

EEC 483 Computer Organization (Spring 2006)

Chapter 3. Arithmetic for Computers

Chansu Yu

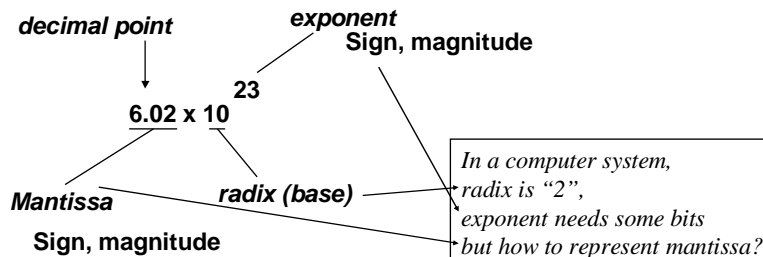
Cleveland State University

Table of Contents

- Ch.1, 4 Introduction & Performance
 - Ch. 2 Instruction: Machine Language
 - 2.1 Introduction
 - 2.2 Operations (arithmetic, memory operations)
 - 2.3 Operands
 - 2.4 Representing instructions
 - 2.5 Logical operations
 - 2.6 Control flow operations
 - 2.7 Supporting procedures
 - 2.8 Beyond numbers
 - 2.9 MIPS addressing
 - 2.10 Starting a program
 - Ch. 3 CPU Implementation: Arithmetic
 - 3.1 Introduction
 - 3.2 Signed and unsigned numbers
 - 3.3 Addition and subtraction
 - 3.4 Multiplication
 - 3.5 Division
 - **3.6 Floating point**
 - Appendix B Constructing an Arithmetic Logic Unit (ALU)
 - Ch. 5 CPU Implementation: All others
 - Ch. 6-9 Advanced topics
- Software interface* (points to Ch. 2)
- Hardware interface* (points to Ch. 3)
- Key parts of this course** (bracketed around Ch. 2 and Ch. 3)

3.6 Floating Point

- We need a way to represent
 - numbers with fractions, e.g., 3.1416
 - very small numbers, e.g., .000000001
 - very large numbers, e.g., 3.15576×10^9



Cleveland State University

3

c.yu91@csuohio.edu

Floating Point – Number System

□ Binary number system

➤ Integer

- $1000_2 = 2^3 = 8$
- $100_2 = 2^2 = 4$
- $10_2 = 2^1 = 2$
- $1_2 = 2^0 = 1$

➤ Fraction

- $0.1_2 = 2^{-1} = 0.5$
- $0.01_2 = 2^{-2} = 0.25$
- $0.001_2 = 2^{-3} = 0.125$

Example: -0.078125

- $\times 2 = 0.15625$
- $\times 2 = 0.3125$
- $\times 2 = 0.625$
- $\times 2 = 1.25$
- $\times 2 = 2.5$
- $\times 2 = 5 = 101_2$

Thus,

$$\begin{aligned}
 -0.078125 &= -101_2 \times 2^{-6} \\
 &= -10.1_2 \times 2^{-5} \\
 &= -1.01_2 \times 2^{-4} \\
 &= -0.101_2 \times 2^{-3} \\
 &= -0.0101_2 \times 2^{-2}
 \end{aligned}$$

Standard form

Cleveland State University

4

c.yu91@csuohio.edu

IEEE 754 Floating-Point Standard

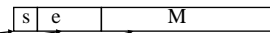
- ❑ Leading “1” bit of significand is implicit
- ❑ Exponent is “biased” to make sorting easier
 - all 0s is smallest exponent all 1s is largest
 - bias of 127 for single precision and 1023 for double precision

- ❑ Summary: $(-1)^{\text{sign}} \times \text{significand} \times 2^{\text{exponent}}$
 $\Rightarrow (-1)^{\text{sign}} \times (1+\text{significand}) \times 2^{\text{exponent} - 127}$

IEEE F.P. $\pm 1.M \times 2^{e-127}$

- ❑ IEEE 754 floating point standard:

- single precision: 1 sign, 8 bit exponent, 23 bit significand => 32-bit
- double precision: 1 sign, 11 bit exponent, 52 bit significand => 64-bit
- Tradeoff between accuracy and range
 - more bits for significand gives more accuracy
 - more bits for exponent increases range



exponent=e-127, mantissa=1.M
 $\Rightarrow (-1)^s \times 1.M \times 2^{e-127}$

IEEE 754 Floating-Point Standard

- ❑ Example:

- $-0.078125 = -1.01_2 \times 2^{-4}$
- Exponent: $e-127 = -4 \Rightarrow e=123=0111\ 1011_2$
- Significand: $1+M = 1.01_2 \Rightarrow M=0.01_2$
- IEEE single precision:
 $10111101101000000000000000000000$

