# COMPUTER ORGANIZATION AND ARCHITECTURE

Course Code : CSE 2151

Credits : 04

# ADDITION EXAMPLE- BINARY

- A=15.5 + B=15.5
- **Step i:** Convert A to binary representation
  - 1111.1 ➔After normalizing we get $1.1111 \times 2^3$
  - E'= 3+127=130 =10000010
  - In IEEE 32-bit format: A= 01000001111100000000000000000000
- **Step ii:** Convert B to binary representation
  - 1111.1 ➔After normalizing we get $1.1111 \times 2^3$
  - E'= 3+127=130 =10000010
  - In IEEE 32-bit format: B= 01000001111100000000000000000000
- **Step 1:** Choose the number with the smaller exponent and shift its mantissa right a number of steps equal to the difference in exponents.
  - Both are of equal exponent. Hence no shift
- **Step 2:**
  - Set the exponent of the result equal to the larger exponent
    - 10000010 (exponent of A or B)

# ADDITION EXAMPLE- BINARY

- **Step 3:**
  - Perform addition on the mantissas and determine the sign of the result

    1.11110000000000000000000 +
    1.11110000000000000000000
    11.11100000000000000000000

- **Step 4:**
  - Normalize the resulting value, if necessary
    - $1.11110000000000000000000 \times 2^1$
  - Adjust resultant exponent E' by adding exponent from the normalized resultant mantissa

    10000010 + (if the exponent is –ve, subtract it from resultant E')
    00000001
    10000011

- In 32-bit format: 0 10000011 11110000000000000000000
- which is 31.0 in decimal

# 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.

# MULTIPLICATION EXAMPLE- BINARY

- A= 0 10000100 0100 × B= 1 00111100 1100

- **Step 1:** add exponents and subtract 127
  - 132+60-127=65. and unsigned representation for 65 is 01000001.
- **Step 2:** Multiply the mantissa. Don't forget hidden bit

```
          1.0100 x 1.1100
           00000
          00000
         10100
        10100
       10100
      ----------
      1000110000        becomes   10.00110000
```

**normalize the result:**

$1.000110000 \times 2^{1}$

**Step 3:** Adjust the exponent : 65+1=66 = 01000010

1 01000010 000110000

# MULTIPLICATION EXAMPLE- BINARY

- A=96.625 × B=12.125
- **Step i:** Convert A to binary representation
  - 1100000.101 ➔After normalizing 1.100000101 X $2^6$
  - E'= 6+127=133 =10000101
  - In IEEE 32-bit format: A= 0 10000101 10000010100000…….
- **Step ii:** Convert B to binary representation
  - 1100.001 ➔After normalizing 1.100001 X $2^3$
  - E'= 3+127=130 = 10000010
  - In IEEE 32-bit format: B= 0 10000010 10000100000……
- **Step 1:** add exponents and subtract 127
  - 133+130-127=136 ➔ 10001000 (unsigned representation)
- **Step 2:** Multiply the mantissa

1.100000101 x 1.100001= 10.01001001100101

After normalizing 1.001001001100101 X $2^1$

1.100000101 x 1.100001

1100000101

0000000000

0000000000

0000000000

0000000000

1100000101

1100000101

1001001001100101

➔10.01001001100101

- **Step 3:** Adjust the exponent:
  - 136+1=137➔ 10001001
- 32-bit representation:
  - 0 10001001 00100100110010 1
  - = 1.00100100110010 1 X $2^{10}$
  - =10010010011.100101
  - =1024+128+16+2+1+ 0.5+.0625+.015625
  - =1171.578125

# DIVISION EXAMPLE- BINARY

- A= 127.03125 ÷ B= 16.9375
- **Step i:** Convert A to binary representation
  - 1111111.00001 ➔After normalizing 1.11111100001 X $2^6$
  - E'= 6+127=133 =10000101
  - In IEEE 32-bit format: A= 0 10000101 11111100001……
- **Step ii:** Convert B to binary representation
  - 10000.1111 ➔After normalizing 1.00001111 X $2^4$
  - E'= 4+127=131 = 10000011
  - In IEEE 32-bit format: B= 0 10000011 00001111000……
- **Step 1:** subtract exponents and add 127
  - 133-131+127=129 ➔ 10000001 (unsigned representation)
- **Step 2:** Divide the mantissa

1.11111100001 ÷ 1.00001111= 1.111

The result is already normalized.

