
Lecture 8: Floating Point

Dragan Mirkovic
Department of Computer Science
University of Houston

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Announcements

- Quiz #5 on Thursday:
 - Arrays and strings (Ch. 10 and 17)
- Today we will start talking about the floating point operations and the associated hardware
 - Chapter 19 in text

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Introduction

- A FP numbers constitute a discrete subset of the continuum of real numbers.
- A floating point calculations are very important in many applications
 - Scientific and engineering applications in particular.
 - Mflop performance measurement unit.
- A logically separate part of the CPU is dedicated to the manipulation of the FP numbers: Floating-Point Unit (FPU)
 - Old CPUs used to have a separate (optional) chip.
 - Starting with Pentium FPU is on the same chip as the rest of the hardware

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

FPU Data

- The FPU manipulates five forms of data:
 - Short floating point (32-bit, float in C/C++)
 - Long floating point (64-bit, double in C/C++)
 - Integer (16-bit, short int in C/C++)
 - Long integer (32-bit, int in C/C++)
 - BCD (Binary Coded Decimal), a form which is close to an integer in ASCII format and is used for translating floating point numbers to and from ASCII. It is 80 bits long.
- Inside the FPU, all numbers are converted to a sixth form:
 - Extended floating-point (80-bit)

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Floating-Point Numbers

- All three floating point formats are analogous to the standard exponential (scientific) notation of the form

$$\pm n.nnnnnn \times 10^{\pm eee}$$

- On a computer this is written as

$$\pm n.nnnnnnE \pm eee$$

- and represented using the binary format:

$$\pm 1.11001110 \times 2^{\pm 1011}$$

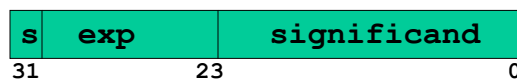
- The FP number has three groups of bits:
 - The **sign** bit (1 = -, 0 = +).
 - The **exponent** bits – positive or negative power of two
 - The **significand** (mantissa of fraction)

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

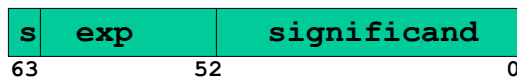
FP Number formats

- Intel processors use three FP binary storage formats specified in the *Standard 754-1985 for Binary Floating-Point Arithmetic* produced by the IEEE organization.

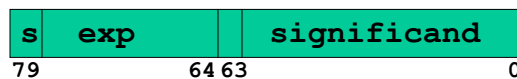
- 32-bit single precision format:



- 64-bit double precision format:



- 80-bit extended precision format:



D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

FP Number Representation

- The sign-magnitude representation is used for the whole number: x and $-x$ differ only in the sign bit.
- The sign of the exponent uses a bias which is 1023 for long fp numbers, and the exponent must lie in the range

$$-1023 \leq \text{exponent} \leq 1024$$

- Hence, $0 \leq \text{exp} \leq 2047$, which fits into 11 bits.
- Since $\log_2 10 \cong 3.3219 \Rightarrow 2^{-1023} \cong 2.225 \times 10^{-308}$
- The other bounds can be obtained in the similar fashion.

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Exponent Representation

Format	Exponent length	Exponent bias	Exponent range
Short	8	127	$-126 \leq \text{exp} \leq 127$
Long	11	1023	$-1022 \leq \text{exp} \leq 1023$
Extended	15	16383	$-16382 \leq \text{exp} \leq 16383$

- The range of the biased exponent is between 1 and $2^l - 2$, where l is the length of the exponent.
- Exponent values $0 \dots 0$ and $1 \dots 1$ are reserved for special encodings

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Range of FP Numbers

Format	Decimal digits of precision	Range
Short	7	$1.18 \times 10^{-38} \leq x \leq 3.4 \times 10^{38}$
Long	15	$2.23 \times 10^{-308} \leq x \leq 1.79 \times 10^{308}$
Extended	19	$3.37 \times 10^{-4932} \leq x \leq 1.18 \times 10^{4932}$

- Precision does not guarantee the accuracy but it is a necessary condition for it
- Accuracy is always less than or equal to the precision