Range of IEEE 754

- IEEE single precision:

0 0111 1111 000000000000000000000000	2^{-1}	2^{-23}	$1.000...000 \times 2^0 = 1$
+ 127 => e=0			
00000000000000000000000000000001			$1.000...001 \times 2^0 = 1 + 2^{-23} = 1 + \epsilon$
00000000000000000000000000000010			$1.000...010 \times 2^0 = 1 + 2^{-22}$
00000000000000000000000000000011			$1.000...011 \times 2^0 = 1 + 2^{-22} + 2^{-23}$
.....			
10000000000000000000000000000000			$1.100...000 \times 2^0 = 1 + 2^{-1} = 1.5$
10000000000000000000000000000001			$1.100...001 \times 2^0 = 1 + 2^{-1} + 2^{-23} = 1.5 + \epsilon$
.....			
11111111111111111111111111111111			$1.111...111 \times 2^0 = 1 + 2^{-1} + \dots + 2^{-23}$ $= 2 - 2^{-23} = 2 - \epsilon$

Range of IEEE 754

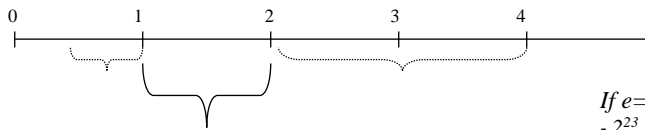
- IEEE single precision

- Exponent:
 - 1~254 (exclude 0 and 255 from 0~255)
 - e = 1~254 - 127 = -126 ~ 127
- Significand:
 - 1~2
 - Is it (1,2), [1,2), (1,2], or [1,2] ? Answer: [1,2) => 1~2-2⁻²³
- Largest positive
 - Exponent: 254 => e=127 } $+(2 - \epsilon) \times 2^{127} \approx 2^{128} \approx 3.4 \times 10^{38}$
 - Significand: all 1's = 2 - ϵ } 0 11111110 1111.....1111
- Smallest positive
 - Exponent: 1 => e=-126 } $+1 \times 2^{-126} = 2^{-126} \approx 1.8 \times 10^{-38}$
 - Significand: all 0's = 1 } 0 00000001 0000.....0000
- Largest & smallest negative ???

Uniformity of IEEE 754

- IEEE single precision:

		2^{-1}	2^{-23}
0 0111 1111	000000000000000000000000		$1.000...000 \times 2^0 = 1$
(e=0)	000000000000000000000001		$1.000...001 \times 2^0 = 1 + 2^{-23} = 1 + \epsilon$
Covers [1,2)		
	111111111111111111111111		$1.111...111 \times 2^0 = 2 - 2^{-23} = 2 - \epsilon$



How many points in [1,2)? Ans: 2^{23} points
 Are they equally spaced? Ans: Yes
 Then, what's the space? Ans: 2^{-23}

If $e=1$, it covers [2,4).
 - 2^{23} points
 - Equally spaced: 2^{-22}
 If $e=-1$, it covers [0.5, 1).
 - 2^{23} points
 - Equally spaced: 2^{-21}

Denormalized Numbers



No points !!!
 How many points? Ans: 2^{23} points
 Are they equally spaced? Ans: Yes
 Then, what's the space? Ans: 2^{-149}

If $e=0$ is also a normalized numbers, exponent= $e-127=-127$
 $\Rightarrow [1,2) \times 2^{-127} = [2^{-127}, 2^{-126})$
 So, what happened to $[0, 2^{-127})$?

- The first interval from zero:
 - Exponent: $1 \Rightarrow e=-126 \Rightarrow [1,2) \times 2^{-126} = [2^{-126}, 2^{-125})$
 - Space= 2^{-149}
 - Numbers are $2^{-126}, 2^{-126}+2^{-149}, 2^{-126}+2^{-148}, \dots$
 - The gap between 0 and the next representable number is much larger than the gaps between nearby representable numbers.
 - How to represent the numbers in $(0, 2^{-126})$?
- IEEE standard uses denormalized numbers to fill in the gap, making the distances between numbers near 0 more alike.

Special Numbers

exponent=0-126=-126, mantissa=0.M
 $\Rightarrow (-1)^s \times 0.M \times 2^{-126} \Rightarrow [0, 2^{-126}]$

$$\pm 0.M \times 2^{e-126}$$

$$\pm 1.M \times 2^{e-127}$$

Exponent	Significand	Number
0	0	0
0	nonzero	\pm Denormalized number
1-254	anything	\pm Normalized number
255	0	\pm Infinity (e.g., 4/0, may continue operating if, for example, X>0)
255	nonzero	NaN (Not a Number) (e.g., 0/0, $\infty-\infty$, $\sqrt{-4}$)

