COMPUTER ORGANIZATION AND ARCHITECTURE

Course Code : CSE 2151

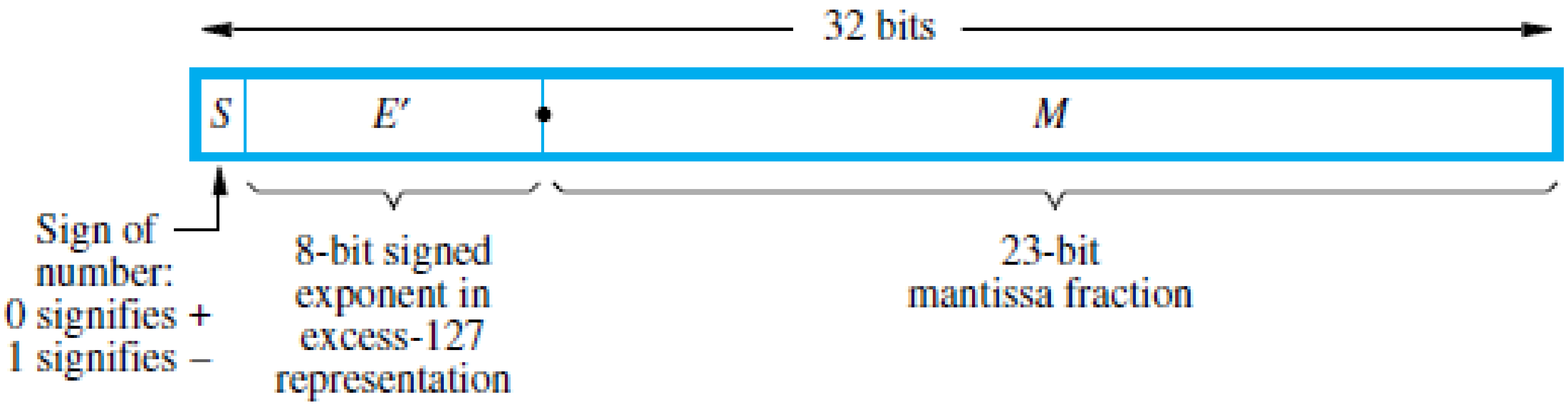
Credits : 04

FLOATING-POINT NUMBERS

- A binary floating-point number is represented by (2008 version of IEEE Standard 754):
  - a sign for the number
  - some significant bits
  - a signed scale factor exponent for an implied base of 2

IEEE STANDARD FLOATING-POINT FORMATS (32 BIT)

- The basic IEEE format is a 32-bit representation that comprises of
  - a sign bit,
  - 23 significant bits, and
  - 8 bits for a signed exponent of the scale factor



$$\text{Value represented} = \pm 1.M \times 2^{E'-127}$$

IEEE STANDARD FLOATING-POINT FORMATS (32 BIT)

- Example:
  - Sign bit: 0, hence +ve number
  - Mantissa, M: 1.00101000000000000000000
  - Exponential, E: E'-127, ➔(E'→00101000_2= 40_{10})

    E: 40-127= -87,

    E: 2^{-87}

0	0 0 1 0 1 0 0 0	0 0 1 0 1 0 ... 0

Value represented = $1.001010\ldots0 \times 2^{-87}$

- Actual binary number:
  - 0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    0000100101
- Corresponding decimal value:
  - 0.00000000000000000000000000747209049425342382085989297035855116746461135335266590118408203125
- In the value represented, what about the digit found to the left side of the decimal point?
  - always be equal to 1
  - can be left out in IEEE floating-point representation

STEPS TO CONVERT 32-BIT REPRESENTATION TO DECIMAL

1. Obtain the mantissa and rewrite the value as V=1.M
2. To obtain E
   i. Convert E' to its equivalent decimal value
   ii. E= E'-127 ➔ 2^E
3. Move point in V towards left or right based on E
   i. Move right if +ve
   ii. Move left if –ve
   iii. Digits before point is the integral part and after is the fractional part.
4. Convert the integral part of binary to decimal equivalent
   i. Multiply each digit separately from left side of point till the first digit by 2^0, 2^1, 2^2, … respectively.
   ii. Find the sum of all the products obtained in step 1.i.
5. Convert the fractional part of binary to decimal equivalent
   i. Divide each digit separately from right side of point till the end by 2^1, 2^2, 2^3, … respectively.
   ii. Find the sum of all the products obtained in step 2.i.
6. Add both integral and fractional part to obtain decimal number.