- The result in IEEE 32-bit format:

0 10000001 11100000000…….

```
                1.1 1 1
            ____________________
1.00001111 | 1.1 1 1 1 1 1 0 0 0 0 1
             1.0 0 0 0 1 1 1 1
             ------------------
             0 1 1 1 0 1 1 0 1 0
               1 0 0 0 0 1 1 1 1
               -----------------
               0 1 1 0 0 1 0 1 1 0
                 1 0 0 0 0 1 1 1 1
                 -----------------
                 0 1 0 0 0 0 1 1 1 1
                   1 0 0 0 0 1 1 1 1
                   -----------------
                   0 0 0 0 0 0 0 0 0
```

# DIVISION EXAMPLE- BINARY

- A= 97.0 ÷ B= 12.125
- **Step i:** Convert A to binary representation
  - 1100001.0 ➔After normalizing 1.100001 X $2^6$
  - E'= 6+127=133 =10000101
  - In IEEE 32-bit format: A= 0 10000101 10000100……
- **Step ii:** Convert B to binary representation
  - 1100.001 ➔After normalizing 1.100001 X $2^3$
  - E'= 3+127=130 = 10000010
  - In IEEE 32-bit format: B= 0 10000010 10000100000……
- **Step 1:** subtract exponents and add 127
  - 133-130+127=130 ➔ 10000010 (unsigned representation)
- **Step 2:** Divide the mantissa: 1.100001 ÷ 1.100001= 1.0

The result is already normalized.

- The result in IEEE 32-bit format:
  0 10000010 00000000000…….

i.e.,1.0 X$2^3$ = 1000 =8 in decimal

```
                1.0
          ------------
1.100001 | 1.100001
           1.100001
          ---------
           0000000
```

# GUARD BITS AND TRUNCATION

- Mantissas of initial operands and final results are limited to 24 bits, including the implicit leading 1
- To attain maximum accuracy in the final results it is important to retain extra bits, often called guard bits, during the intermediate steps.
- Removing guard bits in generating a final result requires that the extended mantissa be truncated to create a 24-bit number that approximates the longer version.

# WAYS OF TRUNCATION: CHOPPING

- Remove the guard bits and make no changes in the retained bits
  - e.g. To truncate from 6 bits to 3 bits:
    - $0.b_{-1}b_{-2}b_{-3}000$ to $0.b_{-1}b_{-2}b_{-3}111$ are truncated to $0.b_{-1}b_{-2}b_{-3}$
- The error in the 3-bit result ranges from 0 to 0.000111
- The error in chopping, ranges from 0 to almost 1 in the least significant position of the retained bits.
- In the example above, it is $b_{-3}$ position. The result of chopping is a *biased approximation* because the error range is not symmetrical about 0

# WAYS OF TRUNCATION: VON NEUMANN ROUNDING

- If the bits to be removed are all 0s,
  - they are simply dropped, with no changes to the retained bits.
- If any of the bits to be removed are 1,
  - the least significant bit of the retained bits is set to 1
- All 6-bit fractions where $b_{-4}b_{-5}b_{-6} \neq 000$ are truncated to $0.b_{-1}b_{-2}1$
- The error in this truncation method ranges between −1 and +1 in the LSB position of the retained bits
- approximation is unbiased because the error range is symmetrical about 0.
- When three guard bits are used, the value 0.001100 is truncated to 0.001

# WAYS OF TRUNCATION: VON NEUMANN ROUNDING

- 0.001 00000 → 0.001 (truncate. 0 error)
- 0.001 11111 → 0.001 (but the value is near 0.010, hence -1 error at LSB position of retaining bits)
- 0.010 11111 → 0.011 (almost nearest value. Almost 0 error)
- 0.010 00001 → 0.011 (+1 error)

