(+)	$7F+6 = 85h$ (8 bits)	Implied 1. (23 bit)
	0	1000,0101	■ 100,1000,1000,0000,0000,0000
Double	(+)	$3FFh + 6 = 405h$ (11 bits)	Implied 1. (52 bit)
	0	100,0000,0101	■ 100,1000,1000,0000, ..., 0000
Extended	(+)	$3FFFh+6 = 4005h$ (15 bits)	Explicit 1. (63 bit)
	0	100,0000,0000,0101	1 ■ 100,1000,1000,0000, ..., 0000

Special Representations

- **Zero:**
 - Exp = All Zero, Mant = All Zero
- **NAN (not a number)** an invalid floating-point result
 - Exp = All Ones, Mant non-zero
- **+ Infinity:**
 - Sign = 0, Exp = All Ones, Mant = 0
- **- Infinity:**
 - Sign = 1, Exp = All Ones, Mant = 0

Converting from Floating-Point Form (example)

Sign = 1

Exponent = 1000,0011

Mantissa = 100,1001,0000,0000,0000

100 = 1000,0011 - 0111,1111 (convert the biased exponent)

1.1001001 x 2⁴ (a normalized binary number)

11001.001 (a de-normalized binary number)

-25.125 (a decimal number)

Variable declaration examples

- Declaring and Initializing Floating Point Variables

Fpvar1	dd	-7.5	; single precision (4 bytes)
Fpvar2	dq	0.5625	; double precision (8 bytes)
Fpvar3	dt	-247.6	; extended precision (10 bytes)
Fpvar4	dt	345.2	; extended precision (10 bytes)
Fpvar5	dt	1.0	; extended precision (10 bytes)

The 80x87 Floating Point Unit

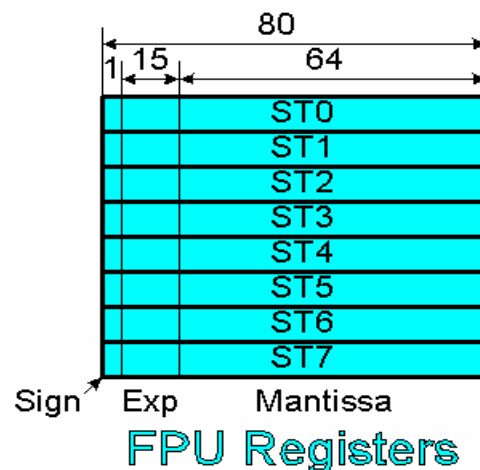
- On-chip hardware for fast computation on floating point numbers
- FPU operations run in parallel with integer operations
- Historically implemented as separate chip
 - 8087-80387 Coprocessor
- 80486DX-Pentium III contain their own internal and fully compatible versions of the 80387
- CPU/FPU Exchange data through memory
 - Converts Single and Double to Extended-precision

Arithmetic Coprocessor Architecture

- Control unit
 - interfaces the coprocessor to the microprocessor-system data bus
- Numeric execution unit (NEU)
 - executes all coprocessor instructions
- NEU has eight-register stack
 - each register is 80-bit wide (extended-precision numbers)
 - data appear as any other form when they reside in the memory system
 - holds operands for arithmetic instructions and the results of arithmetic instructions
 - **FSTSW AX** ;the only instruction (not available for 8087) that allows direct communications between CPU and FPU through the AX register
- Other registers - status, control, tag, exception

Arithmetic Coprocessor FPU Registers

- The eight 80-bit data registers are organized as a stack
- As data items are pushed into the top register, previous data items move into higher-numbered registers, which are lower on the stack
- All coprocessor data are stored in registers in the 10-byte real format
- Internally, all calculations are done on the greatest precision numbers
- Instructions that transfer numbers between memory and coprocessor automatically convert numbers to and from the 10-byte real format



Sample 80x87 FPU Opcodes

FADD:	Addition
FMUL:	Multiplication
FDIV:	Division (divides the destination by the source)
FDIVR:	Division (divides the source by the destination)
FSIN:	Sine (uses radians)
FCOS:	Cosine (uses radians)
FSQRT:	Square Root
FSUB:	Subtraction
FABS:	Absolute Value
FYL2X:	Calculates $Y * \log_2 X(X)$ (Really)
FYL2XP1:	Calculates $Y * \log_2 X(X)+1$ (Really)

Programming the FPU

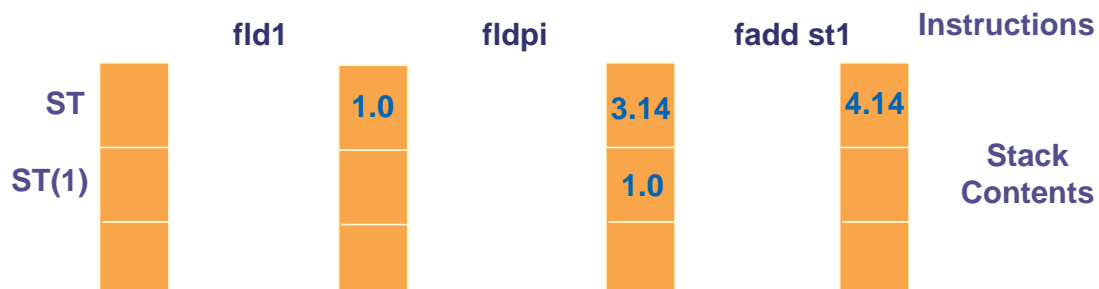
- FPU Registers = ST0 .. ST7
- Format: **OPCODE destination, source**
- Restrictions: One register must be ST0
- Use of ST0 is implicit when not specified otherwise
- **FINIT** - Execute before starting to initialize FPU registers!

