Computer Organization and Architecture

2%	Unit - 1	Data Representation	2 hrs	T1
5%	Unit - 2	Registers Transfer & Micro-operations	5 hrs	T1
6%	Unit - 3	Basic Computers Organization	6 hrs	T1
8%	Unit - 4	Basic Computers Design	4 hrs	T2
18%	Unit - 5	Programming the basic Computers	8 hrs	T2
12%	Unit - 6	Central Processing Unit	8 hrs	T3
10%	Unit - 7	Pipeline Processing	5 hrs	T3
5%	Unit - 8	Computers Arithmatic	2 hrs	T3
14%	Unit - 9	Memory Organization	6 hrs	T4
9%	Unit - 10	Input - Output Organization	4 hrs	T4

Unit-1 : Data Representation : (2%)

→ Fixed Point
→ Floating Point Representation

* Basic Computer Data types :-

→ **Data** : Data refers to an information or a collection of facts.

→ Digital Computers works on binary data which is binary number format (0 or 1).

→ Data are numbers and other binary coded information that are useful to get the required results.

→ Data types found in registers of digital computers are of 3 types :-

(i) Numbers
(ii) Letters of Alphabet
(iii) Other discrete Symbols.

→ Why all types of data are represented in computers registers in binary Coded forms ?

→ Because registers are used to store the data in digital computers.
→ Registers are group of FF and flip-flops are two-state device that can store only 1's and 0's.

* Number Systems :

→ Number System uses a specific 'radix' or 'base'.

→ Radix that is powers of 2 is widely used in computers arithmatics.

→ Commonly used radices in digital computers are :

Radix - 2 : Binary (0 and 1)
Radix - 8 : Octal (0 to 7)
Radix - 16 : Hexadecimal (0 to 15)
Radix - 10 : Decimal (0 to 9)

→ For decimal number system the digit 724.5 is represented as -

$$724.5 = (7 \times 10^2) + (2 \times 10^1) + (4 \times 10^0) + (5 \times 10^{-1})$$

$$= 700 + 20 + 4 + \frac{5}{10}$$

→ Similarly for binary (base-2) number system ; a binary number 101101 is represented as :

$$(1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 45$$

→ To show the equality between decimal & binary we will write

$$(101101)_2 = (45)_{10}$$

→ Similarly we can convert any numbers system to decimal system by representing the numbers in weighted sum format.

→ For Hex numbers $(F3)_{16}$:

$$(F3)_{16} = (15 \times 16^1) + (3 \times 16^0) = (243)_{10}$$

→ Conversion from decimal to its equivalent radix 'r' system is carried out by seperating the numbers into integer and fraction parts.

For example $(41.6875)_{10} = (?)_2$

Integer = 41

2	41	
2	20	1
2	10	0
2	5	0
2	2	1
2	1	0
	0	1

Fraction = 0.6875

$0.6875 \times 2 = 1.3750$
$0.3750 \times 2 = 0.75$
$0.75 \times 2 = 1.5$
$0.5 \times 2 = 1.0$

$(0.6875)_{10} = (0.1011)_2$

$(41)_{10} = (101001)_2$

$(41.6875)_{10} = (101001.1011)_2$

* Complements :

→ Complements are used in digital computers for simplifying the subtraction operation & for logical manipulation.

→ Two types of complements for each base 'r' system :
- (i) r's complement
- (ii) (r-1)'s complement

→ For binary numbers : 2's complement
(2-1)'s complement

→ For decimal numbers : 10's complement
(10-1)'s complement

Examples :

(i) 1's complement of 101101 = 010010

(ii) 2's complement of 101101

= (1's complement of 101101) + 1

$$= \begin{array}{r} 010010 \\ +\quad\quad 1 \\ \hline 010011 \end{array}$$

(iii) 9's complement of 546700

$$= \begin{array}{r} 999999 \\ -\,546700 \\ \hline 453299 \end{array}$$

(iv) 10's complement of 2000

$= (9\text{'s comp. of } 2000) + 1$

$= \begin{array}{r} 9999 \\ -\ 2000 \\ \hline 7999 \end{array} + 1 \Rightarrow 8000$

Second Method : $\begin{array}{r} 10000 \\ -\ 2000 \\ \hline 8000. \end{array}$

1. Fixed-Point Representation :

→ Unsigned Numbers : All +ve numbers including zero

→ Sign bit → It is a bit in signed number representation that indicates the sign (+ve or -ve) of number.
→ It is located at MSB position.