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Examples

- The hexadecimal encoding for minimum and maximum floating point numbers in REAL4 and REAL8 formats

```
00800000      L0 REAL4  1.1754944E-38; Min short fp
7F7FFFFF      L1 REAL4  3.4028235E38; Max short fp
0010000000000000 L2 REAL8  2.2250738585072015E-308; Min long fp
7FEFFFFFFFFFFFFF L3 REAL8  1.7976931348623158E+308; Max long fp
```

- The hexadecimal encoding for minimum and maximum floating point numbers in extended format

```
00018000000000000000 L4 REAL10  3.3621031431120935065E-4932;
7FEFFFFFFFFFFFFF L5 REAL10  1.1897314953572317650E+4932;
```

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

FP Number Normalization

- The significand is almost always normalized, i.e. the first digit of a nonzero number is greater than zero (equal to 1 in binary representation)
- Hence, it can be omitted from the representation.
- Thus, if the significand is $b_{51}b_{50} \dots b_1b_0$, then the number represented is
$$(-1)^s \times 1.b_{51}b_{50} \dots b_1b_0 \times 2^{\text{exp}-1023}$$
- That requires a special representation for zero.
- Extended precision reserves 1 bit for the integer part of the significand
- There are special encodings for several other real numbers and non-numbers

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Normalizing and Denormalizing

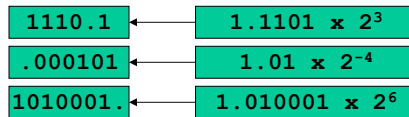
- Most floating-point binary numbers are stored in normalized form
 - This maximizes the precision of the significand.
- Any floating-point binary number can be normalized by shifting the binary point until a single "1" appears to the left of the binary point.
- The exponent expresses the number of positions the binary point is moved left (positive exponent) or moved right (negative exponent).
- Examples:

1110.1	→	1.1101 × 2 ³
.000101	→	1.01 × 2 ⁻⁴
1010001.	→	1.010001 × 2 ⁶

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Denormalizing

- Denormalizing a floating-point binary number reverses the normalizing process.
- Shift the binary point until the exponent is zero.
 - If the exponent is positive n , shift the binary point n positions to the right
 - If the exponent is negative n , shift the binary point n positions to the left, filling leading zeros if necessary.
- Examples:



D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Examples

- Single-Precision Bit Encodings

Binary Value	Biased Exponent	Sign, Exponent, Significand
-1.11	127	1 01111111 1100000000000000000000
+1101.101	130	0 10000010 1011010000000000000000
-.00101	124	1 01111100 0100000000000000000000
+100111.0	132	0 10000100 0011100000000000000000
+.0000001101011	120	0 01111000 1010110000000000000000

- biased exponent range:
1 – 254 (00000001 - 11111110)

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Declaring FP Numbers

- For short (32-bit) FP numbers use `DD` or `REAL4` declarations
- For long (64-bit) FP numbers use `DQ` or `REAL8` declarations
- For extended (80-bit) floats use `DT` or `REAL10` declarations
- Examples:

```

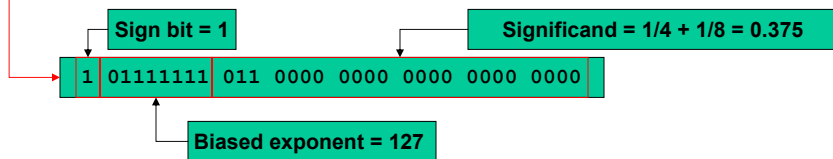
A DD      -1.375;    A negative short fp
B REAL4   54.0E23;   A positive short fp
D DD      -1234;     A negative long int
D1 DD     -1234.;    A negative short fp
E DQ      -1.375;    A negative long fp
F DT      -1.375;    A negative extended fp
G REAL10  1.375;     A positive extended fp
    
```

