CSCI -112
Introduction to computer Systems

Instructor: Santanu Banerjee

Floating Point Operations

Topics

- Floating point formats
- 8ox86 floating point architecture
- Floating Point Instructions – Load & Store
- Floating Point Instructions – Arithmetic
- Floating Point Instructions – Other
- Floating point to/from ASCII
- Single instruction, multiple data instruction
- C/C++ with floating point assembly procedure

Floating Point Formats

Courtesy: UMBC and JBLearning

1011.0011 101100.11×2^{-2}

10.110011×2^{2}

Floating points in Binary system

- Format: 1011.0011
- ZEROs and ONEs only
- Dot ('.') is the binary point.
- The whole number is on the left side part and the fraction part is on the right side of the binary point.
- Like Decimal numbers:
  - The whole or fraction part can be zero
  - Can be represented in scientific notation : …. X 2^n
  - Binary point can be shifted to the left or right by increasing or decreasing the exponent(n)

  $1011.0011 = 1.0110011 \times 2^3 = 101100.11 \times 2^{-2}$

Common Format Components

- Each code is a *normalized* number whose "binary scientific notation" format would be

$$\pm 1.dd\ldots d \times 2^{exp}$$

- Sign bit
  - 0 for positive and 1 for negative
- Exponent field
  - Actual exponent *exp* plus a *bias*
  - Bias gives an alternative to 2's complement
- Fraction ("mantissa") field

$$\pm 1.dd\ldots d \times 2^{exp}$$

$$1011.0011$$

$$+ 1.0110011 \times 2^{3}$$

Convert fraction: Decimal to Binary

1. If fraction is 0, STOP. Otherwise, multiply the decimal fraction by 2. The whole number part of the result is appended to the binary fraction (right of the "binary" point).
2. Discard the whole number part of the previous result. If the result shows a repeating pattern, STOP. Otherwise jump to step 1.

Example: $5.625_{10} = 101.101_2$

5 → 101 (Whole part is simply converted to binary)

.625 → .101 (.625x2 = 1.25 x2 = 0.5 x2 = 1.0)

Decimal 5.625

whole number : 5 : 0101

0101.101

0.625 x 2 = 1.25

0.25 x 2 = 0.50

0.50 x 2 = 1. 0

5.625 ---> 0101.101

IEEE Single Precision Format

- 32-bit format
  - Sign bit
  - 8-bit biased exponent (the actual exponent in the normalized binary "scientific" format plus 127)
  - 23-bit fraction (the fraction in the scientific format without the leading 1 bit)
- Generated by `REAL4` directive

±	Biased Exponent	Fraction
1	8 bit	23 bit

101.101
1.01101 x2 ^{2}

byte
word
dword

1bit --- sign bit

8 bits --- exponential bias

23 bits --- fraction part

0

1000 0001

011010000
000

exponential bias = exponent + bias

127+ 2 =

129

IEEE Double Precision Format

- 64-bit format
  - Sign bit
  - 11-bit biased exponent (the actual exponent in the normalized binary scientific format plus 1023)
  - 52-bit fraction (the fraction in the scientific format without the leading 1 bit)
- Generated by `REAL8` directive

±	Biased Exponent	Fraction
1	11 bit	52 bit

Double Extended Precision Format

- 80-bit format
  - Sign bit
  - 15-bit biased exponent (the actual exponent in a normalized binary scientific format plus 16,383)
  - 64-bit fraction (the fraction in the scientific format *including* the leading 1 bit)
- Generated by `REAL10` directive

±	Biased Exponent	Fraction
1	15 bit	64 bit

Floating Point Formats

format	*total bits*	*exponent bits*	*fraction bits*	*approximate maximum*	*approximate minimum*	*approximate decimal precision*
single	32	8	23	3.40×10^{38}	1.18×10^{-38}	7 digits
double	64	11	52	1.79×10^{308}	2.23×10^{-308}	15 digits
extended double	80	15	64	1.19×10^{4932}	3.37×10^{-4932}	19 digits

- These are for *normalized* numbers
  - Binary scientific notation mantissa written starting with 1 and binary point
- Zero cannot be normalized
  - +0 represented by a pattern of all 0 bits
- Also formats for ± ∞ and NaN ("not a number")

Decimal to floating conversion

- Use a leading bit: 0 for +ve, 1 for –ve
- Disregard (but remember) sign and write the number in binary format.
- Reformat the number in format: 1.<frc bits> X 2^n
- SIGN: 0 or 1 (1 bit)
- EXP : (n + bias) in binary
- FRACTION: Fraction bits (right padded with 0s)
- Concatenate the above three.

Ex:Decimal to floating conversion

Convert -1313.3125 to IEEE 32-bit floating point format.

- The integral part:
  $1313_{10} = 10100100001_2$.
- The fractional part:

0.3125	× 2 = 0.625	(0)	Generate 0 and continue.
0.625	× 2 = 1.25	(1)	Generate 1 and continue with the rest.
0.25	× 2 = 0.5 0	(0)	Generate 0 and continue.
0.5	× 2 = 1.0 1	(0)	Generate 1 and nothing remains.

- So $1313.3125_{10} = 10100100001.0101_2$.
- Normalize: $10100100001.0101_2 = 1.01001000010101_2 \times 2^{10}$.
  - Mantissa is $01001000010101000000000_2$
  - Exponent is $10 + 127 = 137 = 10001001_2$
  - Sign bit is 1.
- So -1313.3125 is:
  - $11000100101001000010101000000000_2$
  - $c4a42a00_{16}$