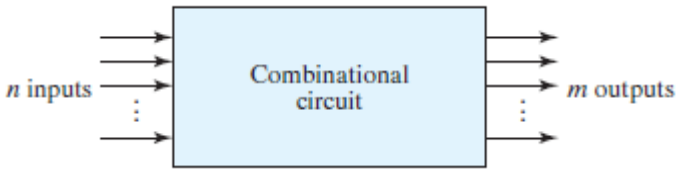
UNIT I

COMBINATIONAL LOGIC

Combinational Circuits – Analysis and Design Procedures - Binary Adder- Subtractor -Decimal Adder - Binary Multiplier - Magnitude Comparator - Decoders – Encoders – Multiplexers - Introduction to HDL – HDL Models of Combinational circuits.

COMBINATIONAL CIRCUITS

- *A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.*
- A combinational circuit performs an operation that can be specified logically by a set of Boolean functions.



Sequential circuits:

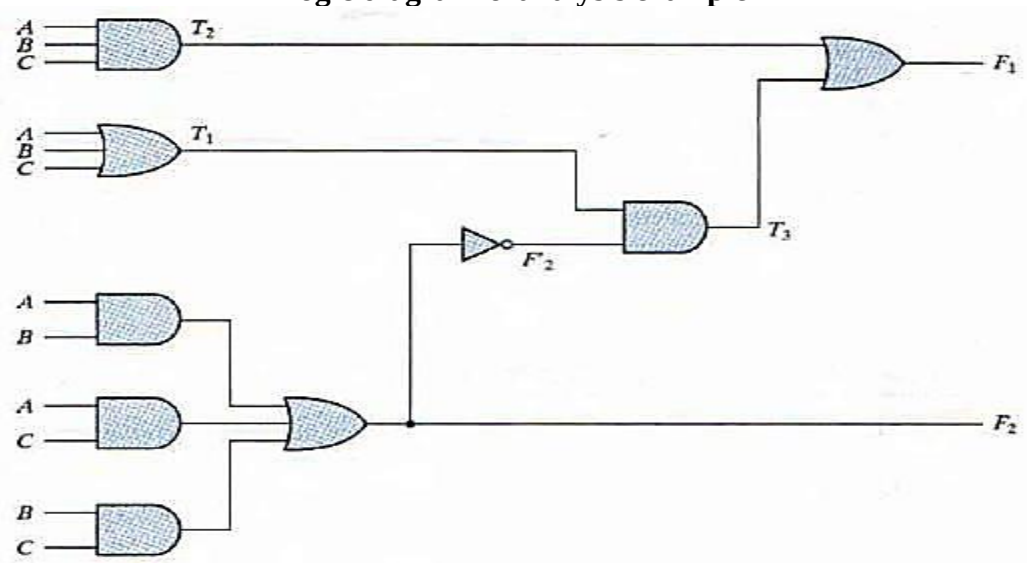
- *Sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements.*
- Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.

ANALYSIS PROCEDURE

Explain the analysis procedure. Analyze the combinational circuit the following logic diagram. (May 2015)

- The analysis of a combinational circuit requires that we determine the function that the circuit implements.
- The analysis can be performed manually by finding the Boolean functions or truth table or by using a computer simulation program.
- The first step in the analysis is to make that the given circuit is combinational or sequential.
- Once the logic diagram is verified to be combinational, one can proceed to obtain the output Boolean functions or the truth table.
- To obtain the output Boolean functions from a logic diagram,
  - ✓ Label all gate outputs that are a function of input variables with arbitrary symbols or names. Determine the Boolean functions for each gate output.
  - ✓ Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols or names. Find the Boolean functions for these gates.
  - ✓ Repeat the process in step 2 until the outputs of the circuit are obtained.
  - ✓ By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Logic diagramforanalysis example



The Boolean functions for the above outputs are,

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

Next, we consider outputs of gates that are a function of already defined symbols:

$$T_3 = F'_2 T_1$$
$$F_1 = T_3 + T_2$$

To obtain F_1 as a function of A, B, and C, we form a series of substitutions as follows:

$$\begin{aligned} F_1 = T_3 + T_2 &= F'_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

❖ Proceed to obtain the truth table for the outputs of those gates which are a function of previously defined values until the columns for all outputs are determined.

Truth Table for the Logic Diagram

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

DESIGNPROCEDURE

Explain the procedure involved in designing combinational circuits.

- The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained.
- The procedure involved involves the following steps,
- ✓ From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
- ✓ Derive the truth table that defines the required relationship between inputs and outputs.
- ✓ Obtain the simplified Boolean functions for each output as a function of the input variables.
- ✓ Draw the logic diagram and verify the correctness of the design.

****

CIRCUITS FOR ARITHMETIC OPERATIONS

Half adder:

Construct a half adder with necessary diagrams. ***(Nov-06,May- 07)***

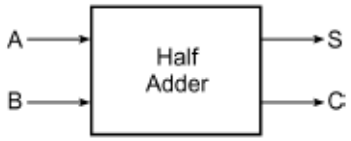
- A half-adder is an arithmetic circuit block that can be used to add two bits and produce two outputs SUM and CARRY.
- The Boolean expressions for the SUM and CARRY outputs are given by the equations

$$\text{SUM } S = A.\overline{B} + \overline{A}.B$$

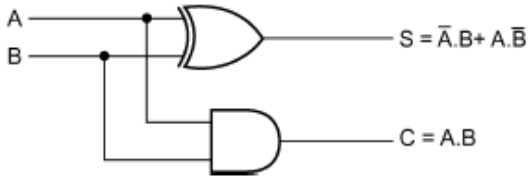
$$\text{CARRY } C = A.B$$

Truth Table:

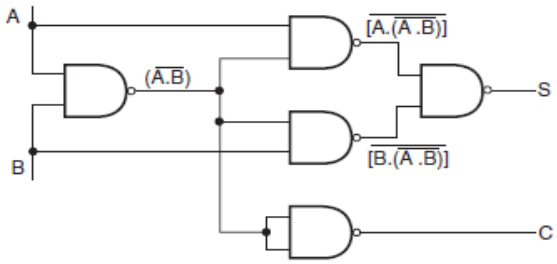
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Logic Diagram:



Half adder using NAND gate:



****

Full adder:

Design a full adder using NAND and NOR gates respectively. ***(Nov -10)***

- *A Full-adder is an arithmetic circuit block that can be used to add three bits and produce two outputs SUM and CARRY.*
- The Boolean expressions for the SUM and CARRY outputs are given by the equations

$$S = \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C}_{in} + A.\overline{B}.\overline{C}_{in} + A.B.C_{in}$$
$$C_{out} = B.C_{in} + A.B + A.C_{in}$$

Truth table:

Input variables			*Outputs*	
X	*A*	*B*	*S*	*C*
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Karnaugh map:

	A′B′	A′B	AB	AB′
X′		1		1
X	1		1	

K-Map for Sum

	A′B′	A′B	AB	AB′
X′			1	
X		1	1	1

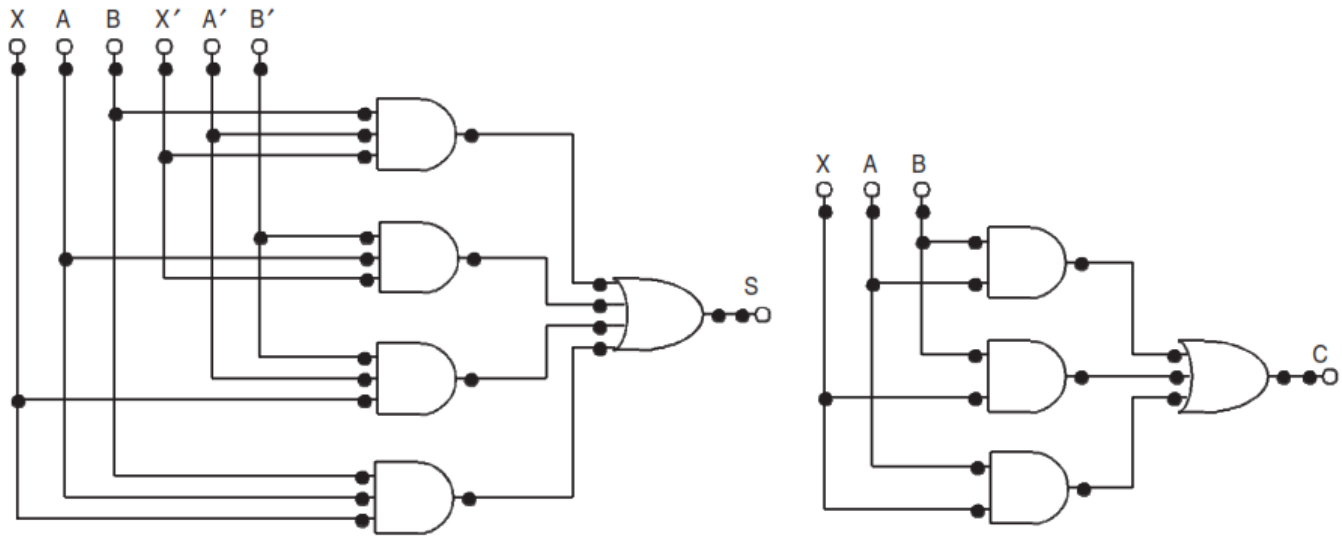
K-Map for Carry

- The simplified Boolean expressions of the outputs are

S = X′A′B + X′AB′ + XA′B′ + XAB

C = AB + BX + AX

Logic diagram:

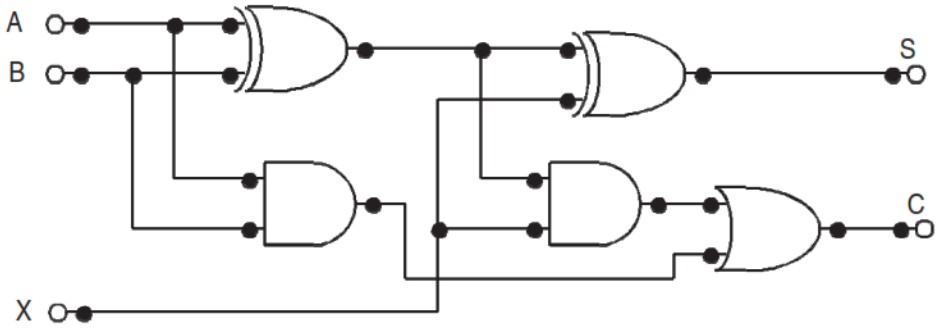


- The Boolean expressions of S and C are modified as follows

$$S = X'A'B + X'AB' + XA'B' + XAB$$
$$= X' (A'B + AB') + X (A'B' + AB)$$
$$= X' (A \oplus B) + X (A \oplus B)'$$
$$= X \oplus A \oplus B$$
$$C = AB + BX + AX = AB + X (A + B)$$
$$= AB + X (AB + AB' + AB + A'B)$$
$$= AB + X (AB + AB' + A'B)$$
$$= AB + XAB + X (AB' + A'B)$$
$$= AB + X (A \oplus B)$$

Full adder using Two half adder:

- Logic diagram according to the modified expression is shown Figure.



****

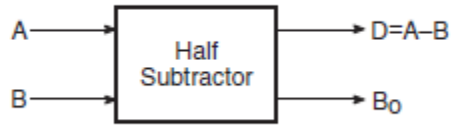
Half subtractor:

Design a half subtractor circuit. ***(Nov-2009)***

- A half-subtractor is a combinational circuit that can be used to subtract one binary digit from anotherto produce a DIFFERENCE output and a BORROW output.
- The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction. The Boolean expression for difference and borrow is:

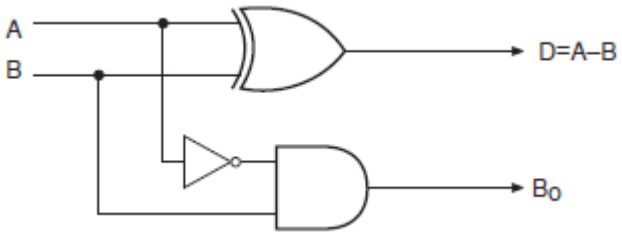
$$D = \overline{A}.B + A.\overline{B}$$

$$B_o = \overline{A}.B$$



A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Logic diagram:



****

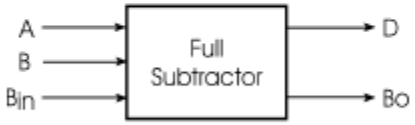
Full subtractor:

Design a full subtractor. ***(Nov-2009,07)***

- A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not.
- As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as Bin .
- There are two outputs, namely the DIFFERENCE output D and the BORROW output Bo. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit. The Boolean expression for difference and barrow is:

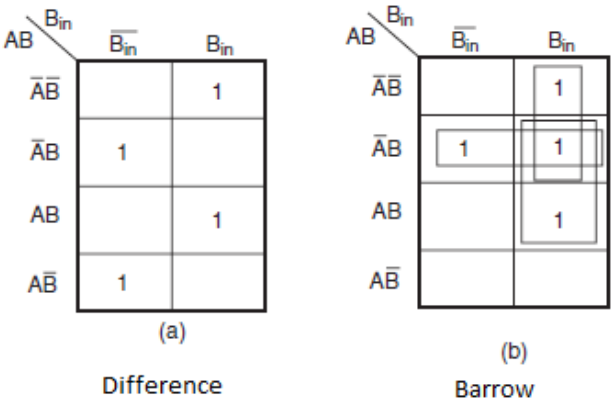
$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$

$$B_o = \overline{A}.B + \overline{A}.B_{in} + B.B_{in}$$

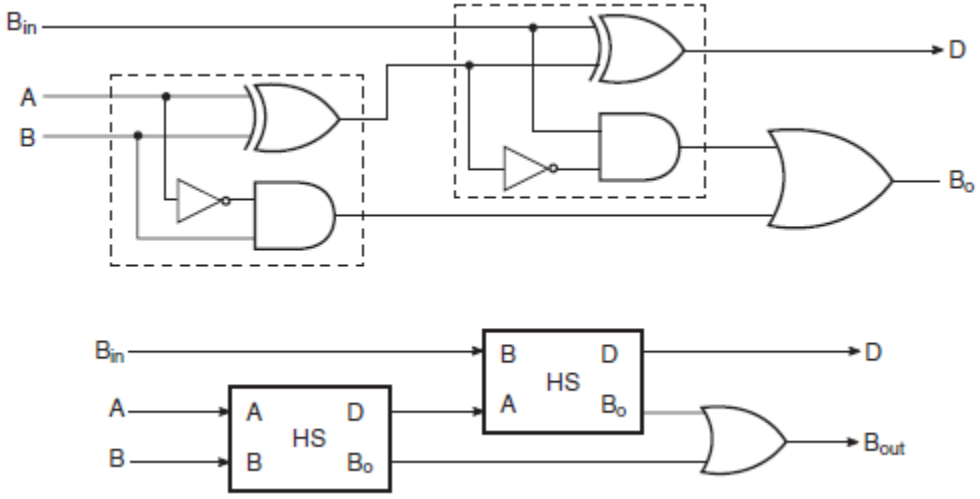


Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_O)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map:



Full subtractor using two half subtractor:



****

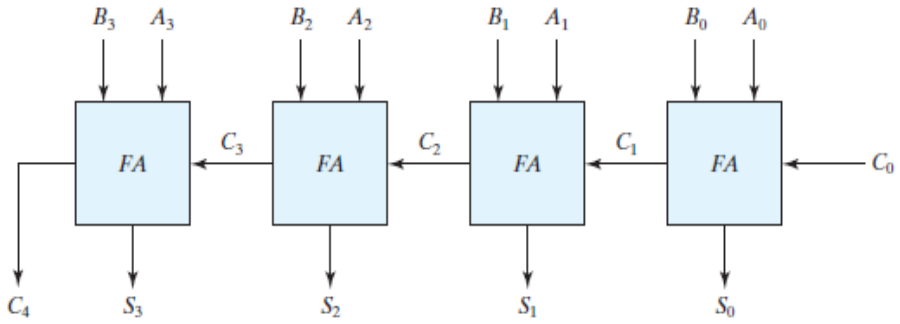
Parallel Binary Adder: (Ripple Carry Adder):

Explain about four bit adder. (or) Design of 4 bit binary adder – subtractor circuit. (Apr – 2019)

- ❖ *A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.*
- ❖ Addition of n-bit numbers requires a chain of n- full adders or a chain of one-half adder and n-1 full adders. In the former case, the input carry to the least significant position is fixed at 0.
- ❖ Figure shows the interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder.
- ❖ The carries are connected in a chain through the full adders. The input carry to the adder is C0, and it ripples through the full adders to the output carry C4. The S outputs generate the required sum bits.

Example: Consider the two binary numbers A = 1011and B = 0011. Their sum S = 1110 is formed with the four-bit adder as follows:

Subscript *i*:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}



- ✓ The carry output of lower order stage is connected to the carry input of the next higher order stage. Hence this type of adder is called ripple carry adder.
- ✓ In a 4-bit binary adder, where each full adder has a propagation delay of tp ns, the output in the fourth stage will be generated only after 4tp ns.
- ✓ The magnitude of such delay is prohibitive for high speed computers.
- ✓ One method of speeding up this process is look-ahead carry addition which eliminates ripple carry delay.

****

Complement of a number:

1's complement:

The 1's complement of a binary number is formed bychanging 1 to 0 and 0 to 1.

Example:

1. The 1's complement of 1011000 is 0100111.
2. The 1's complement of 0101101 is 1010010.

2's complement:

The 2's complement of a binary number is formed by adding 1 with 1's complement of a binary number.

Example:

1. The 2's complement of 1101100 is 0010100
2. The 2's complement of 0110111 is 1001001

Subtraction using 2's complement addition:

- ✓ The subtraction of unsigned binary number can be done by means of complements.
- ✓ Subtraction of A-B can be done by taking 2's complement of B and adding it to A.
- ✓ Check the resulting number. If carry present, the number is positive and remove the carry.
- ✓ If no carry present, the resulting number is negative, take the 2's complement of result and put negative sign.

Example:

Given the two binary numbers X = 1010100 and Y = 1000011, perform the subtraction **(a)** X - Y and **(b)** Y - X by using 2's complements.

Solution:

(a) X = 1010100

2's complement of Y = + 0111101

Sum= 10010001

Discard end carry. *Answer*: X - Y = 0010001

(b) Y = 1000011

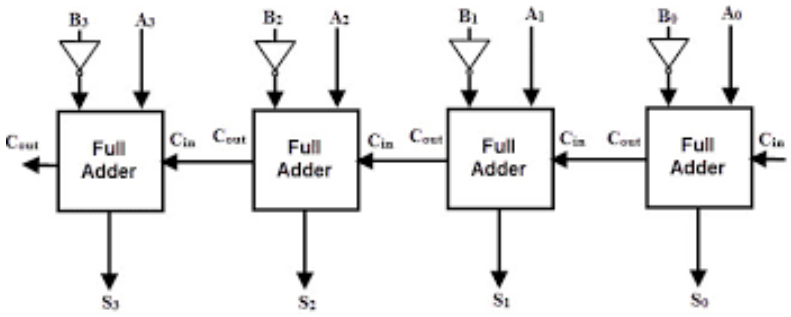
2's complement of X= + 0101100

Sum= 1101111

There is no end carry. Therefore, the answer is Y - X = -(2's complement of 1101111) =-0010001.

****

Parallel Binary Subtractor:



- ✓ The subtraction of unsigned binary numbers can be done most conveniently by meansof complements. The subtraction A - B canbe done by taking the 2's complement of B and adding it to A . The 2's complement canbe obtained by taking the 1's complement and adding 1 to the least significant pair ofbits. The 1's complement can be implemented with inverters, and a 1 can be added tothe sum through the input carry.
- ✓ The circuit for subtracting A - B consists of an adder with inverters placed betweeneach data input B and the corresponding input of the full adder. The input carry Cin mustbe equal to 1 when subtraction is

performed. The operation thus performed becomes *A*,plus the 1's complement of *B* , plus 1. This is equal to *A*plus the 2's complement of *B*.

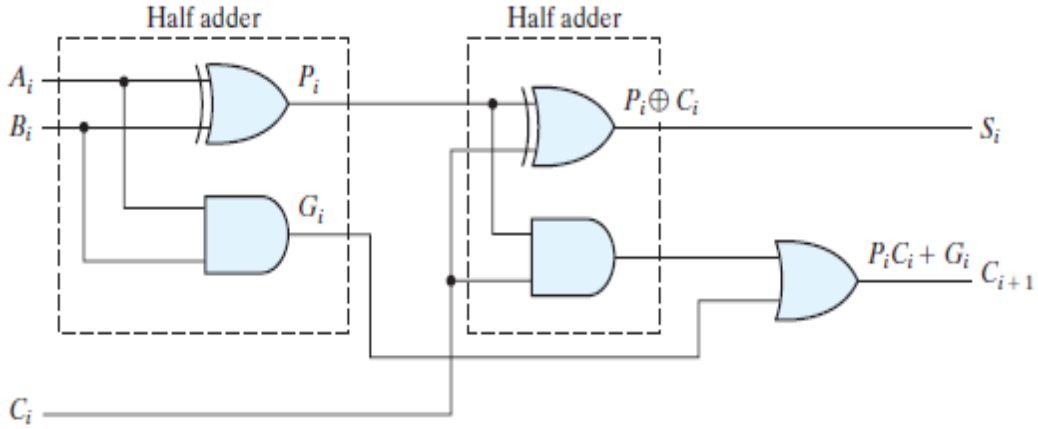
- ✓ For unsigned numbers, that gives *A-B* if *A*>=*B* or the 2's complement of *B* - *A*if *A* <*B*. For signed numbers, the result is *A* - *B*, provided that there is no overflow.

****

Fast adder (or) Carry Look Ahead adder:

Design a carry look ahead adder circuit. ***(Nov-2010)***

- ❖ *The carry look ahead adder is based on the principle of looking at the lower order bits of the augend and addend to see if a higher order carry is to be generated.*
- ❖ It uses two functions carry generate and carry propagate.



Consider the circuit of the full adder shown in Fig. If we define two new binaryvariables

$$P_i = A_i \oplus B_i$$
$$G_i = A_iB_i$$

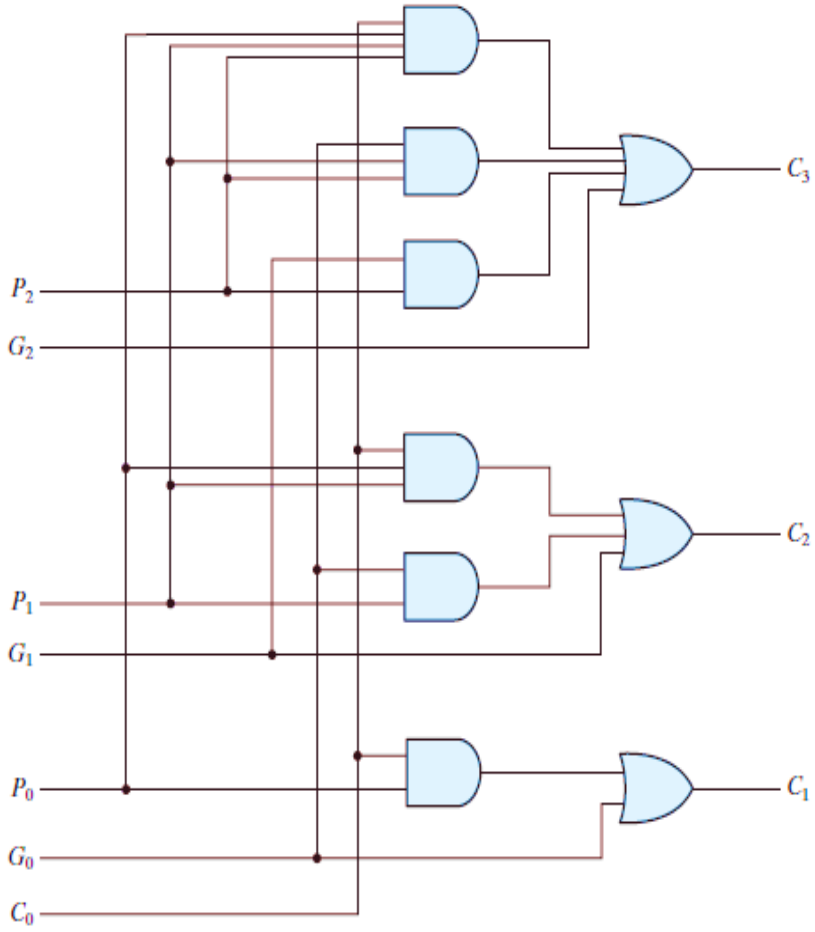
the output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_iC_i$$

Gi is called a carry generate, and it produces a carry of 1 when both Ai and Bi are 1,regardless of the input carry Ci. Pi is called a carry propagate, because it determines whether a carry into stage i will propagate into stage i + 1 (i.e., whether an assertion of Ci will propagate to an assertion of Ci+1).

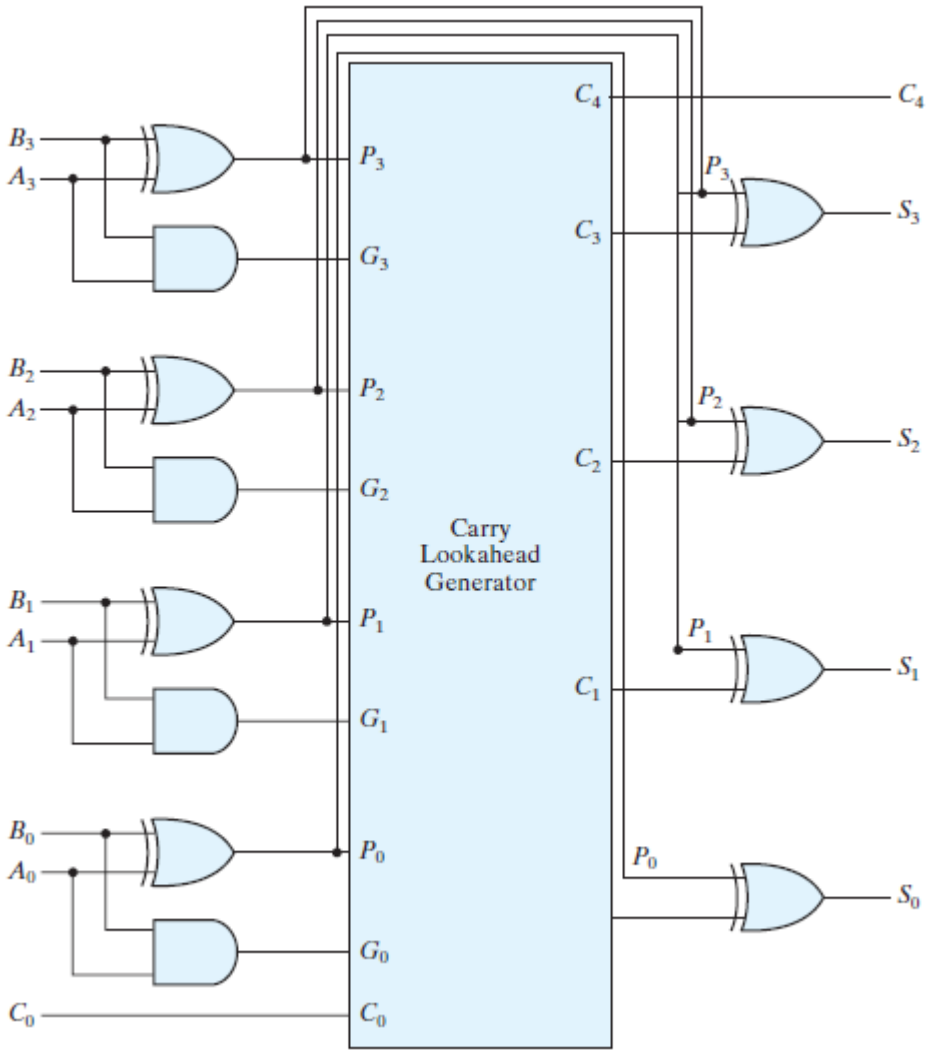
We now write the Boolean functions for the carry outputs of each stage and substitutethe value of each Ci from the previous equations:

$$C_0 = \text{input carry}$$
$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 = P_2P_1P_0C_0$$



Logic diagram of carry lookahead generator

- The construction of a four-bit adder with a carry lookahead scheme is shown in Fig.
- Each sum output requires two exclusive-OR gates.
- The output of the first exclusive-OR gate generates the Pi variable, and the AND gate generates the Gi variable.
- The carries are propagated through the carry look ahead generator and applied as inputs to the second exclusive-OR gate.
- All output carries are generated after a delay through two levels of gates.
- Thus, outputs S1 through S3 have equal propagation delay times. The two-level circuit for the output carry C4 is not shown. This circuit can easily be derived by the equation-substitution method.

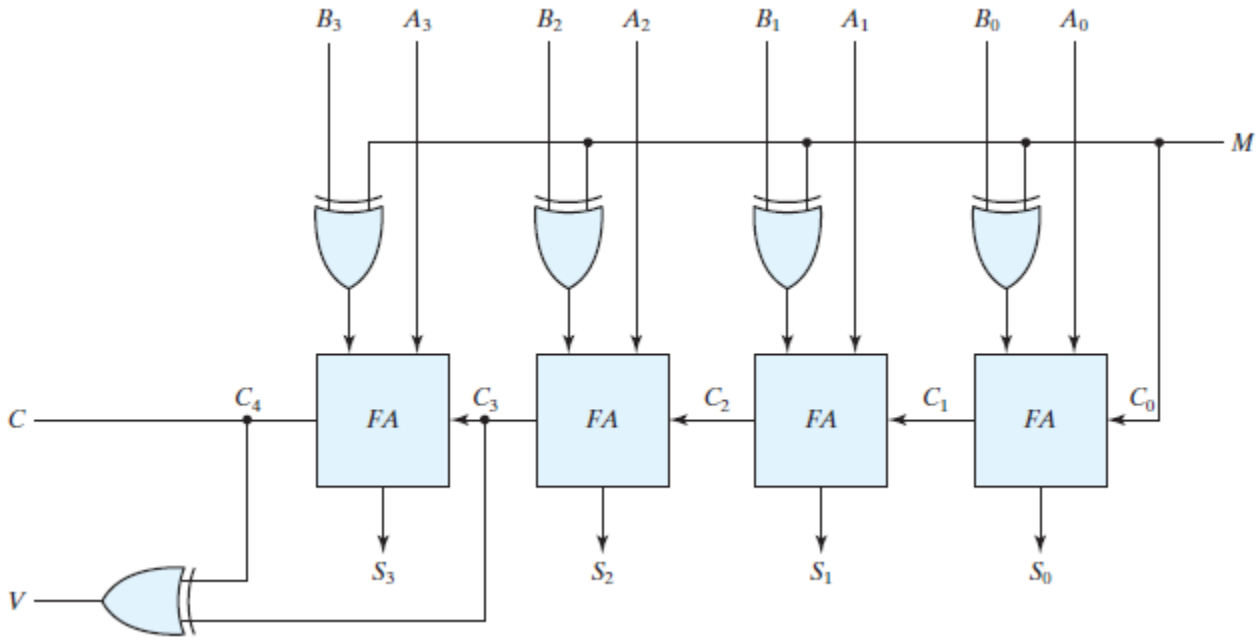


****

4 bit-Parallel adder/subtractor:

Explain about binary parallel / adder subtractor. [NOV – 2019]

- The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder. A four-bit adder–subtractor circuit is shown in Fig.
- The mode input M controls the operation. When $M = 0$, the circuit is an adder, and when $M = 1$, the circuit becomes a subtractor.



- It performs the operations of both addition and subtraction.
- It has two 4bit inputs $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$.
- The mode input M controls the operation when M=0 the circuit is an adder and when M=1 the circuits become subtractor.
- Each exclusive-OR gate receives input *M* and one of the inputs of *B* .
- When *M* = 0, we have *B xor*0 = *B*. The full adders receive the value of *B* , the input carry is 0, and the circuit performs *A* plus *B* . This results in sum $S_3S_2S_1S_0$and carry C_4.
- When *M* = 1, we have *B* xor 1 = *B*' and C_0 = 1. The *B* inputs are all complemented and a 1 is added through the input carry thus producing 2's complement of B.
- Now the data $A_3A_2A_1A_0$will be added with 2's complement of $B_3B_2B_1B_0$to produce the sum i.e., A-B if A≥B or the 2's complement of B-A if A<B.

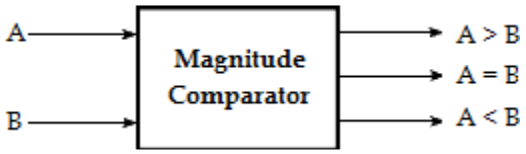
****

Comparators

Design a 2 bit magnitude comparator. ***(May 2006)***

It is a combinational circuit that compares two numbers and determines their relative magnitude. The output of comparator is usually 3 binary variables indicating:

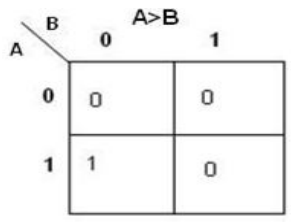
A<B, A=B, A>B



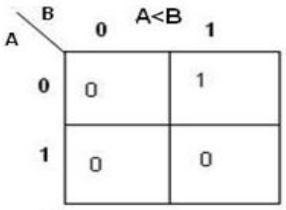
1-bitcomparator: Let's begin with 1bit comparator and from the name we can easily make out that this circuit would be used to compare 1bit binary numbers.

A	B	A>B	A=B	A<B
0	0	0	1	0
1	0	1	0	0
0	1	0	0	1
1	1	0	1	0

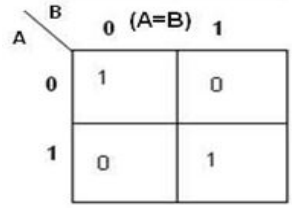
For a 2-bit comparator we have four inputs A1 A0 and B1 B0 and three output E (is 1 if two numbers are equal) G (is 1 when A>B) and L (is 1 when A<B) If we use truth table and K-map the result is



Equation is $A>B = A.\overline{B}$



Equation is $A<B = \overline{A}.B$



The equation is $f(A=B) = \overline{A}.\overline{B} + A.B$
$= A \text{ XNOR } B$

Design of 2 – bit Magnitude Comparator.