# WAYS OF TRUNCATION: ROUNDING

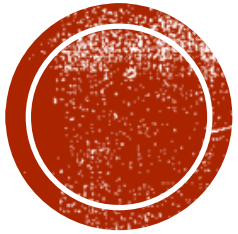
- Achieves the closest approximation to the number being truncated and is an unbiased technique.
- A 1 is added to the LSB position of the bits to be retained if there is a 1 in the MSB position of the bits being removed. Thus,
  - $0.b_{-1}b_{-2}b_{-3}1\ldots$ is rounded to $0.b_{-1}b_{-2}b_{-3} + 0.001$
  - $0.b_{-1}b_{-2}b_{-3}0\ldots$ is rounded to $0.b_{-1}b_{-2}b_{-3}$.
- Except for the case in which the bits to be removed are $10\ldots0$.
  - This is a tie situation; the longer value is halfway between the two closest truncated representations.
- To break the tie
  - choose the retained bits to be the nearest even number
  - the value $0.b_{-1}b_{-2}0100$ is truncated to $0.b_{-1}b_{-2}0$
  - the value $0.b_{-1}b_{-2}1100$ is truncated to $0.b_{-1}b_{-2}1 + 0.001$.
- Also termed as "round to the nearest number or nearest even number in case of a tie"
- The error range is approximately $-1/2$ to $+1/2$ in the LSB position of the retained bits.

# WAYS OF TRUNCATION: ROUNDING

- Best method
- But most difficult to implement because it requires an addition operation and a possible renormalization.
- This rounding technique is the default mode for truncation specified in the IEEE floating-point standard
- When three guard bits are used, using Rounding procedure, the value 0.001100 is truncated to 0.010
- Similarly,
  i. 0.111 011=0.111
  ii. 0.110 011=0.110
  iii. 0.111 101=0.111+0.001=1.000
  iv. 0.111 100=0.111+0.001=1.000
  v. 0.110 100=0.110
  vi. 0.101 100=0.101+0.001=0.110

# IMPLEMENTING ROUNDING

- Requires only three guard bits to be carried along during the intermediate steps in performing an operation.
- The first two of these bits are the two most significant bits of the section of the mantissa to be removed.
- The third bit is the logical OR of all bits beyond these first two bits in the full representation of the mantissa.
- It should be initialized to 0.
- If a 1 is shifted out through this position while aligning mantissas, the bit becomes 1 and retains that value; hence, it is usually called the sticky bit.



# INSTRUCTION SET ARCHITECTURE

- Machine instructions and program execution
- Addressing methods for accessing register and memory operands

MODULE 2

# MEMORY LOCATIONS, ADDRESSES, AND OPERATIONS

- Memory consists of many millions of storage cells, each of which can store 1 bit.
- Data is usually accessed in n-bit groups. n is called word length.