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Hexadecimal encodings

```

0000          .DATA
0000 BFB00000  A DD      -1.375;    A negative short fp
0004 688EEFD3  B REAL4   54.0E23;   A positive short fp
0008 FFFFFB2E  D DD      -1234;     A negative long int
000C C49A4000  D1 DD     -1234.;    A negative short fp
0010 BFF6000000000000  E DQ      -1.375;    A negative long fp
0018 BFFFFB00000000000000  F DT      -1.375;    A neg. extended fp
0022 3FFFB000000000000000  G REAL10  1.375;    A pos. extended fp
0040 00800000  L0 REAL4   1.1754944E-38; Min short fp
0044 7F7FFFFF  L1 REAL4   3.4028235E38; Max short fp
    
```



D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

FPU Integers and BCD Numbers

- The FPU uses the usual 2's complements defined by `DW` and `DD`.
- BCD Numbers are integers and consist of 18 decimal digits and a sign
 - One digit occupies 4 bits (**nibble**)
 - Declared using `TBYTE` (define **ten-byte**)
 - Example:
`A TBYTE -1234567890`
 - Is encoded as
`80000000001234567890h`

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Real Number Encodings

- The IEEE specification includes several real-number and non-number encodings.
 - Positive and negative zero
 - Denormalized finite numbers
 - Normalized finite numbers
 - Positive and negative infinity
 - Non-numeric values (NaN, known as *Not a Number*)
 - Indefinite numbers

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Specific Encodings

- Values often encountered in floating-point operations
- Bit positions marked with the letter x can be either 1 or 0.
- QNaN is a quiet NaN, and SNaN is a signalling NaN.

Value	Sign, Exponent, Significand
Positive Zero	0 00000000 000000000000000000000000
Negative Zero	1 00000000 000000000000000000000000
Positive Infinity	0 11111111 000000000000000000000000
Negative Infinity	1 11111111 000000000000000000000000
QNaN	x 11111111 1xxxxxxxxxxxxxxxxxxxxxxxxxxxx
SNaN	x 11111111 0xxxxxxxxxxxxxxxxxxxxxxxxxxxx ^a

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

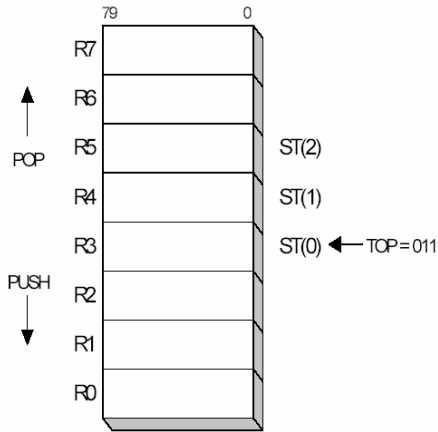
IA-32 FP Architecture

- The original Intel 8086 processor was designed to handle only integer arithmetic.
- Problem for floating-point-intensive applications
- Emulation of floating-point arithmetic through software
 - Severe performance penalty
 - Intel sold a separate floating-point coprocessor chip named the 8087, and upgraded it along with each processor generation.
- Starting with the 80486, it was integrated into the main CPU and renamed as the *Floating-Point Unit* (FPU).

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

FPU Data Registers

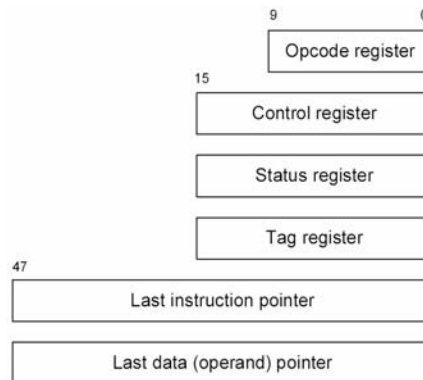
- The FPU has eight 80-bit registers arranged in the form of a register stack.
- A 3-bit field named **TOP** in the FPU status word marks the top of the stack.
- In the example, TOP equals binary 011, identifying R3 as the top of the stack.
- This stack location is also known as ST(0) (or simply ST).