If Sign bit = 0 ⇒ Num is Positive
Sign bit = 1 ⇒ Num is Negative.

→ Rest of the numbers other Sign bit can be represented in 3 ways :-

(i) Signed Magnitude Representation
(ii) Signed 1's complement Representation
(iii) Signed 2's complement Representation

Ex Represent $(+14)_{10}$ in 8-bit binary numbers and $(-14)_{10}$ in 8-bit Signed Magnitude, Signed 1's and Signed 2's complement formats.

+14 = Unsigned Numbers
= 00001110 (in 8-bit)

(-14) in Signed Magnitude Representation
= 1000 1110
↑ MSB = 1 to indicate that the num is -ve.

(-14) in Signed 1's Complement Representation:
= 1 111 0001
1's Complement of (+14)

(-14) in Signed 2's complement Representation:
= 1111 0010
2's Comp. of (+14)

Note: In general Signed 2's complement representation is most widely used format in computer arithmatics.

* Arithmatic Addition:

Rule: Add the two numbers, including their sign bits & discard any carry out of the sign bit position.

Ex Perform arithmatic addition of the given numbers. Use 8-bit representation and 2's complement representation for -ve numbers:

(i) (+6) + (+13)

+6		0000 0110
+13	+	0000 1101
+19		0001 0011

(ii) (-6) + (+13)

2's comp of (+6) = (-6)

8-bit representation of (+6) = 0000 0110

2's comp of (+6) = 1111 1010

-6		1111 1010
+13	+	0000 1101
+7		0000 0111 (Carry discarded)

(iii) (+6) + (-13)

(+13) in 8-bits = 0000 1101

2's comp. of (+13) = 1111 0011

+6		0000 0110
-13	+	1111 0011
-7		1111 1001 → No carry

→ Num is -ve

→ Sign bit = 1

Sign-bit

2's Complement of 11111001 = 00000111
= $(7)_{10}$

So, original -ve no. is = (-7)

(iv) (-6) + (-13)

(-6) and (-13) in 8-bit 2's comp form:

-6		11111010
-13	+	11110011
-19		11101101

⇒ Note:

Range of n-bit Signed Magnitude Num:

$-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

Range of n-bit Signed 1's comp

$-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

Range of n-bit Signed 2's comp

$-(2^{n-1})$ to $+(2^{n-1}-1)$

⇒ For example range of 8-bit Signed 2's Complement numbers:

$-(2^{8-1})$ to $(2^{8-1}-1)$

= -128 to +127

The number out of this range is invalid.

DATE
PAGE

* Overflow Condition :

→ When two numbers of 'n' digits each are added and the sum generates 'n+1' digits, Overflow is occurred.

→ An overflow is a problem in digital computers because the width of registers is finite.

→ A result that contained 'n+1' bits cannot be accommodated in register with a standard length of 'n' bit.

→ Overflow may occur is two numbers added are both positive or both negative.

→ For example : (+70) and (+80) are Stored in two 8-bit registers.

→ The result of an addition : 70 + 80 = +150

→ As we know it is out of the range of (−128 to +127). Since the sum of two numbers exceed the capacity of 8-bit register.

```
+70      0100 0110
+80      0101 0000
-------  ---------
+150     1001 0110
            ↓
         01001 0110
         ↑
         Sign bit for +ve numbers
```

→ Note that 8-bit result that should have been positive but it has -ve sign bit (Means Sign bit = 1)

→ To make it positive an extra bit '0' is required at MSB position, which makes the length of 9-bits.