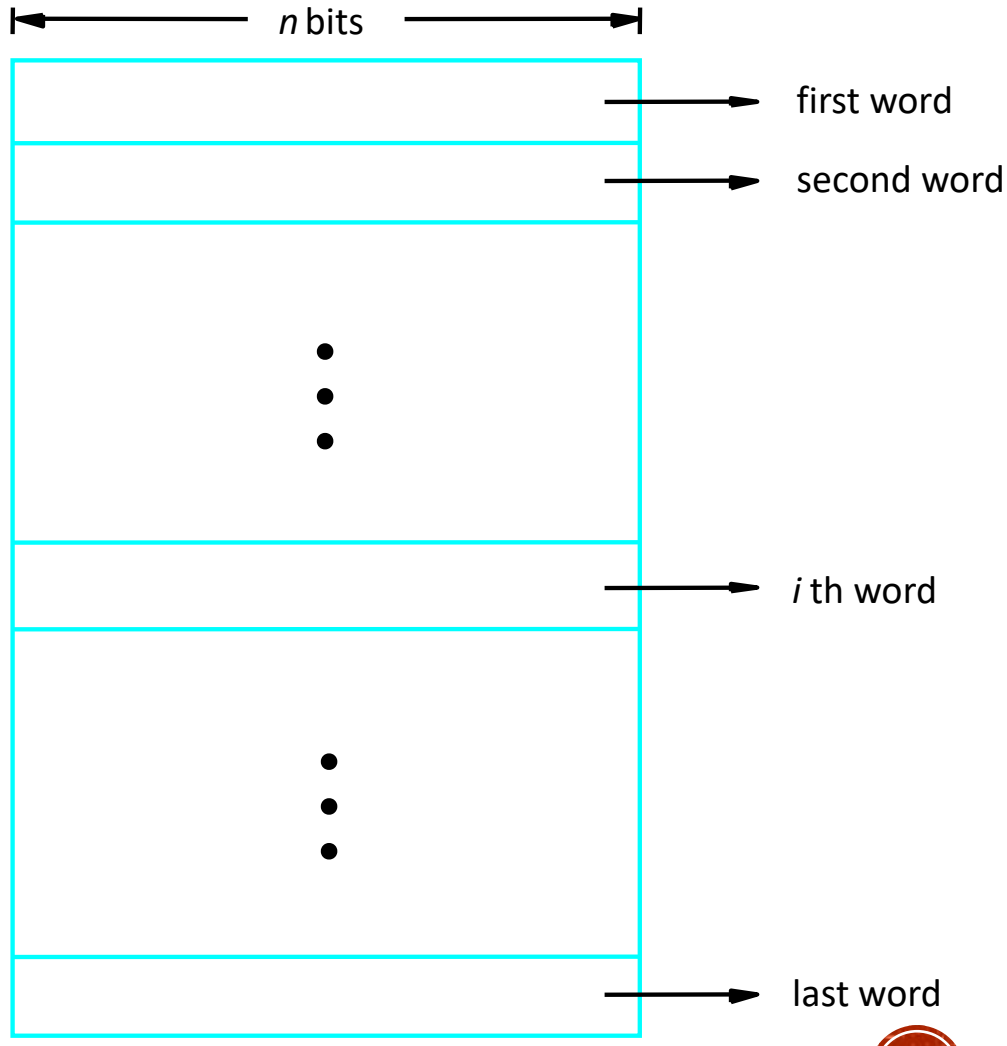
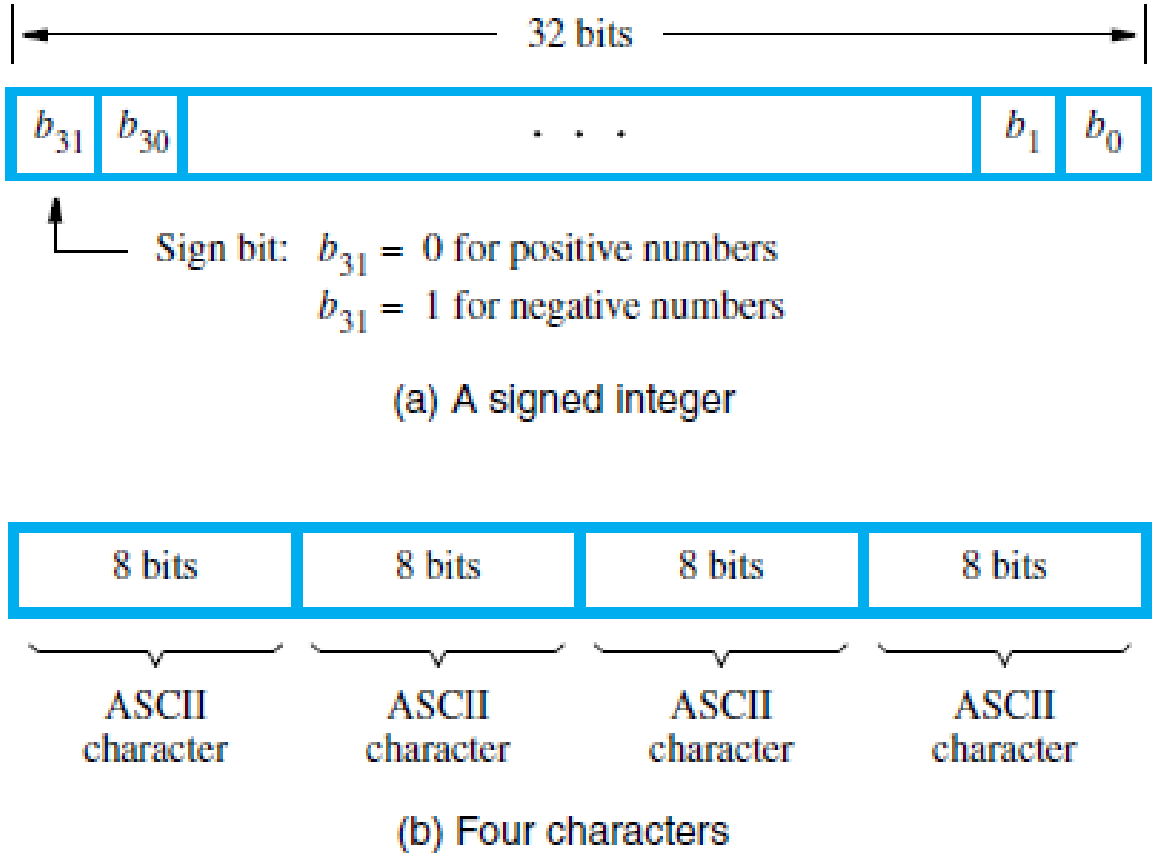



Figure 2.5. Memory words.