D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

FPU Special Registers

- The FPU has six *special-purpose* registers
 - A 10-bit opcode register
 - A 16-bit control register
 - A 16-bit status registers
 - A 16-bit tag word register
 - A 48-bit last instruction pointer register
 - A 48-bit last data (operand) pointer register



D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Form of FP Instructions

- All FP instructions start with an initial 'f'
foperation operands
- A 'b' or 'i' as the second letter indicates a BCD or integer operand
fboperation **BCD** operand
fioperation **Integer** operand
- If the last letter of the opcode is 'p' the FPU stack is popped on the completion of the operation
foperationp operand

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

The FPU Stack

- Important differences between FPU stack and 80x86 stack
 - The FPU stack is in the FPU. The 80x86 stack is in main memory.
 - The FPU stack is always 8 registers long
 - In every FPU computation, one of the operands is **ST**.
 - Trying to load (push) into a non-empty entry in the FPU stack is an error.
 - Trying to compute with an empty entry in the FPU stack is an error.

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

FPU Load Instructions

- FPU Load (Push):
 - `fld mem32 or mem64;` short or long fp load
 - `fld ST(n);` load from another stack entry
 - `fild mem16 or mem32;` int or long load
 - `fbld mem80;` BCD load
- FPU Load (Push) Constants:
 - `fld1` ; load 1
 - `fldz` ; load zero
 - `fldpi` ; load π
 - `fldl2t` ; load $\log_2 10$
 - `fldl2e` ; load $\log_2 e$
 - `fldlg2` ; load $\log_{10} 2$
 - `fldln2` ; load $\log_e 2$

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

FPU Store [and pop] ST

- FPU Store [and pop] Instructions:
 - `fst[p] mem32 or mem64;` store [and pop] ST
 - `fst[p] ST(n);` copy [then pop] ST to ST(n)
 - `fist[p] mem16 or mem32;` store [and pop] ST
as int
 - `fbstp mem80;` store [and pop] ST as BCD
; no non-popping version of `fbstp`

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

The FPU Arithmetic

- There are six basic FPU instruction formats:

Instruction Format	Mnemonic Format	Operands (Dest, Source)	Example
Classical Stack	<i>Fop</i>	{ST(1),ST}	FADD
Classical Stack, extra pop	<i>FopP</i>	{ST(1),ST}	FSUBP
Register	<i>Fop</i>	ST(n),ST ST, ST(n)	FMUL ST(1),ST FDIV ST,ST(3)
Register, pop	<i>FopP</i>	ST(n),ST	FADDP ST(2),ST
Real Memory	<i>Fop</i>	{ST},memReal	FDIVR payRate
Integer Memory	<i>Flop</i>	{ST},memInt	FILD hours

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

The FPU Instructions...

- Operands surrounded by braces {...} are implied operands and are not explicitly coded.
- **ST** denotes **ST (0)**
- The operation may be one of the following:
 - ADD** Add source to destination
 - SUB** Subtract source from destination
 - SUBR** Subtract destination from source
 - MUL** Multiply source by destination
 - DIV** Divide destination by source
 - DIVR** Divide source by destination

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Examples

- Typical FPU stack operations

```
fld op1 ; op1 = 20.0
fld op2 ; op2 = 100.0
fadd
```
- The FADD instruction adds ST(0) to ST(1) and leaves the result at the top of the stack:

	Before		After
ST(0)	100.0	ST(0)	120.0
ST(1)	20.0	ST(1)	

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004

Example

- Add Three Numbers
 - Calculate the sum of three single-precision numbers that are stored in an array.
 - FLD loads from memory into ST(0).
 - FADD adds its operand to the number at ST(0).
 - FSTP stores the number at ST(0) into memory, and pops the value from the stack:

```
.DATA
sngA REAL4 1.5, 3.4, 6.6
sum REAL4 ?
.CODE
fld sngA;   load mem into ST(0)
fadd [sngA+4]; add mem to ST(0)
fadd [sngA+8]; add mem to ST(0)
fstp sum;  store ST(0) to mem
```

D. Mirkovic, COSC 2410: Computer Organization and Programming, Spring 2004