BITS Pilani
Pilani Campus

Module 4 – Number System

Dr. Jagat Sesh Challa
Department of Computer Science & Information Systems

Module Overview

- Data in Computers
- Binary Numbering System
- Binary Arithmetic
- IEEE Floating Point Representation

innovate achieve lead

BITS Pilani
Pilani Campus

Data in Computers

Data in Computers

- Computers understand only two things – 0 and 1
  - Data is stored only the form of combinations of 0s and 1s
- Computers use switches to store 0 or 1
  - **Stores 1 if switch is ON**
  - **Stores 0 if switch is OFF**
- Computers consists of Integrated circuits (IC) of billions of switches; allow storage of huge amounts of information.

Data in Computers

- Kinds of data we store in computers:
  - Integer numbers
    - 1, 3, 94, -103, etc.
  - Floating point numbers
    - 1.00433, 54.9090354598, etc.
  - Characters
    - ‘A’, ‘a’, ‘#’, etc.
  - Strings
    - “Delhi”, “Gopal”, etc.
  - etc.

We will study more deeply

All of them are stored as binary patterns, i.e. strings of 0s and 1s.

innovate achieve lead

BITS Pilani
Pilani Campus

Binary Numbering System

The *human* numbering system

We use digits

0 to 9 -> 10 symbols (digits) (the decimal system)

What is so special about the number 10?

Nothing!

The numbering system for computers

- Computers use binary numbering system
  - i.e. it has only two symbols to represent numbers – 0 and 1
- Just like humans have 10 symbols - (0 to 9)
- Other systems
  - Octal – 8 symbols
    - 0, 1, 2, 3, 4, 5, 6, 7
  - Hexadecimal – 16 symbols
    - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Decimal vs Binary vs Octal vs Hexadecimal: An example

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

How a decimal number is interpreted?

Eg.: 357

Digits ->	3	5	7
Weights->	10^2	10^1	10^0
	MSD		LSD

Perform sum over Digits*Weights:

3*10^2 + 5*10^1 + 7*10^0

= 357

Range of binary numbers

One bit – up to decimal 1

Two bits – up to 3

Three bits – up to 7

In general:

n bits - $2^n - 1$ in decimal

Examples (Worked out on Board)

- Convert 10101 from binary to decimal
- Convert 16 from decimal to binary
- Convert 23 from decimal to binary

Negative Binary Numbers

- How do we signify negative numbers in arithmetic?
- The – symbol to the left of MSD
- So… 1111 is 15, then -1111 should -15
- But… computers cannot handle any symbol apart from 0 and 1

Negative Binary Numbers

- To handle negative numbers: left-most bit is used to indicate the sign.
- Known by various names: Most Significant Bit (**MSB**), sign-bit or high-order bit
- Positive numbers: **MSB is 0**
- Negative numbers: **MSB is 1**

- For 8 bit unsigned numbers: **range of 0 to 255 (2^8 – 1)**
- What about signed numbers?
  - **-127 to 127** or **-128 to 127**

- How does the computer know whether to treat a number as **signed or unsigned?**
  - **It cannot.** It's the programmer's job to tell.

Negative Binary Numbers

Three schemes for representing negative binary numbers:

- Signed-magnitude representation
- One's complement representation
- Two's complement representation

- We will discuss them with respect to 8-bit integers

Signed-magnitude representation

MSB is the sign as usual

7 bits for the magnitude or the value

Range: -127 to 127

Examples:

109	01101101
-109	11101101
127	01111111
-127	11111111

What about zero?

1's complement

As before, MSB indicates the sign.

Negative no. = **One's complement** of the positive number

One's complement → Invert all the bits → 1s to 0s and 0s to 1s

Range: -127 to 127

Examples:

15	00001111
-15	11110000
85	01010101
-85	10101010

What about zero?

2’s complement

2’s complement of a no. = it’s 1’s complement + 1

Range: -128 to 127

Example: **calculating the two’s complement representation of -15**

Decimal 15	00001111
In one’s complement	11110000
Adding 1	+1
In two’s complement	11110001

What about zero?

Most widely used!!

Binary Arithmetic

Binary arithmetic rules

- Addition of binary bits
  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - 1 + 1 = 0 1 is carry. In binary addition carry is discarded
- Subtraction of binary bits:
  - 0 – 0 = 0
  - 1 – 0 = 1
  - 1 – 1 = 0
  - 0 – 1 = 1 Borrow 1 from next high order digit

Note: for a positive number, its binary representation in SMF, 1CT or 2CT – it is the same number itself

Subtraction using 2 CT

- Subtraction using 2CT:
  1. Write the 2CT of the subtrahend
  2. Add it to the minuend
  3. If there is a carry over discard it, the remaining bits gives the result
  4. If there is no carry over, find the 2CT of the result and add –ve sign before it

Subtraction using 2 CT: Example 1

Consider 7 – 4:

1. 2CT (0 1 0 0) = 1 0 1 1 + 0 0 0 1 → 1 1 0 0
2. 0 1 1 1

 + 1 1 0 0

 1 0 0 1 1 → +3

Discard

Substraction using 2CT: Example 2

Consider 4 – 7

1. 2CT(7) = 1 0 0 0 + 0 0 0 1 = 1 0 0 1
2. 0 1 0 0

   + 1 0 0 1

   ---

   1 1 0 1
3. 2CT(1 1 0 1) = 0 0 1 0 + 0 0 0 1 → 0 0 1 1
4. Final answer is -ve of the result obtained i.e., - 3

Addition in 2's Complement

- Overflow
  - Only possible cases:

```
      0 1 1 1 (+7)            1 1 1 1 (-1)
      0 1 0 1 (+5)            0 1 1 1 (+7)
   -------------------     -------------------
   0  1 1 0 0 (- 4)  x     1  0 1 1 0 (+6)
Overflow                Carry

      1 0 1 0 (-6)            1 1 1 0 (-2)
      1 0 0 1 (-7)            1 1 0 1 (-3)
   -------------------     -------------------
   1  0 0 1 1 (+3)  x      1  1 0 1 1 (-5)
Overflow                Carry
```

Addition in 2's Complement

- Observations:
  - When addition is performed on two numbers having same MSB
  - When $C_3 \neq C_4$

$C_{4=1}$	$C_{3=0}$	$C_{2=0}$	$C_{1=0}$	$C_{0=0}$	
	1	0	1	0	(-6)
	1	0	0	1	(-7)
1	0	0	1	1	(+3)

$C_{4=1}$	$C_{3=1}$	$C_{2=0}$	$C_{1=0}$	$C_{0=0}$	
	1	1	1	0	(-2)
	1	1	0	1	(-3)
1	1	0	1	1	(-5)

- *Whenever an overflow occurs MSB gets modified*
- *Addition of two N bits integers B_1 and B_2 causes overflow only if*
  - *$B1 + B2 > +2^{N-1} - 1$ and $B1 + B2 < -2^{N-1}$ addition is*

➢ **Overflows occur when the sum does not fit in the given number of bits**

innovate achieve lead

BITS Pilani
Pilani Campus

IEEE Floating Point Representation

Converting real numbers to binary

- Real numbers: X.Y
  - X – Similar to the way integers are represented (Explained in previous slides)
  - Y
    - Y * 2 = X'.Y'
    - Store X' and Y = Y'
    - Repeat step 1 and 2, for the new fractional value obtained until X'.Y' = 1.0
    - Print all the value of X'.

Example - Convert 10.6875 to binary

$(10)_{10} = (1010)_2$

$(0.6875)_{10} =$

0.6875 * 2 = 1.375 --------→ 1

0.375 * 2 = 0.75 --------→ 0

0.75 * 2 = 1.5 --------→ 1

0.5 * 2 = 1.0 -------→1

$= (1\,0\,1\,0.\,1\,0\,1\,1)_2$

Converting binary real numbers to decimal

- Binary number: X.Y
  - X – Similar to the way integer values are obtained (Explained in previous slides)
  - Y
    - Multiply each bit with the weight of its position, where weight of positions are: 2^{-1}, 2^{-2}, 2^{-3},.....
    - Take the sum of the output
    - E.g. 1 0 1 0 . 1 0 1 1

Integral part Fractional part

Exercise: Convert this binary floating-point number to decimal system

Storing Real number in Computers

Steps to convert a real number in decimal to binary:

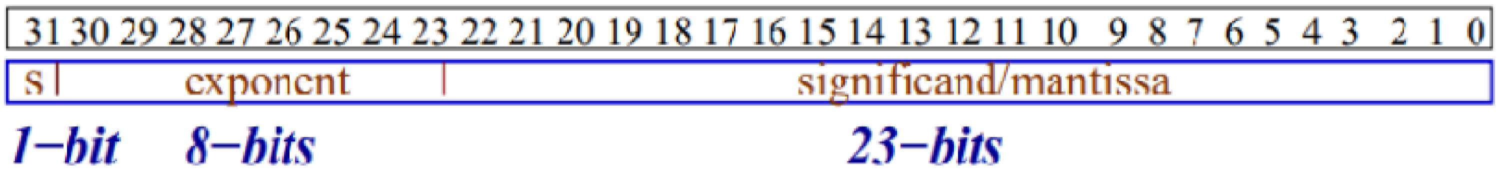
1. First, we convert the real number in decimal to a binary real number.
   - E.g. **10.6875** is converted into **1010.1011**
2. Then we normalize the binary real number.
   - E.g. **1010.1011** can be written as **1.0101011×2^3**
3. Encode the normalized binary real number into **IEEE 754 32-bit or 64-bit floating point format**