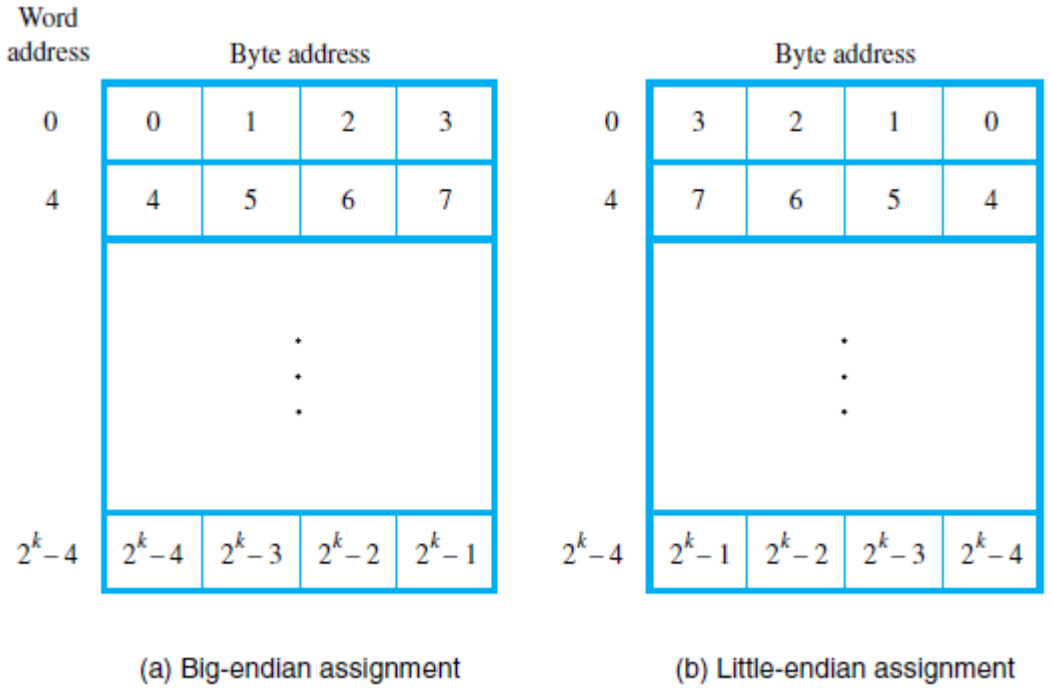
# MEMORY LOCATIONS, ADDRESSES, AND OPERATIONS

- 32-bit word length example
- To retrieve information from memory, either for one word or one byte (8-bit), addresses for each location are needed.
- A k-bit address memory has $2^k$ memory locations, namely 0 - $2^k$-1, called memory space.
- 24-bit memory: $2^{24}$ = 16,777,216 = 16M (1M=$2^{20}$)
- 32-bit memory: $2^{32}$ = 4G (1G=$2^{30}$)
- 1K(kilo)=$2^{10}$
- 1T(tera)=$2^{40}$



# BYTE ADDRESSABILITY

- It is impractical to assign distinct addresses to individual bit locations in the memory.
- The most practical assignment is to have successive addresses refer to successive byte locations in the memory – byte-addressable memory.
- Byte locations have addresses 0, 1, 2, … If word length is 32 bits, the successive words are located at addresses 0, 4, 8,…
- two ways that byte addresses can be assigned across words
  - Big-Endian: lower byte addresses are used for the most significant bytes of the word
  - Little-Endian: opposite ordering. lower byte addresses are used for the less significant bytes of the word




(a) Big-endian assignment (b) Little-endian assignment

# TOPICS COVERED FROM

- Textbook 1:
  - Chapter 9: 9.7.1, 9.7.2
  - Chapter 2: 2.1