32-BIT REPRESENTATION TO DECIMAL: EXAMPLE

- IEEE754 32-bit format: 0 10000001 01000110011001100110011

  S E' M

- S=0, hence +ve number
- E'= 10000001_2=129_{10}
  - E = E'-127 =129-127

    E = 2
- Value represented = 1. 01000110011001100110011 × 2^2

  = 101.00011001100110011001 1

  = 5 . 1

DECIMAL TO 32-BIT REPRESENTATION

- 40.15625
  - S=0, since it is a positive number
  - Mantissa, M,
    - 40 ➔ 101000_2 0.15625 ➔ $.00101_2$
    - $=101000.00101_2$ If point moved towards left, then +ve exponent else -ve exponent
    - $=1.$**0100000101** $\times 2^5$ Ignore 1 before decimal point
    - Therefore, M=**0100000101**0000000000000 Fill the remaining positions in the right with zeroes
  - Exponent, E', in excess-127 representation:
    - E'=E+127
    - =5+127
    - =132
    - $E'=10000100_2$
  - 32-bit representation:
    - 0 10000100 01000001010000000000000

STEPS TO CONVERT DECIMAL TO 32-BIT REPRESENTATION

A. Sign bit, S:
   i. If positive, the first bit will be a 0
   ii. If negative, the first bit will be a 1.

B. Mantissa, M:
   i. Divide the integral part by 2 to get its binary equivalent, I (remainder in bottom-up)
   ii. Multiply the fractional part by 2 to get its binary equivalent, F (value before point in top-down)
   iii. Represent it as I.F
   iv. Adjust the point to obtain 1.M
   v. M must be 23-bit long. Fill the remaining bits in vector with zeroes

STEPS TO CONVERT DECIMAL TO 32-BIT REPRESENTATION

C. Exponent, E', in "Excess 127 form":

i. Count the number of places the binary point needs to be moved until a single digit of 1 sits by itself on the left side of the binary point.

   a. Point moved towards left count is positive else it is negative.

ii. Add 127 to your result above.

   a. 8-bit binary numbers can range from 0 to 255,

   b. exponents in single precision format can range from -126 to +127, that is from 2^{-126} to 2^{127} or,

   c. approximately, 10^{-38} to 10^{38} in size.

   d. In "excess 127 form" negative exponents range from 0 to 126, and positive exponents range from 128 to 255.

   e. 127 represents a power of zero.

iii. Translate the sum (which will always be a positive value after adding 127) into binary form.

iv. This should be represented using 8 bits, so zeros may be added to the left side to ensure a string length of 8 bits.

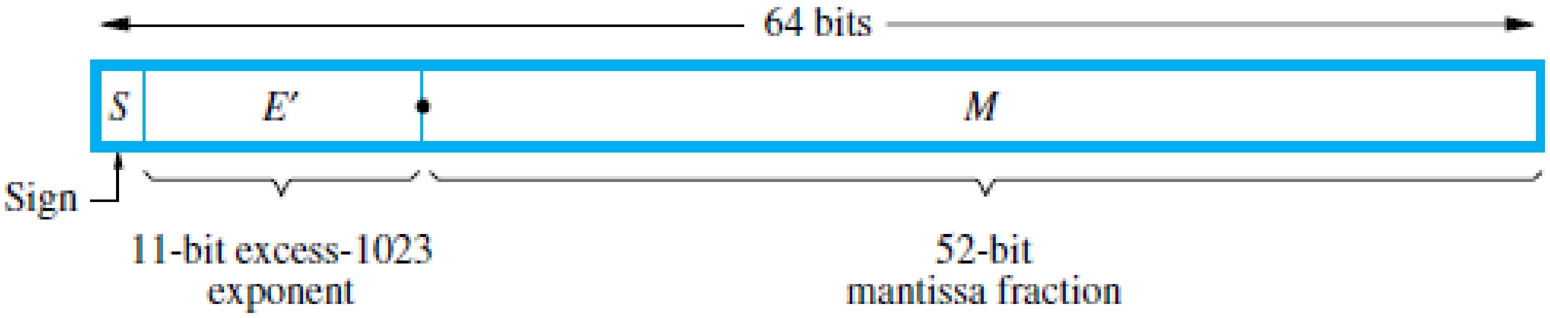
v. This 8-bit string represents the exponent.

