Computer Architecture & Organization (CSE2003)

(Fall 2019-20)

Prof. Anand Motwani
Faculty, SCSE,
email-id: anand.motwani@vitbhopal.ac.in

Contents

- Fixed Point Representation of Numbers (Fixed Point Format) (10)
- Floating Point Representation (11)

- Numerical Exercise
- Quiz / Problems

Session Objectives

At the end of this session student will understand:

- to represent fixed and floating point numbers in the computers
- to perform arithmetic operations with them.

Fixed Point Representation of Numbers

- We learnt the fundamental concepts of how binary could be used to represent real numbers.
- When it comes to storing these numbers though there are two major approaches in modern computing. These are
- **Fixed Point Notation** and
- **Floating Point Notation**.

Fixed Point Representation

- **Fixed Point Notation** is a representation of our fractional number as it is stored in memory. In Fixed Point Notation, the number is stored as a signed integer in two's complement format.

Integer Representation

- Signed Magnitude Method / representation

+7	0 111	-7	1 111
+6	0 110	-6	1 110
+5	0 101	-5	1 101
+4	0 100	-4	1 100
+3	0 011	-3	1 011
+2	0 010	-2	1 010
+1	0 001	-1	1 001
+0	0 000	-0	1 000

Integer Representation

- Signed Magnitude Method / representation
- Drawback of this method:
  - There are two representation of 0.
- Formula of Range:
  - (2^(k-1) – 1) To (2^(k-1) – 1)

Integer Representation

- 1's Complement Method: Positive nos. Are represented in same way as in sign magnitude.
- Negative nos. are represented using 1's complement.

+7	0 111	-7	1 000
+6	0 110	-6	1 001
+5	0 101	-5	1 010
+4	0 100	-4	1 011
+3	0 011	-3	1 100
+2	0 010	-2	1 101
+1	0 001	-1	1 110
+0	0 000	-0	1 111

- 1's Complement Method:
- Drawback:
  - There are two representation of 0.
- Formula of Range:
  - $(2^{(k-1)} - 1)$ To $(2^{(k-1)} - 1)$

Integer Representation

- 2's Complement Method: Positive nos. Are represented in same way as in sign magnitude.
- Negative nos. are represented using 2's complement.

+7	0 111	-8	1 000
+6	0 110	-7	1 001
+5	0 101	-6	1 010
+4	0 100	-5	1 011
+3	0 011	-4	1 100
+2	0 010	-3	1 101
+1	0 001	-2	1 110
0	0 000	-1	1 111

- 2’s Complement Method:
- Advantage:
  - There is only representation of 0.
- Formula of Range:
  - (2^(k-1)) To (2^(k-1) – 1)

$$-(2^{(k-1)})\ to\ (2^{(k-1)} - 1)$$

Conclusion

- Fixed point is a simple yet very powerful way to represent fractional numbers in computer.
- By reusing all integer arithmetic circuits of a computer, fixed point arithmetic is orders of magnitude faster than floating point arithmetic.
- This is the reason why it is being used in many game and DSP applications.
- On the other hand, it lacks the range and precision that floating point number representation offers. You, as a programmer or circuit designer, must do the trade-off.

- Next Slides are added for developing more understanding.

Signed Integer Representation

- The conversions we have so far presented have involved only unsigned numbers.
- To represent signed integers, computer systems allocate the high-order bit to indicate the sign of a number.
  - The high-order bit is the leftmost bit. It is also called the most significant bit.
  - 0 is used to indicate a positive number; 1 indicates a negative number.
- The remaining bits contain the value of the number (but this can be interpreted different ways)

Signed Integer Representation (2)

- There are three ways in which signed binary integers may be expressed:
  - Signed magnitude
  - One's complement
  - Two's complement
- In an 8-bit word, *signed magnitude* representation places the absolute value of the number in the 7 bits to the right of the sign bit.