→ So, 9-bit answer will be correct but cannot be accommodated by 8-bit registers. We say that an overflow occurred.

⇒ Overflow Detection:

→ An overflow can be detected by observing the carry into the sign bit position & carry out of sign bit position.

	$C_7C_6C_5C_4C_3C_2C_1C_0$
+ 70	0100 0110
+ 80	0101 0000
+150	1001 0110

Observe the Carry C_6 which is = 1
and Carry $C_7 = 0$, $C_6 \oplus C_7 = 1$

Overflow detects if $C_6 \oplus C_7 = 1$

DATE
PAGE

* Arithmatic Subtraction

→ Take the 2's complement of Subtrahend (including sign bit) and add it to the minuend (including sign bit)

→ A Carry out of the sign bit is discarded.

$$(\pm A) - (+B) = (\pm A) + (-B)$$
$$(\pm A) - (-B) = (\pm A) + (+B)$$

→ For example,
Consider the subtraction of (-6) and (-13)

$= (-6) - (-13)$

$= (-6) + (+13)$

(-6) → Minuend, (+13) → Subtrahend.

→ As we know $(-6) - (-13) = (+7)$

→ To perform $(-6) + 13$, first we take 2's complement of 6 and add it with 13.

In 8-bit representation,

$(6)_{10} = (0000\ 0110)_2$

$(-6)_{10} = (1111\ 1010)_2$ ← 2's comp.

$$\begin{array}{ll} (-6) & 1111\ 1010 \\ +(+13) & +\ 0000\ 1101 \\ \hline (+7) & 0000\ 0111 \end{array}$$ (Carry Discarded)

Ex Perform the addition operation using Signed 10's Complement representation.
$(+375) + (-240) = ?$

$$\begin{array}{r} 0375 \\ +\ 9760 \\ \hline 0135 \end{array}$$ → 10's complement of 0240

$$= \begin{array}{r} 10000 \\ -\ 0240 \\ \hline 9760 \end{array}$$

→ The sign of decimal no. is usually represented with four bits to represent it in BCD form.

$$\begin{array}{r} 0375 \\ +\ 9760 \\ \hline 0135 \end{array} \qquad \begin{array}{r} 0000\ 0011\ 0111\ 0101 \\ +\ 1001\ 0111\ 0110\ 0000 \\ \hline (1001\ 1010\ 1101\ 0101)_{BCD} \end{array}$$

Invalid BCD.
(Add 0110 to get Valid BCD)

$$\begin{array}{r} 1001\ 1010\ 1101\ 0101 \\ +\ 0000\ 0110\ 0110\ 0000 \\ \hline 1010\ 0001\ 0011\ 0101 \\ 0110\ 0000\ 0000\ 0000 \\ \hline 0000\ 0001\ 0011\ 0101 \end{array} = (0135)_{10}$$

Add 0110

2. Floating Point Representation:

Definition: A floating point number, is a +ve or -ve whole numbers with a decimal point.
For example: 5.5, 0.25, -103.4

→ The name floating point refers that the decimal point can float to any position.

→ The floating point number has two parts:

(i) First part represents a signed, fixed point number called **Mantissa**.

(ii) Second Part represent the position of decimal point called **Exponent**.

Floating Point Numbers

- Mantissa (May be an integer or fraction)
- Exponent

For example: $+6132.789$ is represented in floating point

$= +0.6132789 \times 10^4$

- $+0.6132789$ → Mantissa
- 10^4 → Exponent part

→ In general floating point numbers is always represented as -

$$m \times r^e$$

where m = mantissa
e = exponent
r = radix or base

→ For floating point binary numbers

$$= m \times 2^e$$

→ For example: binary number 1001.11 is represented with 8-bit fraction & 6-bit exponent as follows:

Fraction	Exponent
01001110	000100

$$1001.11 = +(0.100111)_2 \times 2^{+4}$$

Normalized Floating point Numbers:

→ A floating point number is said to be normalized if MSB of mantissa is non zero.