DECIMAL TO 32-BIT REPRESENTATION

- Represent -0.09375 in IEEE754 format
  - **S=1**, since it is a negative number
  - **Mantissa, M:** 0.09375 to binary
    - 0.09375 x 2 = 0.1875 0 (remember, read downwards)
    - 0.1875 x 2 = 0.375 0
    - 0.375 x 2 = 0.75 0
    - 0.75 x 2 = 1.50 1
    - 0.50 x 2 = 1.00 1
   - $= 0.00011_2$
   - $=1.\mathbf{1} \times 2^{-4}$ If point moved towards left, then +ve exponent else -ve exponent. Ignore 1 before point
   - Therefore, M=10000000000000000000000 Fill the remaining positions in the right with zeroes
  - **Exponent, E', in** excess-127 representation:
   - E'=E+127
   - =-4+127
   - =123
   - $E'=1111011_2$= 01111011
  - -0.09375 will be represented in IEEE754 format as
   - 1 01111011 10000000000000000000000

IEEE STANDARD FLOATING-POINT FORMATS (64 BIT)

- IEEE standard also defines a 64-bit representation to accommodate
  - more significant bits, and
  - more bits for the signed exponent, resulting in much higher precision and a much larger range of values
- The 11-bit excess-1023 exponent E' has the
  - range $1 \leq E' \leq 2046$ for normal values, with 0 and 2047 used to indicate special values,
  - The actual exponent E is in the range $-1022 \leq E \leq 1023$
  - providing scale factors of 2^{-1022} to 2^{1023} (approximately $10^{\pm 308}$)



$$\text{Value represented} = \pm 1.M \times 2^{E'-1023}$$

IEEE754 FORMAT: FLOATING-POINT NUMBER

- The full 24-bit string, B, of significant bits, called the mantissa, always has a leading 1, with the binary point immediately to its right. Therefore, the mantissa
  - B = 1.M

    $= 1.b_{-1}b_{-2}\ldots b_{-23}$ has the value

  $V(B) = 1 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \cdot \quad \cdot \quad \cdot + b_{-23} \times 2^{-23}$

- By convention, when the binary point is placed to the right of the first significant bit, the number is said to be normalized.
- Note that the base 2, of the scale factor and the leading 1 of the mantissa are both fixed. They do not need to appear explicitly in the representation.

IEEE754 FORMAT: FLOATING-POINT NUMBER

$$0 \;|\; \underbrace{1\,0\,0\,0\,1\,0\,0\,0}_{\text{excess-127 exponent}} \;\bullet\; 0\,0\,1\,0\,1\,1\,0\,\ldots$$

(There is no implicit 1 to the left of the binary point.)

$$\text{Value represented} = +0.0010110\ldots \times 2^{9}$$

(a) Unnormalized value

$$0 \;|\; 1\,0\,0\,0\,0\,1\,0\,1 \;\bullet\; 0\,1\,1\,0\,\ldots$$

$$\text{Value represented} = +1.0110\ldots \times 2^{6}$$

(b) Normalized version

SPECIAL VALUES: 32-BIT REPRESENTATION

Sl.No.	E'	M	Meaning
1.	=0	= 0	value 0 is represented
2.	=255	= 0	value ∞ is represented
3.	=0	$\neq 0$	denormal numbers are represented. Their value is $\pm 0.M \times 2^{-126}$. There is no implied one to the left of the binary point, and M is any nonzero 23-bit fraction
4.	=255	$\neq 0$	the value represented is called Not a Number (NaN). A NaN represents the result of performing an invalid operation such as 0/0 or $\sqrt{-1}$

