CS2290 – COMPUTER ORGANIZATION AND ARCHITECTURE I

Floating-Point Processing

Dr. Mohammad Mirbagheri

Science Department

Northwestern Polytechnic

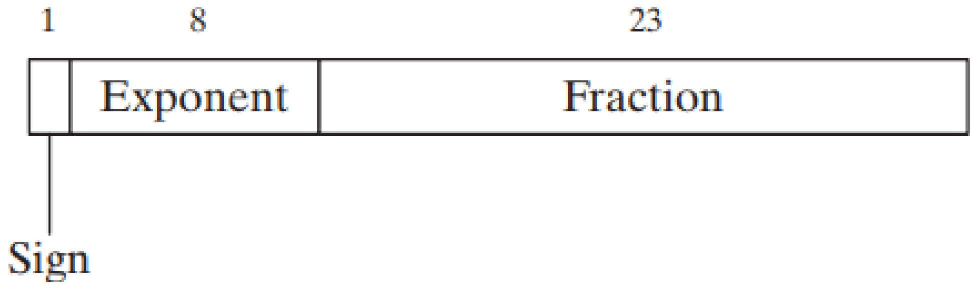
Adapted from slides and textbook by Kip Irvine. Assembly Language for x86 Processors, Pearson, 2015

Copyright © 2024. All rights reserved.

Real Number Encodings

- First Normalize

- 1.101×2^0
  - Sign bit: 0
  - Exponent: 01111111
  - Fraction: 10100000000000000000000



Exponent (E)	Biased (E + 127)	Binary
+5	132	10000100
0	127	01111111
–10	117	01110101
+127	254	11111110
–126	1	00000001
–1	126	01111110

Real Number Encodings

Value	Sign, Exponent, Significand
Positive zero	0 00000000 00000000000000000000000
Negative zero	1 00000000 00000000000000000000000
Positive infinity	0 11111111 00000000000000000000000
Negative infinity	1 11111111 00000000000000000000000

Real Number Encodings

Value	Sign, Exponent, Significand
Positive zero	0 00000000 00000000000000000000000
Negative zero	1 00000000 00000000000000000000000
Positive infinity	0 11111111 00000000000000000000000
Negative infinity	1 11111111 00000000000000000000000
QNaN	x 11111111 1xxxxxxxxxxxxxxxxxxxxxx
SNaN	x 11111111 0xxxxxxxxxxxxxxxxxxxxxx

SNaN significand field begins with 0, but at least one of the remaining bits must be 1

Converting Fractions to Binary Reals

- Express as a sum of fractions having denominators that are powers of 2

Decimal Fraction	Factored As...	Binary Real
1/2	1/2	.1
1/4	1/4	.01
3/4	1/2 + 1/4	.11
1/8	1/8	.001
7/8	1/2 + 1/4 + 1/8	.111
3/8	1/4 + 1/8	.011
1/16	1/16	.0001
3/16	1/8 + 1/16	.0011
5/16	1/4 + 1/16	.0101

Converting Fractions to Binary Reals

- *Binary long division method*
  - First convert the numerator and denominator to binary and then perform long division

convert decimal 0.2 (2/10) to binary using the binary long division method

Converting Single-Precision to Decimal

1. If the MSB is 1, the number is negative; otherwise; positive.
2. The next 8 bits represent the exponent. **Subtract** binary 01111111 (decimal 127), producing the unbiased exponent. Convert the unbiased exponent to decimal.
3. The next 23 bits represent the significand. Notate a “1.”, followed by the significand bits. Trailing zeros can be ignored. Create a floating-point binary number, using the significand, the sign determined in step 1, and the exponent calculated in step 2.
4. Denormalize the binary number produced in step 3.
5. From left to right, use weighted positional notation to form the decimal sum of the powers of 2 represented by the floating-point binary number.

Converting Single-Precision to Decimal

Convert **0 10000010 01011000000000000000000** to Decimal

Floating Point Unit

Floating Point Unit

- FPU has its own set of registers called a ***register stack***.

Floating Point Unit

- FPU has its own set of registers called a ***register stack*.**
- FPU instructions evaluate mathematical expressions in ***postfix*** format

Floating Point Unit

- FPU has its own set of registers called a ***register stack***.
- FPU instructions evaluate mathematical expressions in ***postfix*** format
  - The postfix equivalent
    - (5 * 6) - 4   ?
    - (A + B) * (C + D)   ?

Floating Point Unit

- FPU has its own set of registers called a ***register stack***.
- FPU instructions evaluate mathematical expressions in ***postfix*** format
  - The postfix equivalent
    - (5 * 6) - 4 5 6 * 4 -
    - (A + B) * (C + D) A B + C D + *

Floating Point Unit

- FPU has its own set of registers called a ***register stack.***
- FPU instructions evaluate mathematical expressions in ***postfix*** format
  - The postfix equivalent
    - (5 * 6) - 4 5 6 * 4 -
    - (A + B) * (C + D) A B + C D + *

Evaluating the Postfix Expression 5 6 * 4 – .

Left to Right	Stack		Action
5	5	ST (0)	push 5
5 6	5	ST (1)	push 6
	6	ST (0)	
5 6 *	30	ST (0)	Multiply ST(1) by ST(0) and pop ST(0) off the stack.
5 6 * 4	30	ST (1)	push 4
	4	ST (0)	
5 6 * 4 -	26	ST (0)	Subtract ST(0) from ST(1) and pop ST(0) off the stack.

Rounding

- FPU attempts to round an infinitely accurate result from a floating-point calculation
  - may be impossible because of storage limitations
- We could either round a number up to the next higher value or round it downward

Rounding

- FPU attempts to round an infinitely accurate result from a floating-point calculation
  - may be impossible because of storage limitations
- We could either round a number **up** to the next higher value or round it **downward**
  - Round to nearest even (default); pick the closest even number: e.g. 6.5 rounds to 6, but 7.5 rounds to 8
  - Round down toward negative infinity
  - Round up toward positive infinity
  - Round toward zero (truncate)

Rounding

Method	Precise Result	Rounded
Round to nearest even	1.0111	
Round down toward -∞	1.0111	
Round toward +∞	1.0111	
Round toward zero	1.0111	

Rounding

Method	Precise Result	Rounded
Round to nearest even	1.0111	1.100
Round down toward $-\infty$	1.0111	1.011
Round toward $+\infty$	1.0111	1.100
Round toward zero	1.0111	1.011

Rounding

Method	Precise Result	Rounded
Round to nearest even	-1.0111	
Round down toward $-\infty$	-1.0111	
Round toward $+\infty$	-1.0111	
Round toward zero	-1.0111	

Rounding

Method	Precise Result	Rounded
Round to nearest even	-1.0111	-1.100
Round down toward -∞	-1.0111	-1.100
Round toward +∞	-1.0111	-1.011
Round toward zero	-1.0111	-1.011

Floating-Point Addition

1. Match Exponents
2. Add the Two Mantissas (significands)
3. Normalize the Result
4. Check for Overflow/Underflow
5. Round to available bits
6. May need further normalization; go back to step 3

Floating-Point Addition

1.10101×2^3

$+$

1.0011×2^2

Floating-Point Multiplication

1. Add the two exponents
2. Multiply the mantissas and set the sign
3. Normalize the Result
4. Check for Overflow/Underflow
5. Round to available bits
6. May need further normalization; go back to step 3

Floating-Point Multiplication

1.101×2^3

$\times$

1.01×2^2