→ For example decimal no 350 is normalized but 0350 is not.

→ A binary no : 101.1 can be represented in floating point numbers as follows :

$101.1 = 1011 \times 2^{-1}$ (Point shifted towards right)

$101.1 = 10.11 \times 2^{1}$ (Point shifted towards left)

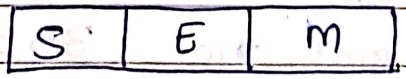
→ Normalization done in two ways :
(i) Explicit
(ii) Implicit

→ Standard formats for explicit & implicit representation :

Explicit :	$(-1)^S \times 0.M \times 2^E$
Implicit :	$(-1)^S \times 1.M \times 2^E$

Here S = Sign bit = 0 or 1
(0 for +ve, 1 for -ve)
M = Mantissa
E = Exponent.

* Standard IEEE format for floating point arithmatic :



Sign bit Expo Mantissa

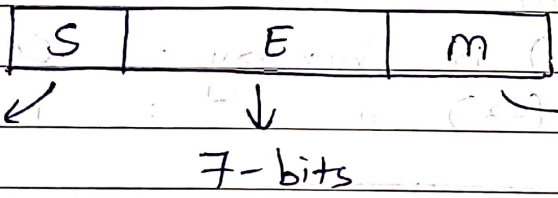
→ Most of the binary floating point representations follow IEEE-754 Standard.

→ The data type 'float' uses IEEE-32 bit format.

Ex Represent a numbers $(101.1)_2$ in floating point numbers with 1 bit of sign, 7-bits for exponent and 24-bits for mantissa.

$$101.1 = \underbrace{0.1011}_{M} \times 2^{\overbrace{3}^{E}} \quad \text{(Normalized)}$$

Num is positive so, Sign bit (S) = 0



0	0000011	1011.... (20 zeros
S	E	M
(1)	(7)	(24)

→ Total 32 bits

Ex Represent $(5.875)_{10}$ in explicit and implicit floating point numbers.

$$(5.875)_{10} = (101.111)_2$$

For explicit : 0.101111×2^3

For implicit : 1.01111×2^2

In a Standard ~~As~~ ANSI - 32 bit format,

Explicit :

0	000.00.11	1011110..(17 zeros)
S	E	M
(1)	(7)	(24)

Implicit :

0	0000010	011110....(18 zeros)
S	E	m

Ex $x = 0.0001101001101$, Represent x in implicit & explicit floating point numbers.

Explicit : $x = 0.1101001101 \times 2^{-3}$

0	E	m
S		

Exponent = -3

(3) in binary = 0011

(-3) " = 1101 (2's comp)

0	1101	1101001101
S	E	m

Implicit : $x = 1.1010011101 \times 2^{-4}$

$(4)_{10}$ in binary = 0100

$(-4)_{10}$ in binary : 1100 (2's comp).

0	1100	1010011101
S	E	m

* ANSI 32-bit floating point byte format:

SEEEEEEE	. MMMMMMMM	MMMMMMMM	MMMMMMMM
(Byte-1)	(Byte-2)	(Byte-3)	(Byte-4)

Here S = Sign of Mantissa
E = Exponent bits
M = Mantissa bits

Ex Express $(13)_{10}$ and $(-17)_{10}$ in Standard floating point byte format.

$(13)_{10} = (1101)_2$

$= 0.1101 \times 2^4$

$= 00000100.\ 11010000\ \ 00000000\ \ 00000000$

$(-17) = -10001$

$= -0.10001 \times 2^5$

$= 10000101.\ 10001000.\ 00000000\ \ 00000000$

Unsigned Integers : [Integers] Ex : 5, 10, 2

Signed Integers : [Sign | Integers] Ex : -5, +2

Unsigned fixed point : [Integers | Fraction] Ex : 3.7

Signed fixed point : [Sign | Integers | Fraction]
Ex : -2.5, +1.3

Floating point : [Sign | Expo | Mantissa]
Ex : -0.3×10^{-2}

Examples for Practice

Ex-1 Perform the subtraction with following unsigned decimal numbers by taking 10's complement of subtrahend.