EXCEPTIONS

- In conforming to the IEEE Standard, a processor must set exception flags if any of the following conditions arise when performing operations
  - underflow,
  - overflow,
  - divide by zero,
  - Inexact: name for a result that requires rounding in order to be represented in one of the normal formats
  - Invalid: exception occurs if operations such as 0/0 or $\sqrt{-1}$ are attempted
- When an exception occurs, the result is set to one of the special values.

ADDITION EXAMPLE- BINARY

- A=96.625 + B=12.125
- **Step i:** Convert A to binary representation
  - 1100000.101 ➔After normalizing we get 1.100000101 X 2^6
  - E'= 6+127=133 =10000101
  - In IEEE 32-bit format: A= 01000010110000010100000…….
- **Step ii:** Convert B to binary representation
  - 1100.001 ➔After normalizing we get 1.100001 X 2^3
  - E'= 3+127=130 = 10000010
  - In IEEE 32-bit format: B= 01000001010000100000……
- **Step 1:** Choose the number with the smaller exponent and shift its mantissa right a number of steps equal to the difference in exponents (Shift point to left).
  - Shift the mantissa of smaller number, B, to the right by 3 bits we get
    - 1.10000100000 ---Original mantissa (with hidden bit considered)
    - 0.11000010000 ---shifting by 1 bit
    - 0.011000010000 ---shifting by 2 bits
    - 0.001100001000 ---shifting by 3 bits

ADDITION EXAMPLE- BINARY

- **Step 2:**
  - Set the exponent of the result equal to the larger exponent
    - 10000101 (exponent of A)
- **Step 3:**
  - Perform addition on the mantissas and determine the sign of the result

$$
\begin{array}{r}
1.100000101000000\ + \\
\underline{0.001100001000000} \\
1.101100110000000
\end{array}
$$

- **Step 4:**
  - Normalize the resulting value, if necessary
    - Result is already normalized
- In 32-bit format: 0 10000101 101100110000
- which is 108.75 in decimal

ADD/SUBTRACT RULE

1. Choose the number with the smaller exponent
2. Shift its mantissa right to the number of steps equal to the difference in exponents. (Shift point to left)
3. Set the exponent of the result equal to the larger exponent.
4. Perform addition/subtraction on the mantissas and determine the sign of the result.
5. Normalize the resulting value, if necessary.

ADDITION EXAMPLE- BINARY

- 0.25 = 0 01111101 00000000000000000000000 + 100 = 0 10000101 10010000000000000000000
- **Step 1:** align radix points
  - shifting the mantissa LEFT by 1 bit DECREASES THE EXPONENT by 1 and RIGHT by 1 INCREASES THE EXPONENT by 1
  - we want to shift the mantissa right, because the bits that fall off the end should come from the least significant end of the mantissa
    - choose to shift the .25, since we want to increase it's exponent.
    - shift by

```
  10000101
 -01111101
 ---------
  00001000   (8) places.
```

00000000000000000000000 (original value)

10000000000000000000000 (shifted 1 place) (note that hidden bit is shifted into msb of mantissa)

01000000000000000000000 (shifted 2 places)

00100000000000000000000 (shifted 3 places)

0010000000000000000000 (shifted 4 places)

00001000000000000000000 (shifted 5 places)

00000100000000000000000 (shifted 6 places)

00000010000000000000000 (shifted 7 places)

00000001000000000000000 (shifted 8 places)

ADDITION EXAMPLE- BINARY

- Step 2: add (don't forget the hidden bit for the 100)