The truth table of 2-bit comparator is given in table below

Truth table:

Inputs				Outputs		
A_3	A_2	A_1	A_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

K-Map:

For A>B

A_1A_0 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$A>B = A_0B_1'B_0' + A_1B_1' + A_1A_0B_0'$

For A=B

A_1A_0 \ B_1B_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

$A=B = A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0B_1B_0 + A_1A_0'B_1B_0'$

$= A_1'B_1' (A_0'B_0' + A_0B_0) + A_1B_1 (A_0B_0 + A_0'B_0')$

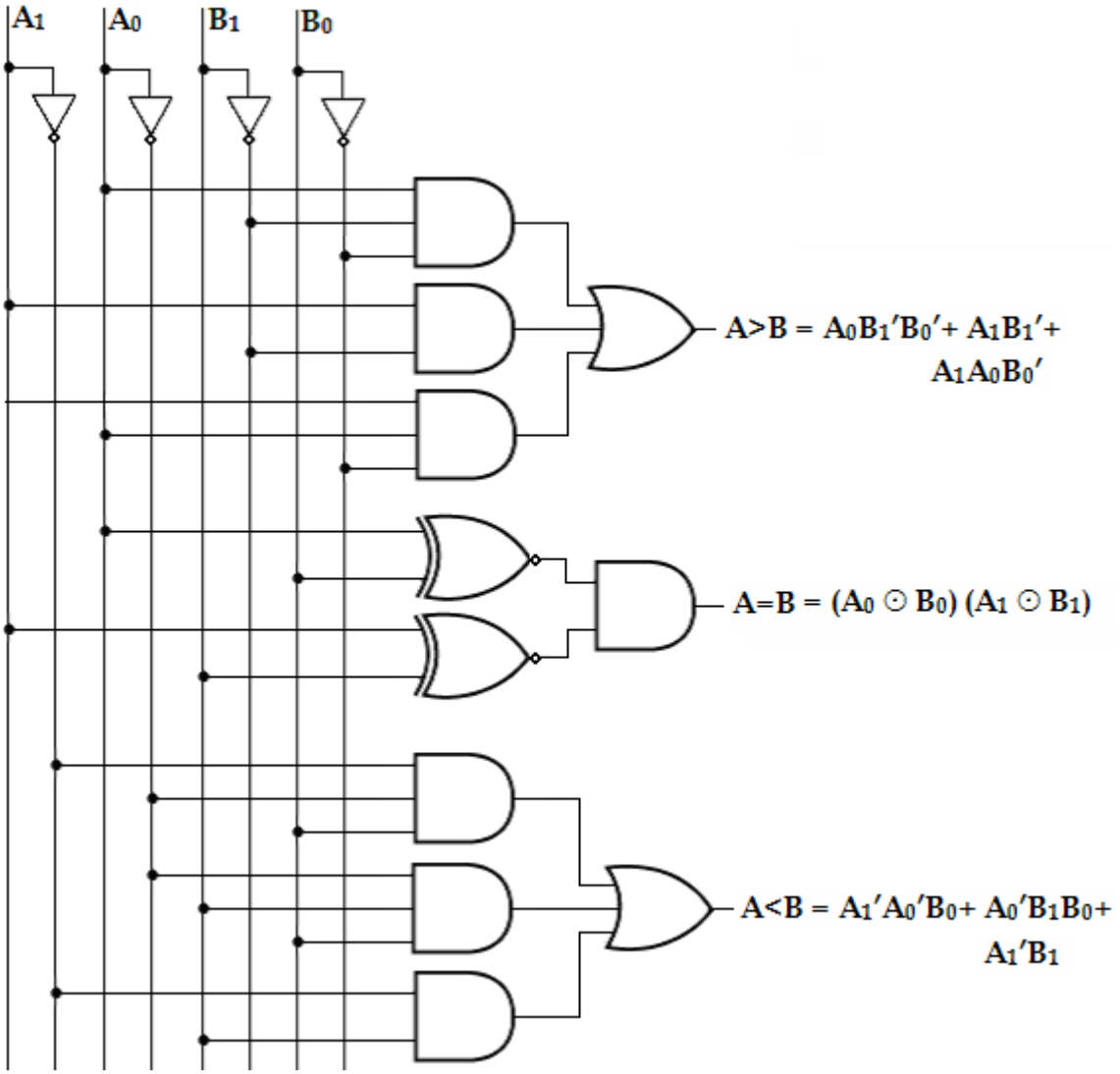
$= (A_0 \odot B_0)(A_1 \odot B_1)$

For A<B

A_1A_0 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$A<B = A_1'A_0'B_0 + A_0'B_1B_0 + A_1'B_1$

Logic Diagram:



****

4 bit magnitude comparator:

Design a 4 bit magnitude comparators. (Apr – 2019)

Input

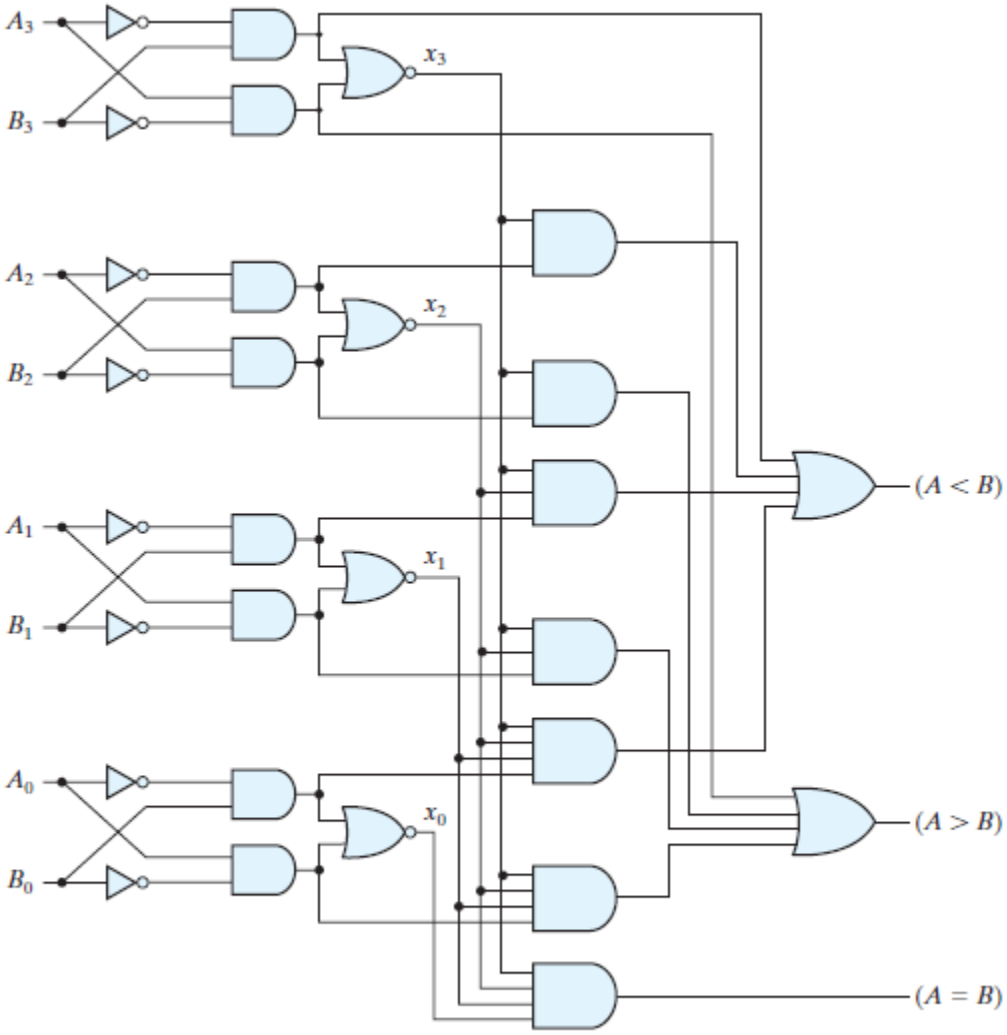
$$A = A_3\ A_2\ A_1\ A_0$$
$$B = B_3 B_2 B_1 B_0$$

Function Equation

$$(A = B) = x_3x_2x_1x_0$$

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1' + x_3x_2x_1A'n_0B_0'$$



Four-bit magnitude comparator

****

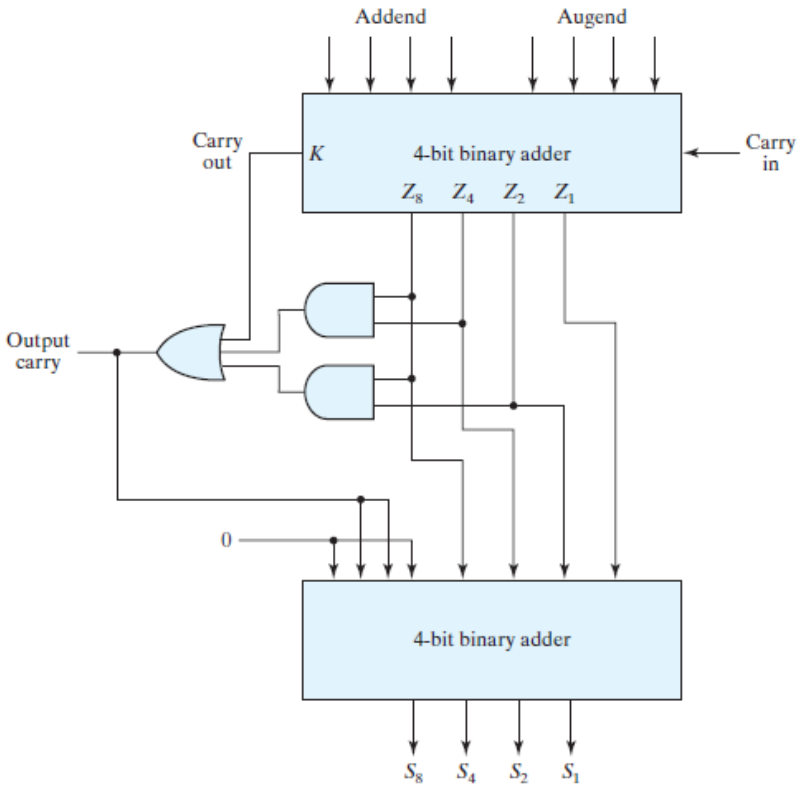
BCD Adder:

Design to perform BCD addition.(or) What is BCD adder? Design an adder to perform arithmetic addition of two decimal bits in BCD. (May -08)(Apr 2017,2018)[Nov – 2019]

- Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 9 + 9 + 1 = 19, the 1 in the sum being an input carry.
- Suppose we apply two BCD digits to a four-bit binary adder. The adder will form the sum in binary and produce a result that ranges from 0 through 19. These binary numbers are listed in Table and are labeled by symbols K, Z8, Z4, Z2, and Z1. K is the carry, and the subscripts under the letter Z represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code.

Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19



- A BCD adder that adds two BCD digits and produces a sum digit in BCD is shown in Fig. The two decimal digits, together with the input carry, are first added in the top four-bit adder to produce the binary sum.

- When the output carry is equal to 0, nothing is added to the binary sum. When it is equal to 1, binary 0110 is added to the binary sum through the bottom four-bit adder.
- The condition for a correction and an output carry can be expressed by the Boolean function

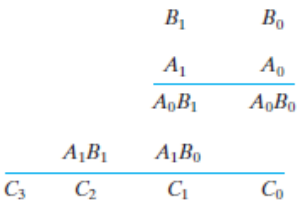
  $C = K + Z_8Z_4 + Z_8Z_2$
- The output carry generated from the bottom adder can be ignored, since it supplies information already available at the output carry terminal.
- A decimal parallel adder that adds n decimal digits needs n BCD adder stages. The output carry from one stage must be connected to the input carry of the next higher order stage.

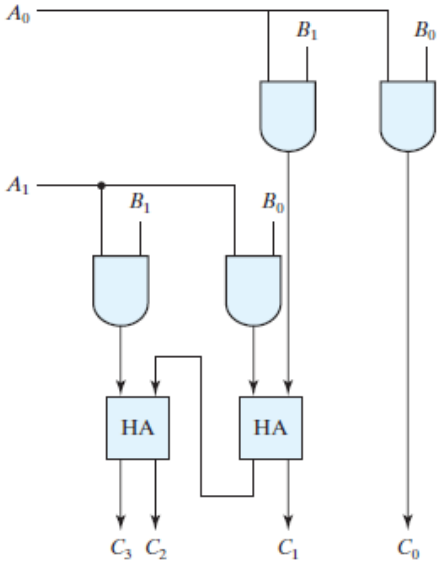
****

Binary Multiplier:

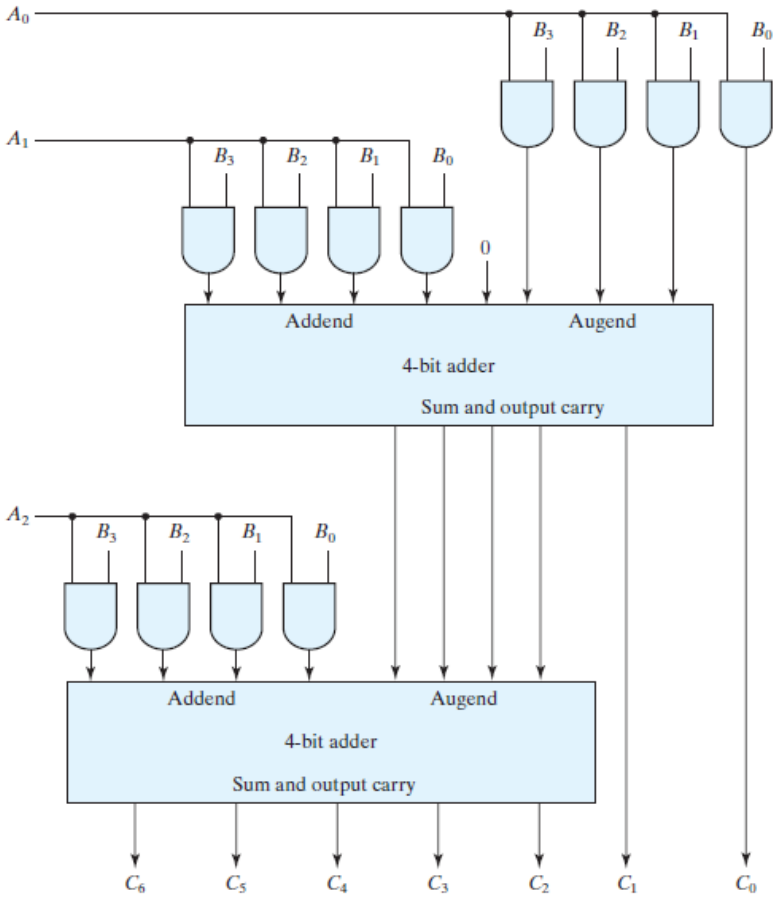
Explain about binary Multiplier.

- Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers. The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit. Each such multiplication forms a partial product.
- Successive partial products are shifted one position to the left. The final product is obtained from the sum of the partial products.





- A combinational circuit binary multiplier with more bits can be constructed in a similar fashion.
- A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier.
- The binary output in each level of AND gates is added with the partial product of the previous level to form a new partial product. The last level produces the product.



****

CODE CONVERSION

Design a binary to gray converter. ***(Nov-2009)(Nov***

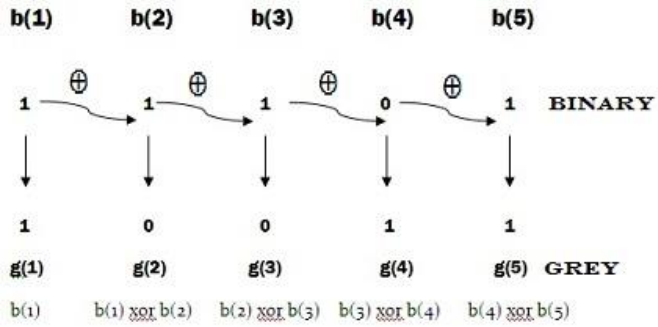
2017)

Binary to Grayconverter

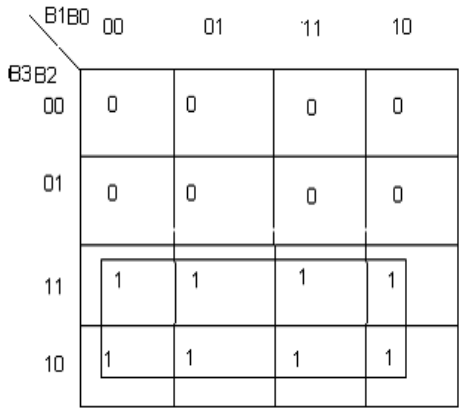
Gray code is unit distance code.
Input code: Binary [B_3 B_2 B_1 B_0]
output code: Gray [G_3 G_2 G_1 G_0]

Truth Table

B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

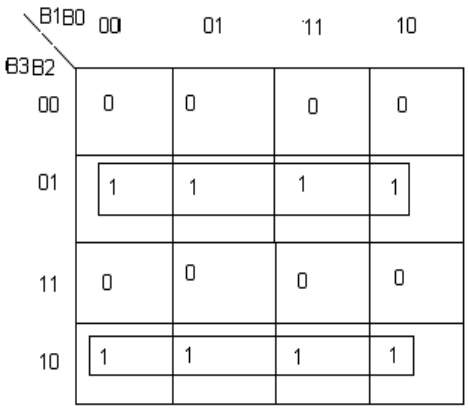


K-MAP FORG3:



G3=B3

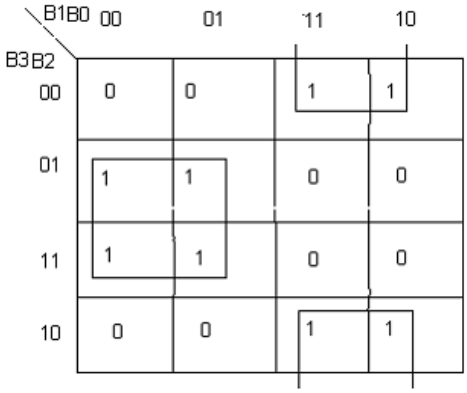
K-MAP FORG2:

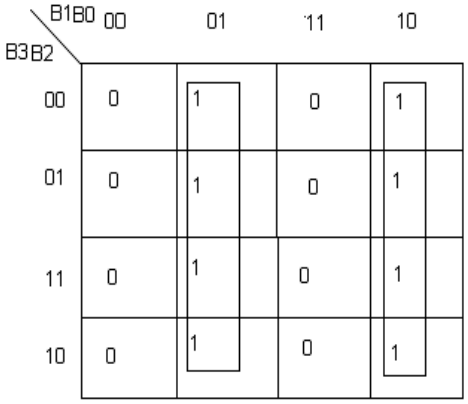


G2=B3'B2+B3B2'=B3 $\oplus$B2

K-MAP FORG1:

K-MAP FORG0:

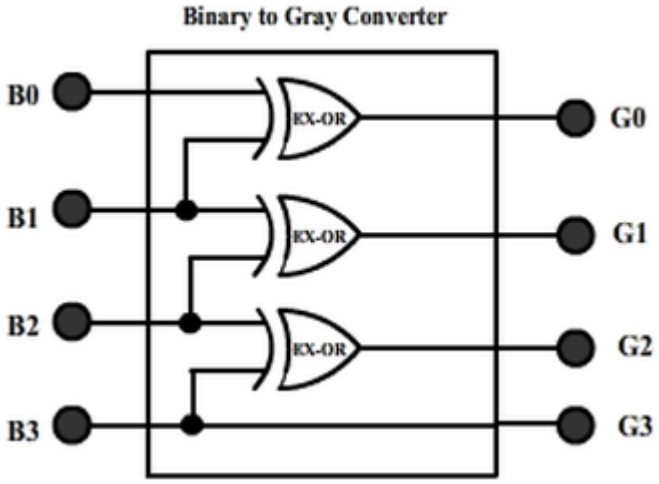




G1=B1'B2+B1B2'=B1 ⊕B2

G0=B1' B0+B1B0'=B1 ⊕B0

Logic diagram:



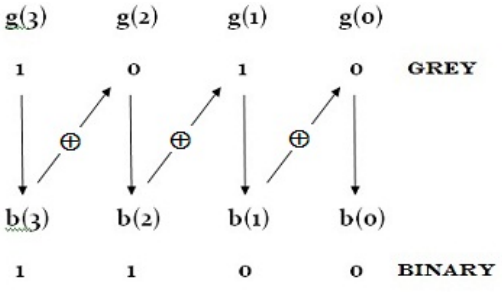
Gray to Binary converter:

Design a gray to binary converter.(OR) Design a combinational circuit that converts a four bit gray code to a four bit binary number using exclusive – OR gates. (Nov-2009) [NOV – 2019]

Gray code is unit distance code.

Input code: Gray [G_3 G_2 G_1 G_0]

output code: Binary [B_3 B_2 B_1 B_0]



i.e

$b(3) = g(3)$

$b(2) = b(3) \oplus g(3)$

$b(1) = b(2) \oplus g(1)$

$b(0) = b(1) \oplus g(0)$

Truth Table:

Gray code				Natural-binary code			
G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

K-Map:

For B_3

G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$B_3 = G_3$

For B_2

G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$B_2 = G_3'G_2 + G_3G_2' = G_3 \oplus G_2$

For B_1

G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

For B_0

G_3G_2 \ G_1G_0	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

From the above K-map,

$B_3 = G_3$

$B_2 = G_3'G_2 + G_3G_2'$

$B_2 = G_3 \oplus G_2$

$B_1 = G_3'G_2'G_1 + G_3'G_2G_1' + G_3G_2G_1 + G_3G_2'G_1'$

$= G_3' (G_2'G_1 + G_2G_1') + G_3 (G_2G_1 + G_2'G_1')$

$= G_3' (G_2 \oplus G_1) + G_3 (G_2 \oplus G_1)'$ $[x \oplus y = x'y + xy'], [(x \oplus y)' = xy + x'y']$

$B_1 = G_3 \oplus G_2 \oplus G_1$

$B_0 = G_3'G_2' G_1'G_0 + G_3'G_2 G_1 G_0' + G_3G_2G_1 G_0 + G_3G_2G_1 G_0' + G_3'G_2G_1'G_0' + G_3G_2 G_1'G_0' + G_3'G_2G_1G_0 + G_3G_2 G_1 G_0.$

$= G_3'G_2' (G_1'G_0 + G_1G_0') + G_3G_2 (G_1'G_0 + G_1G_0') + G_1'G_0' (G_3'G_2 + G_3G_2') + G_1G_0 (G_3'G_2 + G_3G_2').$

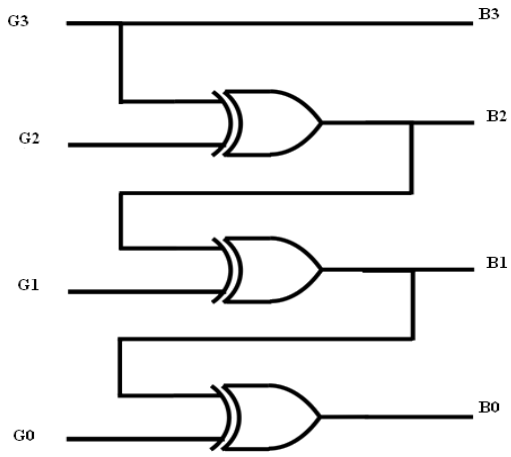
$= G_3'G_2' (G_0 \oplus G_1) + G_3G_2 (G_0 \oplus G_1) + G_1'G_0' (G_2 \oplus G_3) + G_1G_0 (G_2 \oplus G_3).$

$= G_0 \oplus G_1 (G_3'G_2' + G_3G_2) + G_2 \oplus G_3 (G_1'G_0' + G_1G_0)$

$= (G_0 \oplus G_1) (G_2 \oplus G_3)' + (G_2 \oplus G_3) (G_0 \oplus G_1)$ $[x \oplus y = x'y + xy']$

$B_0 = (G_0 \oplus G_1) \oplus (G_2 \oplus G_3)$.

Logic Diagram:



BCD to Excess -3 converter:

Design a combinational circuits to convert binary coded decimal number into an excess-3 code.

- Excess-3 code is modified form of BCD code. ***(Nov-06,09,10, May-08,10)***
- Excess -3 code is derived from BCD code by adding 3to each coded number.

Truth table:

Decimal	BCD code				Excess-3 code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

K-Map:

For E_3

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	x	x	x	x
10	1	1	x	x

$E_3 = B_3 + B_2 (B_0 + B_1)$

For E_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	x	x	x	x
10	0	1	x	x

$E_2 = B_2B_1'B_0' + B_2' (B_0 + B_1)$

For E_1

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	x	x	x	x
10	1	0	x	x

$E_1 = B_1'B_0' + B_1B_0$

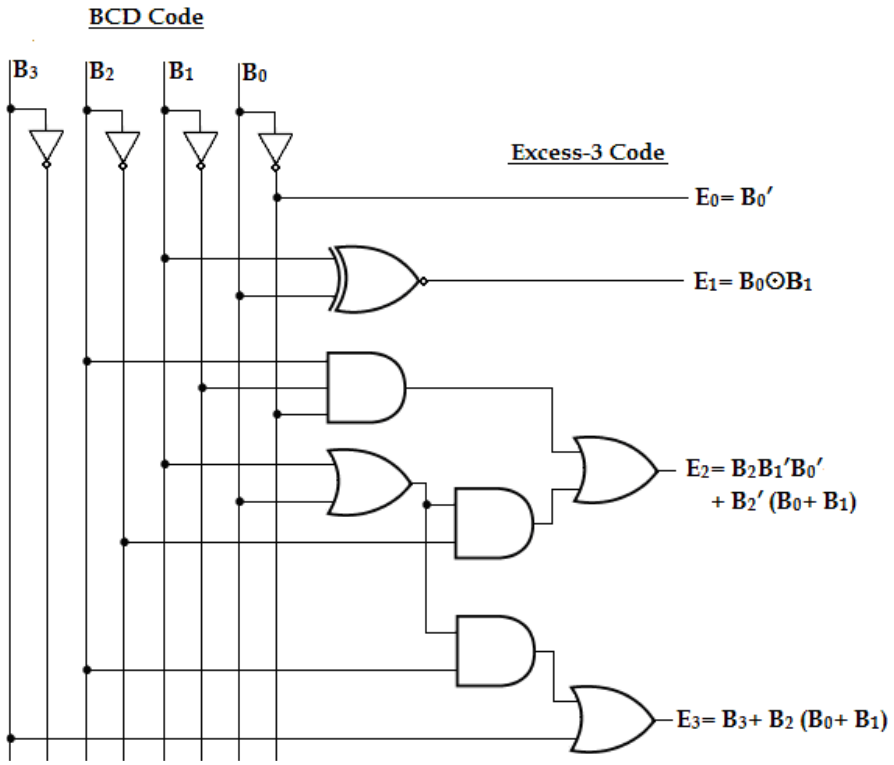
$= B_1 \odot B_0$

For E_0

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	x	x	x	x
10	1	0	x	x

$E_0 = B_0'$

Logic Diagram



Excess -3 to BCD converter:

Design a combinational circuit to convert Excess-3 to BCD code. (May 2007)

Truth table:

Decimal	Excess-3 code				BCD code			
	E_3	E_2	E_1	E_0	B_3	B_2	B_1	B_0
3	0	0	1	1	0	0	0	0
4	0	1	0	0	0	0	0	1
5	0	1	0	1	0	0	1	0
6	0	1	1	0	0	0	1	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	0	1	0	1
9	1	0	0	1	0	1	1	0
10	1	0	1	0	0	1	1	1
11	1	0	1	1	1	0	0	0
12	1	1	0	0	1	0	0	1

K-map simplification

For B_0

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	1	0	0	1
11	1	X	X	X
10	1	0	0	1

$B_0 = \overline{E}_0$

For B_1

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	1	0	1
11	0	X	X	X
10	0	1	0	1

$B_1 = \overline{E}_1E_0 + E_1\overline{E}_0$
$= E_1 \oplus E_0$

For B_2

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	0	1	0
11	0	X	X	X
10	1	1	0	1

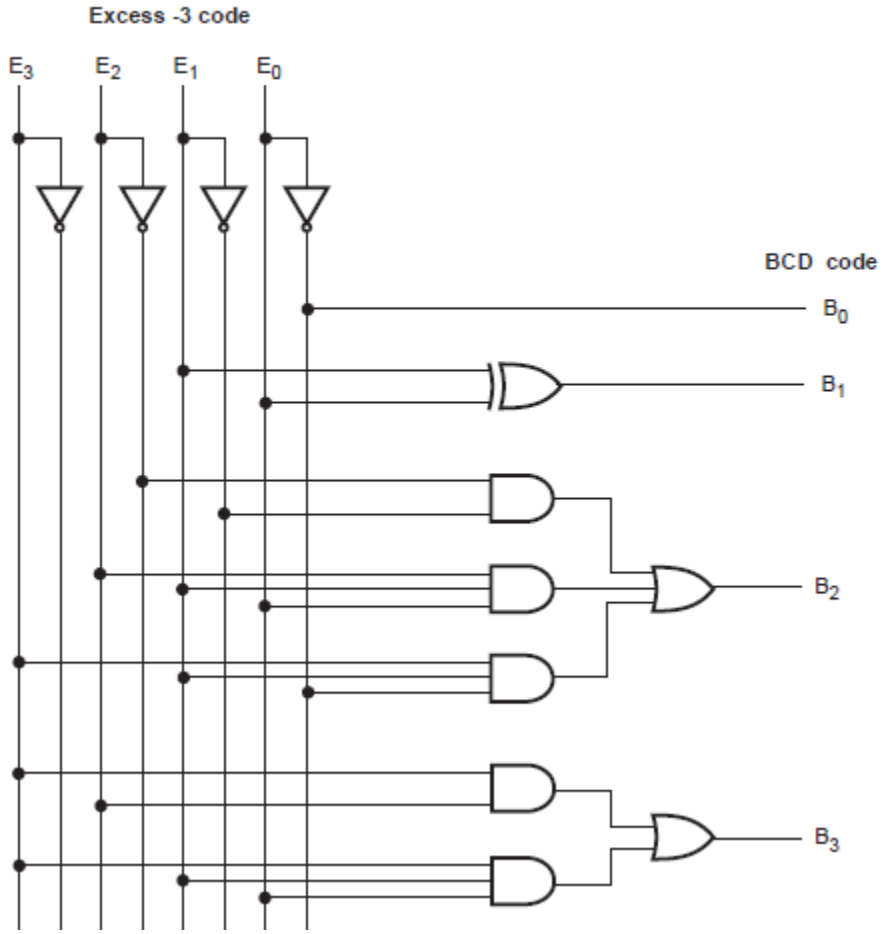
$B_2 = \overline{E}_2\overline{E}_1 + E_2E_1E_0 + E_3E_1\overline{E}_0$

For B_3

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	0	0	0
11	1	X	X	X
10	0	0	1	0

$B_3 = E_3E_2 + E_3E_1E_0$

Logic diagram



Design Binary to BCD converter.

Truth table:

Decimal	Binary Code				BCD Code				
	D	C	B	A	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

K-map:

For B_0

DC \ BA	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$B_0 = A$

For B_1

DC \ BA	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	0	0
10	0	0	0	0

$B_1 = DCB' + D'B$

For B_2

DC \ BA	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	1	1
10	0	0	0	0

$B_2 = D'C + CB$

For B_3

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

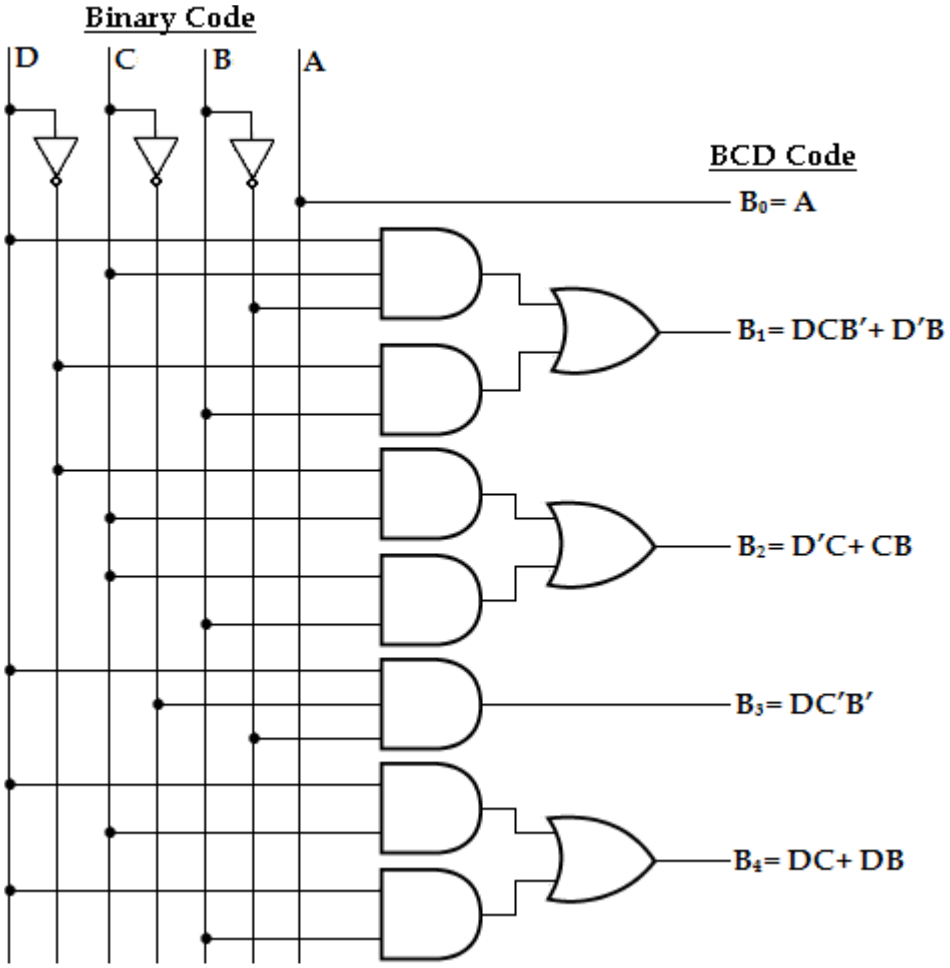
$B_3 = DC'B'$

For B_4

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$B_4 = DC + DB$

Logic diagram:



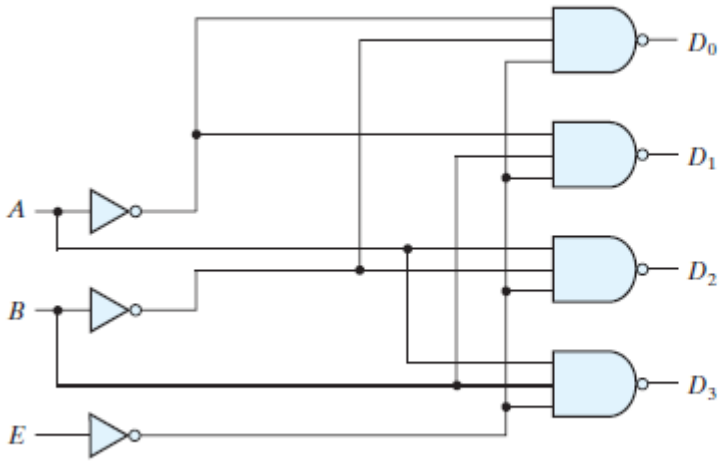
****

Decoder:

Explain about decoders with necessary diagrams. ***(Apr 2018)(Nov 2018)***

- A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n -bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.
- The purpose of a decoder is to generate the 2^n (or fewer) minterms of *n* input variables, shown below for two input variables.

2 to 4 decoder:



(a) Logic diagram

E	*A*	*B*	D_0	D_1	D_2	D_3
1	*X*	*X*	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

(b) Truth table

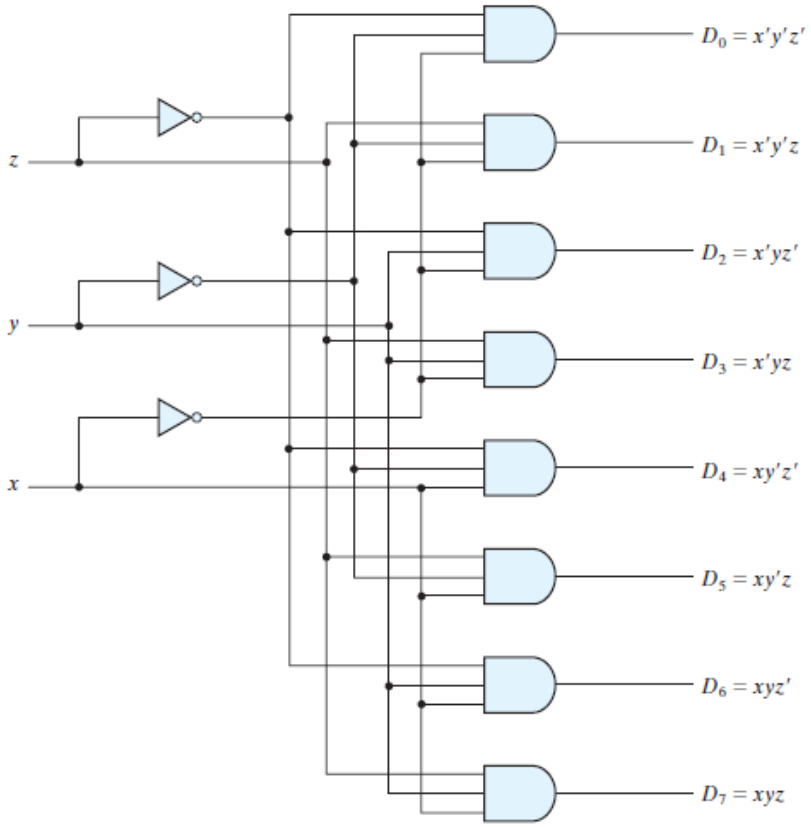
3 to 8 Decoder:

Design 3 to 8 line decoder with necessary diagram. ***May -10)***

Truth table:

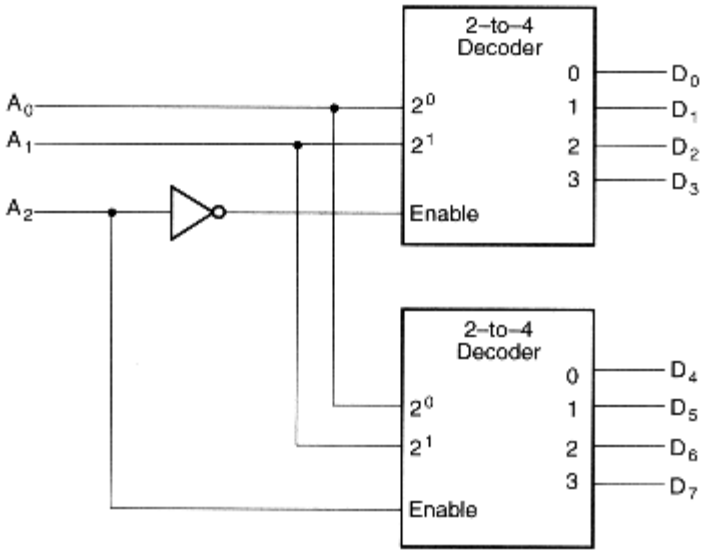
Inputs			Outputs							
x	*y*	*z*	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Logic diagram:



Design for 3 to 8 decoder with 2 to 4 decoder:

- Not that the two to four decoder design shown earlier, with its *enable* inputs can be used to build a three to eight decoder as follows.

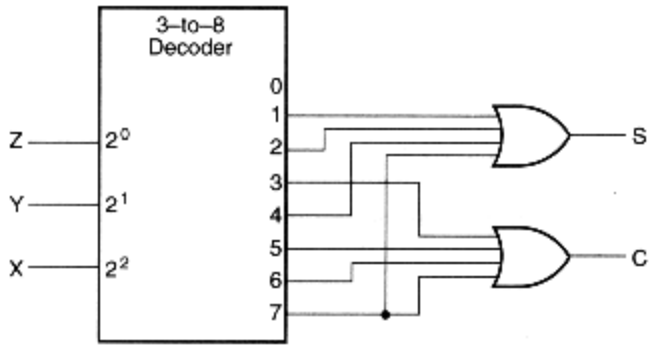


Implementation of Boolean function using decoder:

- Since the three to eight decoder provides all the minterms of three variables, the realisation of a function in terms of the sum of products can be achieved using a decoder and OR gates as follows.

Example*: Implement full adder using decoder.*

Sum is given by $\sum m(1, 2, 4, 7)$ while *Carry* is given by $\sum m(3, 5, 6, 7)$ as given by the minterms each of the OR gates are connected to.