Let us study the IEEE 754 Floating Point Representation

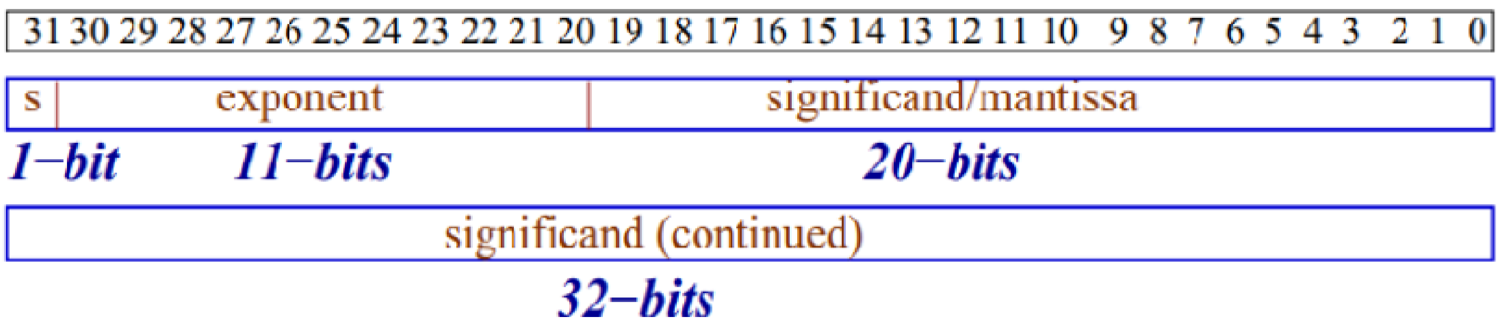
IEEE 754 floating point representation

- Three pieces of information are required:
  - Sign of the number (s)
  - Significant value (m)
  - Signed exponent of 10 (e)
- The data type float uses IEEE 32-bit single precision format and the data type double uses IEEE 64-bit double precision format

IEEE 754 floating point representation (contd.)



Single precision (32 bit)



Double precision (64 bit)

IEEE floating point representation

- In single precision (32 bits):
  - The MSB indicates the sign bit (s): 1 is -ve and 0 is +ve.
  - The next 8 bits: Stores the value of the signed exponent (e)
  - Remaining 23: Stores the significand also know as *mantissa* (m).
  - Value of a floating number IN 32 bits is

$$(-1)^s \times 1.m \times 2^{e-127}$$

  - e – 127: Excess 127 representation, meaning if the exponent value is 5, it will be stored as 5 + 127 = 132 in the exponent field

Storing 10.6875 int IEEE 754 32-bit Floating Point Format

10.6875 → 1010.1011 → 1.0101011×2^3

binary Normalised binary

↓

$s = 0$

$e = (3+127)_2=(130)_2 = 10000010$

$m = 01010110000000000000000$

which is

0 10000010 01010110000000000000000

IEEE 754 32-bit Floating Point representation

Conversion to decimal real number (Worked out on board)

Convert the following number to decimal:

1 00000111 11000000000000000000000

It is

$-1.75 \times 2^{(7-127)} = -1.316554 \times 10^{-36}$

IEEE floating point representation

Why excess 127?

- Helps in natural ordering in the values stored in the exponent field
- Consider 2^{-126} and 2^{+126}
  - Case 1: In absence of Excess 127:
    - -126 = 10000010
    - +126 = 01111110
      - While -126 is a smaller value the exponent field contains larger value (10000010 > 01111110)
  - Case 2: In presence of Excess 127:
    - -126 is written as 1 (1 – 127 = 6) → 00000001
    - +126 is written as 253 (253 – 127 = 126) → 11111101
    - -126 is a smaller value and so is the value stored in the exponent (00000001 < 11111101)

Example 1

Convert 12.375 into 32-bit IEEE 754 Floating Point Format

1. $(12)_{10} + (0.375)_{10} = (1100)_2 + (0.011)_2 = (1100.011)_2$
2. Shift $(1100.011)_2$ by 3 digits to normalize the value

$$(12.375)_{10} = (1.100011)_2 * 2^3$$

s = 0

Exponent = 130 (represented in excess 127)

Mantissa = 100011

FP Representation → 0| 10000010 | 1000110000……

Example 2

Convert the following binary number in 32-bit IEEE 754 floating point format into decimal:

1 1011 0110 011 0000 0000 0000 0000 0000

$(-1)^{1} \times 2^{10110110 - 01111111} \times 1.011$

$= -\ 1.375 \times 2^{55}$

$= -49539595901075456.0$

$= -\ 4.9539595901075456 \times 10^{16}$

Example 3 (Work out on board)

Convert -10.7 into 32-bit IEEE 754 Floating Point Format

You will see a recurring pattern of bits that is never-ending.

What should you do?

Store only the bits that can fit your 23-bits mantissa. Rest are truncated.

This leads to approximation.

innovate achieve lead

BITS Pilani
Pilani Campus

Thank you

Q & A