Infinity and NaNs

result of operation *overflows*, i.e., is larger than the largest number that can be represented (underflow is also possible)

overflow is not the same as divide by zero (raises a different exception)

\pm infinity **S** 1...1 0...0

It may make sense to do further computations with infinity
 e.g., X/0 > Y may be a valid comparison

Not a number, but not infinity (e.g. sqrt(-4))
 invalid operation exception (unless operation is = or \neq)

NaN **S** 1...1 non-zero \leftarrow HW decides what goes here

NaNs propagate: f(NaN) = NaN

Extra Bits for Rounding

- ❑ "Floating Point numbers are normally approximations for a number they can't really represent."
- ❑ IEEE 754 offers several modes of rounding to let the programmer pick the desired approximation. => requires extra bits: "guard" and "round"
- ❑ How many extra bits?: As if computed the result exactly and rounded.
- ❑ $2.56 \times 10^0 + 2.34 \times 10^2$ (only three digits for significant)
 - + $\left. \begin{array}{l} 0.0256 \\ 2.3400 \end{array} \right\}$ with no rounding bits=> 2.36
 - with: guard holds "5", round holds "6"=> $2.36 + 0.0056 = 2.37$
- ❑ **Sticky bit**
 - set if there are nonzero bits to the right of the round bit
 - $2.56 \Rightarrow 0.02$ (0.0256), 5 in guard, 6 in round, 0 in sticky
 - $2.56 \Rightarrow 0.00$ (0.00000256), 0 in guard, 0 in round, 1 in sticky

Rounding Digits

IEEE Standard:

four rounding modes: round to nearest (default)
round towards plus infinity
round towards minus infinity
round towards 0

round to nearest:

round digit < B/2 then truncate
> B/2 then round up (add 1 to ULP: unit in last place)
= B/2 then round to nearest even digit

it can be shown that this strategy minimizes the mean error introduced by rounding

Operations with Floating Numbers

- Add
- Subtraction
- Multiplication
- Division

- Rounding

Basic Addition Algorithm

$$3.45 \times 10^{-1} + 2.57 \times 10^{-3}$$

$$(0.345) \quad (0.00257)$$

⇒ Align the decimal points, first

$$\Rightarrow 345 \times 10^{-3} + 2.57 \times 10^{-3}$$

$$\Rightarrow (345 + 2.57) \times 10^{-3}$$

$$\Rightarrow 347.57 \times 10^{-3} \text{ or } 3.4757 \times 10^{-1}$$

Basic Addition Algorithm

For addition (or subtraction) this translates into the following steps:

- (1) compute $Y_e - X_e$ (getting ready to align binary point)
- (2) right shift X_m that many positions to form $X_m 2^{X_e - Y_e}$
- (3) compute $X_m 2^{X_e - Y_e} + Y_m$

if representation demands normalization, then a normalization step follows:

- (4) left shift result, decrement result exponent (e.g., 0.001xx...)
right shift result, increment result exponent (e.g., 101.1xx...)
continue until MSB of data is 1 (NOTE: Hidden bit in IEEE Standard)

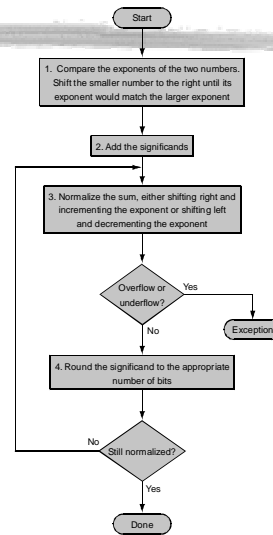
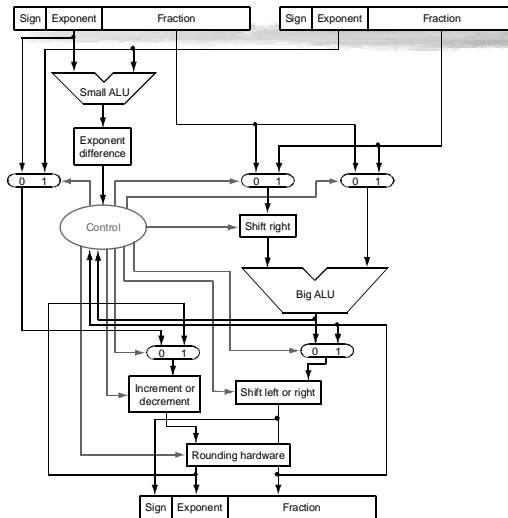
- (5) doubly biased exponent must be corrected:

$X_e = 7$	$X_e = 1111$	$= 15$	$= 7 + 8$
$Y_e = -3$	$Y_e = 0101$	$= 5$	$= -3 + 8$
Excess 8	10100	20	4 + 8 + 8

extra subtraction step of the bias amount

- (6) if result is 0 mantissa, may need to set the exponent to zero by special step