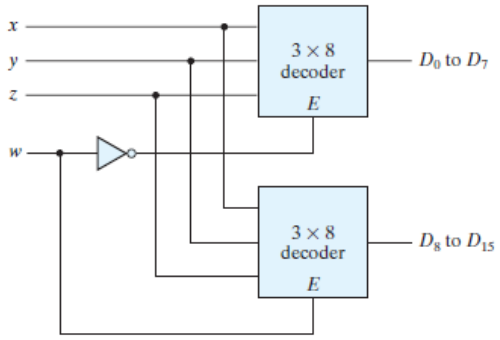


Solution :

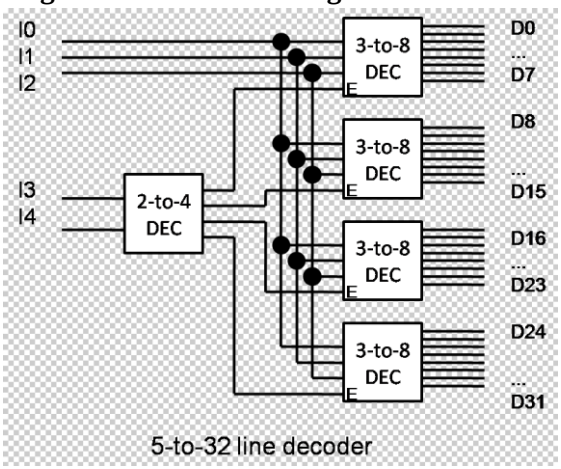
Step 1 : Truth table

Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Design for 4 to 16 decoder using 3 to 8 decoder:



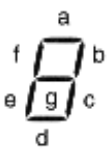
Design 5 to 32 decoder using 3 to 8 and 2 to 4 decoder:



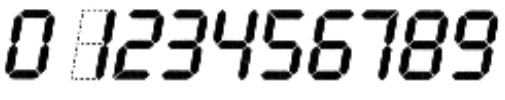
****

BCD to seven segment decoder

Design a BCD to seven segment code converter. ***(May-06,10, Nov- 09)***



(a) Segment designation



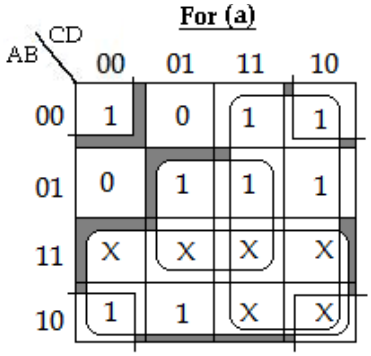
(b) Numeric designation for display

Truth table:

	BCD code				7-Segment code						
Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

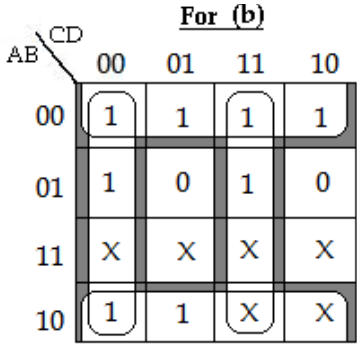
K-Map:

For (a)



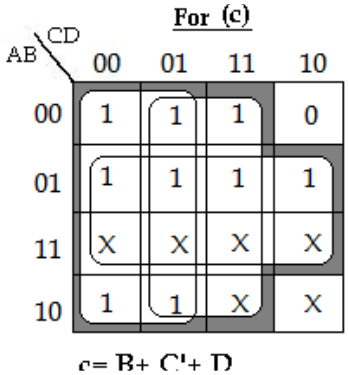
a= A+ C+ BD+ B'D'

For (b)



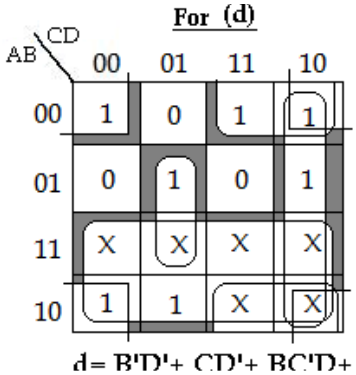
b= B'+ C'D'+ CD

For (c)



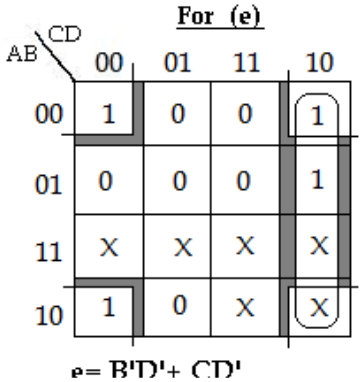
c= B+ C'+ D

For (d)



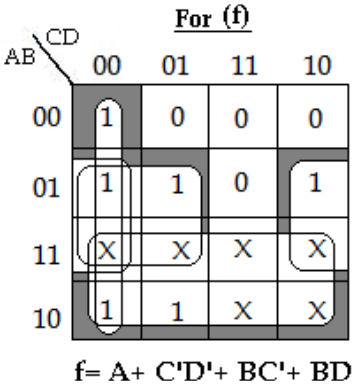
d= B'D'+ CD'+ BC'D+ B'C+ A

For (e)



e= B'D'+ CD'

For (f)



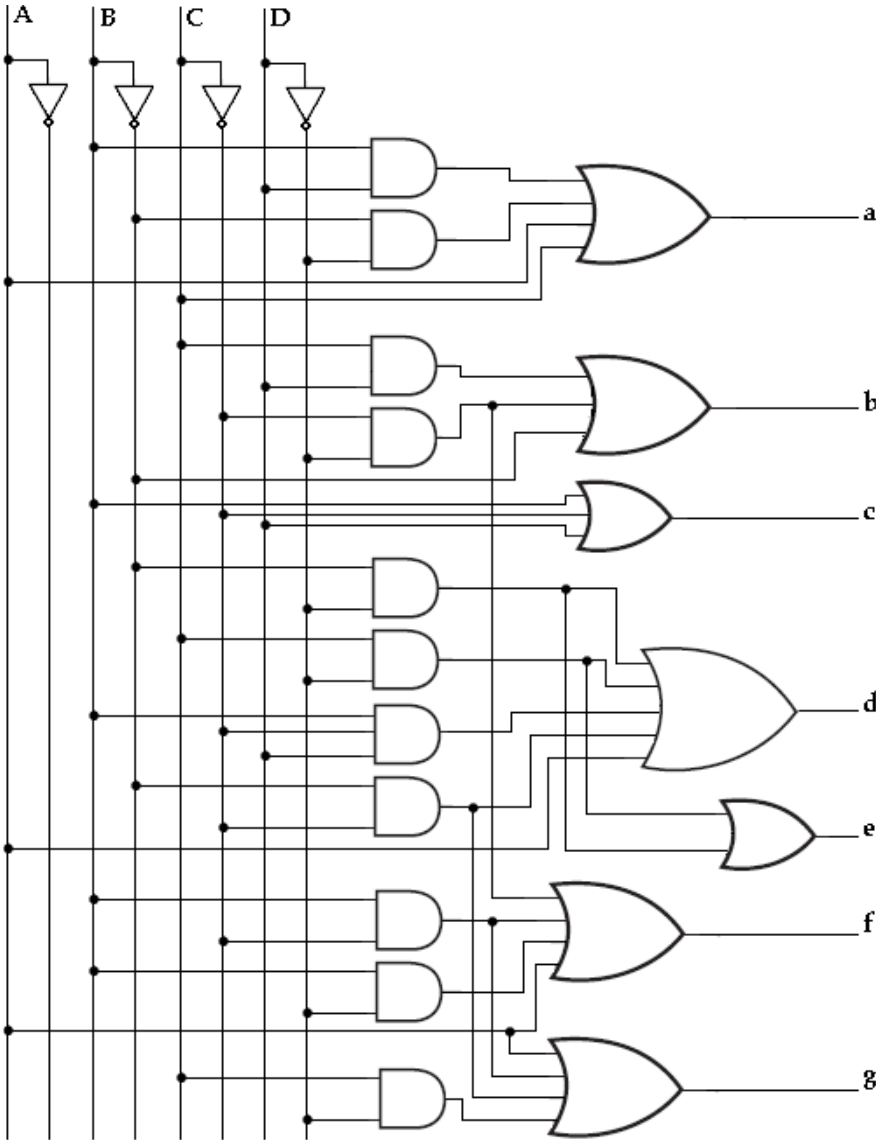
f= A+ C'D'+ BC'+ BD'

For (g)

AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

g= A+ BC'+ B'C+ CD'

Logic Diagram:



- The specification above requires that the output be zeroes (none of the segments are lighted up) when the input is not a BCD digit.
- In practical implementations, this may defer to allow representation of hexadecimal digits using the seven segments.

****

Encoder:

Explain about encoders. (Nov 2018)

- An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The output lines, as an aggregate, generate the binary code corresponding to the input value.

Octal to Binary Encoder:

- The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1, 3, 5, or 7.
- Output y is 1 for octal digits 2, 3, 6, or 7, and output x is 1 for digits 4, 5, 6, or 7. These conditions can be expressed by the following Boolean output functions:

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

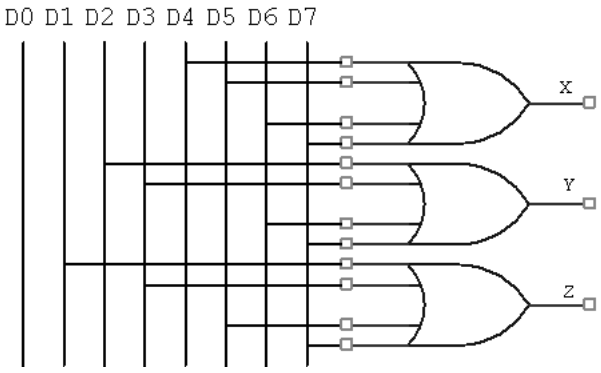
The encoder can be implemented with three OR gates.

Truth table:

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; but this output is the same as when D0 is equal to 1. The discrepancy can be resolved by providing one more output to indicate whether at least one input is equal to 1.

Logic Diagram:



****

Priority Encoder:

Design a priority encoder with logic diagram.(or) Explain the logic diagram of a 4 – input priority encoder. (Apr – 2019)

A priority encoder is an encoder circuit that includes the priority function. The operationof the priority encoder is such that if two or more inputs are equal to 1 at the same time,the input having the highest priority will take precedence.

Truth table:

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Modified Truth table:

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	x	x	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
1	1	0	0			
0	0	1	0	1	0	1
0	1	1	0			
1	0	1	0			
1	1	1	0			
0	0	0	1	1	1	1
0	0	1	1			
0	1	0	1			
0	1	1	1			
1	0	0	1			
1	0	1	1			
1	1	0	1			
1	1	1	1			

K-Map:

For x

D_0D_1 \ D_2D_3	00	01	11	10
00	x	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

$x = D_2 + D_3$

For y

D_0D_1 \ D_2D_3	00	01	11	10
00	x	1	1	0
01	1	1	1	0
11	1	1	1	0
10	0	1	1	0

$y = D_3 + D_1D_2$

For V

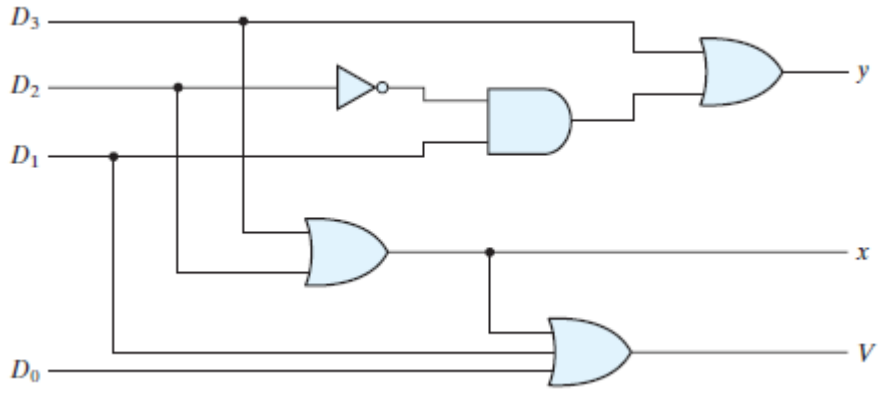
D_0D_1 \ D_2D_3	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$V = D_0 + D_1 + D_2 + D_3$

Logic Equations:

$$x = D_2 + D_3$$
$$y = D_3 + D_1 D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$

Logic diagram:



****

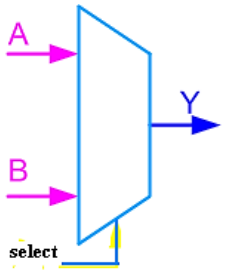
Multiplexer: (MUX)

Design a 2:1 and 4:1 multiplexer.

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

2 to 1 MUX:

A 2 to 1 line multiplexer is shown in figure below, each 2 input lines A to B is applied to one input of an AND gate. Selection lines S are decoded to select a particular AND gate. The truth table for the 2:1 mux is given in the table below.

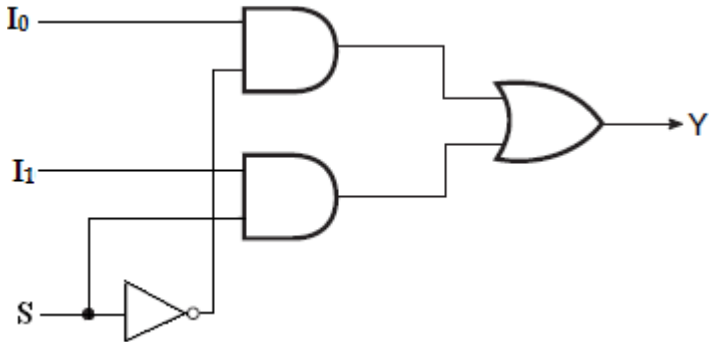


- To derive the gate level implementation of 2:1 mux we need to have truth table as shown in figure. And once we have the truth table, we can draw the K-map as shown in figure for all the cases when Y is equal to '1'.

Truth table:

S	Y
0	I_0
1	I_1

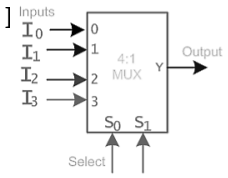
Logic Diagram:

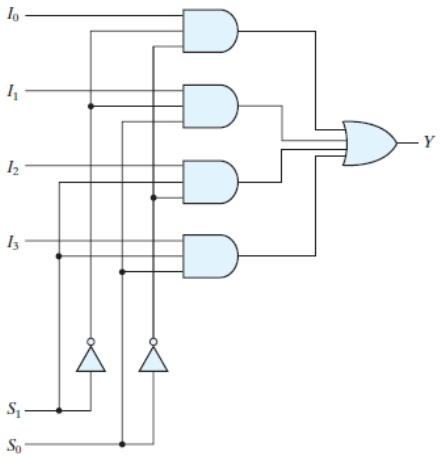


4 to 1 MUX:

- A 4 to 1 line multiplexer is shown in figure below, each of 4 input lines I0 to I3 is applied to one input of an AND gate.

- Selection lines S0 and S1 are decoded to select a particular AND gate.
- The truth table for the 4:1 mux is given in the table below.





Truth Table:

SELECT INPUT		OUTPUT
S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

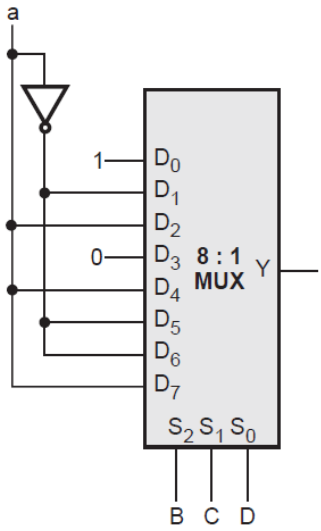
Problems :

Example: Implement the Boolean expression using MUX

F(A,B,C,D) = ∑m(0,1,5,6,8,10,12,15) **(Apr 2017, Nov 2017)**

Solution : Implementation table :

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{a}$	(0)	(1)	2	3	4	(5)	(6)	7
a	(8)	9	(10)	11	(12)	13	14	(15)
	1	$\bar{a}$	a	0	a	$\bar{a}$	$\bar{a}$	a



Example: Implement the boolean function using Multiplexer. [NOV – 2019]

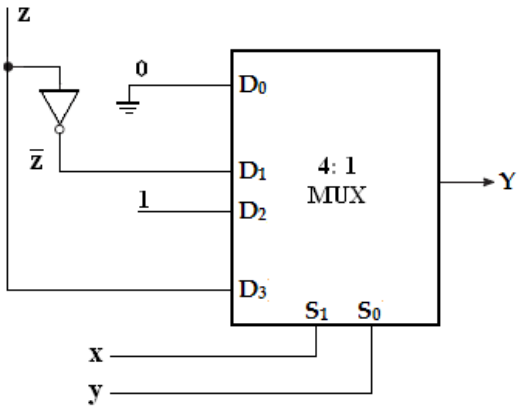
F (x, y, z) = Σm (1, 2, 6, 7)

Solution:

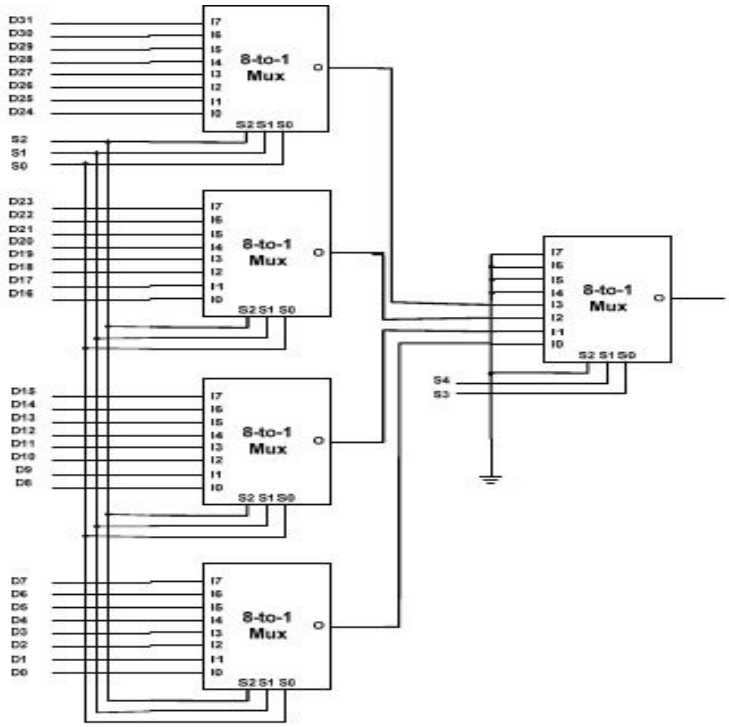
Implementation table:

	D_0	D_1	D_2	D_3
$\bar{z}$	0	(1)	(2)	3
z	4	5	(6)	(7)
	0	$\bar{z}$	1	z

Multiplexer Implementation:



Example: 32:1 Multiplexer using 8:1 Mux (Nov 2018) (Apr – 2019)

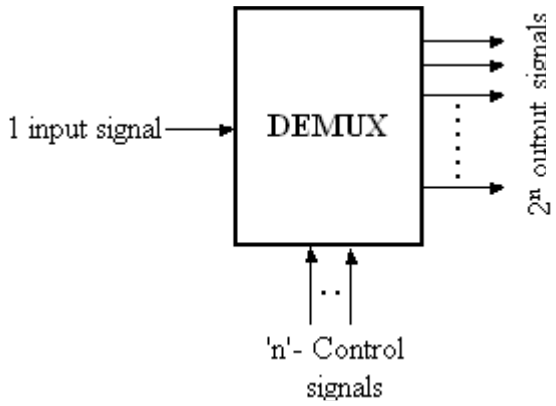


DEMULTIPLEXERS:

Explain about demultiplexers.

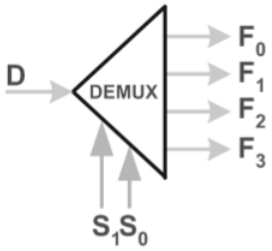
- The de-multiplexer performs the inverse function of a multiplexer, that is it receives information on one line and transmits its onto one of 2n possible output lines.

❖ The selection is by n input select lines. Example: 1-to-4 De-multiplexer

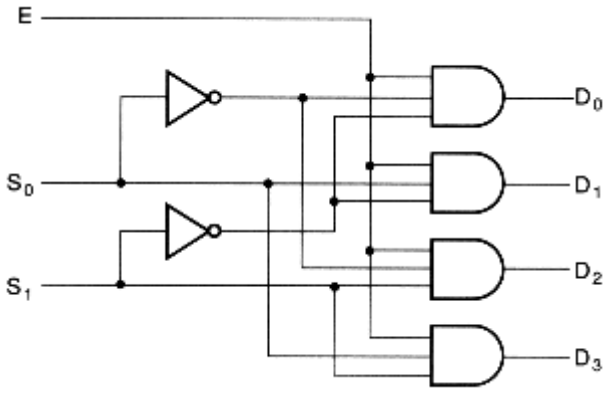


Truth table

S1	S0	F0	F1	F2	F3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D



Logic Diagram:



Truth Table:

INPUT				OUTPUT			
E	D	S0	S1	Y0	Y1	Y2	Y3
1	1	0	0	1	0	0	0
1	1	0	1	0	1	0	0
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

Example:

1. **Implement full adder using De-multiplexer.**

Solution :

Step 1 : Truth table

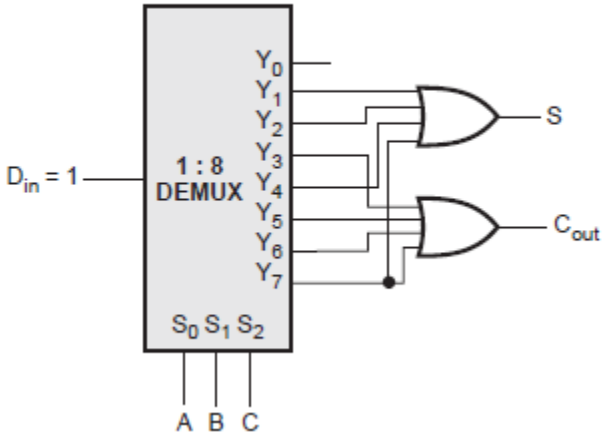
Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step 2 : For full adder

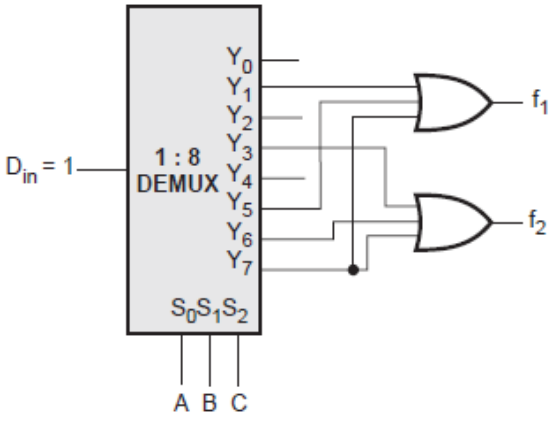
Carry = $C_{out} = \sum m\ (3, 5, 6, 7)$

and Sum = $S = \sum m\ (1, 2, 4, 7)$

Step 3 : When $D_{in} = 1$, the demultiplexer gives minterms at the output.



2. Implement the following functions using de-multiplexer.
f1 (A,B,C) = ∑m(1,5,7), f2 (A,B,C) = ∑m(3,6,7)
Solution:



****

Parity Checker / Generator:

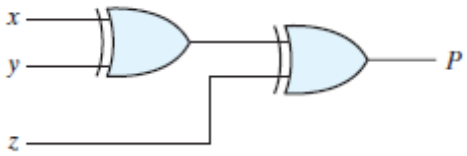
- A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The message, including the parity bit, is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted.
- The circuit that generates the parity bit in the transmitter is called a *parity generator.* The circuit that checks the parity in the receiver is called a *parity checker.*
- In even parity system, the parity bit is '0' if there are even number of 1s in the data and the parity bit is '1' if there are odd number of 1s in the data.
- In odd parity system, the parity bit is '1' if there are even number of 1s in the data and the parity bit is '0' if there are odd number of 1s in the data.

3-bit Even Parity generator:
Truth Table:

Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$P = x \oplus y \oplus z$$

Logic Diagram:

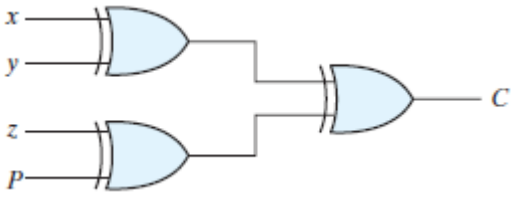


4-bit Even parity checker:
Truth Table:

Four Bits Received				Parity Error Check
x	y	z	P	C
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

$$C = x \oplus y \oplus z \oplus P$$

Logic Diagram:



INTRODUCTION TO HDL

- In electronics, a **hardware description language or HDL** is any language from a class of computer languages and/or programming languages for formal description of digital logic and electronic circuits.
- HDLs are used to write executable specifications of some piece of hardware.
- A simulation program, designed to implement the underlying semantics of the language statements, coupled with simulating the progress of time, provides the hardware designer with the ability to model a piece of hardware before it is created physically.
- *Logic synthesis* is the process of deriving a list of components and their interconnection (called net list) from the model of a digital system.
- *Logic Simulation* is the representation of the structure and behavior of a digital logic synthesis through the use of a computer.
- The standard HDLs that supported by IEEE.
  - ✓ VHDL (very High Speed Integrated Circuit HDL)
  - ✓ Verilog HDL

HDL MODELS OF COMBINATIONAL CIRCUITS

The Verilog HDL model of a combinational circuit can be described in any one of the following modeling styles,

✓Gate level modeling-using instantiations of predefined and user defined primitive gates.
✓Dataflow modeling using continuous assignment with the keyword **assign**.
✓Behavioral modeling using procedural assignment statements with the keyword **always**.

Gate level modeling

In this type, a circuit is specified by its logic gates and their interconnections. Gate level modeling provides a textual description of a schematic diagram. The verilog HDL includes 12basic gates as predefined primitives. They are and, nand, or, nor, xor, xnor, not &buf.

HDL

```
// Gate-level description of two-to-four-line decoder
// Refer to Fig. 4.19 with symbol E replaced by enable, for clarity.

module decoder_2x4_gates (D, A, B, enable);
 output      [0: 3]      D;
 input                   A, B;
 input                   enable;
 wire                    A_not, B_not, enable_not;

 not
   G1  (A_not, A),
   G2  (B_not, B),
   G3  (enable_not, enable);
 nand
   G4  (D[0], A_not, B_not, enable_not),
   G5  (D[1], A_not, B, enable_not),
   G6  (D[2], A, B_not, enable_not),
   G7  (D[3], A, B, enable_not);
endmodule
```

Data flow modeling

Data flow modeling of combinational logic uses a number of operators that act on operands to produce desired results. Verilog HDL provides about 30 different operators. Data flow modeling uses continuous assignments and the keyword **assign**. A continuous assignment is a statement that assigns a value to a net. The data type family **net** is used to represent a physical connection between circuit elements.

HDL for2-to-4 line decoder

Symbol	Operation
+	binary addition
−	binary subtraction
&	bitwise AND
|	bitwise OR
^	bitwise XOR
~	bitwise NOT
==	equality
>	greater than
<	less than
{ }	concatenation
? :	conditional

```
module decoder_2x4_df (
  output      [0: 3]      D,
  input                   A, B,
                          enable
);

  assign      D[0] = ~(~A & ~B & ~enable),
              D[1] = ~(~A & B & ~enable),
              D[2] = ~(A & ~B & ~enable),
              D[3] = ~(A & B & ~enable);
endmodule
```

Behavioral modeling

- Behavioral modeling represents digital circuits at a functional and algorithmic level. It is used mostly to describe sequential circuits, but can also be used to describe combinational circuits.
- Behavioral descriptions use the keyword **always,** followed by an optional event control expression and a list of procedural assignment statements.

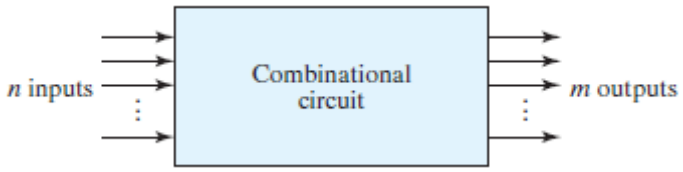
```
// Behavioral description of two-to-one-line multiplexer

module mux_2x1_beh (m_out, A, B, select);
  output      m_out;
  input       A, B, select;
  reg         m_out;

  always      @(A or B or select)
    if (select == 1) m_out = A;
    else m_out = B;
  endmodule
```

TWO MARK QUESTIONS & ANSWERS

1) Define combinational logic. (May 2008, 2016)

A combinational circuit consists of logic gates whose outputs at any time are determined from only the

present combination of inputs. A combinational circuit performs an operation that canbe specified logically by a set of Boolean functions.



2) What are sequential circuits?

Sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements. Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.

3) Write the design procedure for combinational circuits?

The procedure involves the following steps:

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

4) What is Half adder?

A half-adder is an arithmetic circuit block that can be used to add two bits and produce two outputs SUM and CARRY.

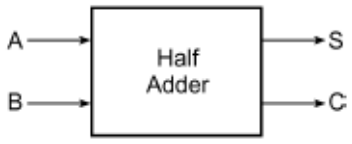
The Boolean expressions for the SUM and CARRY outputs are given by the equations

$$\text{SUM } S = A.\overline{B} + \overline{A}.B$$

$$\text{CARRY } C = A.B$$

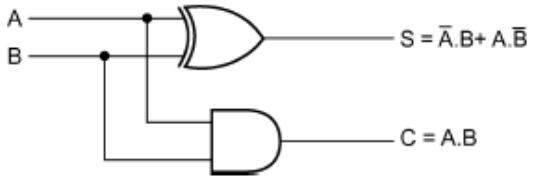
Truth Table:

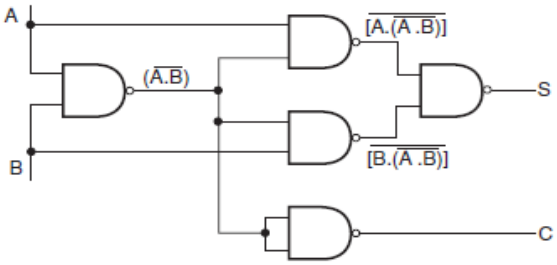
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



5) Draw the logic diagram of half adder using NAND gate. (May 2006,13)

Logic Diagram: **Half adder using NAND gate:**





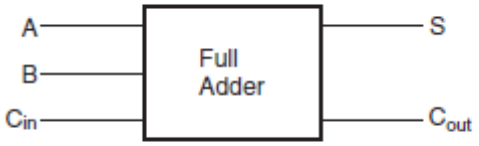
6) What is Full adder? Draw the truth table of full adder. (Apr 2018)

A Full-adder is an arithmetic circuit block that can be used to add three bits and produce two outputs SUM and CARRY.

The Boolean expressions for the SUM and CARRY outputs are given by the equations

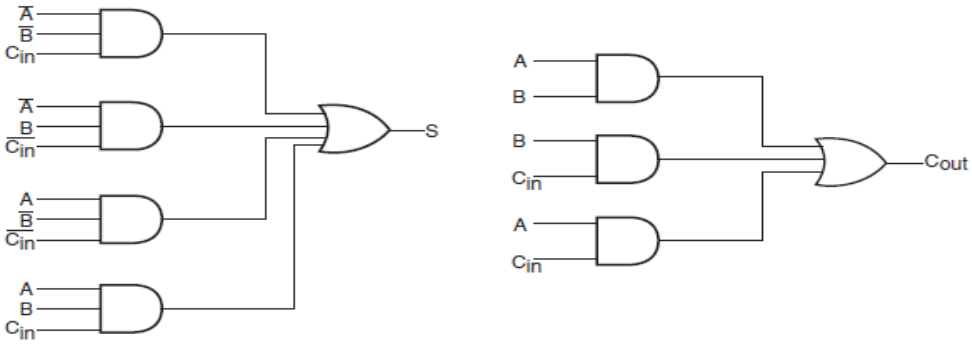
$$S = \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C}_{in} + A.\overline{B}.\overline{C}_{in} + A.B.C_{in}$$

$$C_{out} = B.C_{in} + A.B + A.C_{in}$$



A	B	C_{in}	SUM (S)	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

7) Draw the Logic diagram of full adder.

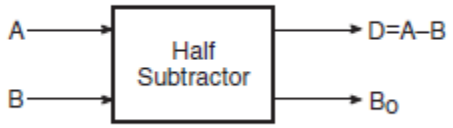


8) What is Half subtractor? (May 2005)

A half-subtractor is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction. The Boolean expression for difference and borrow is:

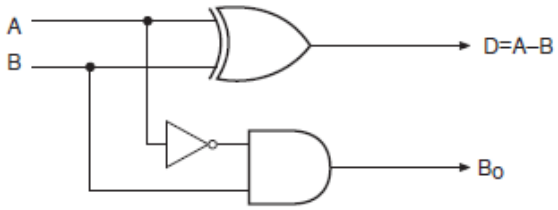
$$D = \overline{A}.B + A.\overline{B}$$

$$B_0 = \overline{A}.B$$

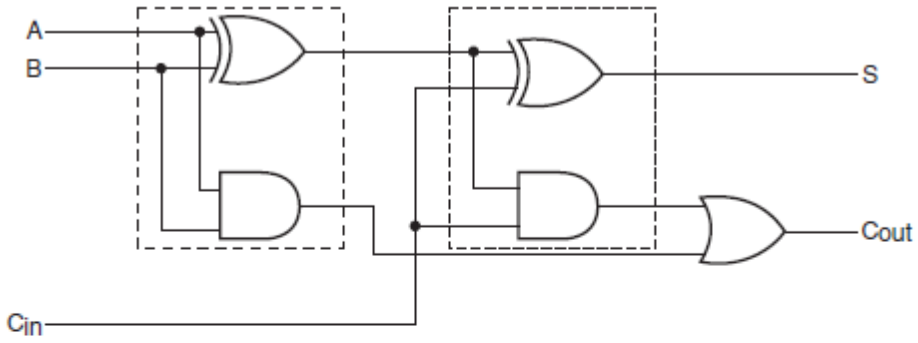


A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

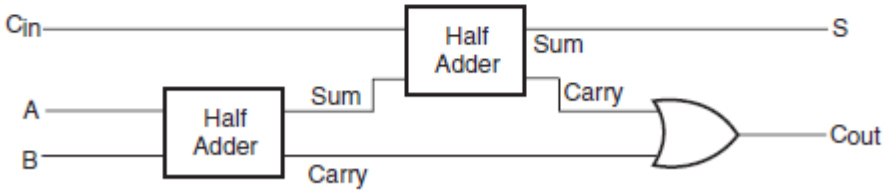
Logic diagram:



9) Draw Full adder using Two half adder. (Apr – 2019)



(a)



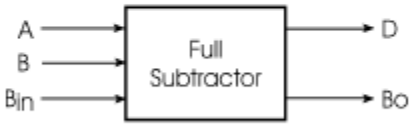
(b)

10) What is Full subtractor? Write the truth table of full subtractor. (Nov 2017)

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not. As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as Bin . There are two outputs, namely the DIFFERENCE output D and the BORROW output Bo. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit. The Boolean expression for difference and barrow is:

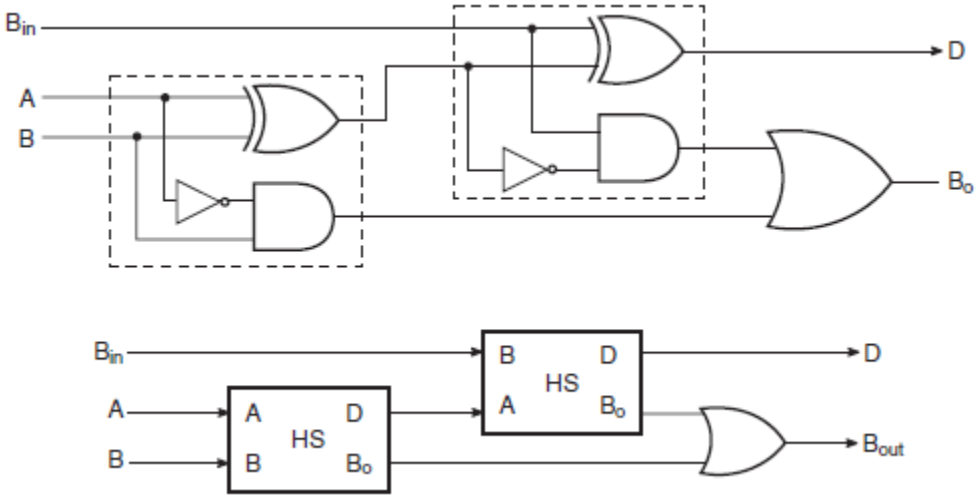
$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$

$$B_o = \overline{A}.B + \overline{A}.B_{in} + B.B_{in}$$



Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_O)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

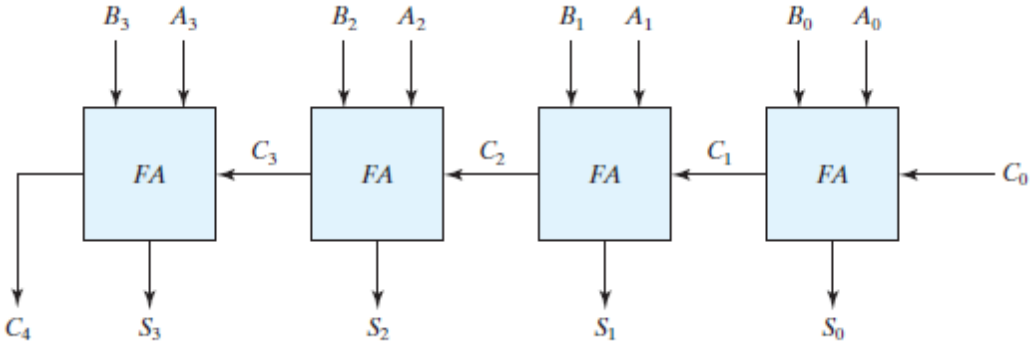
11) Draw Full subtractor using two half subtractor.