Programming the FPU (cont.)

- FLD size[MemVar]
 - Load ST0 with MemVar &
 - Push all other ST_i to ST_(i+1) for i=0..6
- FSTP size[MemVar]
 - Move top of stack (ST0) to memory &
 - Pop all other ST_i to ST_(i-1) for i = 1..7
- FST size[MemVar]
 - Move top of stack to memory
 - Leave result in ST0

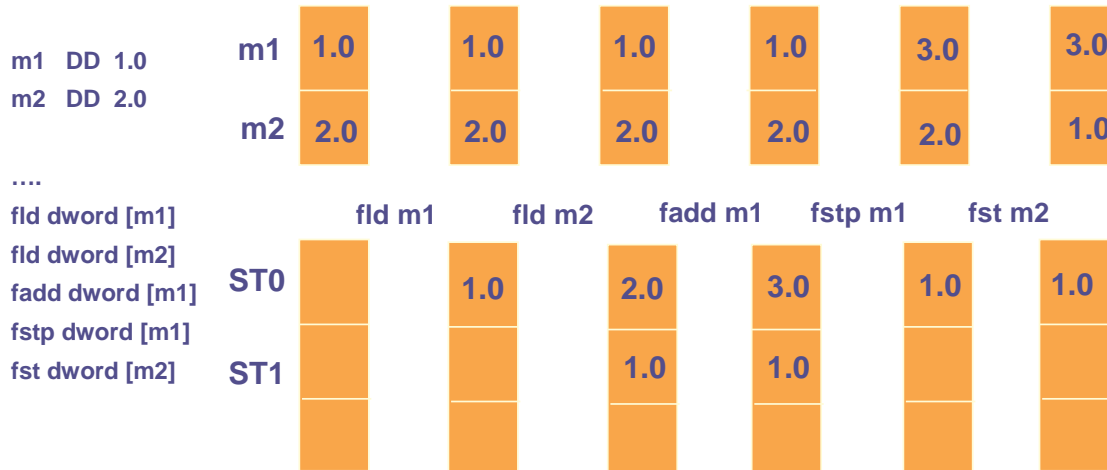
FPU Programming Classical-Stack Format

- Instructions treat the coprocessor registers like items on the stack
- ST0 (the top of the stack) is the source operand
- ST1, the second register, is the destination



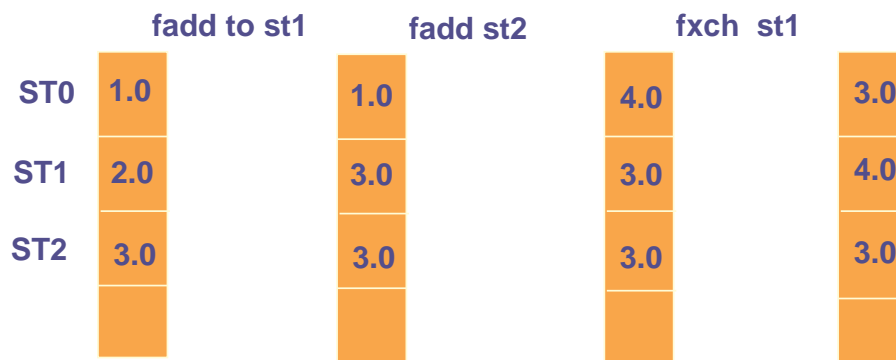
FPU Programming Memory Format

- The stack top (ST0) is always the implied destination



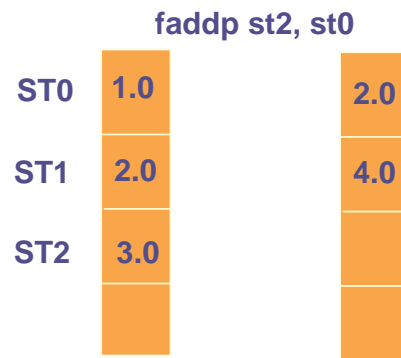
FPU Programming Register Format

- Instructions treat coprocessor registers as registers rather than stack elements



FPU Programming Register-Pop Format

- Coprocessor registers are treated as a modified stack
 - the source register must always be a stack top



FPU Programming Timing Issues

- A coprocessor instruction following a processor instruction
 - coordinated by assembler for 8086 and 8088
 - coordinated by processor on 80186 - Pentium
- A processor instruction that accesses memory following a coprocessor instruction that accesses the same memory
 - for 8087 need include **WAIT or FWAIT** to ensure that coprocessor finishes before the processor begins
 - assembler is doing this automatically

Calculating the area of a circle

```
RAD    DD    2.34, 5.66, 9.33, 234.5, 23.4    fmul st1    ; multiply ST0 = ST0 x ST1
AREA   RESD  5                                fstp dword [AREA + di]
..START                                     add  si, 4
    mov  si, 0    ;source element 0          add  di, 4
    mov  di, 0    ;destination element 0     loop  .Main1
    mov  cx, 5    ;count of 5
    finit
    fldpi          ; pi to ST0
    mov  ax, 4C00h
    int  21h

.Main1:
    fld  dword [RAD + si] ;radius to ST0
    fmul st0          ; square radius
```

Floating Point Operations Calculating $\text{Sqrt}(x^2 + y^2)$

```
X          dd    4.0
Y          dd    3.0
Z          resd  1
                                fld  dword [Y]
                                fld  st0
                                fmulp st1, st0
..start
    mov  ax, cs    ; Initialize DS=CS
    mov  ds, ax
                                faddp st1, st0
                                fsqrt
                                fstp  dword [Z]
    fld  dword [X]
    fld  st0
    fmulp st1, st0
                                mov  ax, 4C00h    ;Exit to DOS
                                int  21h
```