① 5250 − 1321

Taking 10's comp of 1321

$$= \begin{array}{r} 10000 \\ -\ 1321 \\ \hline 8679 \end{array} \qquad \begin{array}{r} 5250 \\ +\ 8679 \\ \hline 1)\ 3929 \end{array} \text{ (Ignore Carry)}$$

② 20 − 100

10's comp of 100 $= \begin{array}{r} 1000 \\ -\ 100 \\ \hline 900 \end{array}$

$$\begin{array}{r} 020 \\ +\ 900 \\ \hline 920 \end{array}$$ (No carry, means no. is −ve)

Again 10's comp of 920 $= \begin{array}{r} 1000 \\ -\ 920 \\ \hline 080 \end{array}$

Answer = $\boxed{-080}$

EQ.2 Perform the subtraction with following unsigned binary no by taking 2's Complement of subtrahend

① 11010 − 1101

```
 11010          Taking 2's comp of 01101
-01101                  = 10011
```

Now,
```
  11010
+ 10011
-------
1)01101   (Ignore Carry)
```

26 − 13 = 13.

② 100 − 110000

2's comp of 110000 = 010000

```
  000100
+ 010000
--------
  010100   (No carry, Num is -ve)
```

2's comp of 010100 = 101100

Ans = −101100

4 − 48 = −44

Ex Consider the signed binary number to be 10111011. What is the decimal equivalent of this num if it is in Sign magnitude form, or 1's comp or 2's comp form.

for Signed Magnitude format, first left most bit is Sign bit and remaining bits represent magnitude.

$(10111011)_2$ in sign magnitude form

$= -(0111011)_2$

$= -59$

In 1's comp form,

$10111011 \xrightarrow[\text{Comp}]{1's} 01000100 = (68)_{10}$

$= -68$

In 2's comp form;

$10111011 \xrightarrow[\text{Comp}]{2's} 01000101 = (69)_{10}$

$= -69$

Ex Express (-53.5) using 32-bit floating point format in which 1-bit sign bit, 8 bit for exponent and 23-bit for fractional part.

$(-53.5)_{10} = (-110101.1)_2$

$= (-0.1101011) \times 2^6$

S	E	M

1	00000110	11010110 .... (15 zeros)

Ex Let R1 and R2 be two 4-bit registers that store numbers in 2's comp form. For the operation R1 + R2 which one of the following values of R1 & R2 gives an arithmatic overflow?

(A) R1 = 1011, R2 = 1110
(B) R1 = 1100, R2 = 1010
(C) R1 = 0011, R2 = 0100
(D) R1 = 1001, R2 = 1111

Check all the options

Option: 1: R1 = 1011 = −0101 = −5
R2 = 1110 = −0010 = −2
−7

Range for 2's comp no: -2^{n-1} to $2^{n-1}-1$
$= -2^{4-1}$ to $2^{4-1}-1$
$= -8$ to 7

Here $R_1 + R_2 = -7$ (No overflow)

Option: 2: R1 = 1100, R2 = 1010

R1 = 1100 = −0100 = −4
R2 = 1010 = −0110 = −6
−10 (Out of Range)

Overflow Occurred.

Ex: What is the 16 bit pattern which represent (−13.5) in normalized Signed Magnitude fraction. S = 1 bit, Exponent = 7 bits and Mantissa = 8 bits. Represent it in Hex also.

Here −13.5 is −ve number so, S = 1

13.5 in binary = 1101.1
$= 0.11011 \times 2^4$

S	E	m
1	7	8

1	0000100	11011000
S	E	m

1 0000100 11011000

$= (84D8)_{16}$