12) What is Parallel Binary Adder (Ripple Carry Adder)?

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.It can beconstructed with full adders connected in cascade, with the output carryfrom each full adder connected to the input carry of the next full adder in the chain.

13) Draw the logic diagram for four bit binary parallel adder.



14) What is 1's complement of a number?

The 1's complement of a binary number is formed bychanging 1 to 0 and 0 to 1.

Example:

1. The 1's complement of 1011000 is 0100111.
2. The 1's complement of 0101101 is 1010010.

15) What is 2's complement of a number?

The 2's complement of a binary number is formed by adding 1 with 1's complement of a binary number.

Example:

1) The 2's complement of 1101100 is 0010100

2) The 2's complement of 0110111 is 1001001

16) How Subtraction of binary numbers perform using 2's complement addition?

- ✓ The subtraction of unsigned binary number can be done by means of complements.
- ✓ Subtraction of A-B can be done by taking 2's complement of B and adding it to A.
- ✓ Check the resulting number. If carry present, the number is positive and remove the carry.
- ✓ If no carry present, the resulting number is negative, take the 2's complement of result and put negative sign.

17) Given the two binary numbers X = 1010100 and Y = 1000011, perform the subtraction

(a) X - Y and (b) Y - X by using 2's complements.

Solution:

(c) X = 1010100

2's complement of Y = + 0111101

Sum= 10010001

Discard end carry. *Answer*: $X - Y$ = 0010001

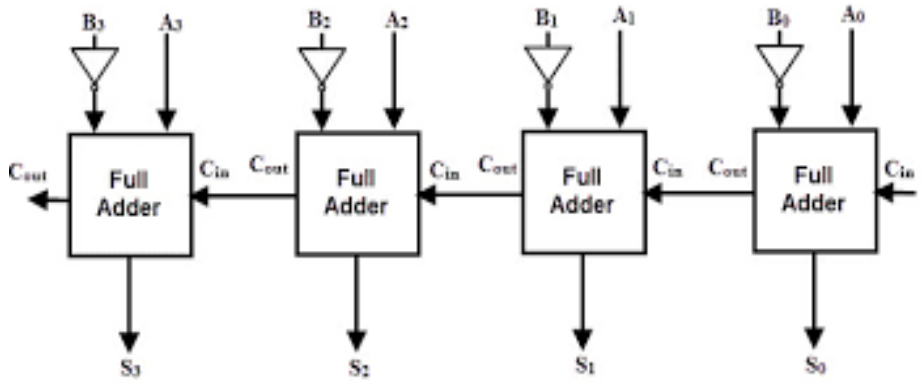
(d) Y = 1000011

2's complement of X= + 0101100

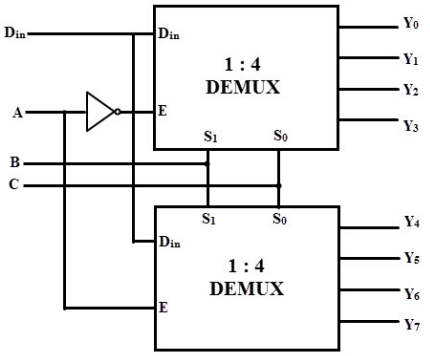
Sum= 1101111

There is no end carry. Therefore, the answer is $Y - X$ = -(2's complement of 1101111) =-0010001.

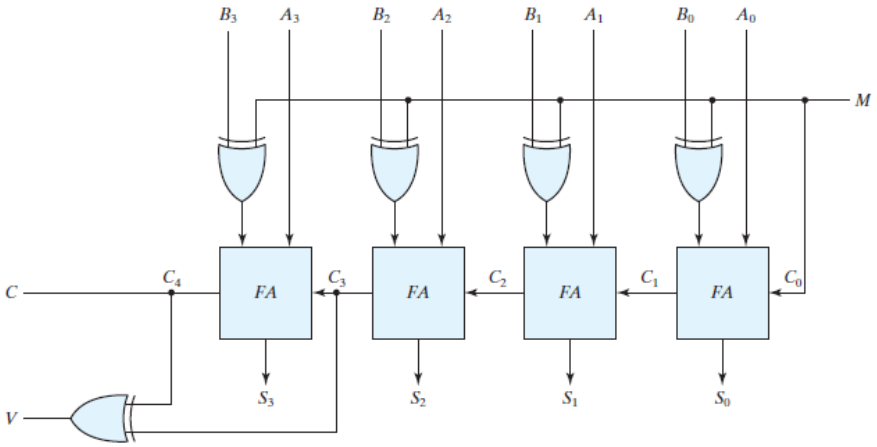
18) Draw the logic diagram of Parallel Binary Subtractor.



19) Draw 1:8 Demux using two 1:4 demux. (Nov 2018)



20) Draw the logic diagram of 2's complement adder/subtractor. (May 2013)



The mode input *M* controls the operation. When *M* = 0, the circuit is an adder, and when *M* = 1, the circuit becomes a subtractor.

21) What is Magnitude Comparator? [NOV – 2019]

The comparison of two numbers is an operation that determines whether one number is greater than, less than, or equal to the other number. A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes.

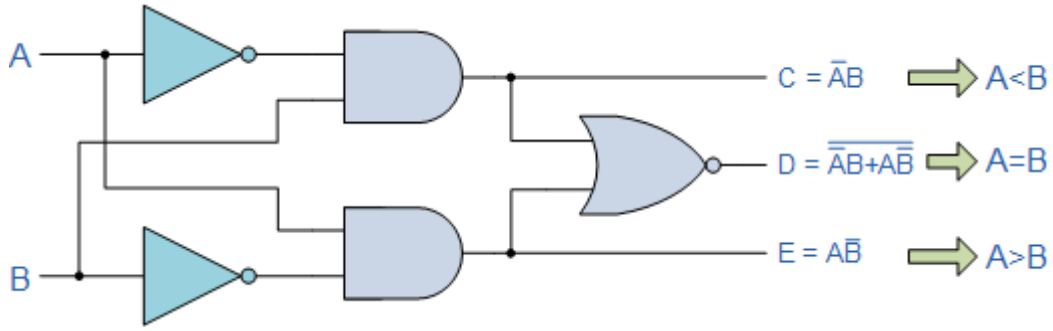
The outcome of the comparison is specified by three binary variables that indicate whether A > B, A = B, or A < B.

22) Design a 1-bit Magnitude Comparator.

Truth table:

Inputs		Outputs		
B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Logic Circuits:



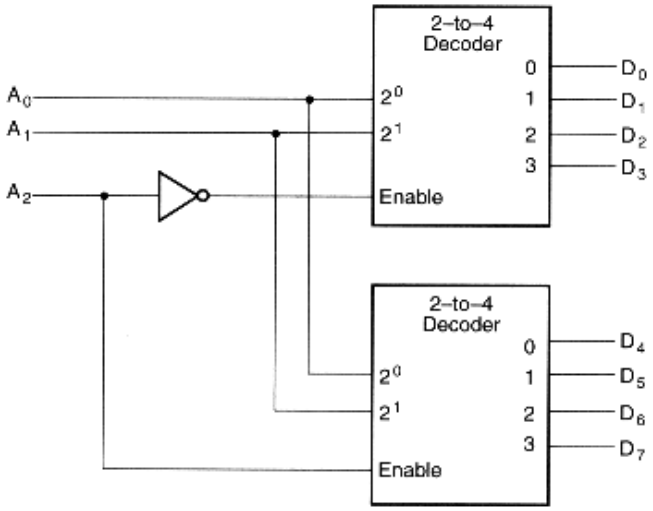
23) What is Decoder? What are binary decoders? (Nov 2017)

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n -bit coded information hasunused combinations, the decoder may have fewer than 2^n outputs.

The purpose of a decoder is to generate the 2^n (or fewer) minterms of *n* input variables, shown below for two input variables.

24) Design a 3 to 8 decoder with 2 to 4 decoder.

Not that the two to four decoder design shown earlier, with its *enable* inputs can be used to build a three to eight decoder as follows.



25) What is Encoder? (May 2012)

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The output lines, as an aggregate, generate the binary code corresponding to the input value.

26) What is Priority Encoder? (Apr 2017)

A priority encoder is an encoder circuit that includes the priority function. The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

27) Define Multiplexer (MUX) (or) Data Selector. (Dec 2006, May 2011) [NOV – 2019]

A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

28) What is De-multiplexer?

The de-multiplexer performs the inverse function of a multiplexer, that is it receives information on one line and transmits its onto one of 2^n possible output lines. The selection is by *n* input select lines.

29) What is Parity?

A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The message, including the parity bit, is transmitted and then checked at the receiving

end for errors. An error is detected if the checked parity does not correspond with the one transmitted.

30) What is Parity Checker / Generator:

The circuit that generates the parity bit in the transmitter is called a *parity generator*. The circuit that checks the parity in the receiver is called a *parity checker*.

31) What is even parity and odd parity?

In even parity system, the parity bit is '0' if there are even number of 1s in the data and the parity bit is '1' if there are odd number of 1s in the data.

In odd parity system, the parity bit is '1' if there are even number of 1s in the data and the parity bit is '0' if there are odd number of 1s in the data.

31) Give the applications of Demultiplexer.

i) It finds its application in Data transmission system with error detection.

ii) One simple application is binary to Decimal decoder.

32) Mention the uses of Demultiplexer.

Demultiplexer is used in computers when a same message has to be sent to different receivers. Not only in computers, but any time information from one source can be fed to several places.

33) Give other name for Multiplexer and Demultiplexer.

Multiplexer is otherwise called as Data selector.

Demultiplexer is otherwise called as Data distributor.

34) What is the function of the enable input in a Multiplexer?

The function of the enable input in a MUX is to control the operation of the unit.

35) List out the applications of decoder? (Dec 2006)

a. Decoders are used in counter system.

b. They are used in analog to digital converter.

c. Decoder outputs can be used to drive a display system.

36) What is the Application of Mux?

1. They are used as a data selector to select one output of many data inputs.
2. They can be used to implement combinational logic circuits
3. They are used in time multiplexing systems.
4. They are used in frequency multiplexing systems.
5. They are used in A/D & D/A Converter.
6. They are used in data acquisition system.

37) List out the applications of comparators?

a. Comparators are used as a part of the address decoding circuitry in computers to select a specific input/output device for the storage of data.

b. They are used to actuate circuitry to drive the physical variable towards the reference value.

c. They are used in control applications.

38) What is carry look-ahead addition?

The speed with which an addition is performed limited by the time required forthe carries to propagate or ripple through all of the stage of the adder. One method ofspeeding up the process is by eliminating the ripple carry delay.

39) What is the Difference between Decoder & Demux.?

S.No	Decoder	Demux
1	Decoder is a many inputs to many Outputs	Demux is a single input to many outputs
2	There are no selection lines.	The selection of specific output line is controlled by the value of selection lines.

40) How Binary to Gray Code Conversion done?

Consider b1, b2, b3, b4 and b5 is the Binary Number and it is need be converted into Grey Code.

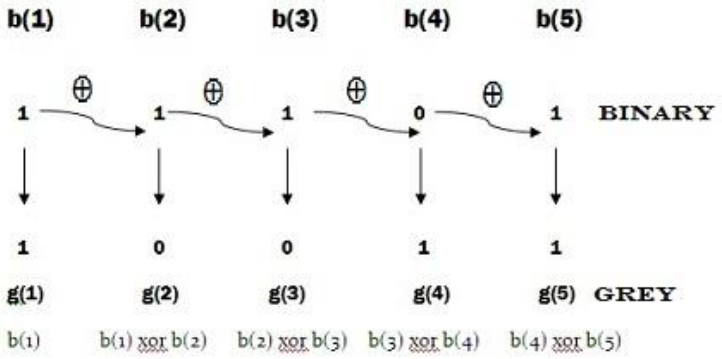
1. Write Most Significant Bit (MSB) is same as the MSB in Binary Number.

2. The second bit of the Grey code can be found by performing the Exclusive-OR (EX-OR) operation between the First and second bits of the Binary Number.

3. The Third bit of the Grey code can be found by performing the Exclusive-OR (EX-OR) operation between the Third and Second bits of the given Binary Number; and so on

EX-OR Operation:

1. Both the bits are 0 or 1 then the output of EX-OR gate will be 0.
2. Any one of the bit in two bits is 1 then the output of EX-OR gate will be 1.

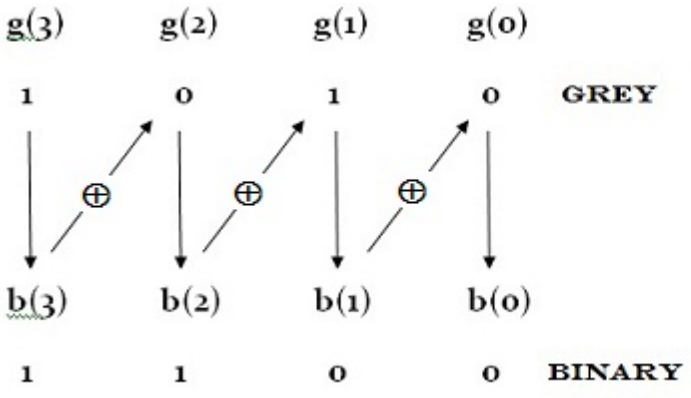


41) How Gray Code to Binary Conversion done?

Consider g0, g1, g2 and g3 is the Gray Code and it is need be converted into Binary Number. The steps for Binary to Gray Code Conversion needs to be reversed to find out the equivalent Binary Number

1. The Most Significant Bit (MSB) of the Binary is same as the First MSB of the Gray Code.

2. If the second Gray Bit is 0 then the second bit of the Binary is bit will be same as that of the First Binary bit; if the Second Gray Bit is 1 then the Second Bit of the Binary will be inverse of its previous binary bit. Refer the below image for easy understanding of Gray to Binary Conversion



i.e

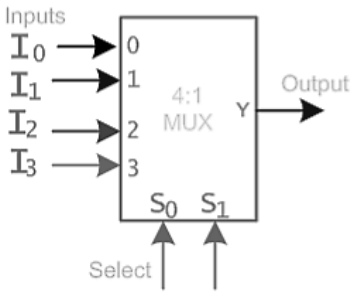
$b(3) = g(3)$

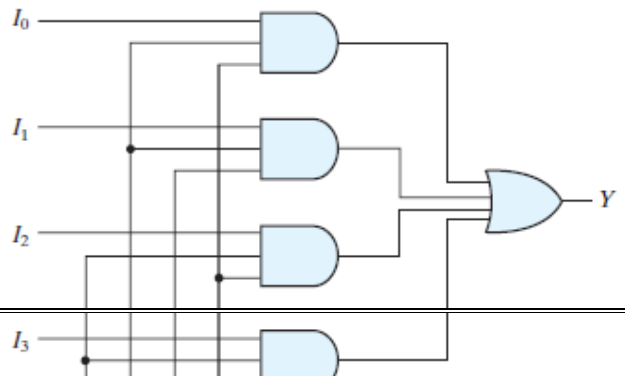
$b(2) = b(3) \oplus g(3)$

$b(1) = b(2) \oplus g(1)$

$b(0) = b(1) \oplus g(0)$

32) Draw the circuit for 4 to 1 line multiplexer. (Apr 2017) [NOV – 2019]
Logic Diagram:





Truth Table:

SELECT INPUT		OUTPUT
S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

42) What do you meant by HDL?

Hardware description language (HDL)- hardware description language or HDL is any language from a class of computer languages and/or programming languages for formal description of digital logic and electronic circuits.

43) List the Verilog HDL model of a combinational circuits.

✓Gate level modeling-using instantiations of predefined and user defined primitive gates.
✓Dataflow modeling using continuous assignment with the keyword **assign**.
✓Behavioral modeling using procedural assignment statements with the keyword **always**.

44) What is meant by Gate level modeling?

In this type, a circuit is specified by its logic gates and their interconnections. Gate level modeling provides a textual description of a schematic diagram.

45) What is meant by data flow modeling?

Data flow modeling of combinational logic uses a number of operators that act on operands to produce desired results. Verilog HDL provides about 30 different operators.

46) What is meant by Behavioral modeling?

Behavioral modeling represents digital circuits at a functional and algorithmic level. It is used mostly to describe sequential circuits, but can also be used to describe combinational circuits.

47) What is Verilog?

Verilog is a general purpose hardware descriptor language. It has similar in syntax to the C programming language. It can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the switch level.

48) Define logic synthesis and simulation.

Logic synthesis is the process of deriving a list of components and their interconnection (called netlist) from the model of a digital system.

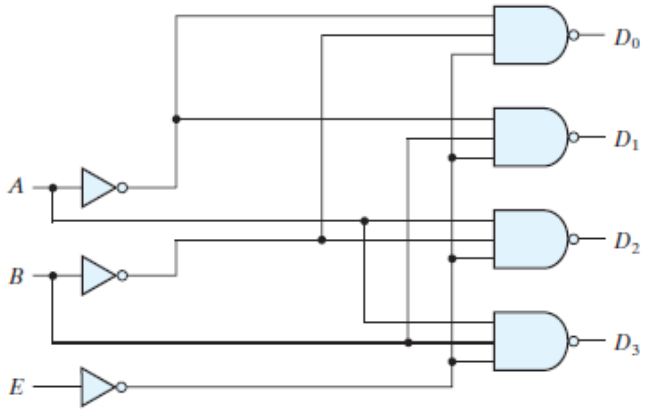
Logic Simulation is the representation of the structure and behavior of a digital logic synthesis through the use of a computer.

49) List the standard HDLs that supported by IEEE.

- ✓ VHDL (very High Speed Integrated Circuit HDL)
- ✓ Verilog HDL

50) Write the truth table of 2 to 4 line decoder and draw its logic diagram. (Apr – 2019)

2 to 4 decoder:



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

(b) Truth table

51) State the different modeling techniques used in HDL. (Apr 2018)

- Gate level modeling
- Data flow modeling
- Behavioral modeling

UNIT III

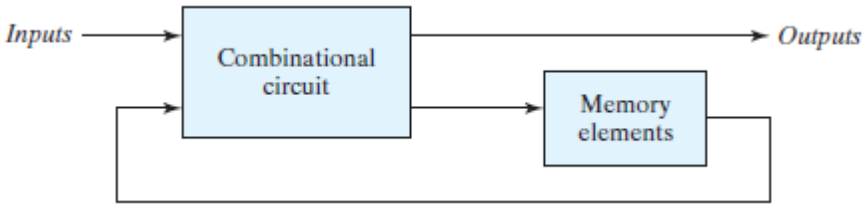
SYNCHRONOUS SEQUENTIAL LOGIC

Sequential Circuits - Storage Elements: Latches , Flip-Flops - Analysis of Clocked Sequential Circuits - State Reduction and Assignment - Design Procedure - Registers and Counters - HDL Models of Sequential Circuits

SEQUENTIAL CIRCUITS

Sequential circuits:

- Sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements.
- Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.



Types of sequential circuits:

There are two main types of sequential circuits, and their classification is a function ofthe timing of their signals.

1. **Synchronous sequential circuit:**

   It is a system whose behaviorcan be defined from the knowledge of its signals at discrete instants of time.

2. **Asynchronous sequential circuits:**

   The behaviorof an asynchronous sequential circuit depends upon the input signals at any instant of timeand the order in which the inputs change. The storage elements commonly used in asynchronoussequential circuits are time-delay devices.

LATCHES AND FLIP FLOPS

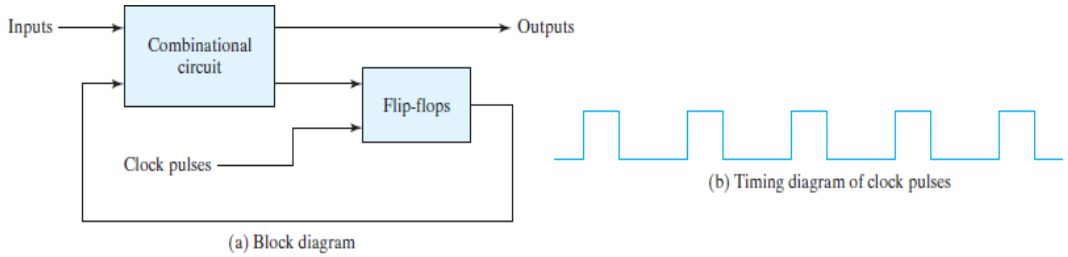
Flip-Flop:

- *The storage elements (memory) used in clocked sequential circuits are called flipflops.* A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1.
- A sequential circuit may use many flip-flops to store as many bits as necessary. The block diagram of a synchronous clocked sequential circuit is shown in Fig.

- A storage element in a digital circuit can maintain a binary state indefinitely (as long as power is delivered to the circuit), until directed by an input signal to switch states.
- The major differences among various types of storage elements are in the number of inputs they possess and in the manner in which the inputs affect the binary state.

Latch:

- *The storage elements that operate with signal levels (rather than signal transitions) are referred to as latches; those controlled by a clock transition are flip-flops.*Latches are said to be level sensitive devices; flip-flops are edge-sensitive devices.

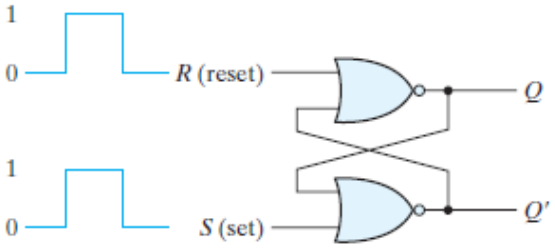


Synchronous clocked sequential circuit

SR Latch: Using NOR gate

Realize SR Latch using NOR and NAND gates and explain its operation.

- The *SR* latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates, and two inputs labeled *S* for set and *R* for reset.
- The *SR* latch constructed with two cross-coupled NOR gates is shown in Fig.



(a) Logic diagram

S	*R*	*Q*	*Q'*	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	(forbidden)

(b) Function table

SR latch with NOR gates

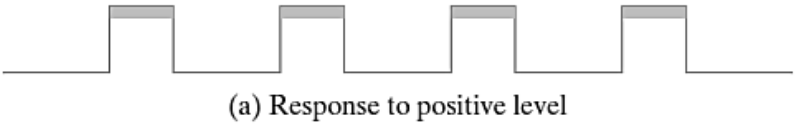
- The latch has two useful states. When output $Q = 1$ and $Q' = 0$, the latch is said to be in the *set state* . When $Q = 0$ and $Q' = 1$, it is in the *reset state* . Outputs Q and Q' are normally the complement of each other.
- However, when both inputs are equal to 1 at the same time, a condition in which both outputs are equal to 0 (rather than be mutually complementary) occurs.
- If both inputs are then switched to 0 simultaneously, the device will enter an unpredictable or undefined state or a metastable state. Consequently, in practical applications, setting both inputs to 1 is forbidden.

FLIP FLOPS

Triggering of Flip Flops:

Explain about triggering of flip flops in detail.

- The state of a latch or flip-flop is switched by a change in the control input. This momentary change is called a *trigger*, and the transition it causes is said to trigger the flip-flop.



(a) Response to positive level

(b) Positive-edge response

(c) Negative-edge response

Level Triggering:

- SR, D, JK and T latches are having enable input.
- Latches are controlled by enable signal, and they are level triggered, either positive level triggered or negative level triggered as shown in figure (a).
- The output is free to change according to the input values, when active level is maintained at the enable input.

Edge Triggering:

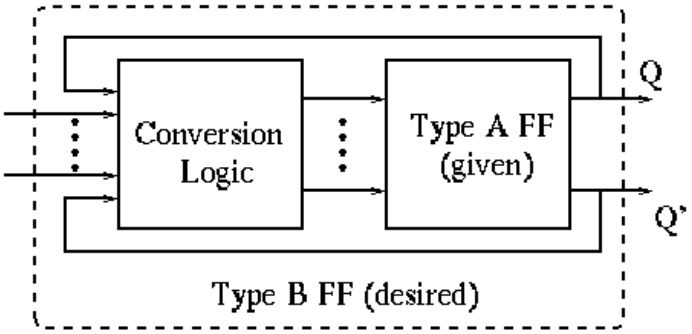
- A clock pulse goes through two transitions: from 0 to 1 and the return from 1 to 0.
- As shown in above Fig (b) and (c)., the positive transition is defined as the positive edge and the negative transition as the negative edge.

****

Explain the operation of flipflops.(Nov 2017)

FLIP FLOP CONVERSIONS

The purpose is to convert a given type A FF to a desired type B FF using some conversion logic.



The key here is to use the excitation table, which shows the necessary triggering signal (S,R, J,K, D and T) for a desired flipflop state transition $Q_t \rightarrow Q_{t+1}$:

Excitation table for all flip flops:

Q_t	Q_{t+1}	S	R	D	J	K	T
0	0	0	X	0	0	X	0
0	1	1	0	1	1	X	1
1	0	0	1	0	X	1	1
1	1	X	0	1	X	0	0

1. SR Flip Flop to JK Flip Flop

The truth tables for the flip flop conversion are given below. The present state is represented by Qp and Qp+1 is the next state to be obtained when the J and K inputs are applied.

For two inputs J and K, there will be eight possible combinations. For each combination of J, K and Qp, the corresponding Qp+1 states are found. Qp+1 simply suggests the future values to be obtained by the JK flip flop after the value of Qp.

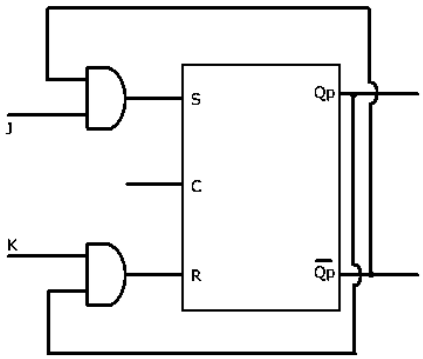
The table is then completed by writing the values of S and R required to get each Qp+1 from the corresponding Qp. That is, the values of S and R that are required to change the state of the flip flop from Qp to Qp+1 are written.

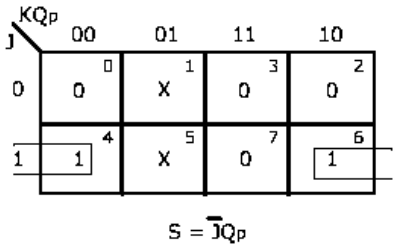
S-R Flip Flop to J-K Flip Flop

Conversion Table

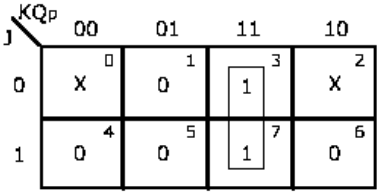
J-K Inputs		Outputs		S-R Inputs	
J	K	Qp	Qp+1	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Logic Diagram





$S = \overline{J}Qp$



$R = KQn$

2.JK Flip Flop to SR Flip Flop

This will be the reverse process of the above explained conversion. S and R will be the external inputs to J and K. As shown in the logic diagram below, J and K will be the outputs of the combinational circuit. Thus, the values of J and K have to be obtained in terms of S, R and Qp. The logic diagram is shown below.

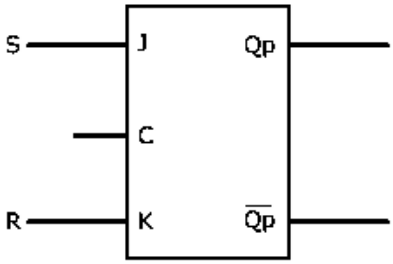
A conversion table is to be written using S, R, Qp, Qp+1, J and K. For two inputs, S and R, eight combinations are made. For each combination, the corresponding Qp+1 outputs are found. The outputs for the combinations of S=1 and R=1 are not permitted for an SR flip flop. Thus the outputs are considered invalid and the J and K values are taken as "don't cares".

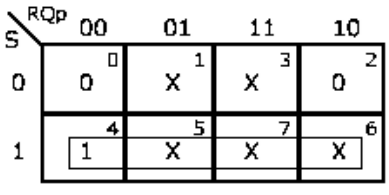
J-K Flip Flop to S-R Flip Flop

Conversion Table

S-R Inputs		Outputs		J-K Inputs	
S	R	Qp	Qp+1	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	Invalid		Dont care	
1	1	Invalid		Dont care	

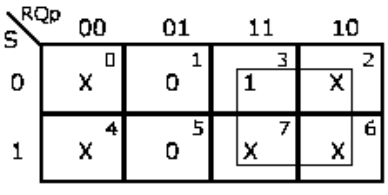
Logic Diagram





J=S

K-maps



K=R

3.SR Flip Flop to D Flip Flop

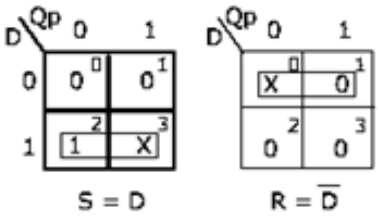
As shown in the figure, S and R are the actual inputs of the flip flop and D is the external input of the flip flop. The four combinations, the logic diagram, conversion table, and the K-map for S and R in terms of D and Qp are shown below.

S-R Flip Flop to D Flip Flop

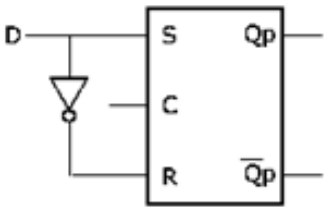
Conversion Table

D Input	Outputs		S-R Inputs	
	Qp	Qp+1	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

K-maps



Logic Diagram



4.D Flip Flop to SR Flip Flop

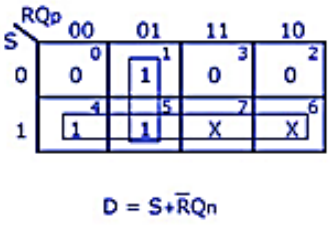
D is the actual input of the flip flop and S and R are the external inputs. Eight possible combinations are achieved from the external inputs S, R and Qp. But, since the combination of S=1 and R=1 are invalid, the values of Qp+1 and D are considered as "don't cares". The logic diagram showing the conversion from D to SR, and the K-map for D in terms of S, R and Qp are shown below.

D Flip Flop to S-R Flip Flop

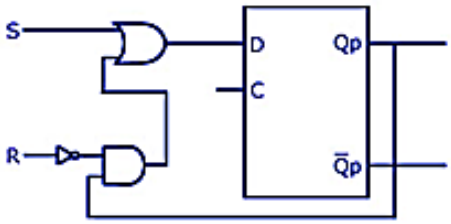
Conversion Table

S-R Inputs		Outputs		D Input
S	R	Qp	Qp+1	
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	Invalid		Dont care
1	1	Invalid		Dont care

K-map

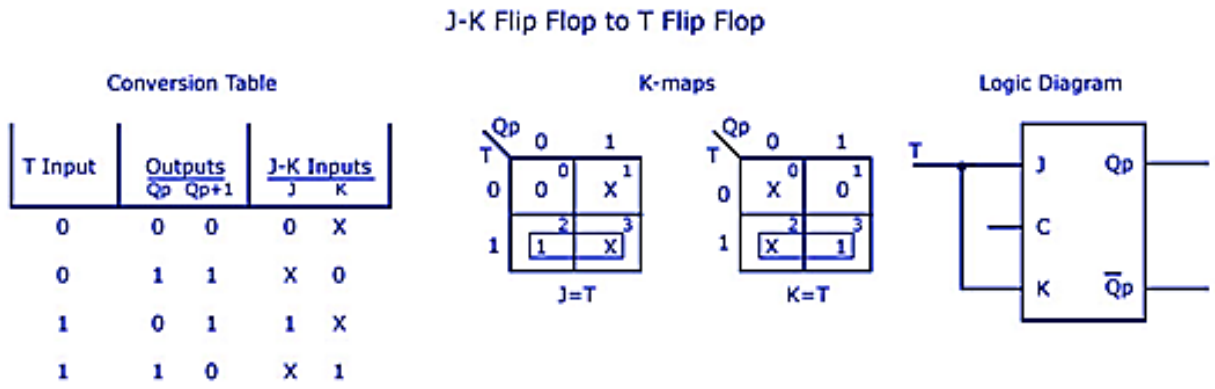


Logic Diagram



5.JK Flip Flop to T Flip Flop

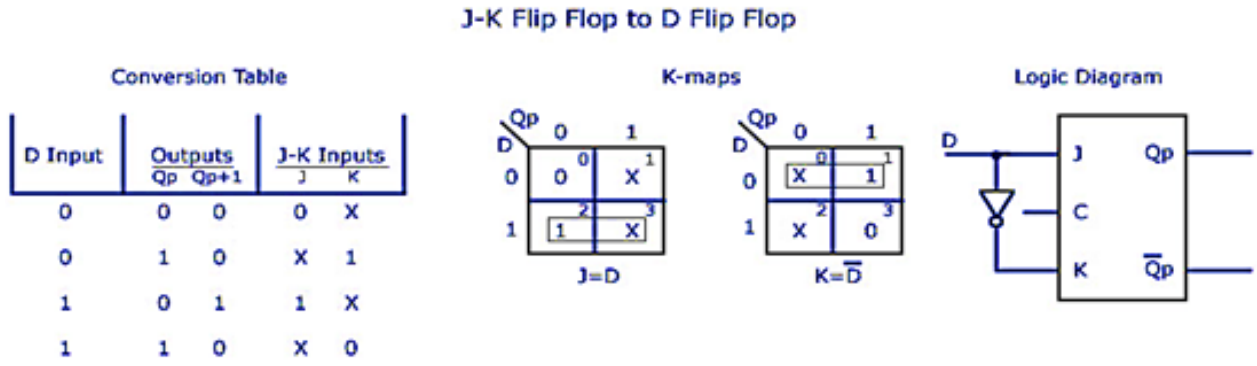
J and K are the actual inputs of the flip flop and T is taken as the external input for conversion. Four combinations are produced with T and Qp. J and K are expressed in terms of T and Qp. The conversion table, K-maps, and the logic diagram are given below.



T Input	Outputs Qp	Qp+1	J-K Inputs J	K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

6.JK Flip Flop to D Flip Flop

D is the external input and J and K are the actual inputs of the flip flop. D and Qp make four combinations. J and K are expressed in terms of D and Qp. The four combination conversion table, the K-maps for J and K in terms of D and Qp, and the logic diagram showing the conversion from JK to D are given below.



D Input	Outputs Qp	Qp+1	J-K Inputs J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	0	X	0

7.D Flip Flop to JK Flip Flop

AUQ: How will you convert a D flip-flop into JK flip-flop? (AUQ: Dec 2009,11,Apr 2017)

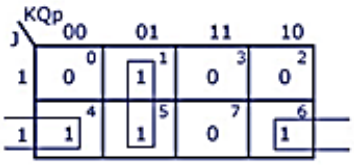
In this conversion, D is the actual input to the flip flop and J and K are the external inputs. J, K and Qp make eight possible combinations, as shown in the conversion table below. D is expressed in terms of J, K and Qp.The conversion table, the K-map for D in terms of J, K and Qp and the logic diagram showing the conversion from D to JK are given in the figure below.

D Flip Flop to J-K Flip Flop

Conversion Table

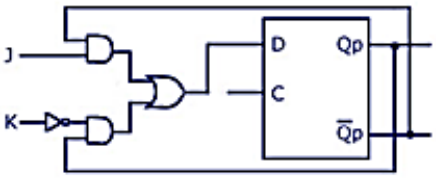
J-K Input J	K	Outputs Qp	Qp+1	D Input
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

K-map



$D = J\overline{Q}p + \overline{K}Qp$

Logic Diagram



****

MEALY AND MOORE MODELS

Write short notes on Mealy and Moore models in sequential circuits.

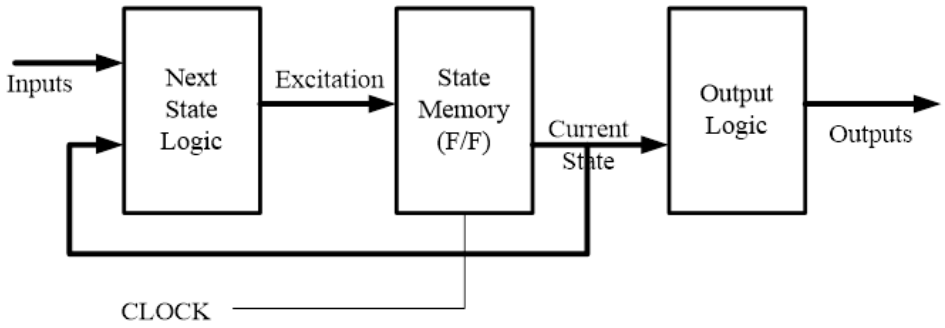
- In synchronous sequential circuit the outputs depend upon the order in which its input variables change and can be affected at discrete instances of time.