  1.10010000000000000000000 (100) +

  <u>0.00000001000000000000000 (.25)</u>

  1.10010001000000000000000

- Step 3: normalize the result (get the "hidden bit" to be a 1)
  - Result is already normalized

- In 32-bit format: 0 10000101 10010001000000000000000

SUBTRACTION EXAMPLE- BINARY

- A=96.625 - B=12.125
- Convert A to binary representation
  - 1100000.101 ➔After normalizing we get 1.100000101 X 2^6
  - E'= 6+127=133 =10000101
  - In IEEE 32-bit format: 01000010110000010100000…….
- Convert B to binary representation
  - 1100.001 ➔After normalizing we get 1.100001 X 2^3
  - E'= 3+127=130 = 10000010
  - In IEEE 32-bit format: 01000001010000100000……
- **Step 1:**
- Choose the number with the smaller exponent and shift its mantissa right a number of steps equal to the difference in exponents.
  - A=0 10000101 10000010100000……. B=0 10000010 10000100000……
- Shift the mantissa of smaller number, B, to the right by 3 bits we get
  - 1.10000100000 ---Original mantissa (with hidden bit considered)
  - 0.11000010000 ---shifting by 1 bit
  - 0.011000010000 ---shifting by 2 bits
  - 0.001100001000 ---shifting by 3 bits

SUBTRACTION EXAMPLE- BINARY

- **Step 2:**
  - Set the exponent of the result equal to the larger exponent
    - 10000101 (exponent of A)
- **Step 3:**
  - Perform subtraction on the mantissas and determine the sign of the result

    1.10000010100000 -
    <u>0.00110000100000</u>
    1.01010010000000
- **Step 4:**
  - Normalize the resulting value, if necessary
    - Result is already normalized
- In 32-bit format: 0 10000101 01010010000000
- which is 84.5 in decimal

SUBTRACTION EXAMPLE- BINARY

- 0.25 = 0 01111101 00000000000000000000000 + 100 = 0 10000101 10010000000000000000000
- **Step 1:** align radix points
  - shifting the mantissa LEFT by 1 bit DECREASES THE EXPONENT by 1 and RIGHT by 1 INCREASES THE EXPONENT by 1
  - we want to shift the mantissa right, because the bits that fall off the end should come from the least significant end of the mantissa
    - choose to shift the .25, since we want to increase it's exponent.
    - shift by

```
 10000101
-01111101
 00001000   (8) places.
```

00000000000000000000000 (original value)

10000000000000000000000 (shifted 1 place) (note that hidden bit is shifted into msb of mantissa)

01000000000000000000000 (shifted 2 places)

00100000000000000000000 (shifted 3 places)

0010000000000000000000 (shifted 4 places)

00001000000000000000000 (shifted 5 places)

00000100000000000000000 (shifted 6 places)

00000010000000000000000 (shifted 7 places)

00000001000000000000000 (shifted 8 places)

SUBTRACTION EXAMPLE- BINARY

- Step 2: add (don't forget the hidden bit for the 100)

  1.10010000000000000000000 (100) -

  <u>0.00000001000000000000000 (.25)</u>

  1.10001111000000000000000

- Step 3: normalize the result (get the "hidden bit" to be a 1)
  - Result is already normalized

- In 32-bit format: 0 10000101 10001111000000000000000

  133 1.10001111000000000000000

  133-127=6

  $=1.10001111000000000000000 \times 2^6$

  =1100011.1100000000000000000

  = 99.75

MULTIPLY AND DIVIDE RULE

- Multiply Rule
  A. Add the exponents and subtract 127 to maintain the excess-127 representation.
  B. Multiply the mantissas and determine the sign of the result.
  C. Normalize the resulting value, if necessary.

- Divide Rule
  A. Subtract the exponents and add 127 to maintain the excess-127 representation.
  B. Divide the mantissas and determine the sign of the result.
  C. Normalize the resulting value, if necessary.

TOPICS COVERED FROM

- Textbook 1:
  - Chapter 1: 1.4.2,
  - Chapter 9: 9.7, 9.7.1