Floating point addition



Other Arithmetic

- ❑ Sub ~ Add
- ❑ Multiply
 - Exponent and significand are calculated separately
- ❑ Division
 - Very challenging
 - Many solutions for fast & accurate division
 - TI: Newton's iteration with extra-precise reciprocal
 - IBM: multiply-add
 - Cyrix: SRT (square root) division

Floating Points in MIPS

- ❑ Registers (Fig.3.19, p207)
 - \$f0, ..., \$f31 for fp operations
 - \$f0, ..., \$f31 are used in pairs for double precision operations (\$f0 = \$f0 | \$f1)
- ❑ Instructions (Fig.3.19/3.20, pp207-208)
 - add.s / add.d, sub.s/sub.d, mul.s/mul.d, div.s/div.d
 - Do mul/div use Hi/Lo registers? Ans: NO (ex: mul.d \$f2, \$f4, \$f6)
 - lwcl \$f1,100(\$s2) / swcl \$f1,100(\$s2) => load/store
 - bclt / bclf => branch if FP condition is true/false
 - c.x.s / c.x.d => set condition after FP comparison (x = eq, neq, lt, le, gt or ge) (eg: c.lt.s \$f2, \$f4) (eg: bclt label)

*There is a
destination register*

*There is no
destination register*

*There is no
registers to compare*

History

- ❑ 1980 Intel 8087
 - First processor implementing IEEE 754
 - Coprocessor to Intel 8086
 - Stack architecture

- ❑ Since 80386
 - Incorporated into the main chip

Pentium Bug

- ❑ Pentium FP Divider uses algorithm to generate multiple bits per steps
 - FPU uses most significant bits of divisor & dividend/remainder to guess next 2 bits of quotient
 - Guess is taken from lookup table: -2, -1,0,+1,+2 (if previous guess too large a remainder, quotient is adjusted in subsequent pass of -2)
 - Guess is multiplied by divisor and subtracted from remainder to generate a new remainder
 - Called SRT division after 3 people who came up with idea
- ❑ Pentium table uses 7 bits of remainder + 4 bits of divisor = 2^{11} entries
- ❑ 5 entries of divisors omitted: 1.0001, 1.0100, 1.0111, 1.1010, 1.1101 from PLA (fix is just add 5 entries back into PLA: cost \$200,000)
- ❑ Self correcting nature of SRT => string of 1s must follow error
 - e.g., 1011 1111 1111 1111 1111 1011 1000 0010 0011 0111 1011 0100
(2.99999892918)
- ❑ Since indexed also by divisor/remainder bits, sometimes bug doesn't show even with dangerous divisor value

Pentium bug appearance

- ❑ First 11 bits to right of decimal point always correct: bits 12 to 52 where bug can occur (4th to 15th decimal digits)
- ❑ FP divisors near integers 3, 9, 15, 21, 27 are dangerous ones:
 - $3.0 > d \parallel 3.0 - 36 \times 2^{-22}$, $9.0 > d \parallel 9.0 - 36 \times 2^{-20}$
 - $15.0 > d \parallel 15.0 - 36 \times 2^{-20}$, $21.0 > d \parallel 21.0 - 36 \times 2^{-19}$
- ❑ 0.333333×9 could be problem
- ❑ In Microsoft Excel, try $(4,195,835 / 3,145,727) * 3,145,727$
 - = 4,195,835 => not a Pentium with bug
 - = 4,195,579 => Pentium with bug (assuming Excel doesn't already have SW bug patch)
 - Rarely noticed since error in 5th significant digit
 - Success of IEEE standard made discovery possible:
 - all computers should get same answer

Pentium Bug Time line (Section 3.8)

- ❑ June 1994: Intel discovers bug in Pentium: takes months to make change, reverify, put into production: plans good chips in January 1995 4 to 5 million Pentiums produced with bug
- ❑ Scientist suspects errors and posts on Internet in September 1994
- ❑ Nov. 22 Intel Press release: "Can make errors in 9th digit ... Most engineers and financial analysts need only 4 of 5 digits. Theoretical mathematician should be concerned. ... So far only heard from one."
- ❑ Intel claims happens once in 27,000 years for typical spread sheet user:
 - 1000 divides/day x error rate assuming numbers random
- ❑ Dec 12: IBM claims happens once per 24 days: Bans Pentium sales
 - 5000 divides/second x 15 minutes = 4,200,000 divides/day
 - IBM statement: <http://www.ibm.com/Features/pentium.html>
 - Intel said it regards IBM's decision to halt shipments of its Pentium processor-based systems as unwarranted.
- ❑ Dec. 21: Intel announced "To owners of Pentium processor-based computers and the PC community: We at Intel wish to sincerely apologize for our handling of the recently publicized Pentium processor flaw." Cost \$500M.