General Models:

- There are two models in sequential circuits. They are:
    1. Mealy model
    2. Moore model

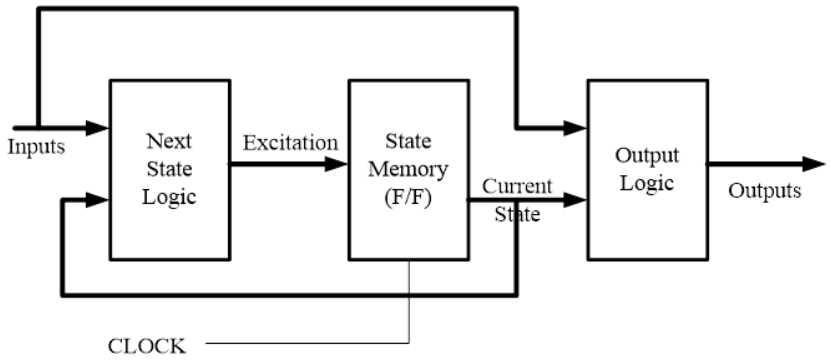
Moore machine:

- In the Moore model, the outputs are a function of present state only.



Mealy machine:

- In the Mealy model, the outputs are a function of present state and external inputs.

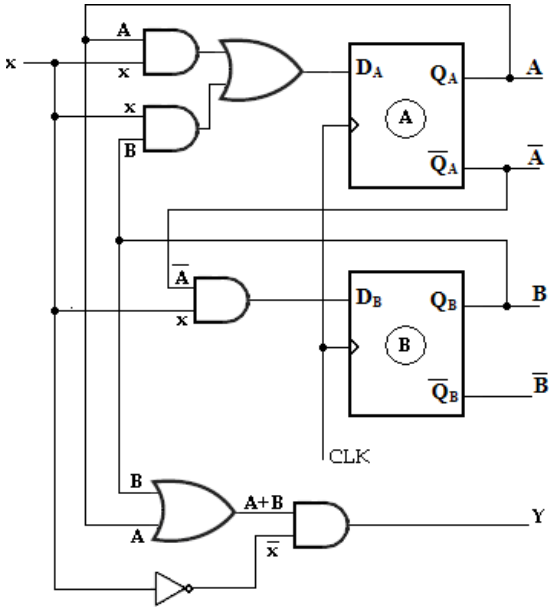


Difference between Moore model and Mealy model.

Sl.No	Moore model	Mealy model
1	Its output is a function of present state only.	Its output is a function of present state as well as present input.
2	Input changes does not affect the output.	Input changes may affect the output of the circuit.
3	It requires more number of states for implementing same function.	It requires less number of states for implementing same function.

Example:
A sequential circuit with two 'D' Flip-Flops A and B, one input (x) and one output (y).
The Flip-Flop input functions are:
D_A= Ax+ Bx
D_B= A'x and
the circuit output function is, Y= (A+ B) x'.
(a) Draw the logic diagram of the circuit, (b) Tabulate the state table, (c) Draw the state diagram.

Solution:



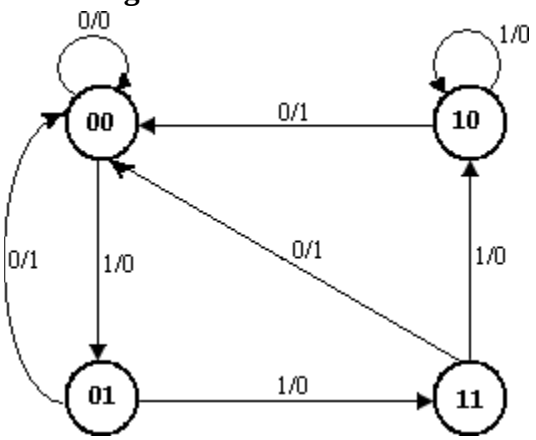
State table:

Present state		Input	Flip-Flop Inputs		Next state		Output
A	B	x	D_A= Ax+Bx	D_B= A′x	A(t+1)	B(t+1)	Y= (A+B)x′
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	0	0	0	1
0	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	1
1	1	1	1	0	1	0	0

Present state		Next state				Output	
		x= 0		x= 1		x= 0	x= 1
A	B	A	B	A	B	Y	Y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

Second form of state table

State diagram:



****

TWO MARKS

1. Difference between Combinational & Sequential Circuits.

S .no	Combinational Circuits	Sequential Circuits
1	The output at all times depends only on the present combination of input variables.	The output not only depends on the present input but also depends on the past history input variables.
2	Memory unit is not Required	Memory unit is required to store the past history of input variable
3	Clock input is not needed.	Clock input is needed.

4	Faster in Speed	Speed is Slower
5	Easy to design. Eg:Mux, Demux, Encoder, Decoder, Adders, Subtractors.	Difficult to design. Eg: Shift Register, Counters.

2. What are the classifications of sequential circuits?

The sequential circuits are classified on the basis of timing of their signals in to two types. They are 1) Synchronous sequential circuit.2) Asynchronous sequential circuit.

3. **Define Latch.**

The basic unit for storage is Latch. A Latch maintain its output state either at 1or 0 until directed by an input signal to change its state.

4. Define a flip flop.

A flip-flop is a storage device capable of storing one bit of information. It has two states either 0 or 1. It is also called bistable multivibrator.

5. What are the different types of flip-flop?

The various types of flip flops are 1). SR flip-flop 2). D flip-flop 3). JK flip-flop 4). T flip-flop

6. What is the main difference between a latch and flip flop?

- ✓ The output of latch changes immediately when its input changes.
- ✓ The output of a flip-flop changes only when its clock pulse is active and its input changes. Input changes do not affect output if its clock is not activated.

7. State few application of Flip-Flop.

- ✓ Used as a memory element.
- ✓ Used as delay elements.
- ✓ Data transfer
- ✓ Used as a building block in sequential circuits such as counters and registers.

8. What is the operation of D flip-flop?

In D flip-flop during the occurrence of clock pulse if D=1, the output Q is set and if D=0, the output is reset. Set – 1, Reset – 0.

9. What is the operation of JK flip-flop?

When K input is low and J input is high the Q output of flip-flop is set.

When K input is high and J input is low the Q output of flip-flop is reset.

When both the inputs K and J are low the output does not change

When both the inputs K and J are high it is possible to set or reset the flip-flop(ie) the output toggle onthe next positive clock edge.

10. What is the operation of T flip-flop? (Nov 2018)

T flip-flop is also known as Toggle flip-flop. 1). When T=0 there is no change in the output. 2). When T=1 the output switch to the complement state (ie) the output toggles.

11. Define race around condition.

In JK flip-flop output is fed back to the input. Therefore change in the output results change in the input. Due to this in the positive half of the clock pulse if both J and K are high then output toggles continuously. This condition is called 'race around condition'.

12. What is triggering? What is the need for trigger in flip-flop?

A flip-flop is made to change its state by application of a clock pulse after giving inputs. This is called triggering. The clock (triggering input) is given to synchronize the change in the output with it.

13. What is meant by level and edge-triggering? (Nov 2017) (Apr – 2019)

- ✓ If flip-flop changes its state when the clock is positive (high) or negative (low) then, that flip-flop is said to be *level triggering flip-flop*.
- ✓ If the flip-flop changes its state at the positive edge (rising edge) or negative edge (falling edge) of the clock is sensitive to its inputs only at this transition of the clock then flip-flop is said to be *edge triggered flip-flop.*

14. How do you eliminate race around condition in JK flip flop. ?

Using master-slave flip-flop which consists of two flip-flops where one circuit serves as a master and the other as a slave race around condition in JK flip flop is eliminated .

15. Define rise time.

The time required to change the voltage level from 10% to 90% is known as rise time (t_r).

16. Define fall time.

The time required to change the voltage level from 90% to 10% is known as fall time (t_f).

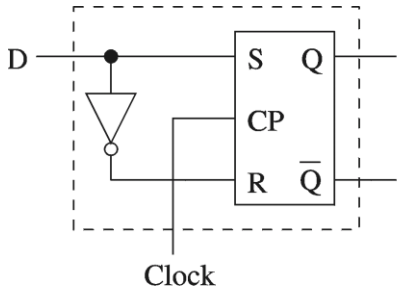
17. Define skew and clock skew.

The phase shift between the rectangular clock waveforms is referred to as skew and the time delay between the two clock pulses is called clock skew.

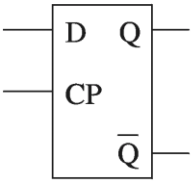
18. Define setup time.

The setup time is the minimum time required to maintain a constant voltage levels at the excitation inputs of the flip-flop device prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip flop.

19. Draw the logic diagram and write the function table of D Latch. (Apr 2019)



(a) Circuit diagram



(b) Logic symbol

D	Q_{n+1}
0	0
1	1

(c) Truth table

20. Define hold time.

The hold time is the minimum time for which the voltage levels at the excitation inputs must remain constant after the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip flop.

21. Define propagation delay.

A propagation delay is the time required to change the output after the application of the input

22. Explain the flip-flop excitation tables for RS FF.

In RS flip-flop there are four possible transitions from the present state to theNext state. They are

1). 0→0 transition: This can happen either when R=S=0 or when R=1 and S=0.

2). 0→ 1 transition: This can happen only when S=1 and R=0.

3). 1→0 transition: This can happen only when S=0 and R=1.

4). 1→1 transition: This can happen either when S=1 and R=0 or S=0 and R=0.

23. Give some applications of clocked RS Flip-flop.

Clocked RS flip flops are used in Calculators & Computers.

It is widely used in modern electronic products.

24. What is the drawback of SR Flipflop? How is this minimized? (Apr 2018)

In SR flipflop when both S and R inputs are one it will generate a Undetermined state.This is Minimized by providing feedback path or by using JK flip flop.

25. How many flip flops are required to build a Binary counter that counts from 0 to 1023?

2^{10}= 1024 hence 10 flipflops are required.

26. State the difference between latches and flipflops. (Apr 2019)

Latches	Flip Flops
Latches are building blocks of sequential circuits and these can be built from logic gates	Flip flops are also building blocks oof sequential circuits. But, these can be built from the latches.
Latch continuously checks its inputs and changes its output correspondingly.	Flip flop continuously checks its inputs and changes its output correspondingly only at times determined by clocking signal
The latch is sensitive to the duration of the pulse and can send or receive the data when the switch is on	Flipflop is sensitive to a signal change. They can transfer data only at the single instant and data cannot be changed until next signal change. Flip flops are used as a register.
It is based on the enable function input	It works on the basis of clock pulses
It is a level triggered, it means that the output of the present state and input of the next state depends on the level that is binary input 1 or 0.	It is an edge triggered, it means that the output and the next state input changes when there is a change in clock pulse whether it may a +ve or -ve clock pulse.

27. What is mealy and Moore circuit? Or what are the models used to represent clocked sequential circuits?

- ✓ *Mealy circuit* is a network where the output is a function of both present state and input.
- ✓ *Moore circuit is* a network where the output is function of only present state

****

COUNTERS

Counter:

- A counter is a register (group of Flip-Flop) capable of counting the number of clock pulse arriving at its clock input.
- A counter that follows the binary number sequence is called a binary counter.
- Counter are classified into two types,
    1. Asynchronous (Ripple) counters.
    2. Synchronous counters.
- In ripple counter, a flip- flop output transition serves as clock to next flip-flop.

- With an asynchronous circuit, all the bits in the count do not all change at the same time.

- In a synchronous counter, all flip-flops receive common clock.
  - With a synchronous circuit, all the bits in the count change synchronously with the assertion of the clock
- A counter may count up or count down or count up and down depending on the input control.

Uses of Counters:

The most typical uses of counters are

- ✓ To count the number of times that a certain event takes place; the occurrence of event to be counted is represented by the input signal to the counter
- ✓ To control a fixed sequence of actions in a digital system
- ✓ To generate timing signals
- ✓ To generate clocks of different frequencies

Modulo 16 ripple /Asynchronous Up Counter

Explain the operation of a 4-bit binary ripple counter.

- The output of up-counter is incremented by one for each clock transition.
- A 4-bit asynchronous up-counter consists of 4JK Flip-Flops.
- The external clock signal is connected to the clock input of the first FlipFlop.
- The clock inputs of the remaining Flip-Flops are triggered by the Q output of the previous stage.
- We know that in JK Flip-Flop, if J=1 , K=1 and clock is triggered the past output will be complemented.
- Initially, the register is cleared, $Q_DQ_CQ_BQ_A$ =0000.
- During the first clock pulse, Flip-Flop A triggers, therefore Q_A=1, Q_B=Q_C=Q_D=0.

$$Q_DQ_CQ_BQ_A=0001$$

- At the second clock pulse FLipFlop A triggers, therefore Q_A changes from 1 to 0, which triggers FlipFlop B, therefore Q_B=1,Q_A=Q_C=Q_D=0

$$Q_DQ_CQ_BQ_A=0010$$

- At the third clock pulse FlipFlop A triggers, therefore Q_A changes from 0 to 1, This never triggers FlipFlop B because 0 to 1 transition gives a positive edge triggering,but here the FlipFlops are triggered only at negative edge(1 to 0 transition) therefore Q_A=Q_B=1, Q_C=Q_D=0.

$$Q_DQ_CQ_BQ_A=0011$$

- At the fourth clock pulse Flip-Flop A triggers, therefore Q_A changes from 1 to 0, This triggers FlipFlop B therefore Q_B changes from 1 to 0. The change in Q_B from 1 to 0 triggers C Flip-Flop,

- Therefore Q_C changes from 0 to 1. Therefore $Q_A=Q_B=Q_D=0$, $Q_C=1$.

$Q_DQ_CQ_BQ_A=0100$

Truth table:

CLK	Outputs			
	Q_D	Q_C	Q_B	Q_A
-	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Truth table for 4-bit asynchronous up-counter

Timi

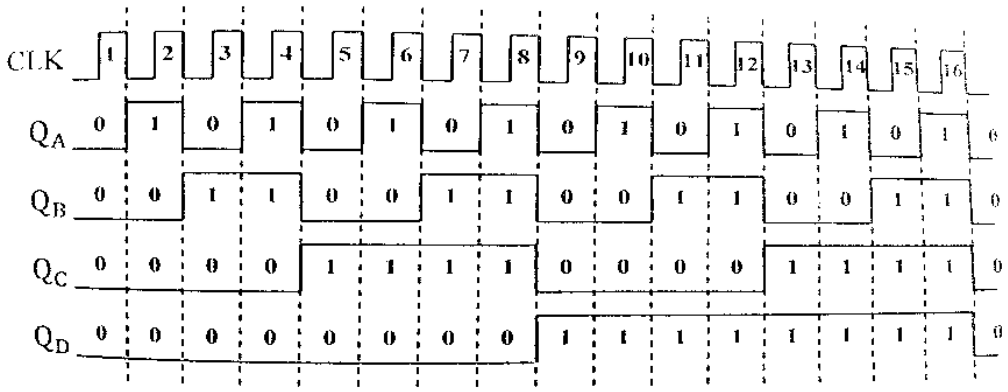


Figure 4.37 Timing diagram of 4-bit asynchronous up-counter.

****

Modulo 16 /4 bit Ripple Down counter/ Asynchronous Down counter

Explain about Modulo 16 /4 bit Ripple Down counter.

- The output of down-counter is decremented by one for each clock transition.
- A 4-bit asynchronous down-counter consists of 4JK Flip-Flops.

- The external clock signal is connected to the clock input of the first Flip-Flop.
- The clock inputs of the remaining Flip-Flops are triggered by the $\overline{Q}$ output of the previous stage.
- We know that in JK Flip-Flop, if J=1 , K=1 and clock is triggered the past output will be complemented.

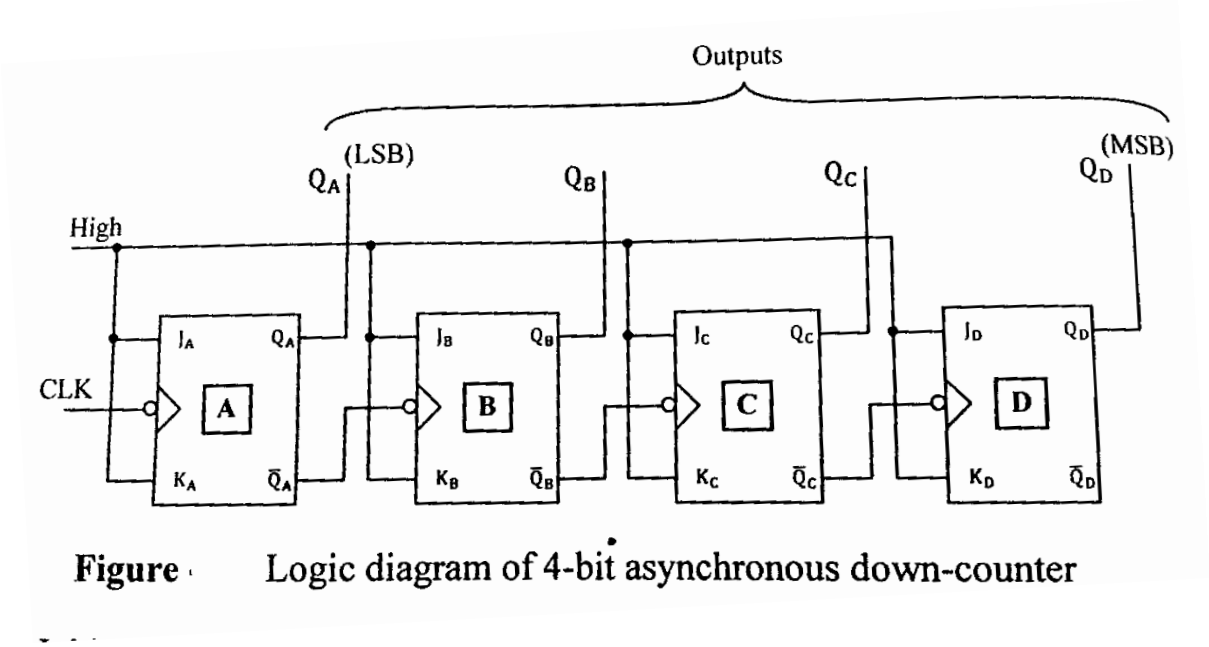


Figure Logic diagram of 4-bit asynchronous down-counter

- Initially, the register is cleared, $Q_DQ_CQ_BQ_A$ =0000.
- During the first clock pulse, Flip-Flop A triggers, therefore Q_A changes from 0 to 1 also $\overline{Q_A}$ changes from 1 to 0.This triggers Flip-Flop B, therefore Q_B changes from 0 to 1, also $\overline{Q_B}$ changes from 1 to 0which triggers Flip-FlopC. Hence Q_C changes from 0 to 1 and $\overline{Q_C}$ changes from 1 to 0, which further triggers, Flip-Flop D.

$$Q_DQ_CQ_BQ_A=1111$$

$$\overline{Q_D}\,\overline{Q_C}\,\overline{Q_B}\,\overline{Q_A}=0000$$

- During the second clock pulse Flip-Flop A triggers, therefore Q_A changes from 1 to 0 also $\overline{Q_A}$ changes from 0 to 1 which never triggers B Flip-Flop. Therefore C and D Flip-Flop are not triggered.

$$Q_DQ_CQ_BQ_A =1110$$

- The same procedure repeats until the counter decrements upto 0000.

CLK	Outputs			
	Q_D	Q_C	Q_B	Q_A
-	0	0	0	0
1	1	1	1	1
2	1	1	1	0
3	1	1	0	1
4	1	1	0	0
5	1	0	1	1
6	1	0	1	0
7	1	0	0	1
8	1	0	0	0
9	0	1	1	1
10	0	1	1	0
11	0	1	0	1
12	0	1	0	0
13	0	0	1	1
14	0	0	1	0
15	0	0	0	1
16	0	0	0	0

Table ... Truth table for 4-bit asynchronous down-counter

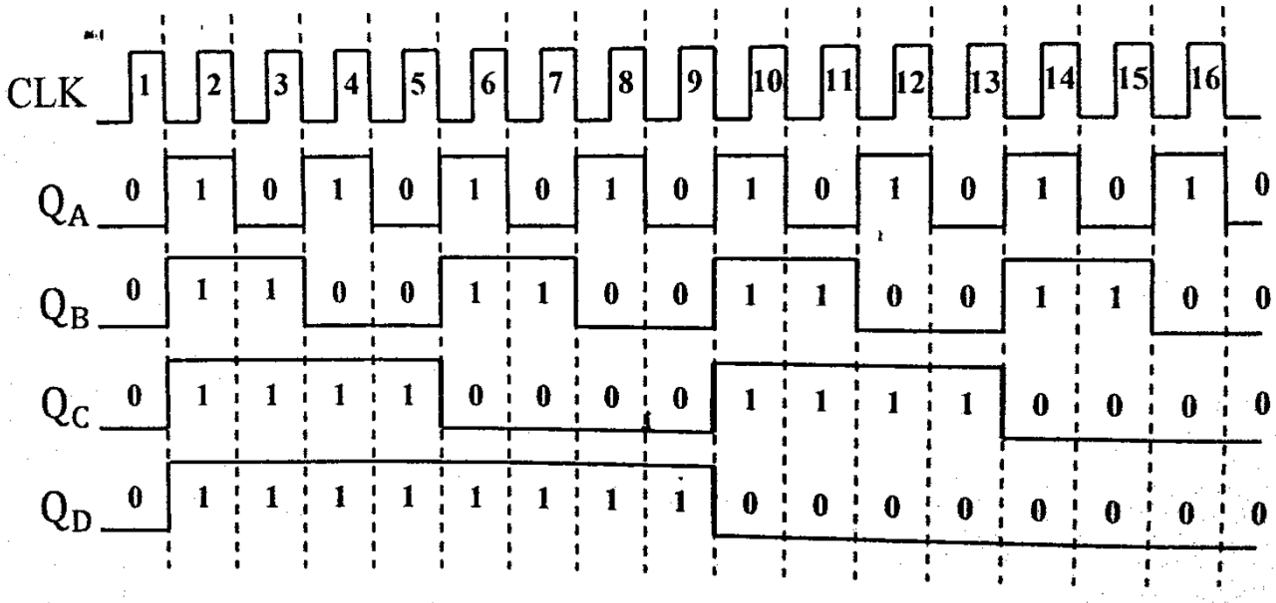


Figure 4) Timing diagram of 4-bit asynchronous down-counter.

****

Asynchronous Up/Down Counter:

Explain about Asynchronous Up/Down counter.

- The up-down counter has the capability of counting upwards as well as downwards. It is also called multimode counter.
- In asynchronous up-counter, each flip-flop is triggered by the normal output Q of the preceding flip-flop.
- In asynchronous down counter, each flip-flop is triggered by the complement output $\overline{Q}$ of the preceding flip-flop.
- In both the counters, the first flip-flop is triggered by the clock output.

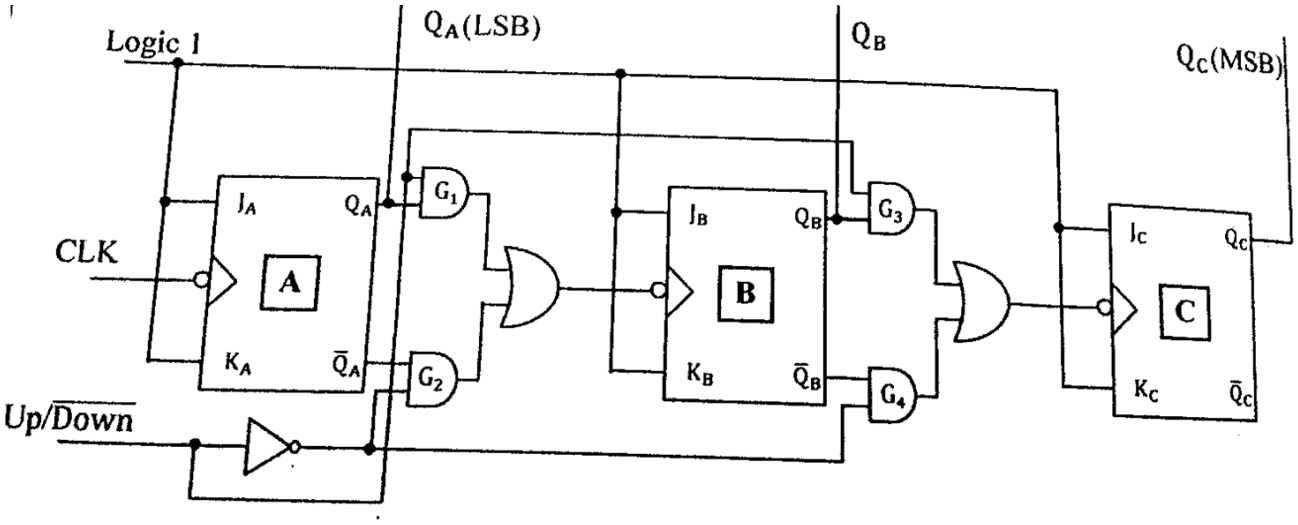


Figure 3-bit asynchronous up/down-counter

- *If Up/$\overline{Down}$ =1,* the 3-bit asynchronous up/down counter will perform up-counting. It will count from 000 to 111. If Up/$\overline{Down}$ =1 gates G_2 and G_4 are disabled and gates G_1 and G_3 are enabled. So that the circuit behaves as an up-counter circuit.
- *If Up/$\overline{Down}$ =0,* the 3-bit asynchronous up/down counter will perform down-counting. It will count from 111 to 000. If Up/$\overline{Down}$ =0 gates G_2 and G_4 are enabled and gates G_1 and G_3 are disabled. So that the circuit behaves as an down-counter circuit.

Up/$\overline{Down}$ = 1

Q_C	Q_B	Q_A
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Up/$\overline{Down}$ = 0

Table : Truth table for 3-Bit asynchronous Up/Down-counter

****

4- bitSynchronous up-counter:

Explain about 4-bit Synchronous up-counter.

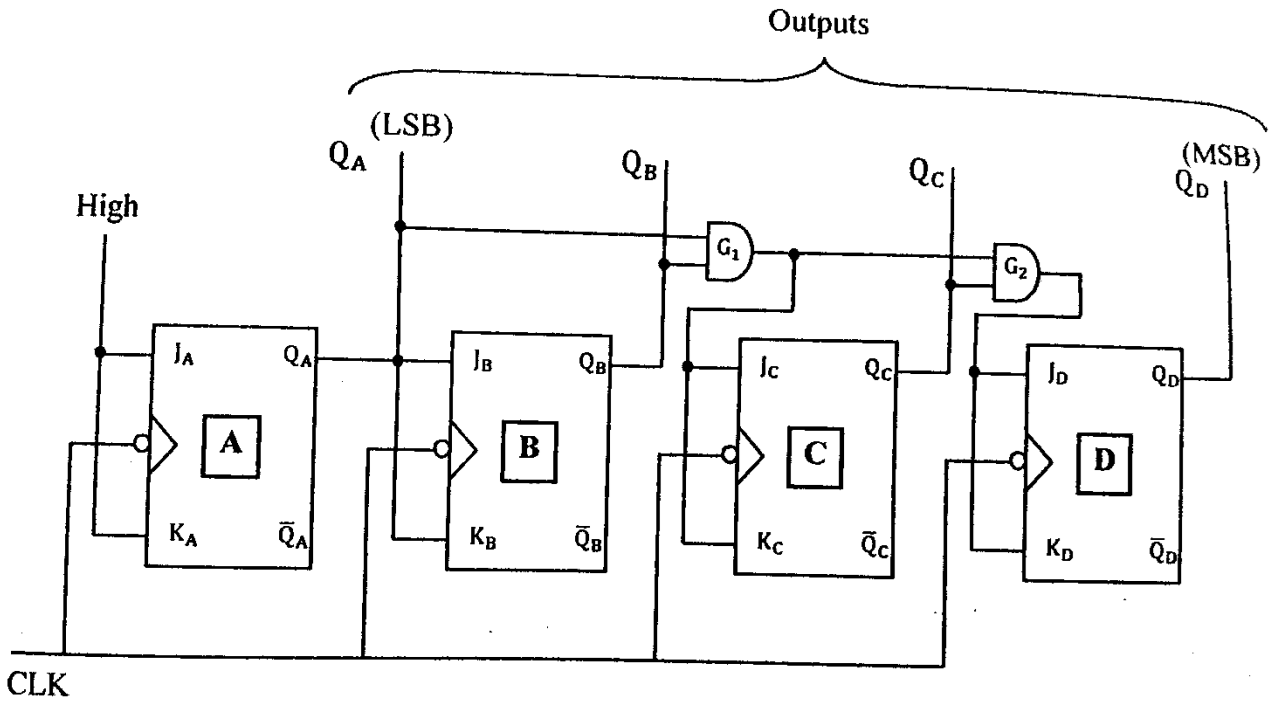


Figure Logic diagram of 4-bit Synchronous up-counter

- In JK Flip-Flop, If J=0, K=0 and clock is triggered, the output never changes. If J=1 and K=1 and the clock is triggered, the past outpit will be complemented.

Initially the register is cleared $Q_DQ_CQ_BQ_A$= 0000.

During the first clock pulse, J_A= K_A = 1, Q_A becomes 1, Q_B, Q_C, Q_D remains 0.

$Q_DQ_CQ_BQ_A$= 0001.

During second clock pulse, J_A= K_A = 1, Q_A=0.

J_B= K_B = 1, Q_B =1, Q_C, Q_D remains 0.

$Q_DQ_CQ_BQ_A$= 0010.

During third clock pulse, J_A= K_A = 1, Q_A=1.

J_B= K_B = 0, Q_B =1, Q_C, Q_D remains 0.

$Q_DQ_CQ_BQ_A$= 0011.

During fourth clock pulse, J_A= K_A = 1, Q_A=0.

J_B= K_B = 1, Q_B =0

J_C= K_C = 1, Q_C=1

Q_D remains 0

$Q_DQ_CQ_BQ_A$= 0100.

The same procedure repeats until the counter counts up to 1111.

CLK	Outputs			
	Q_D	Q_C	Q_B	Q_A
-	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Table Truth table for 4-bit synchronous up-counter

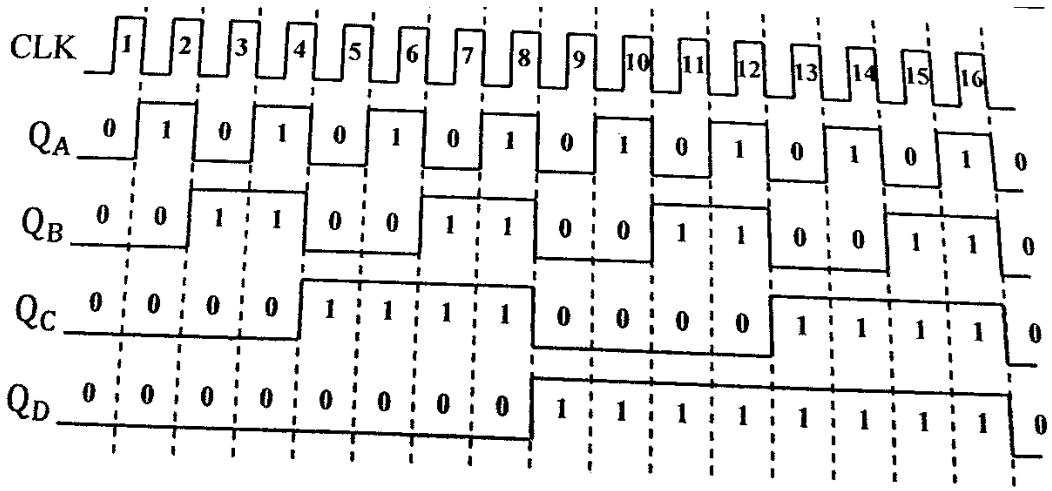


Figure Timing diagram of 4-bit synchronous up-counter

****

4- bit Synchronous down-counter:

Explain about 4-Bit Synchronous down counter.

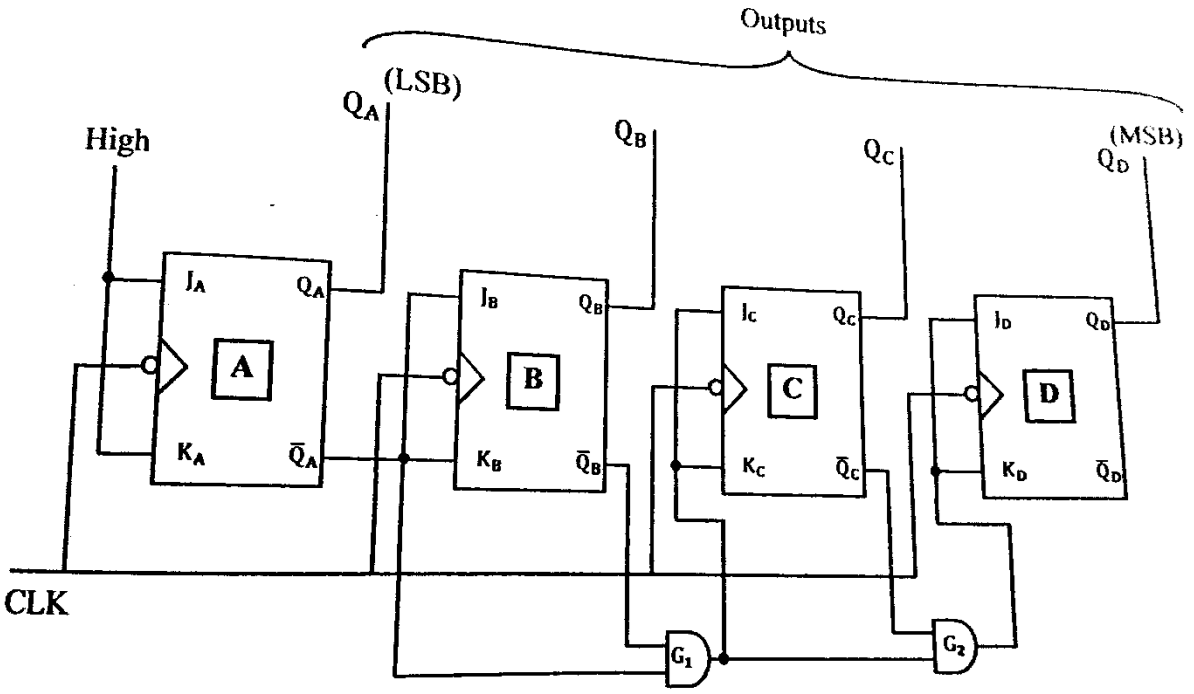


Figure 3 Logic diagram of 4-bit synchronous down-counter

In JK Flip-Flop, If J=0, K=0 and clock is triggered, the output never changes. If J=1 and K=1 and the clock is triggered, the past outpit will be complemented.

Initially the register is cleared $Q_DQ_CQ_BQ_A = 0000$

$\overline{Q}_D\overline{Q}_C\overline{Q}_B\overline{Q}_A = 1111$

During the first clock pulse, $J_A = K_A = 1$, $Q_A = 1$

$J_B = K_B = 1$, $Q_B = 1$

$J_C = K_C = 1$, $Q_C = 1$

$J_D = K_D = 1$, $Q_D = 1$

$Q_DQ_CQ_BQ_A = 1111$

$\overline{Q}_D\overline{Q}_C\overline{Q}_B\overline{Q}_A = 0000$

During the second clock pulse, $J_A = K_A = 1$, $Q_A = 0$

$J_B = K_B = 0$, $Q_B = 1$

$J_C = K_C = 0$, $Q_C = 1$

$J_D = K_D = 0$, $Q_D = 1$

$Q_DQ_CQ_BQ_A = 1110$

$\overline{Q}_D\overline{Q}_C\overline{Q}_B\overline{Q}_A = 0001$

During the second clock pulse, $J_A = K_A = 1$, $Q_A = 1$

$J_B = K_B = 1$, $Q_B = 0$

$J_C = K_C = 0$, $Q_C = 1$

$J_D = K_D = 0$, $Q_D = 1$

$Q_D Q_C Q_B Q_A = 1101$

The process repeats until the counter down-counts up to 0000.

CLK	Outputs			
	Q_D	Q_C	Q_B	Q_A
-	0	0	0	0
1	1	1	1	1
2	1	1	1	0
3	1	1	0	1
4	1	1	0	0
5	1	0	1	1
6	1	0	1	0
7	1	0	0	1
8	1	0	0	0
9	0	1	1	1
10	0	1	1	0
11	0	1	0	1
12	0	1	0	0
13	0	0	1	1
14	0	0	1	0
15	0	0	0	1
16	0	0	0	0

Table Truth table of 4-bit synchronous down-counter

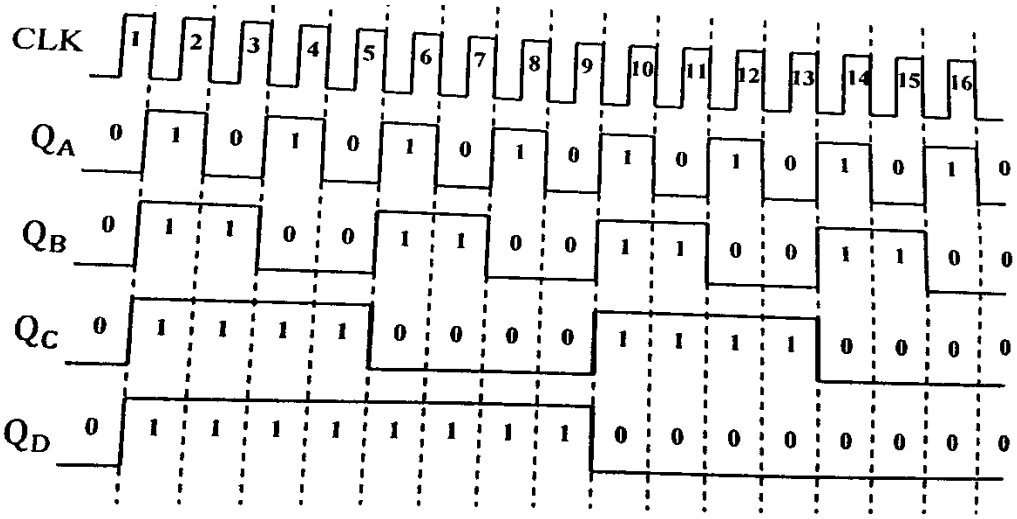


Figure ... Timing diagram of 4-bit synchronous down-counter

****

Modulo 8 Synchronous Up/Down Counter:

Explain about Modulo 8 Synchronous Up/Down Counter.