Two’s complement

- To express a value in two’s complement representation:
  - If the number is positive, just convert it to binary and you’re done.
  - If the number is negative, find the one’s complement of the number and then add 1.
- Example:
  - In 8-bit binary, 3 is: `00000011`
  - -3 using one’s complement representation is: `11111100`
  - Adding 1 gives us -3 in two’s complement form: `11111101`.

Two's Complement

- +3 = 00000011
- +2 = 00000010
- +1 = 00000001
- +0 = 00000000
- -1 = 11111111
- -2 = 11111110
- -3 = 11111101

How to: 2's Complement

- +2 = 0010
- reverse numbers

  1101
- Then add 1 (binary)

  1110
- So 1110 = -2

- Can positive 8, be represented in 4 bits?

No, because –8 is 1000

  Called an Overflow!

- -7 = 1001

  0110 (reversed)

  0111 (add 1)

+7 = 0111

Benefits

- One representation of zero
- Arithmetic works easily
- Negating is fairly easy
  - 3 = 00000011
  - Boolean complement gives 11111100
  - Add 1 to LSB 11111101

Solution to Assignment

Bit Pattern				Number Represented (n)	n / 2
1	1	1	1	-1	-0.5
1	1	1	0	-2	-1
1	1	0	1	-3	-1.5
1	1	0	0	-4	-2
1	0	1	1	-5	-2.5
1	0	1	0	-6	-3
1	0	0	1	-7	-3.5
1	0	0	0	-8	-4
0	1	1	1	7	3.5
0	1	1	0	6	3
0	1	0	1	5	2.5
0	1	0	0	4	2
0	0	1	1	3	1.5
0	0	1	0	2	1
0	0	0	1	1	0.5
0	0	0	0	0	0

Entry-Ticket: Quiz

- For 5 bits register, positive largest number that can be stored is _______ and negative lowest number that can be stored is ____________.

Floating Point (Real Numbers) Representation

- Two's complement representation deal with signed integer values only.
- Without modification, these formats are not useful in scientific or business applications that deal with real number values.
- Floating-point representation solves this problem.

Floating-Point Representation

- If we are clever programmers, we can perform floating-point calculations using any integer format.
- This is called *floating-point emulation*, because floating point values aren't stored as such; we just create programs that make it seem as if floating-point values are being used.
- Most of today's computers are equipped with specialized hardware that performs floating-point arithmetic with no special programming required.
  - Not embedded processors!

Floating-Point Representation

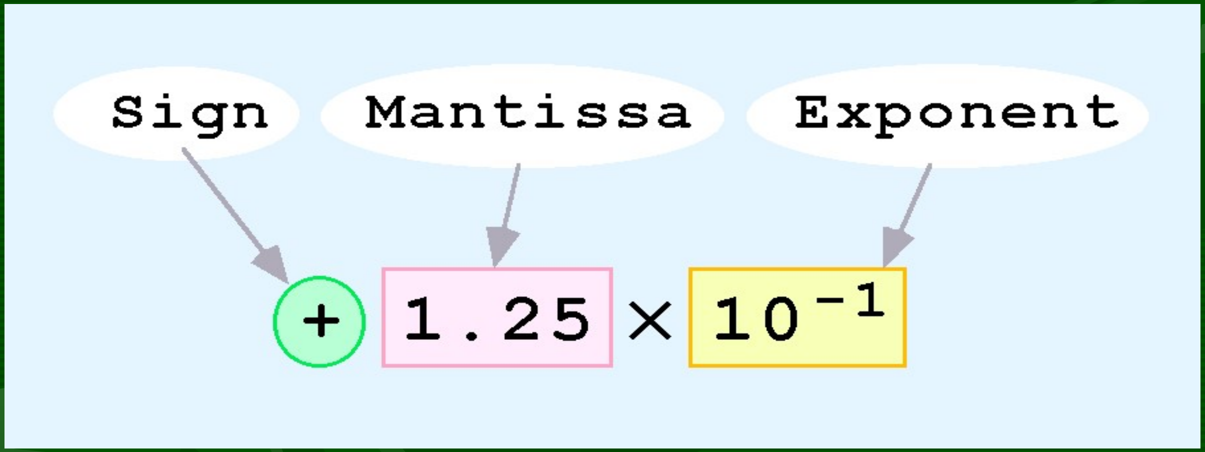
- Floating-point numbers allow an arbitrary number of decimal places to the right of the decimal point.
  - For example: $0.5 \times 0.25 = 0.125$
- They are often expressed in scientific notation.
  - For example:

    $0.125 = 1.25 \times 10^{-1}$

    $5{,}000{,}000 = 5.0 \times 10^{6}$

Floating-Point Representation

- Computers use a form of scientific notation for floating-point representation
- Numbers written in scientific notation have three components:



Floating Point Representation

	Exponent	Mantissa
Sign Bit	Decimal / Binary Point No.	Signed / Fixed Point No.
		May be fraction /integer

Floating point is represented in the form

$M * r^e$

Floating Point Representation

- Only the Mantissa and exponent are physical represented in Register (including sign)
- A floating point number is represented in similar manner except the base 2 for exponent.
- Sign bit 0 ~ **+** and 1 ~ **-**
- Mantissa stores fraction part.
- Exponent stored in bias form.

Floating Point Representation

- Exponent stored in bias form.
- The bias is calculated as:
- If k bits are used to represent exponent then:
  – Bias no. = $(2^{(k-1)} – 1)$
  – Range of exponent = $- (2^{(k-1)} – 1)$ to $2^{(k-1)}$

- Example: 7 bits are used for storing the exponent then bias = ? , Range =
- Note: here we always store exponent in positive.

- Biased number is also called excess no.
- Since exponent is stored in biased form so bias number is added to the actual exponent of the given no.
- Actual no. Can be calculated from the contents of the registers by using following formula:
- Actual Number = $(-1)^{s} (1+m) * 2^{(e\text{-}bias)}$
- s is sign bit, m is mantissa and e is exponent.

Numerical Problem

- Question: Represent +0.125 if 5 bits are used to represent exponent and 6 bits for mantissa.
- Solution steps:
- 1. Calculate bias
- 2. Calculate binary of given decimal no.
- 3. Normalize the binary no.
- 4. Calculate exponent.
- 5. Calculate Mantissa
- Represent the no. With as positive (i.e. Use 0 as sign bit)

Solution

- 1. Bias = 15
- 2. Binary no. = $(0.001)_2$
- 3. Normalized value = $1.0 * 2^{-3}$
- Calculate exponent: -3 + 15 = 12 ~ 1100
- Mantissa: No. to right of binary point is 0. So, mantissa is 0.
- No. is positive so sign bit is 0.
- Answer =

0	**0 1 1 0 0**	**0 0 0 0 0 0**

- Q. Represent 52.21875 in 32-bit binary floating point format.
- 52.21875 = 110100.00111 =
- $.11010000111 \times 2^{6}$.
- Normalized 23 bit mantissa = 0.11010000111000000000000.
- As excess representation is being used for exponent, it is equal to 127 + 6 = 133.
- Thus the representation is $52.21875 = 0.11010000111 \times 2^{133} = 0.110100000111 \times 2^{10000101}$.
- The 32-bit string used to store 52.21875 in a computer will thus be

0	10000101	11010000111000000000000

The solution as per IEEE 754 standard

- Q. Represent 52.21875 in 32-bit binary floating point format.
- 52.21875 = 110100.00111 =
- 1.1010000111×2^5 .
- Normalized 23 bit mantissa = .1010000111 0000000000000.
- As excess representation is being used for exponent, it is equal to 127 + 5 = 132.
- Thus the representation is $52.21875 = 0.1010000111 \times 2^{132} = 0.10100000111 \times 2^{10000100}$.
- The 32-bit string used to store 52.21875 in a computer will thus be
-

0	10000100	10100001110000000000000