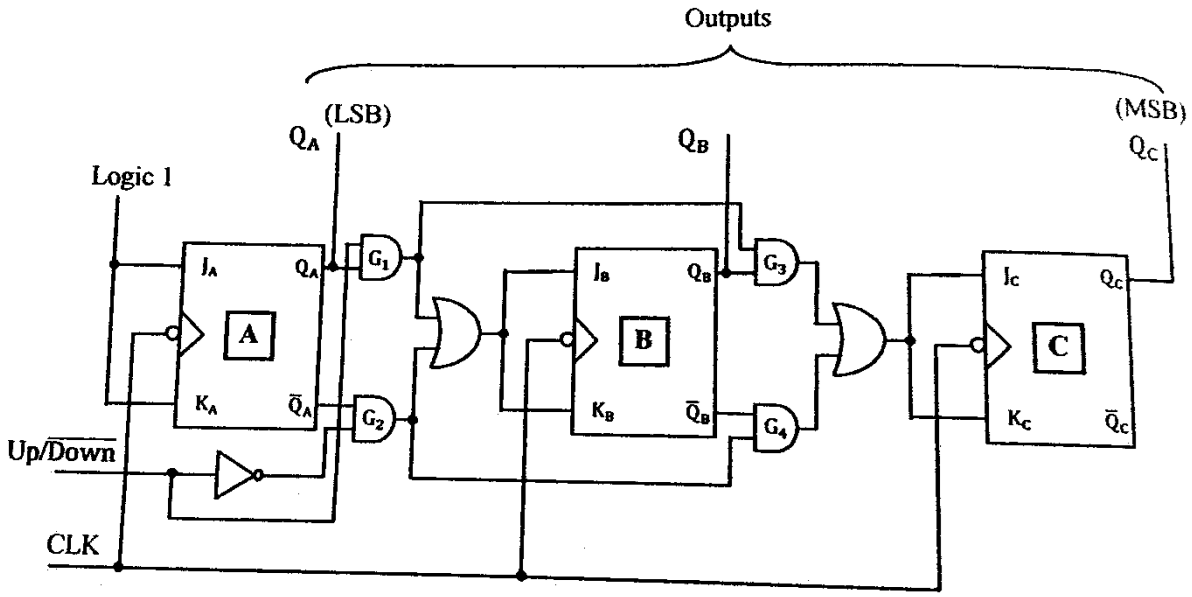


Figure : 3-bit synchronous up/down-counter

In synchronous up-counter the Q_A output is given to J_B, $K_B$and Q_A. Q_B is given to J_C, K_C. But in synchronous down –counter $\overline{Q_A}$output is given toJ_B, K_B and $\overline{Q_A}$. $\overline{Q_B}$ is given to J_C, K_C.

A control input Up/$\overline{\text{Down}}$ is used to select the mode of operation.

If *Up/$\overline{\text{Down}}$ =1*, the 3-bit asynchronous up/down counter will perform up-counting. It will count from 000 to 111. If Up/$\overline{\text{Down}}$ =1 gates G_2 and G_4 are disabled and gates G_1 and G_3 are enabled. So that the circuit behaves as an up-counter circuit.

If *Up/$\overline{\text{Down}}$ =0*, the 3-bit asynchronous up/down counter will perform down-counting. It will count from 111 to 000. If Up/$\overline{\text{Down}}$ =0 gates G_2 and G_4 are enabled and gates G_1 and G_3 are disabled. So that the circuit behaves as an down-counter circuit.

Up/$\overline{\text{Down}}$ = 1

Q_C	Q_B	Q_A
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Up/$\overline{\text{Down}}$ = 0

Table : Truth table for 3-Bit asynchronous Up/Down-counter

****

TWO MARKS

1. What is counter?

A counter is a register (group of Flip-Flop) capable of counting the number of clock pulse arriving at its clock input.

2. What is binary counter?

A counter that follows the binary number sequence is called a binary counter.

3. State the applications of counters.

1. Used as a memory Element.
2. Used as a Delay Element.
3. Used as a basic building block in sequential circuits such as counters and registers.
4. Used for Data Transfer, Frequency Division & Counting.

4. List the types of counters.

Counter are classified into two types,

- ✓ Asynchronous (Ripple) counters.
- ✓ Synchronous counters.

5. Give the comparison between synchronous & Asynchronous counters. (Nov/Dec 2009, Nov 2017)

S.No	Asynchronous counters	Synchronous counters
1.	In this type of counter flip-flops are connected in such a way that output of 1st flip-flop drives the clock for the next flip - flop.	In this type there is no connection between output of first flip-flop and clock input of the next flip – flop
2	All the flip-flops are not clocked simultaneously	All the flip-flops are clocked simultaneously
3	Logic circuit is very simple even for more number of states	Design involves complex logic circuit as number of states increases
4	Counters speed is low.	Counters speed is high.

6. State the Steps or Design procedure for Synchronous Counter.

Preparation of 1). State Diagram

2). State Table

3). State Assignment

4). Excitation Table (Consider which Memory Unit Using)

5). K-Map

6). Circuit Diagram

7. What is modulo-N counter?

A modulo–*n*counter will count n states. For example a mod-6 counter will count the sequence 000,001,010,011,100,101 and then recycles to 000. Mod -6 counter skips 110 and 111 states and it goes through only six different states.

****

DESIGN OF RIPPLE COUNTERS

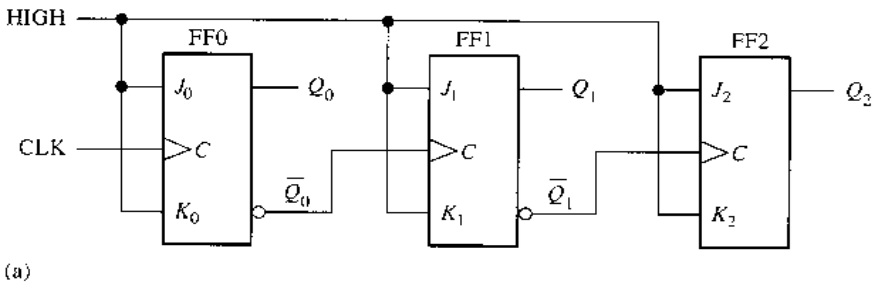
3-Bit Asynchronous Binary Counter/ modulo -7 ripple counter:

Design a 3-bit binary counter using T-flip flops. [NOV – 2019]

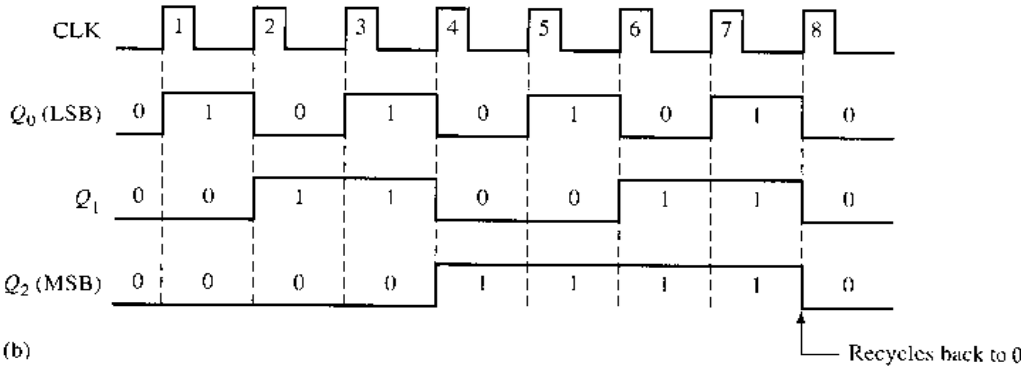
Explain about 3-Bit Asynchronous binary counter. ***(Nov -2009)***

The following is a three-bit asynchronous binary counter and its timingdiagram for one cycle. It works exactly the same way as a two-bitasynchronous binary counter mentioned above, except it has eight statesdue to the third flip-flop.

Clock Pulse	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0



(a)



(b)

Asynchronous counters are commonly referred to as ripple counters forthe following reason: The effect of the input clock pulse is first "felt" byFFO. This effect cannot get to FF1 immediately because of thepropagation delay through FF0. Then there is the propagation delaythrough FF1 before FF2 can be triggered. Thus, the effect of an inputclock pulse "ripples" through the counter, taking some time, due topropagation delays, to reach the last flip-flop.

****

ANALYSIS OF CLOCKED SEQUENTIAL CIRCUIT

Design and analyze of clocked sequential circuit with an example.

The analysis of a sequential circuit consists of obtaining a table or a diagram for the time sequence of inputs, outputs and internal states.

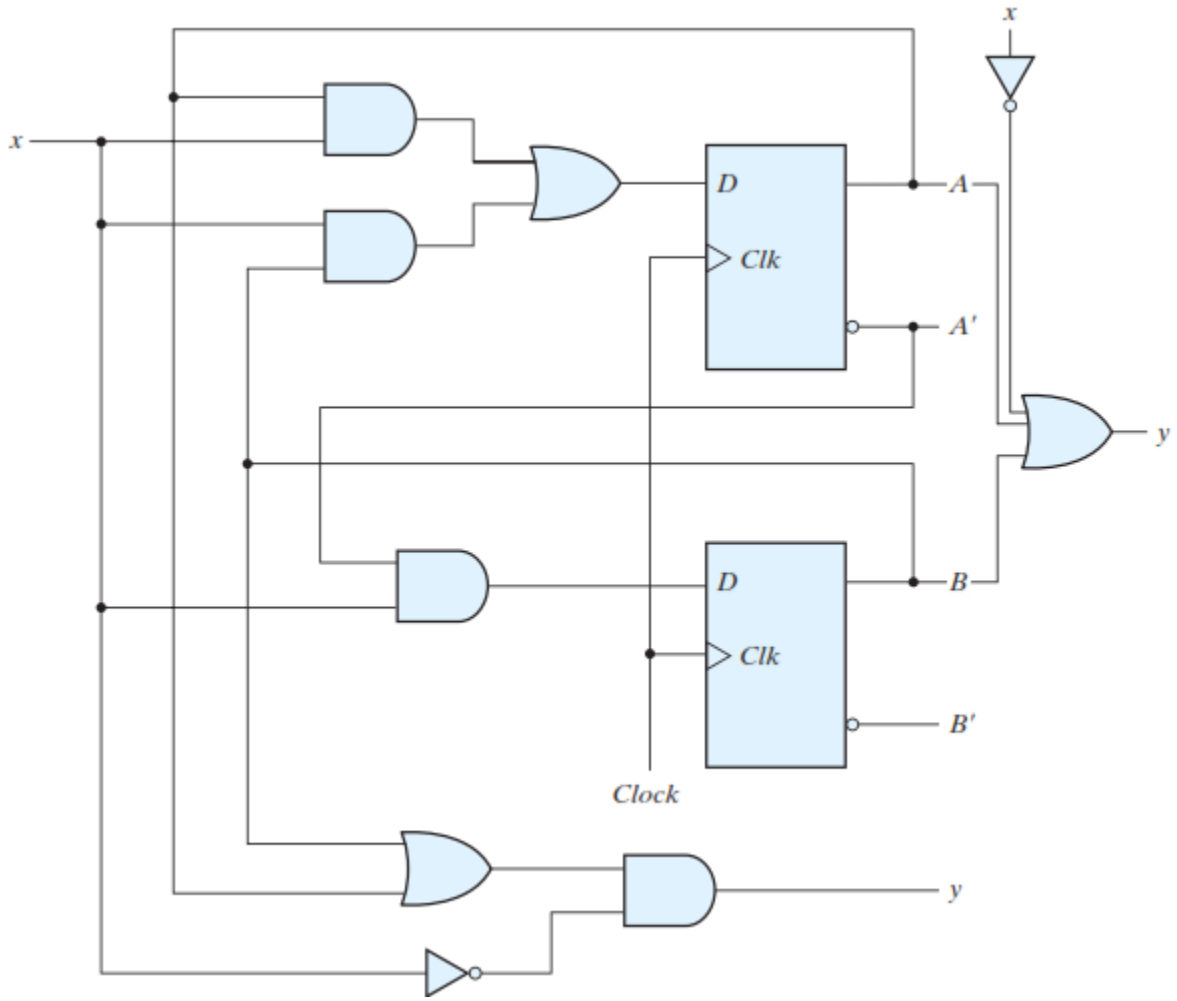


Fig: Example of sequential circuit

Consider the sequential circuit is shown in figure. It consists of two D flip-flops A and B, an input x and an output y.

A state equation specifies the next state as function of the present state and inputs.

$$A(n+1)= A(n)x(n)+B(n)x(n)$$

$$B(n+1)= A\overline{(n)}x(n)$$

They can be written in simplified form as,

$$A(n+1) = A\underline{x} + Bx$$

$$B(n+1) = Ax$$

The present state value of the output can be expressed algebraically as,

$$y(n)=(A+B)\overline{x}$$

DESIGN OF SYNCHRONOUS COUNTERS

Design and analyze of clocked sequential circuit with an example.

The procedure for designing synchronous sequential circuit is given below,

1. *From the given specification, Draw the state diagram.*
2. *Plot the state table.*
3. *Reduce the number of states if possible.*
4. *Assign binary values to the states and plot the transition table by choosing the type of Flip-Flop.*
5. *Derive the Flip flop input equations and output equations by using K-map.*
6. *Draw the logic diagram.*

State Diagram:

- State diagram is the *graphical representation of the information available in a state table.*
- In state diagram, a state is represented by a circle and the transitions between states are indicated by directed lines connecting the circles.

State Table:

- A state table gives the time sequence of inputs, outputs ad flip flops states. The table consists of four sections labeled present state, next state, input and output.
- The present state section shows the states of flip flops A and B at any given time 'n'. The input section gives a value of x for each possible present state.
- The next state section shows the states of flip flops one clock cycle later, at time n+1.

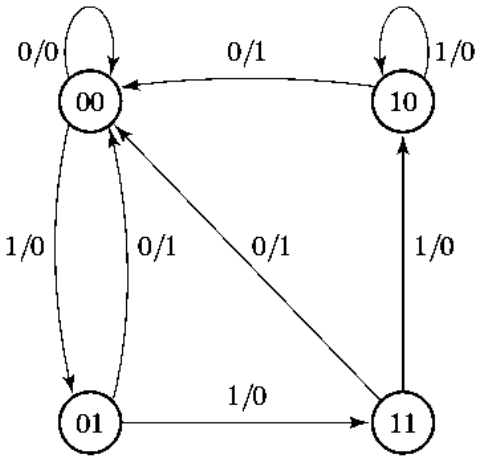
The state table for the circuit is shown. This is derived using state equations.

Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

The above state table can also be expressed in different forms as follows.

Present State		Next State				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

The state diagram for the logic circuit in below figure.



Flip-Flop Input Equations:

The part of the circuit that generates the inputs to flip flops is described algebraically by a set of Boolean functions called flip flop input equations.

The flip flop input equations for the circuit is given by,

$$D_A = Ax + Bx$$
$$D_B = Ax$$

****

TWO MARKS

1. **Define state diagram.**

   State diagram is the graphical representation of the information available in a state table. In state diagram, a state is represented by a circle and the transitions between states are indicated by directed lines connecting the circles.

2. **What is the use of state diagram?**

   i) Behavior of a state machine can be analyzed rapidly.

   ii) It can be used to design a machine from a set of specification.

3. **What is state table? (Nov 2018)**

A stable table is a table that represents relationship between inputs, outputs and flip-flop states, is called state table. Generally it consists of four section present state, next state, input and output.

4. **What is a state equation?**

A state equation also called, as an application equation is an algebraic expression that specifies the condition for a flip-flop state transition. The left side of the equation denotes the next state of the flip-flop and the right side, a Boolean function specifies the present state.

5. **Define sequential circuit.**

Sequential circuits are circuits in which the output variables dependent not only on the present input variables but they also depend up on the past output of these input variables.

6. **What do you mean by present state?**

The information stored in the memory elements at any given time defines the present state of the sequential circuit.

7. **What do you mean by next state?**

The present state and the external inputs determine the outputs and the next state of the sequential circuit.

8. **Define synchronous sequential circuit.**

SynchronousSequential circuits are circuits in which the signals can affect the memory elements only atdiscrete instant of time.

9. **What are the steps for the design of asynchronous sequential circuit?**

i) Construction of primitive flow table
ii) Reduction of flow table
iii) State assignment is made
iv) Realization of primitive flow table

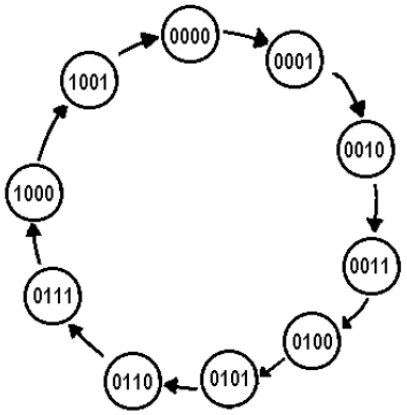
****

***Design of a Synchronous Decade Counter Using JK Flip- Flop* (Apr 2018, Nov 2018)**

A synchronous decade counter will count from zero to nine and repeat thesequence.

State diagram:

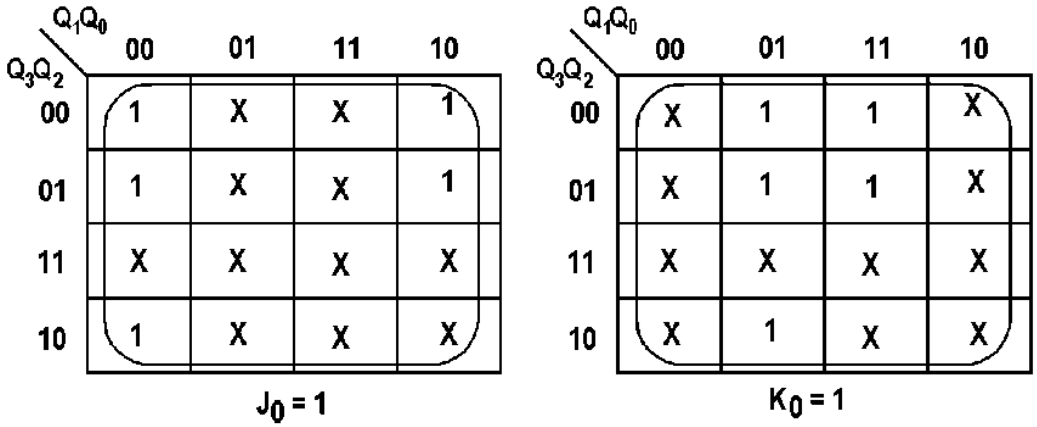
The state diagram of this counter is shown in Fig.



Excitation table:

Present State				Next State				Output							
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	0	0	0	0	X	1	0	X	0	X	X	1

K-Map:



Q_3Q_2 \ Q_1Q_0	00	01	11	10
00		1	X	X
01		1	X	X
11	X	X	X	X
10			X	X

$J_1 = \overline{Q}_3 Q_0$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	X	1	
01	X	X	1	
11	X	X	X	X
10	X	X	X	X

$K_1 = \overline{Q}_3 Q_0$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00			1	
01	X	X	X	X
11	X	X	X	X
10			X	X

$J_2 = Q_1 Q_0$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	X	X	X
01			1	
11	X	X	X	X
10			X	X

$K_2 = Q_1 Q_0$

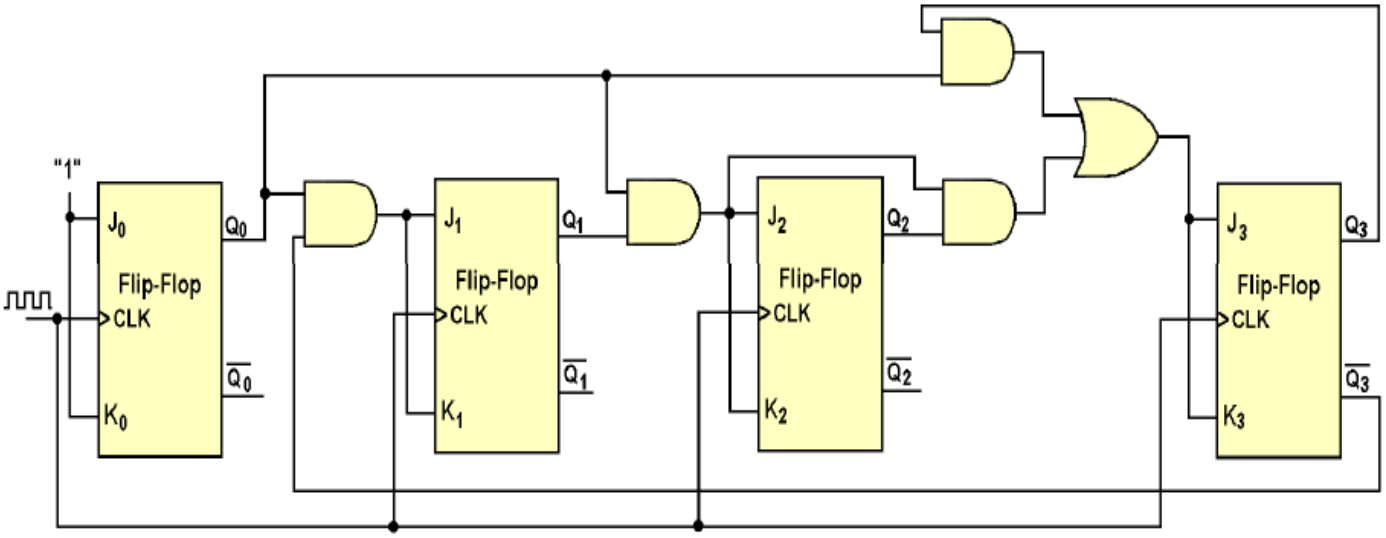
Q_3Q_2 \ Q_1Q_0	00	01	11	10
00				
01			1	
11	X	X	X	X
10	X	X	X	X

$J_3 = Q_3 Q_0 + Q_2 Q_1 Q_0$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	X	X	X	X
10		1	X	X

$K_3 = Q_3 Q_0 + Q_2 Q_1 Q_0$

Logic Diagram:

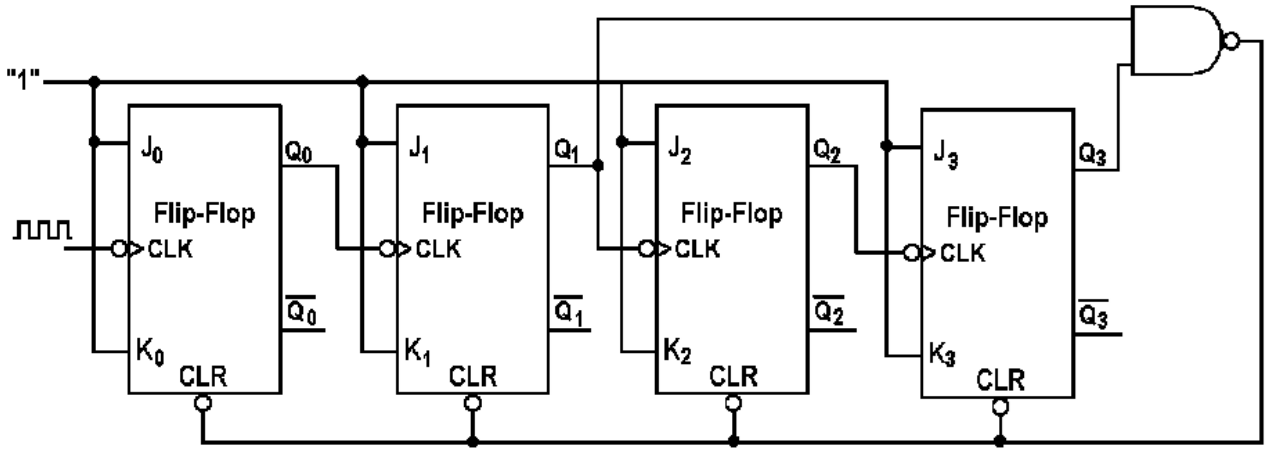


****

Design of an Asynchronous Decade Counter Using JK Flip- Flop.

An asynchronous decade counter will count from zero to nine and repeat thesequence. Since the JK inputs are fed from the output of previous flip-flop,therefore, the design will not be as complicated as the synchronous version.

At the ninth count, the counter is reset to begin counting at zero. The NAND gateis used to reset the counter at the ninth count. At the ninth count the outputs offlip-flop Q3 and Q1 will be high simultaneously. This will cause the output ofNAND to go to logic “0” that would reset the flip-flip. The logic design of thecounter is shown in Fig.

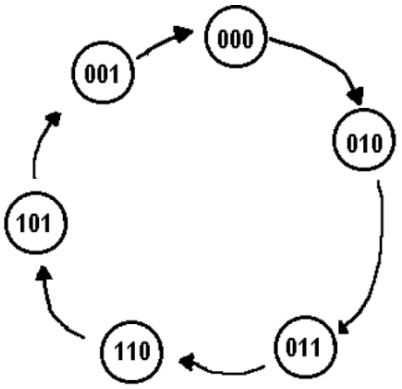


****

Design of a Synchronous Modulus-Six Counter Using SR Flip-Flop(Nov 2017)

The modulus six counters will count 0, 2, 3, 6, 5, and 1 and repeat the sequence.This modulus six counter requires three SR flip-flops for the design.

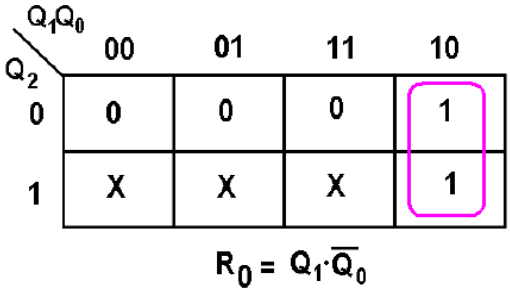
State diagram:



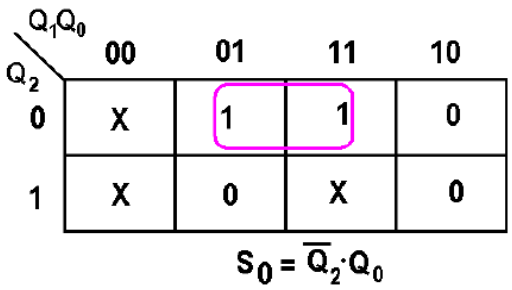
Truth table:

[Present State			Next State			Output					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	R_2	S_2	R_1	S_1	R_0	S_0
0	0	0	0	1	0	0	X	1	0	0	X
0	1	0	0	1	1	0	X	X	0	1	0
0	1	1	1	1	0	1	0	X	0	0	1
1	1	0	1	0	1	X	0	0	1	1	0
1	0	1	0	0	1	0	1	0	X	X	0
0	0	1	0	0	0	0	X	0	X	0	1

K-Map:



$R_0 = Q_1 \cdot \overline{Q_0}$



$S_0 = \overline{Q_2} \cdot Q_0$

Q_2 \ Q_1Q_0	00	01	11	10
0	1	0	X	X
1	X	0	X	0

$R_1 = \overline{Q}_1 \cdot \overline{Q}_0$

Q_2 \ Q_1Q_0	00	01	11	10
0	0	X	0	0
1	X	X	X	1

$S_1 = Q_2$

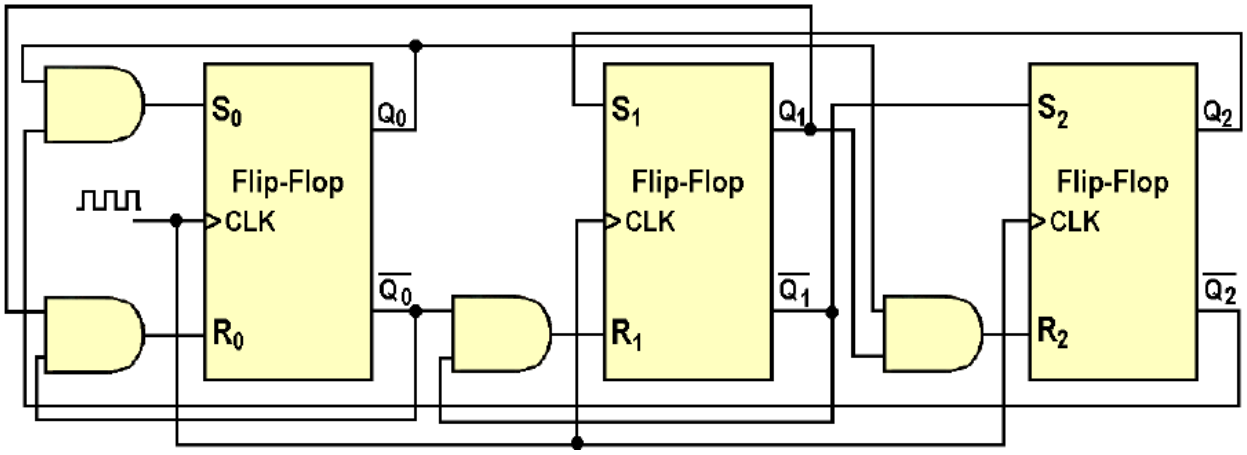
Q_2 \ Q_1Q_0	00	01	11	10
0	0	0	1	0
1	X	0	X	X

$R_2 = Q_1 \cdot Q_0$

Q_2 \ Q_1Q_0	00	01	11	10
0	X	X	0	X
1	X	1	X	0

$S_2 = \overline{Q}_1$

Logic Diagram:



SHIFT REGISTERS

Explain various types of shift registers. (or) Explain the operation of a 4-bit bidirectional shift register. (Or) What are registers? Construct a 4 bit register using D-flip flops and explain the operations on the register. (or) With diagram explain how two binary numbers are added serially using shift registers. (Apr – 2019)[NOV – 2019]

- A register is simply a group of Flip-Flops that can be used to store a binary number.
- There must be one Flip-Flop for each bit in the binary number.
- For instance, a register used to store an 8-bit binary number must have 8 Flip-Flops.
- The Flip-Flops must be connected such that the binary number can be entered (shifted) into the register and possibly shifted out.
- A group of Flip-Flops connected to provide either or both of these functions is called a ***shift register***.
- A register capable of shifting the binary information held in each cell to its neighboring cell in a selected direction is called a shift register.

- There are four types of shift registers namely:
    1. Serial In Serial Out Shift Register,
    2. Serial In Parallel Out Shift Register
    3. Parallel In Serial Out Shift Register
    4. Parallel In Parallel Out Shift Register

1.Serial In Serial Out Shift Register

- The block diagram of a serial out shift register is as below.

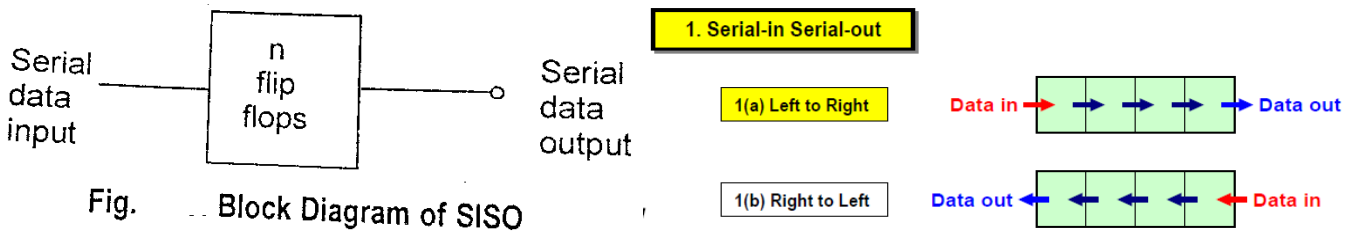


Fig. Block Diagram of SISO

- As seen, it accepts data serially .i.e., one bit at a time on a single input line. It produces the stored information on its single output also in serial form.
- Data may be shifted left using shift left register or shifted right using shift right register.

Shift Right Register

The circuit diagram using D flip-fops is shown in figure

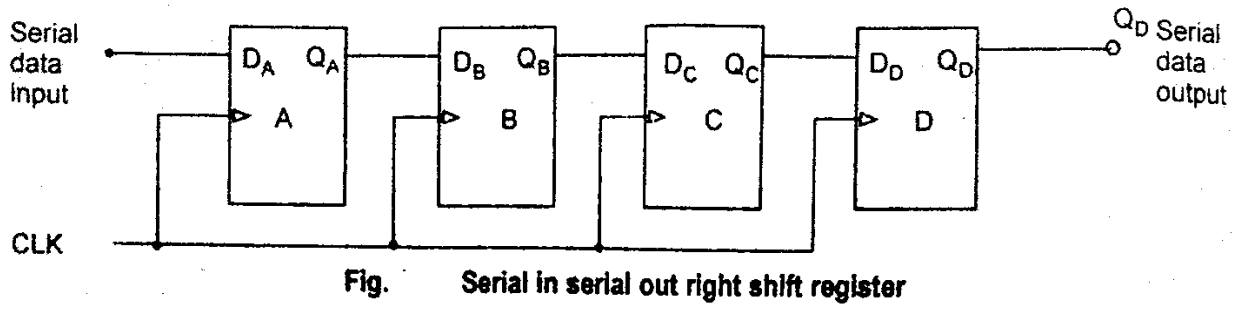


Fig. Serial in serial out right shift register

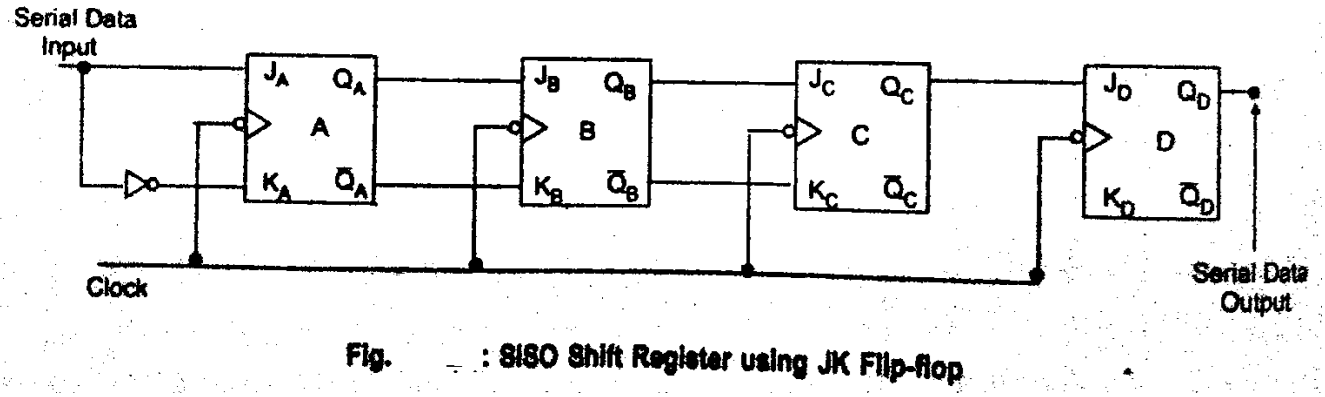


Fig. : SISO Shift Register using JK Flip-flop

- As shown in above figure,the clock pulse is applied to all the flip-flops simultaneously.
- The output of each flip-flop is connected to D input of the flip-flop at its right.
- Each clock pulse shifts the contents of the register one bit position to the right.
- New data is entered into stage A whereas the data presented in stage D are shifted out.

- For example, consider that all stages are reset and a steady logical 1 is applied to the serial input line.
- When the *first clock pulse* is applied, flip-flop A is set and all other flip-flops are reset.
- When the *second clock pulse* is applied,the '1' on the data input is shifted into flip-flop A and '1' that was in flip flop A is shifted to flip-flop B.
- This continues till all flip-flop sets.
- The data in each stage after each clock pulse is shown in table below

Shift Pulse	Serial Data Input	Q_A	Q_B	Q_C	Serial Output Q_D
0	1	0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	1	1	1	1	1

Shift Left Register

The figure below shows the shift left register using D flip-flops.

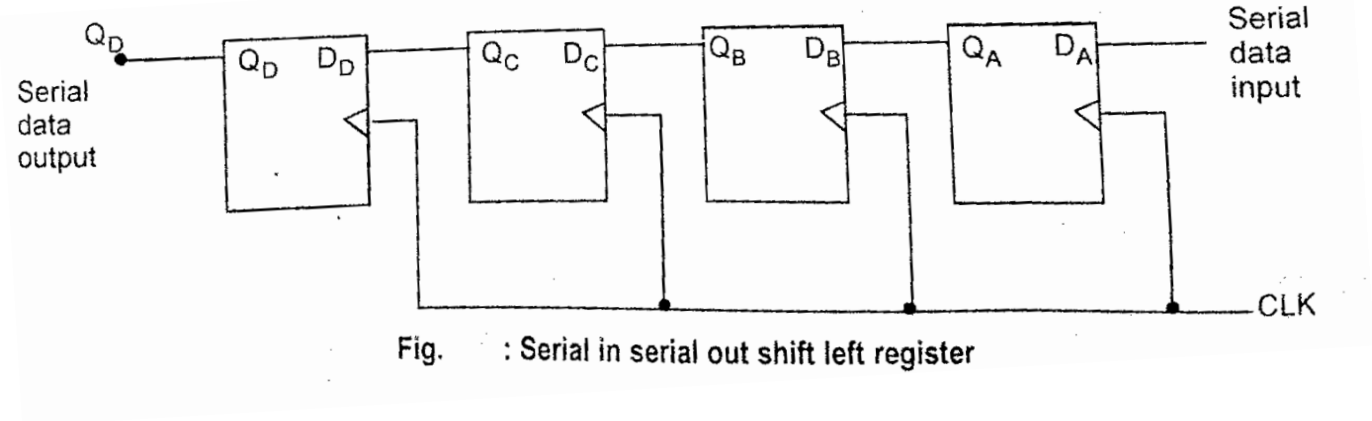
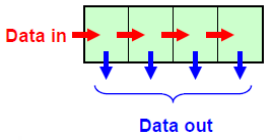


Fig. : Serial in serial out shift left register

- The clock is applied to all the flip-flops simultaneously. The output of each flip-flop is connected to D input of the flip-flop at its left.
- Each clock pulse shifts the contents of the register one bit position to the left.
- Let us illustrate the entry of the 4-bit binary number 1111 into the register beginning with the right most bit.
- When the *first clock pulse* is applied, flip flop A is set and all other flip-flops are reset.
- When *second clock pulse* is applied, '1' on the data input is shifted into flip-flop A and '1' that was in flip flop A is shifted toflip-flop B. This continues fill all flip-flop are set.
- The data in each stage after each clock pulse is shown in table below.

Q_D	Q_C	Q_B	Q_A	Serial Input Data	Clock Pulse
0	0	0	0	1	0
0	0	0	1	1	1
0	0	1	1	1	2
0	1	1	1	1	3
1	1	1	1	1	4

2. Serial in Parallel out shift register:



A 4 bit serial in parallel out shift register is shown in figure.

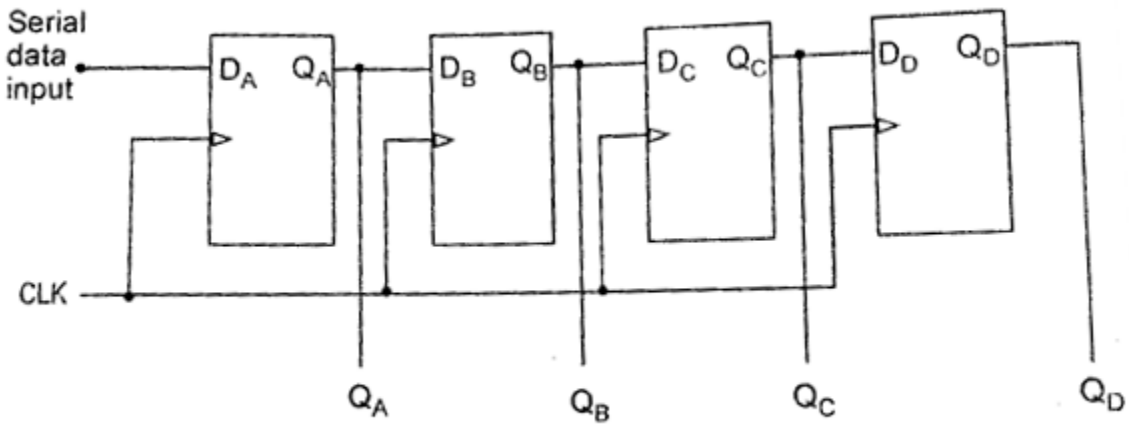


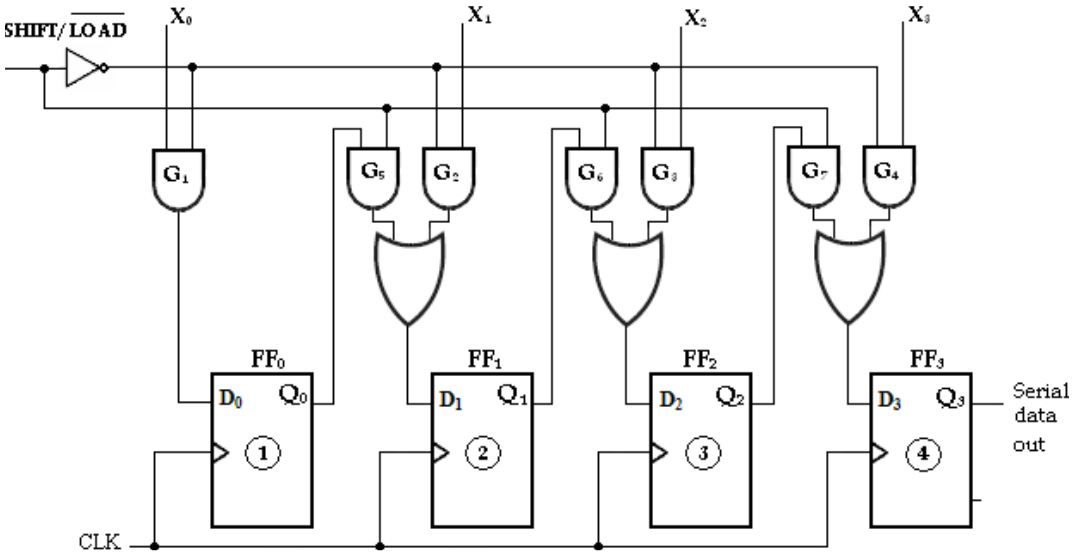
Fig. 3.42: Serial in parallel out shift register

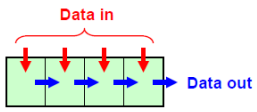
- It consists of one serial input and outputs are taken from all the flip-flops simultaneously.
- The output of each flip-flop is connected to D input of the flip-flop at its right. Each clock pulse shifts the contents of the register one bit position to the right.
- For example, consider that all stages are reset and a steady logical '1' is applied to the serial input line.
- When the *first clock pulse* is applied flip flop A is set and all other flip-flops are reset.
- When the *second pulse* is applied the '1' on the data input is shifted into flip flop A and '1' that was in flip flop A is shifted into flip-flop B. This continues till all flip-flops are set. The data in each stage after each clock pulse is shown in table below.

Shift Pulse	Serial Data Input	Parallel Outputs			
		Q_A	Q_B	Q_C	Q_D
0	1	0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	1	1	1	1	1

3. Parallel In Serial Out Shift register:

- For register with parallel data inputs, register the bits are entered simultaneously into their respective stages on parallel lines.
- A four bit parallel in serial out shift register is shown in figure. Let A,B,C and D be the four parallel data input lines and SHIFT/$\overline{\text{LOAD}}$ is a control input that allows the four bits of data to be entered in parallel or shift the serially.





- When SHIFTS/$\overline{\text{LOAD}}$ is low, gates G1 through G3 are enabled, allowing the data at parallel inputs to the D input of its respective flip-flop. When the clock pulse is applied the flip-flops with D=1 will set and those with D=0 will reset, thereby storing all four bits simultaneously.

- When SHIFT/$\overline{\text{LOAD}}$is high. AND gates G1 through G3 are disabled and gates G4 through G6are enabled, allowing the data bits to shifts right from one stage to next. The OR gates allow either the normal shifting operation or the parallel data entry operation, depending on which AND gates are enabled by the level on the SHIFT/$\overline{\text{LOAD}}$ input.

Parallel In Parallel OutShift Register:

- In parallel in parallel out shift register, data inputs can be shifted either in or out of the register in parallel.
- A four bit parallel in parallel out shift register is shown in figure.Let A, B, C, D be the four parallel data input lines and Q_A,Q_B,Q_C and Q_D be four parallel data output lines. The SHIFT/$\overline{\text{LOAD}}$ is the control input that allows the four bits data to enter in parallel or shift the serially.

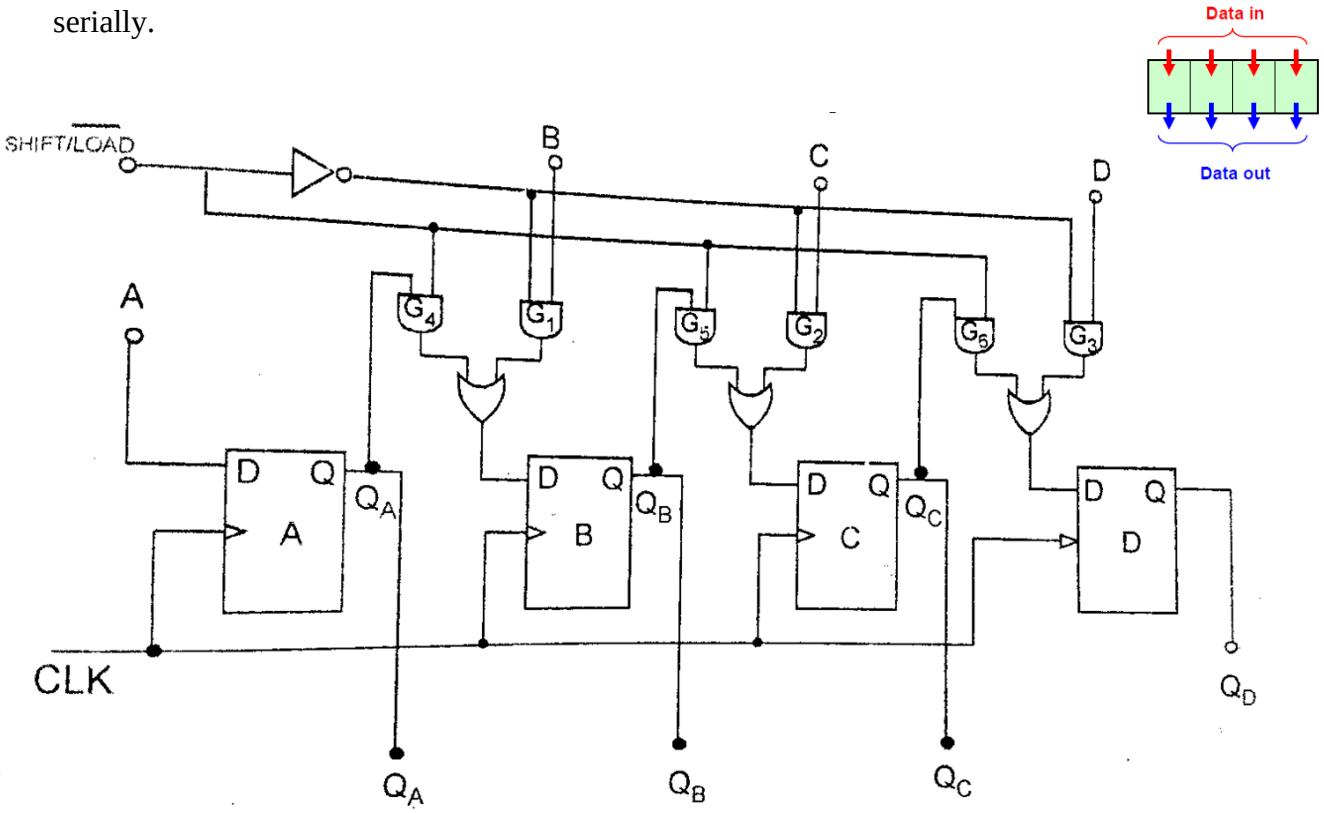


Fig. : Parallel in parallel out shift register

- When *SHIFT/$\overline{\text{LOAD}}$ is low*, gates G1 through G3 are enabled, allowing the data at parallel inputs to the D input of its respective flip-flop. When the clock pulse is applied, the flip-flops with D =1 will*set* those with D=0 will *reset* thereby storing all four bits simultaneously. These are immediately available at the outputs Q_A,Q_B,Q_C and Q_D.

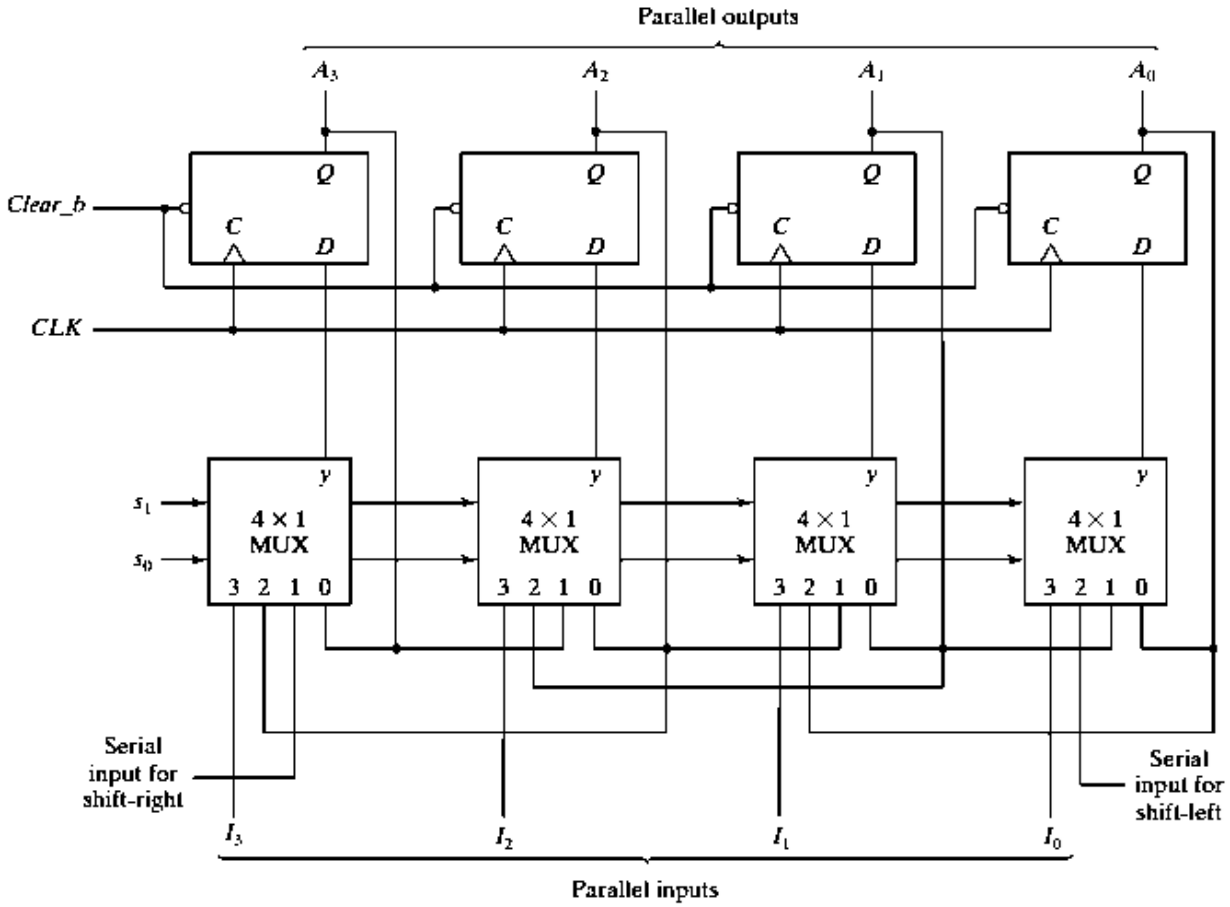
- When *SHIFT/$\overline{\text{LOAD}}$ is high*, gates G1, through G3 are disabled and gates G4 through G6 are enabled allowing the data bits to shift right from one stage to another. The OR gates allow either the normal shifting operation or the parallel data entry operation, depending on which AND gates are enabled by the level on the SHIFT/$\overline{\text{LOAD}}$ input.

****

Universal Shift Register:

Explain about universal shift register.(Apr -2018)

- A register that can shift data to right and left and also has parallel load capabilities is called universal shift register.
- It has the following capabilities.
    1. A clear control to clear the register to 0.
    2. A clock input to synchronize the operations.
    3. A shift right control to enable the shift right operation and the associated serial input and output lines.
    4. A shift left control to enable the shift left operation and the associated serial input and output lines.
    5. A parallel load control to enable a parallel transfer and the *n* input lines.
    6. *n* parallel output lines.
    7. A control state that leaves the information in the register unchanged in the presence of the clock.



- The diagram of 4-bit universal shift register that has all that capabilities listed above is shown in figure. It consists of four D flip-flop and four multiplexers.All the multiplexers have two common selection inputs S_1 and S_0. Input 0 is selected when S_1S_0=00, input 1 is selected when S_1S_0=01 and similarly for other two inputs.
- The selection inputs control the mode of operation of the register. When S_1S_0=00, the present value of the register is applied to the D inputs of the flip-flop. The next clock pulse transfers into each flip-flop the binary value it held previously, and no change of state occurs.
- When S_1S_0=01,terminal 1 of the multiplexer inputs has a path to be the D inputs of the flip-flops. This causes a shift right operation, with the serial input transferred into flip-flop A_3.
- When S_1S_0=10, a shift left operation results with the other serial input going into flip-flop A_0. Finally, when S_1 S_0 = 11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock edge. The function table is shown below.

Mode Control		
s_1	s_0	Register Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

****

SHIFT REGISTER COUNTERS:

Explain about Johnson and Ring counter. (Nov 2018)

Most common shift register counters are Johnson counter and ring counter.

Johnson counter:

- A 4 bit Johnson counter using D flip-flop is shown in figure. It is also called shift counter or twisted counter.

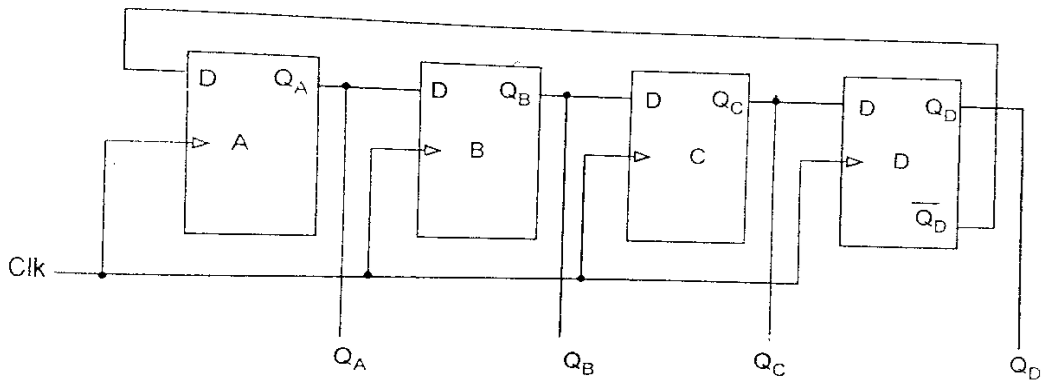


Fig. : Johnson Counter

- The output of each flip-flop is connected to D input of the next stage. The inverted output of last flip-flop $\overline{Q_D}$is connected to the D input of the first flip-flop A.
- Initially, assume that the counter is reset to 0. i.e., Q_A Q_B Q_C Q_D =0000. The value at D_B = D_C=D_D=0, whereas D_A =1 since $\overline{Q_D}$.
- When the *first clock pulse is* applied, the first flip-flop A is set and the other flip-flops are reset. i.e., Q_A Q_B Q_C Q_D =1000.
- When the *second clock pulse* is applies, the counter is Q_A Q_B Q_C Q_D = 1100. This continues and the counter will fill up with 1's from left to right and then it will fill up with 0's again.
- The sequence of states is shown in the table. As observed from the table, a 4-bit shift counter has 8 states. In general, an *n*-flip-flop Johnson counter will result in *2n* states.

Clock Pulse	Q_A	Q_B	Q_C	Q_D	$\overline{Q_D}$
0	0	0	0	0	1
1	1	0	0	0	1
2	1	1	0	0	1
3	1	1	1	0	1
4	1	1	1	1	0
5	0	1	1	1	0
6	0	0	1	1	0
7	0	0	0	1	0
0	0	0	0	0	1

The timing diagram of Johnson counter is as follows:

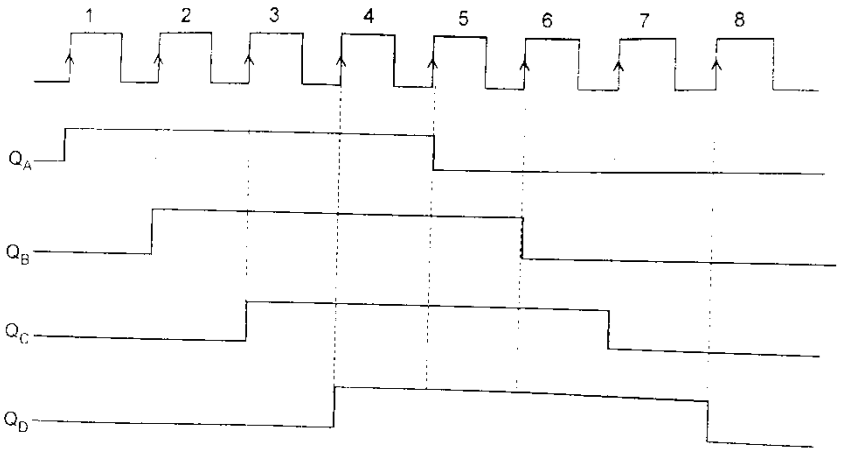


Fig. : Timing Diagram of Johnson Counter

Ring Counter:

A 4- bit ring counter using D Flip-Flop is shown in figure.

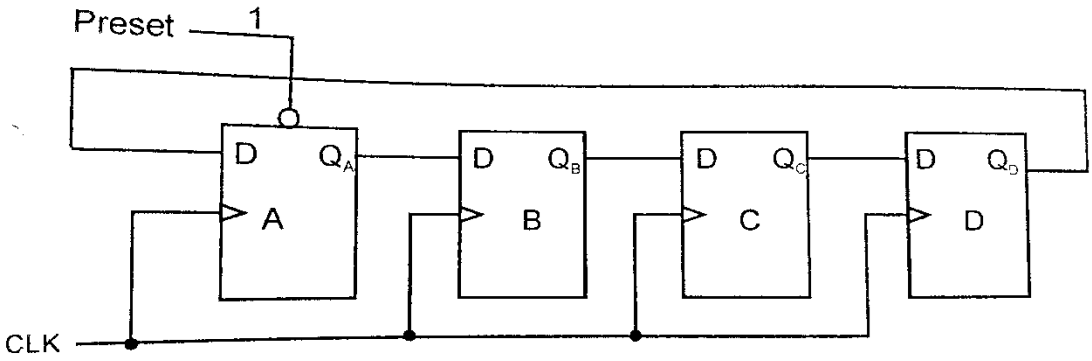


Fig. : Ring Counter

- As shown in figure, the true output of flip-flop D. i.e., Q_D is connected back to serial input of flip-flop A.
- Initially, 1 preset into the first flip-flop and the rest of the flip-flops are cleared i.e., $Q_AQ_BQ_CQ_D$=1000.
- When the *first clock pulse is applied,* the second flip-flop is set to 1while the other three flip flops are reset to 0.
- When the second clock pulse is applied, the '1' in the second flip-flop is shifted to the third flip-flop and so on.
- The truth table which describes the operation of the ring counter is shown below.

Clock Pulse	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
0	1	0	0	0

- As seen a 4-bit ring counter has 4 states. In general, an *n*-bit ring counter has *n* states. Since a single '1' in the register is made to circulate around the register, it is *called a ring counter.* The timing diagram of the ring counter is shown in figure.

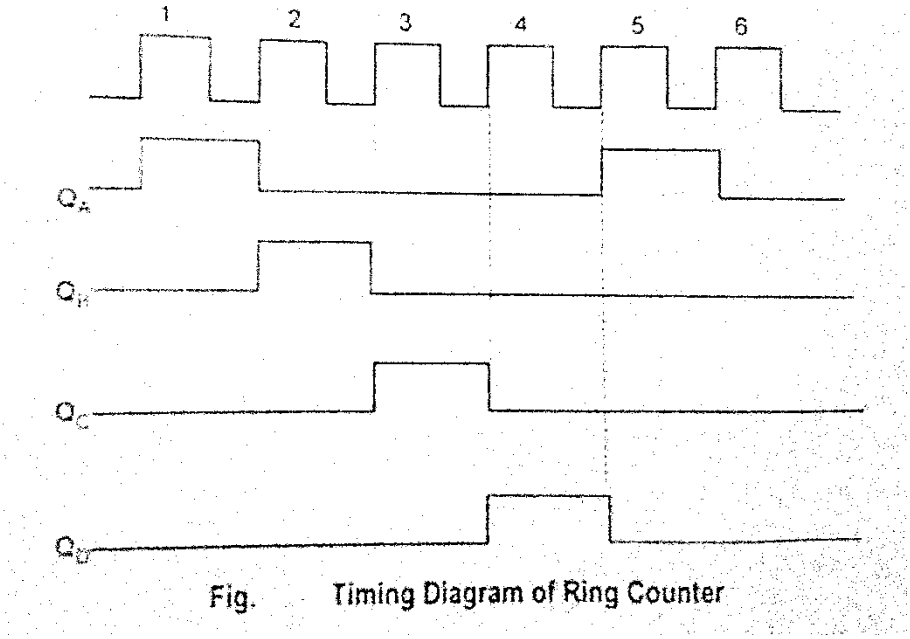


Fig. Timing Diagram of Ring Counter

****

TWO MARKS

1. **Define registers.**

A register is a group of flip-flops. An-bit register has a group of n flip-flops and is capable of storing any binary information/number containing n-bits.

2. **Define shift registers**.

A register capable of shifting its binary information in one or both directions is called as a shift register. It consists of a chain of flip flops in cascade, with the output of one flip flop connected to the input of the next flip-flop

3. **What are the different types of shift registers?[Nov 2010,April 2007,Apr 2018, Nov 2018]**

- ✓ Serial In Serial Out Shift Register
- ✓ Serial In Parallel Out Shift Register
- ✓ Parallel In Serial Out Shift Register
- ✓ Parallel In Parallel Out Shift Register
- ✓ Bidirectional Shift Register

4. **State the applications of shift register**.

Shift registers are widely used in

- ✓ Time delay circuits
- ✓ As Serial to parallel converter
- ✓ As Parallel to serial converters
- ✓ As Counters

5. **Define Shift Register Counter**.

A shift register can also be used as a counter. A shift register with the serial output connection back to the serial input is called Shift register counter

6. What is bi-directional shift register and unidirectional shift register?

A register capable of shifting both right and left is called bi-directional shift register. A register capable of shifting only one direction is called unidirectional shift register.

7. What are the two types of shift register counters?[April/May 2007,Nov/Dec 2006,2011,2012]

There are 2 types of shift Register counters are:

Ring counter:

A ring counter is a circular shift register with only one flip flop being set, at any particular time, all others are cleared.

Johnson counters:

The Johnson counter is K-bit switch-tail rings counter*2k* decodinggates to provide outputs for *2k* t i m i n g s i g n a l s .

8. How can a SIPO shift register is converted in to SISO shift register? (Apr/May 2010)

By taking output only on the Q output of last flip flop SIPO shift register is converted in to SISO shift register.

9. What is bi-directional shift register and unidirectional shift register?

A register capable of shifting both right and left is called bi-directional shift register. A register capable of shifting only one direction is called unidirectional shift register.

10. What is sequence generator?

The sequential circuit used to repeat a particular sequence repeatedly is called Sequence generator.

****

HDL FOR SEQUENTIAL CIRCUITS

Write coding in HDL for various flip-flops.

D Flip Flops

```
module DFF (q, d, clock, clock, reset);
input clock, reset, d;
output q;
reg q;
always @ (posedge clock, negedge reset)
begin
   if (~ reset)
            q <= 1'b0;
   else
            q <= d;
   end
end module
```

T Flip-Flop

```
module TFF (q, t, clock, reset);
input clock, reset, t;
output q;
reg q;
always @ (posedge clock, negedge reset)
begin
   if (~ reset)    (// same as if (reset==0))
   q <= 1'b0;
   else if (t)
   q <= ~q;
   end
end module
```

J-K Flip Flop

```
module JKFF (q, j, k, clock, reset);
input clock, reset, j, k;
output q;
```

```
reg q;
always @ (posedge clock, negedge reset)
begin
   if (~ reset)
   q <= 1'b0;
   else
            begin
                     case ({j, k})
                     2'b00 : q <= q;
                     2'b01 : q <= 0;
                     2'b10 : q <= 1;
                     2'b11 : q <= ~q;
                     end case
            end
    end
end module.
```

T flip flop from D flip flop and gates

```
module T_FF (Q, T, CLK, RST);
   output Q;
   input T, CLK, RST'
   wire DT
   assign DT = Q ^ T                  // T flip flop characteristic equation is Q(t + 1) = Q ⊕ T.
   DFF TF1 (Q, DT, CLK, RST);  //Instantiate D flip flop.
endmodule.
```

JK flip flop from D flip flop and gates

```
module JK_FF (Q, J, K, CLK, RST);
   wire JK;
   assign JK = (J&~Q) | (~K &Q); // JK flip flop characteristic equation is Q(t+1) = JQ̄+K̄Q
   //Instantiate D flip flop
   DFF JK1 (Q, JK, CLK, RST);
endmodule
```

Ripple Counter using T flip flop

```
module ripple counter (q, t, CLK, reset);
input t, CLK, reset ;
output [3:0] q;
// Instantiate t flip flop
   TFF t1 (q[0], t, CLK, reset);
   TFF t2 (q[1], t, q[0], reset);
   TFF t3 (q[2], t, q[1], reset);
   TFF t4 (q[3], t, q[2], reset);
end module
```

Synchronous Counter

```
module synchronous counter (CLK, reset, q);
input CLK, reset;
output [3:0]q;
reg [3:0]q;
always @ (posedge reset, posedge CLK)
  begin
        if (reset)
        q <= 4'b0000;
        else
        q <= q + 1'b1;
  end
end module
```

Ripple Counter using D-flip flop

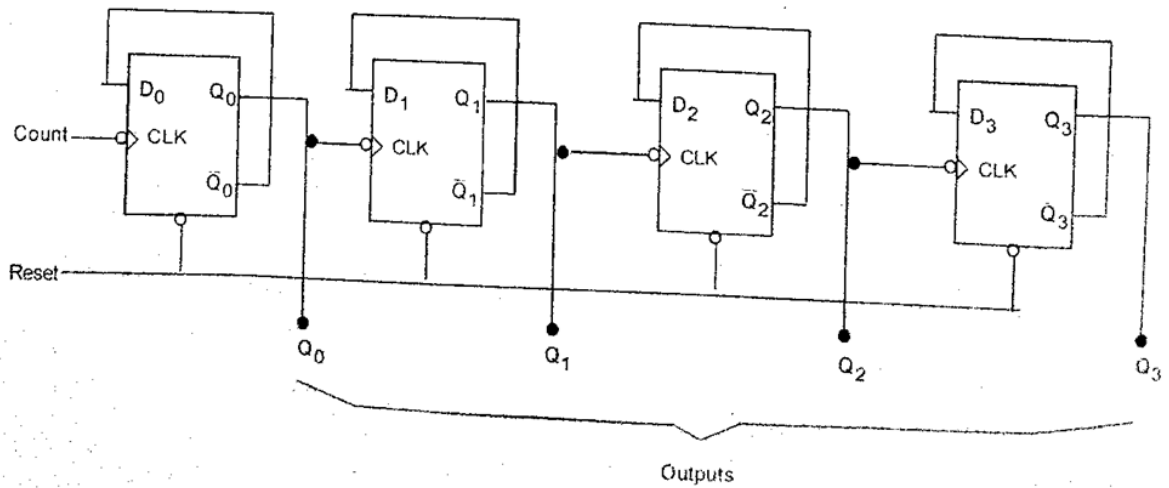


Fig. 3.50 Ripple counter using D-flipflop

```
module ripple_counter (Q3, Q2, Q1, Q0, count, reset);
   output Q3, Q2, Q1, Q0 ;
   input count, reset;
//instantiate complementing flip flop.
   comp_DFF F0 (Q0, count, reset);
   comp_DFF F1 (Q1, Q0, reset);
   comp_DFF F2 (Q2, Q1, reset);
   comp_DFF F3 (Q3, Q2, reset);
endmodule.
//complementing D-flip flop
module comp_DFF (Q, CLK, reset);
   output Q;
   input CLK, RESET;
   reg Q;
   always @(negedge CLK, posedge Reset)
   if (~Reset), Q <=1'b0;
     else Q <= #2 ~Q                    //intra-assignment delay
endmodule
```

Universal Shift Register

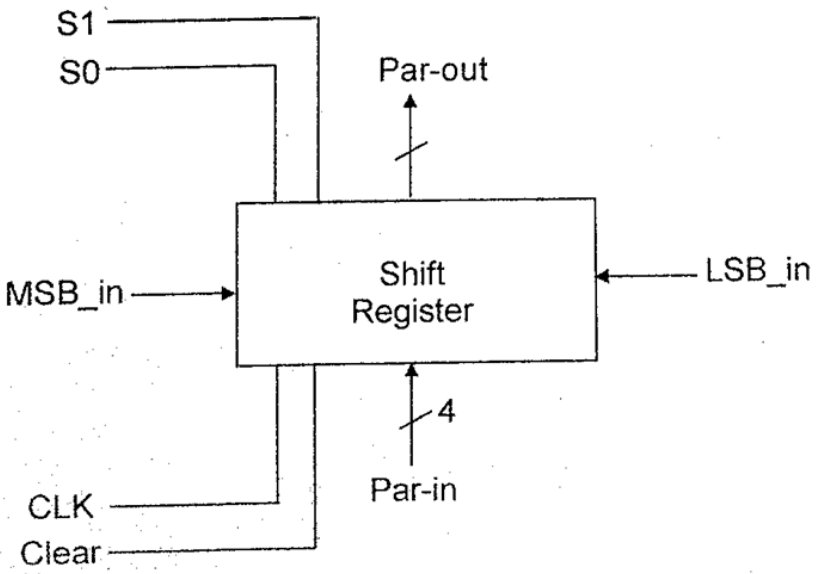


Fig. 3.51 Four Bit Universal Shift Register

Function Table		
Mode Control		
S_1	S_0	Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

```
module shift register (S1, S0, LSB_in, MSB_in, Par_in, CLK, Clear, par_out);
input S1, S0, LSB_in, MSB_in, Clk, Clear;
input [3:0] par_in;
output [3:0] par_out;
reg [3:0] par_out;
always @ (posedge CLK, negedge Clear)
begin
   if (~ clear)
            par_out ⇐ 4'b000;
   else
            case ({S1, S0})
            2'b00 : par_out ⇐ par_out;
            2'b01 : par_out ⇐ {MSB_in, par_out [3:1]};
            2'b10 : par_out ⇐ {par_out [2:0], LSB_in};
            2'b11 : par_out ⇐ par_in;
            endcase
   end
endmodule.
```

Test Bench:

D flip flop

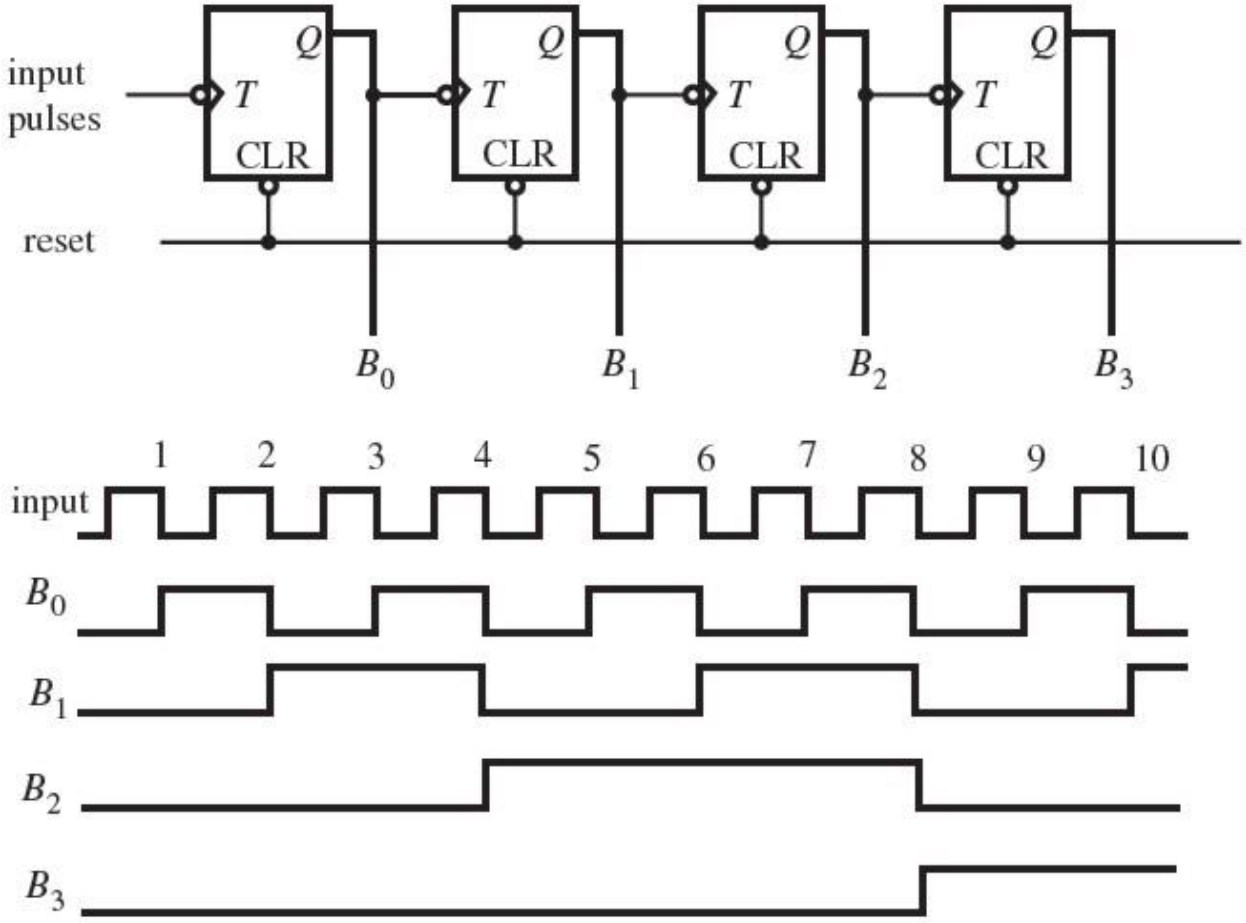
```
module DFF_test bench;
  wire tq;
  reg tclock, treset, td;
  DFF d1 (tclock, treset, td, tq);          //Instantiate D-flip flop module
  initial   begin
                td = 0;
                tclock = 0;
                treset = 0;
            #3  treset = 1;
            end
  always #3 tclock = ~tclock;
  always #5 d = ~d
  initial #100 $ stop;
endmodule.
```

Synchronous Counter

```
module synchronouscounter_test;
  wire [3;0]tq;
  reg tCLK, trest;
  synchronous counter SC1 (tclk, treset, tq);     //Instantiate synchronous counter module
  initial   begin
                tclk = 0;
                treset = 0;
            #5  treset = 1;
            #5  treset = 0;
            end
  always #5 tCLK = ~tCLK
  initial #200 $stop
endmodule
```

Write the VHDL Code for 4-Bit Binary Up Counter and explain. (Apr 2019)
VHDL Code for 4-Bit Binary Up Counter

The clock inputs of all the flip-flops are connected together and are triggered by the input pulses. Thus, all the flip-flops change state simultaneously (in parallel).



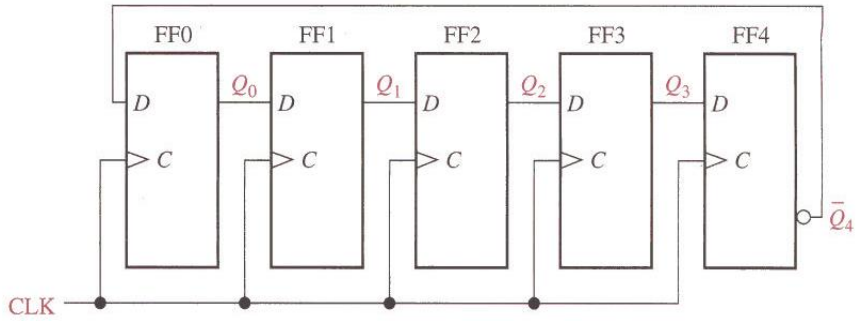

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity vhdl_binary_counter is
port(C, CLR : in std_logic;
Q : out std_logic_vector(3 downto 0));
end vhdl_binary_counter;
architecture bhv of vhdl_binary_counter is
signal tmp: std_logic_vector(3 downto 0);
begin
process (C, CLR)
begin
if (CLR=’1′) then
tmp <= "0000";
elsif (C’event and C=’1′) then
tmp <= tmp + 1;
```

```
end if;
end process;
Q <= tmp;
end bhv;
```

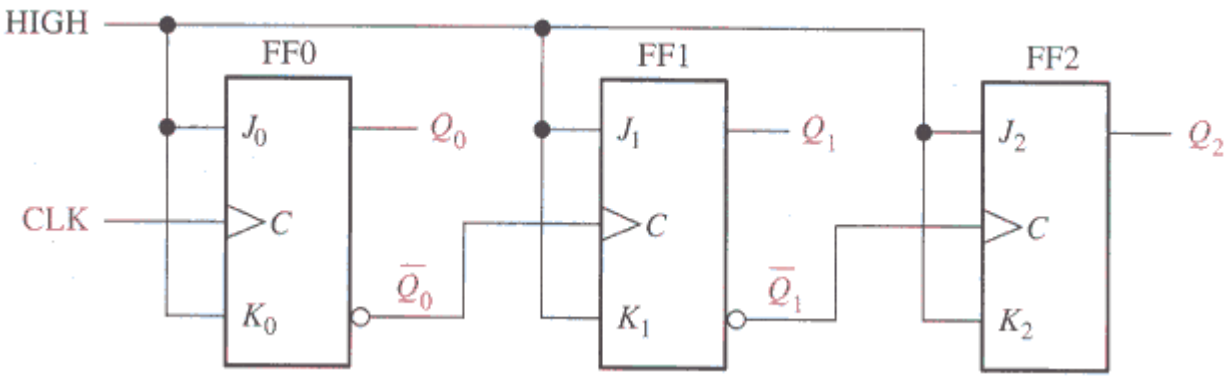
****

TWO MARKS

1. **Draw 5-bit Johnson counter.**



2. **Draw the diagram of 3-bit ripple counter.**



3. **Give few applications of shift register.**
   - ✓ Serial to parallel converter
   - ✓ Parallel to serial converter
   - ✓ As a counter
   - ✓ To introduce delay in a digital circuit.

****

UNIT III COMPUTER FUNDAMENTALS

Functional Units of a Digital Computer: Von Neumann Architecture – Operation and Operands of Computer Hardware Instruction – Instruction Set Architecture (ISA): Memory Location, Address and Operation – Instruction and Instruction Sequencing – Addressing Modes, Encoding of Machine Instruction – Interaction between Assembly and High Level Language.

1. FUNCTIONAL UNITS OF A DIGITAL COMPUTER

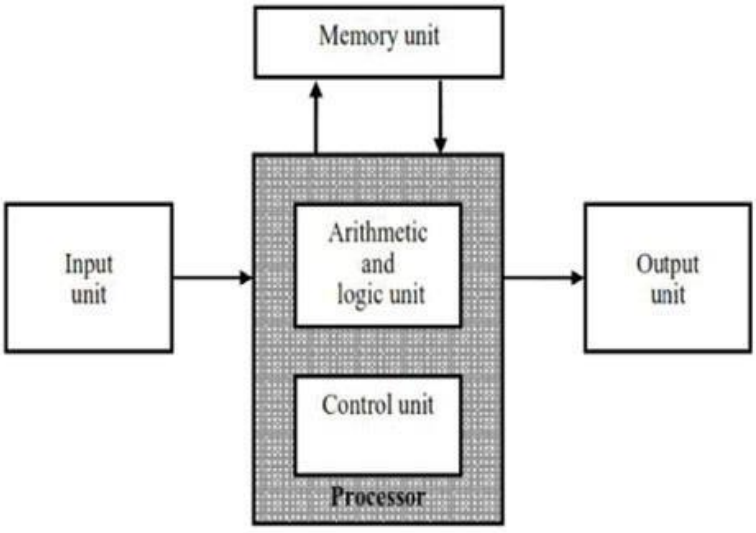
1. Explain in detail about the components of a computer system. (12 or 16) (Nov/Dec2014)(Nov/Dec2015) (May/June2016) (Nov/Dec 2016) Apr/ May 2018

Components of a computer system:

The five basic components of computer systems are,

- Input unit
- Output unit
- Arithmetic and logic unit
- Memory unit
- Control unit

- These units are interconnected by electrical cables to permit communication between them. This allows the computer to function as a system.



Input Unit:

- A computer must receive both data and program statements to function properly and be able to solve problems. The method of feeding data and programs to a computer is accomplished by an input device.

- Computer input devices read data from a source, such as magnetic disks, and translate that data into electronic impulses for transfer into the CPU. Some typical input devices are a keyboard, a mouse or a scanner.

Output Unit

- The output unit is the counterpart of the input unit. Its function is to send processed results to the outside world.
- The most familiar example of such a device is a printer. Printers employ mechanical impact heads, inkjet streams, or photocopying techniques, as in laser printers, to perform the printing. It produces printers capable of printing as **many as 10,000** lines per minute.
- This is a tremendous speed for a mechanical device but is still very slow compared to the electronic speed of a processor unit. Monitors, Speakers, Headphones and projectors are also some of the output devices.
- Some units, such as graphic displays, provide both an output function and an input function. The dual role of input and output of such units are referred with single name as I/O unit in many cases**.**
- **Speakers, Headphones and projectors are some of the output devices. Storage devices such as hard disk, floppy disk, flash drives** are also used for input as well as output.

Memory Unit

- The function of the memory unit is to store programs and data. There are two classes of storage, called **primary and secondary**. **Primary storage**is a fast memory that operates at electronic speeds.
- Programs must be stored in the memory while they are being executed. The memory contains a large number of semiconductor storage cells, each capable of storing one bit of information.
- These cells are rarely read or written as individual cells but instead are processed in groups of **fixed size called words.**
- The memory is organized so that the contents of one word, containing n bits, can be stored or retrieved in one basic operation.
- To provide easy access to any word in the memory, a distinct address is associated with each word location. Addresses are numbers that identify successive locations.
- A given word is accessed by specifying its address and issuing a control command that starts the storage or retrieval process. The number of bits in each word is often referred to as the word length of the computer.
- Typical **word lengths range from 16 to 64 bits.** The capacity of the memory is one factor that characterizes the size of a computer.

- Programs must reside in the memory during execution. Instructions and data can be written into the memory or read out under the controller of the processor.
- It is essential to be able to access any word location in the memory as quickly as possible. Memory in which any location can be reached in a short and fixed amount of time after specifying its address is **called random-access Memory (RAM).**
- The time required to access one word is called the **memory access time.** This time is fixed, independent of the location of the word being accessed. It typically ranges from a few nanoseconds (ns) to about 100 ns for modem RAM units.
- The memory of a computer is normally implemented as a Memory hierarchy of three or four levels of semiconductor RAM units with different speeds and sizes.
- The small, fast, RAM units are called **caches**. They are tightly coupled with the processor and are often contained on the same integrated circuit chip to achieve high performance.
- The largest and slowest unit is referred to as the main Memory. Although primary storage is essential, it tends to be expensive.

Thus additional, cheaper, secondary storage is used when large amounts of data and many programs have to be stored, particularly for information that is access infrequently. A wide selection of secondary storage deviceis available, including magnetic disks and tapes and optical disks

Arithmetic and Logic Unit(ALU):

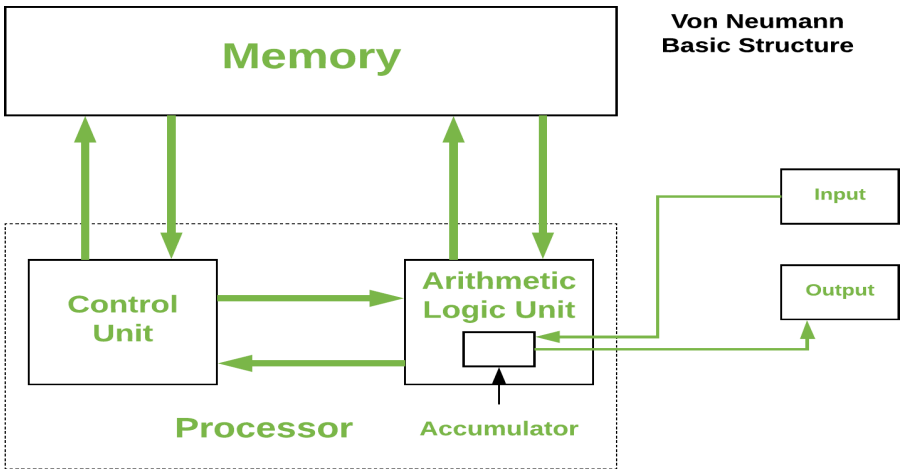
- **ALU** is a digital circuit that performs two types of operations**arithmetic** and **logical**.
- Arithmetic operations are the fundamental mathematical operations consisting of addition, subtraction, multiplication and division. Logical operations consists of comparisons. (i.e) Two pieces of data are compared to see whether one is equal to, less than, or greater than the other.
- The ALU is afundamental building block of the central processing unit of a computer. Memory enables a computer to store, at least temporarily, data and programs.
- Memory also known as the primary storage or main memory - is a part of the microcomputer that holds data for processing, instructions for processing the data (the program) and information (processed data).
- Part of the contents of the memory is held only temporarily. (i.e)It is stored only as long as the microcomputer is turned on. When you turn the machine off, the contents are lost.
- The control unit instructs the arithmetic-logic unit which operation to perform and then sees that the necessary numbers are supplied. The control and arithmetic & logic units are many times faster than other devices connected to a computer system.

Control Unit (CU):

- It is the part of a CPU that directs its operation. The control unit instructs the rest of the computer system how to carry out a program‘s instructions.
- It directs the movement of electronic signals between memories, which temporarily holds data, instructions & processed information and the ALU.
- It also directs these control signals between the CPU and input/output devices. The control unit is the circuitry that controls the flow of information through the processor, and coordinates the activities of the other units within it.

1.1 VON NEUMANN ARCHITECTURE

- V**on Neumann Architecture** also known as the Von Neumann model, the computer consisted of a CPU, memory and I/O devices.
- The program is stored in the memory. The CPU fetches an instruction from the memory at a time and executes it.
- Thus, the instructions are executed sequentially which is a slow process. Neumann m/c are called control flow computer because instruction are executed sequentially as controlled by a program counter.
- To increase the speed, parallel processing of computer have been developed in which serial CPU‘s are connected in parallel to solve a problem. Even in parallel computers, the basic building blocks are Neumann processors.
- The von Neumann architecture is a design model for a stored-program digital computer that uses a processing unit and a single separate storage structure to hold both instructions and data.
- It is named after mathematician and early computer scientist John von Neumann.
- Such a computer implements a universal Turing machine, and the common ―referential model‖ of specifying sequential architectures, in contrast with parallel architectures.



2. OPERATION AND OPERANDS OF COMPUTER HARDWARE INSTRUCTION

2. Explain about operations operands of computer hardware instruction.

Operation of the computer hardware

- Every computer must be able to perform arithmetic. The MIPS assembly language notation

  add a, b, c

- Instructs a computer to add the two variables b and c and to put their sum in a. This notation is rigid in that each MIPS arithmetic instruction performs only one operation and must always have exactly three variables.
- The following code shows an equivalent MIPS code**: ADD $s1, $s2, $s3**the sum of b and c is placed in a.
- Here, the variables a,b and c are assumed to be stored in the register $s1, $s2 and $s3 all arithmetic immediate value are signed extended.

MIPS Assembly Language

MIPS Operands

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Three register operands
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Three register operands
	add immediate	`addi $s1,$s2,20`	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	`lw $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	`sw $s1,20($s2)`	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	`lh $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	`lhu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	`sh $s1,20($s2)`	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	`lb $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	`lbu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	`sb $s1,20($s2)`	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	`ll $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	`sc $s1,20($s2)`	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	`lui $s1,20`	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	`and $s1,$s2,$s3`	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	`or $s1,$s2,$s3`	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	`nor $s1,$s2,$s3`	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	`andi $s1,$s2,20`	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	`ori $s1,$s2,20`	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	`sll $s1,$s2,10`	$s1 = $s2 << 10	Shift left by constant
	shift right logical	`srl $s1,$s2,10`	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	`beq $s1,$s2,25`	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1,$s2,25`	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for `beq`, `bne`
	set on less than unsigned	`sltu $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	`slti $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	`sltiu $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to $ra	For switch, procedure return
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000	For procedure call

Name	Example	Comments
32 registers	`$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at`	Fast locations for data. In MIPS, data must be in registers to perform arithmetic, register `$zero` always equals 0, and register `$at` is reserved by the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

- All arithmetic operations have exactly three operands, no more and no less, conforms have this conforms to the philosophy of keeping the hardware simple. This situation illustrates the first of four underlying principles of hardware design.

Design Principle 1: Simplicity favors regularity.

Compiling Two C Assignment Statements into MIPS

Example 1:

- This segment of a C program contains the five variables a, b, c, d, and e. Since Java evolved from C, this example and the next few work for either high-level programming language:

```
a = b + c;
d = a - e;
```

Answer

- The translation from C to MIPS assembly language instructions are performed by the compiler. Show the MIPS code produced by a compiler.
- A MIPS instruction operates on two source operands and places the result in one destination operand.
- Hence, the two simple statements above compile directly into these two MIPS assembly language instructions:

```
add a, b, c
sub d, a, e
```

Compiling a complex C Assignment into MIPS

Example 2:

- A somewhat complex statement contains the five variables f, g, h, i, and j:
- What might a C compiler produce?

```
f = (g + h) - (i + j);
```

Answer

- The compiler must break this statement into several assembly instructions, since only one operation is performed per MIPS instruction.
- The first MIPS instruction calculates the sum of g and h. We must place the result somewhere, so the compiler creates a temporary variable, called t0:

```
add t0,g,h # temporary variable t0 contains g + h
```

- Although the next operation is subtract, we need to calculate the sum of i and j before we can subtract.
- Thus, the second instruction places the sum of i and j in another temporary variable created by the compiler, called t1:

```
add t1,i,j  # temporary variable t1 contains i + j
```

- Finally, the subtract instruction subtracts the second sum from the first and places the difference in the variable f, completing the compiled code:

```
sub f,t0,t1 # f gets t0 - t1, which is (g + h) - (i + j)
```

Note : ‘ #’symbol indicate the comment line

Operands of the Computer Hardware

Over View

- **Memory operand**
- **Constant or immediate operands**
- **Index register**

- In MIPS instruction set architecture, operand can either in register or memory. Most of the arithmetic and logical instructions use register operands.
- Registers are limited number of special location built directly in hardware and they are visible to the programmer when the computer is completed.
- The size of a register in the MIPS architecture is 32 bits; groups of 32 bits occur so frequently that they are given the name **word** in the MIPS architecture**.**
- **Word**the natural unit of access in a computer, usually a group of 32 bits; corresponds to the size of a register in the MIPS architecture.
- The reason for the limit of 32 registers may be found in the second of our four underlying design principles of hardware technology:

Design Principle 2: Smaller is faster.

- A very large number of registers may increase the clock cycle time simply because it takes electronic signals longer when they must travel farther.
- Use fewer register to conserve energy.

Example:

Compiling a C Assignment Using Registers

- It is the compiler‘s job to associate program variables with registers. Take, for instance, the assignment statement from our earlier example:

```
f = (g + h) - (i + j);
```

- The variables f, g, h, i, and j are assigned to the registers $s0, $s1, $s2, $s3, and $s4, respectively. What is the compiled MIPS code?

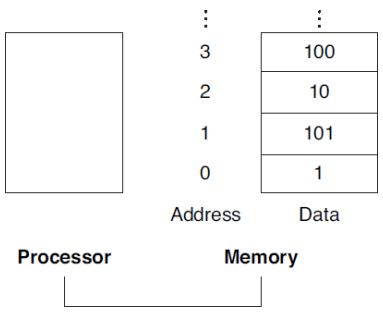
Answer

- The compiled program is very similar to the prior example, except we replace the variables with the register names mentioned above plus two temporary registers, $t0 and $t1, which correspond to the temporary variables above:

```
add $t0,$s1,$s2 # register $t0 contains g + h
add $t1,$s3,$s4 # register $t1 contains i + j
sub $s0,$t0,$t1 # f gets $t0 - $t1, which is (g + h)-(i + j)
```

Memory Operands

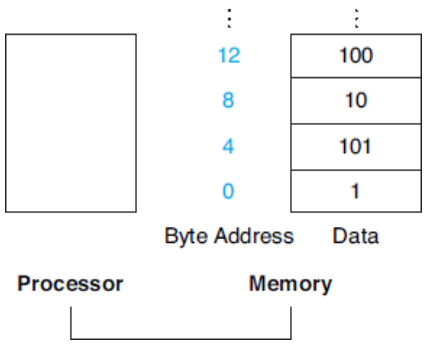
- Programming languages have simple variables that contain single data elements, as in these examples, but they also have more complex data structures—arrays and structures.
- These complex data structures can contain many more data elements than there are registers in a computer.
- The processor can keep only a small amount of data in registers, but computer memory contains billions of data elements.
- Hence, data structures (arrays and structures) are kept in memory.
- As explained above, arithmetic operations occur only on registers in MIPS instructions; thus, MIPS must include instructions that transfer data between memory and registers. Such instructions are called**data transfer instructions**.
- To access a word in memory, the instruction must supply the memory **address**.
- Memory is just a large, single-dimensional array, with the address acting as the index to that array, starting at 0. For example, in the followingFigure, the address of the third data element is 2, and the value of Memory[2] is 10



Memory addresses and contents of memory at those locations

- The data transfer instruction that copies data from memory to a register is traditionally called **load.**
- The format of the load instruction is the name of the operation followed by the register to be loaded, then a constant and register used to access memory.

- The sum of the constant portion of the instruction and the contents of the second register forms the memory address. The actual MIPS name for this instruction is lw, standing for l**oad word.**



Actual MIPS memory addresses and contents of memory for those words.

Alignment restriction

- In MIPS, words must start at addresses that are multiples of 4. This requirement is called an **alignment restriction**
- alignment restriction A requirement that data be aligned in memory on natural boundaries .many architecture have alignment restriction

Big endian and little Endian

- 8 bit bytes are divided into two parts:
- Address of the left most byte is called –big endian‖ and right most byte is called –little endian –

Compiling an Assignment When an Operand Is in Memory

Example 1:

- Let‘s assume that A is an array of 100 words and that the compiler has associated the variables g and h with the registers $s1 and $s2 as before.
- Let‘s also assume that the starting address, or base address, of the array is in $s3. Compile this C assignment statement:

```
g = h + A[8];
```

MIPS Code

- In the given statement, there is a single operation. Whereas, one of the operands is in memory, so we must carry this operation in two steps:

  Step 1: load the temporary register($s3) + 8

  Step 2: perform addition with h(($s2)), and store result in g($s1)

```
lw   $t0,8($s3) # Temporary reg $t0 gets A[8]

add  $s1,$s2,$t0 # g = h + A[8]
```

- The constant in a data transfer instruction (8) is called the offset, and the register added to form the address ($s3) is called the**base register.**

Example 2:

Compiling Using Load and Store

- What is the MIPS assembly code for the C assignment statement below?

```
A[12] = h + A[8];
```

- Assume variable h is associated with register $s2 and the base address of the array A is in $s3.

MIPS code

```
lw   $t0,32($s3)  # Temporary reg $t0 gets A[8]
add  $t0,$s2,$t0  # Temporary reg $t0 gets h + A[8]
```

- The final instruction stores the sum into A[12], using 48 (4 × 12) as the offset and register $s3 as the base register

```
sw   $t0,48($s3)  # Stores h + A[8] back into A[12]
```

- Load word and store word are the instructions that copy words between memory and registers in the MIPS architecture. Other brands of computers use other instructions along with load and store to transfer data.

Constant or Immediate Operands

- Constant variables are used as one of the operand for many arithmetic operation in MIPS architecture
- The constants would have been placed in memory whenthe program was loaded.
- To avoid load instruction used in arithmetic instruction we can use one operand is a constant
- This quick add instruction with one constant operand is called add immediate or addi. To add 4 to register $s3, we just write

Design Principle 3: Make the common case fast.

```
lw $t0, AddrConstant4($s1)  # $t0 = constant 4
add $s3,$s3,$t0             # $s3 = $s3 + $t0 ($t0 == 4)
```

- Assuming that $s1 + AddrConstant4 is the memory address of the constant 4.

```
addi    $s3,$s3,4           # $s3 = $s3 + 4
```

Advantage of constant operands

- It uses less energy
- It performs operation in more fast

Index register

- The register in the data transfer instructions was originally invented to hold an index of an array with the offset used for the starting address of an array. Thus, the base register is also called the index register.

3. INSTRUCTION SET ARCHITECTURE (ISA)

3. Discuss about ISA.

- The addressing methods that are commonly used for accessing operands in memory locations and processor registers are also presented.
- The emphasis here is on basic concepts. We use a generic style to describe machine instructions and operand addressing methods that are typical of those found in commercial processors.
- A sufficient number of instructions and addressing methods are introduced to enable us to present complete, realistic programs for simple tasks.
- These generic programs are specified at the assembly-language level, where machine instructions and operand addressing information are represented by symbolic names.
- A complete instruction set, including operand addressing methods, is often referred to as the instruction set architecture (ISA) of a processor.
- The vast majority of programs are written in high-level languages such as C, C++, or Java.
- To execute a high-level language program on a processor, the program must be translated into the machine language for that processor, which is done by a compiler program.
- Assembly language is a readable symbolic representation of machine language.

3.1 Memory Locations and Addresses

- The memory consists of many millions of storage cells, each of which can store a bit of information having the value 0 or 1.
- Because a single bit represents a very small amount of information, bits are seldom handled individually.
- The usual approach is to deal with them in groups of fixed size. For this purpose, the memory is organized so that a group of n bits can be stored or retrieved in a single, basic operation.
- Each group of n bits is referred to as a word of information, and n is called the word length. The memory of a computer can be schematically represented as a collection of words, Modern computers have word lengths that typically range from 16 to 64 bits.
- If the word length of a computer is 32 bits, a single word can store a 32-bit signed number or four ASCII-encoded characters, each occupying 8 bits, as shown in Figure.
- A unit of 8 bits is called a byte. Machine instructions may require one or more words for their representation.

- We will discuss how machine instructions are encoded into memory words in a later section, after we have described instructions at the assembly-language level.
- Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires distinct names or addresses for each location.
- It is customary to use numbers from 0 to 2k − 1, for some suitable value of k, as the addresses of successive locations in the memory.
- Thus, the memory can have up to 2k addressable locations. The 2k addresses constitute the address space of the computer. For example, a 24-bit address generates an address space of 224 (16,777,216) locations.
- This number is usually written as 16M (16 mega), where 1M is the number 220 (1,048,576). A 32-bit address creates an address space of 232 or 4G (4 giga) locations, where 1G is 230.

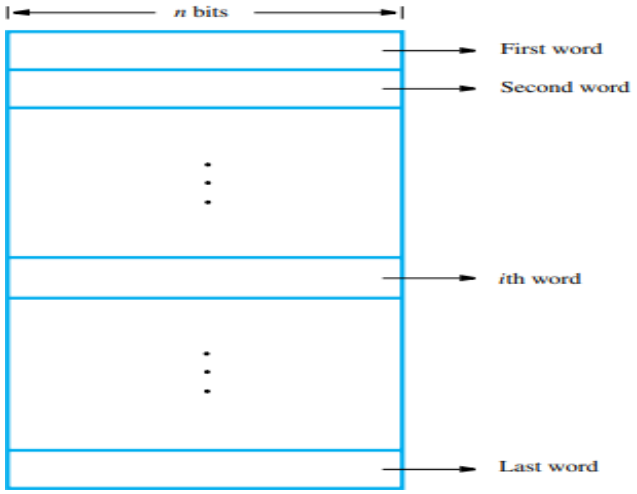


Fig 3.1 Memory words

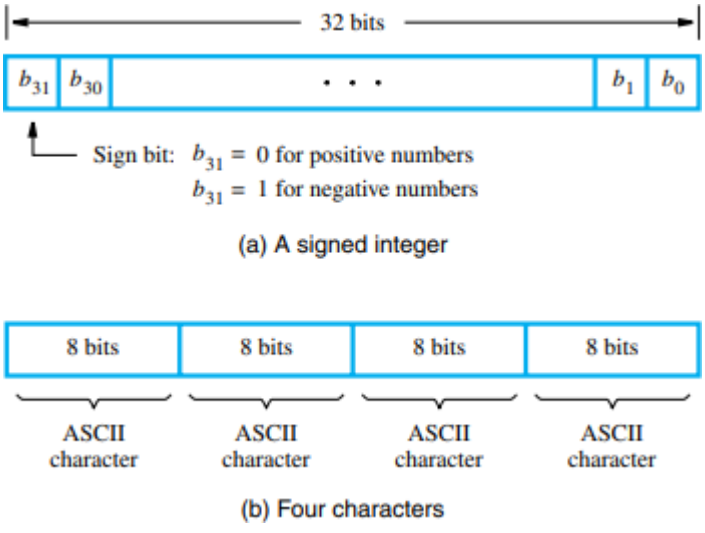


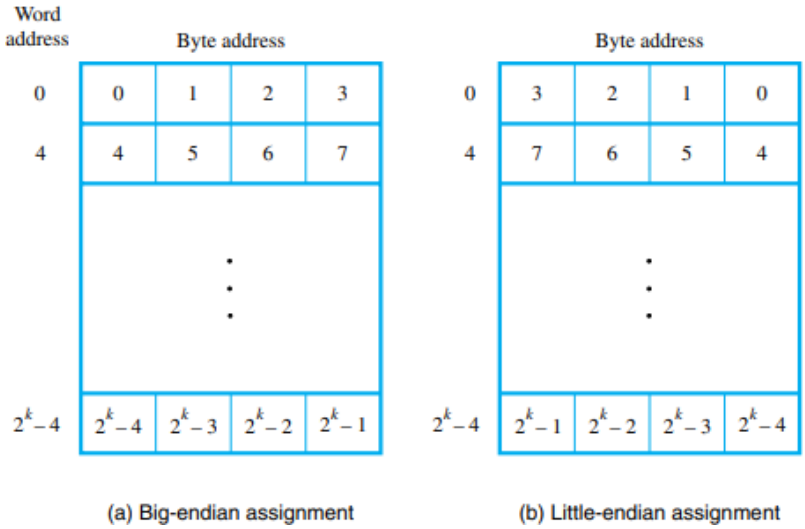
Figure 3.2 Examples of encoded information in a 32-bit word

3.1.1 Byte Addressability

- A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits. 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. This is the assignment used in most modern computers. The term byte-addressable memory is used for this assignment. Byte locations have addresses 0, 1, 2,....
- Thus, if the word length of the machine is 32 bits, successive words are located at addresses 0, 4, 8, , with each word consisting of four bytes.

3.1.2 Big-Endian and Little-Endian Assignments

- The name big-endian is used when lower byte addresses are used for the more significant bytes (the leftmost bytes) of the word.
- The name little-endian is used for the opposite ordering, where the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word.
- The words –more significant‖ and –less significant‖ are used in relation to the weights (powers of 2) assigned to bits when the word represents a number.
- Both little-endian and big-endian assignments are used in commercial machines. In both cases, byte addresses 0, 4, 8,..., are taken as the addresses of successive words in the memory of a computer with a 32-bit word length.
- These are the addresses used when accessing the memory to store or retrieve a word.



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

3.1.3 Word Alignment

- In the case of a 32-bit word length, natural word boundaries occur at addresses 0, 4, 8,..., as shown in Figure 2.3. We say that the word locations have aligned addresses if they begin at a byte address that is a multiple of the number of bytes in a word.
- For practical reasons associated with manipulating binary-coded addresses, the number of bytes in a word is a power of 2. Hence, if the word length is 16 (2 bytes), aligned words begin at byte

addresses 0, 2, 4,..., and for a word length of 64 (23 bytes), aligned words begin at byte addresses 0, 8, 16,....

- There is no fundamental reason why words cannot begin at an arbitrary byte address. In that case, words are said to have unaligned addresses.
- But, the most common case is to use aligned addresses, which makes accessing of memory operands more efficient.

3.1.4 Accessing Numbers and Characters

- A number usually occupies one word, and can be accessed in the memory by specifying its word address. Similarly, individual characters can be accessed by their byte address.
- For programming convenience it is useful to have different ways of specifying addresses in program instructions.

3.2 Memory Operations

- Both program instructions and data operands are stored in the memory. To execute an instruction, the processor control circuits must cause the word (or words) containing the instruction to be transferred from the memory to the processor.
- Operands and results must also be moved between the memory and the processor. Thus, two basic operations involving the memory are needed, namely, Read and Write.
- The Read operation transfers a copy of the contents of a specific memory location to the processor. The memory contents remain unchanged.
- To start a Read operation, the processor sends the address of the desired location to the memory and requests that its contents be read.
- The memory reads the data stored at that address and sends them to the processor. The Write operation transfers an item of information from the processor to a specific memory location, overwriting the former contents of that location.
- To initiate a Write operation, the processor sends the address of the desired location to the memory, together with the data to be written into that location.
- The memory then uses the address and data to perform the write.

4. INSTRUCTION AND INSTRUCTION SEQUENCING

4. Discuss about instruction and instruction sequencing.

- The tasks carried out by a computer program consist of a sequence of small steps, such as adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen.

- **A computer must have instructions capable of performing 4 types of operations:**

  1) Data transfers between the memory and the registers (MOV, PUSH, POP, XCHG).

  2) Arithmetic and logic operations on data (ADD, SUB, MUL, DIV, AND, OR, NOT).

  3) Program sequencing and control (CALL.RET, LOOP, INT).

  4) I/0 transfers (IN, OUT).

REGISTER TRANSFER NOTATION (RTN)

Here we describe the transfer of information from one location in a computer to another. Possible locations that may be involved in such transfers are memory locations, processor registers, or registers in the I/O subsystem.

- Most of the time, we identify such locations symbolically with convenient names.
- The possible locations in which transfer of information occurs are:
  1) Memory-location
  2) Processor register &
  3) Registers in I/O device.

Location	Hardware Binary Address	Example	Description
Memory	LOC, PLACE, NUM	R1 ← [LOC]	Contents of memory-location LOC are transferred into register R1.
Processor	R0, R1 ,R2	[R3] ← [R1]+[R2]	Add the contents of register R1 &R2 and places their sum into R3.
I/O Registers	DATAIN, DATAOUT	R1 ← DATAIN	Contents of I/O register DATAIN are transferred into register R1.

ASSEMBLY LANGUAGE NOTATION

• To represent machine instructions and programs, assembly language format is used.

Assembly Language Format	Description
Move LOC, R1	Transfer data from memory-location LOC to register R1. The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten.
Add R1, R2, R3	Add the contents of registers R1 and R2, and places their sum into register R3.

BASIC INSTRUCTION TYPES

Instruction Type	Syntax	Example	Description	Instructions for Operation C<-[A]+[B]
Three Address	Opcode Source1,Source2,Destination	Add A,B,C	Add the contents of memory-locations A & B. Then, place the result into location C.	
Two Address	Opcode Source, Destination	Add A,B	Add the contents of memory-locations A & B. Then, place the result into location B, replacing the original contents of this location. Operand B is both a source and a destination.	Move B, C Add A, C
One Address	Opcode Source/Destination	Load A	Copy contents of memory-location A into accumulator.	Load A Add B Store C
		Add B	Add contents of memory-location B to contents of accumulator register & place sum back into accumulator.	
		Store C	Copy the contents of the accumulator into location C.	
Zero Address	Opcode [no Source/Destination]	Push	Locations of all operands are defined implicitly. The operands are stored in a pushdown stack.	Not possible

INSTRUCTION EXECUTION & STRAIGHT LINE SEQUENCING

- The program is executed as follows:

1) Initially, the address of the first instruction is loaded into PC.

2) Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called Straight-Line sequencing.

3) During the execution of each instruction, PC is incremented by 4 to point to next instruction.

- There are 2 phases for Instruction Execution:

1) Fetch Phase: The instruction is fetched from the memory-location and placed in the IR.

2) Execute Phase: The contents of IR is examined to determine which operation is to be performed. The specified-operation is then performed by the processor.

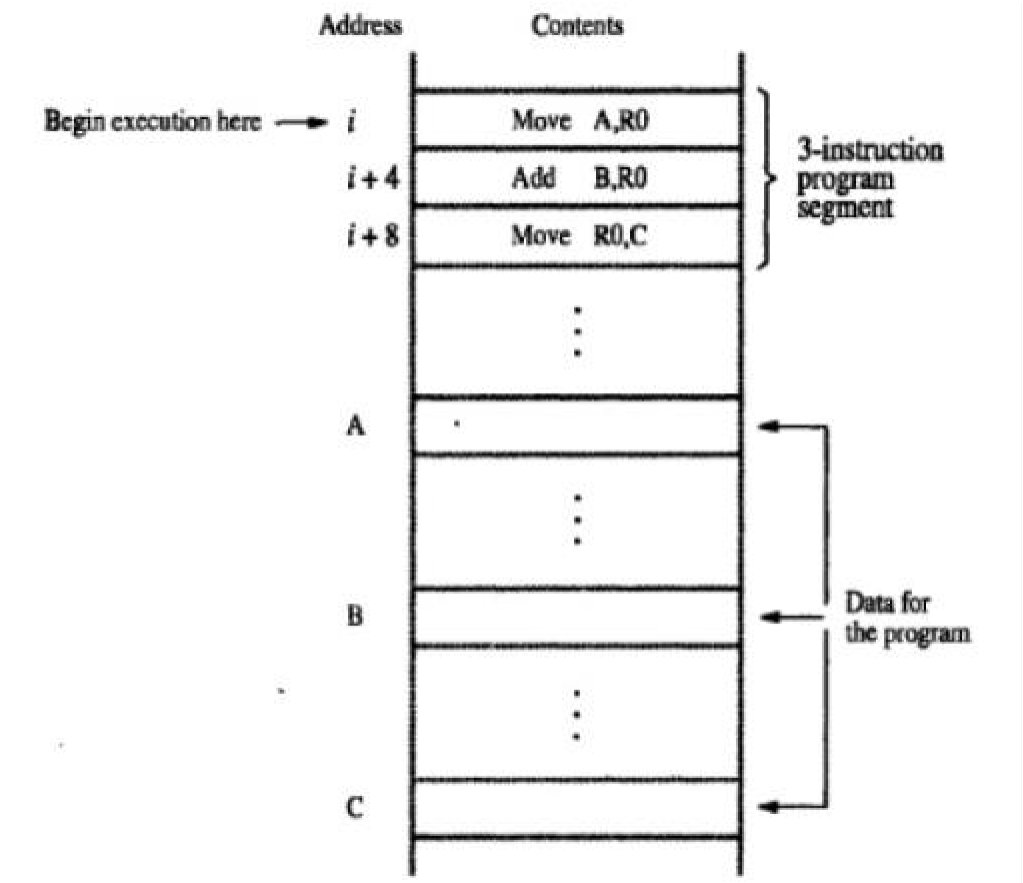
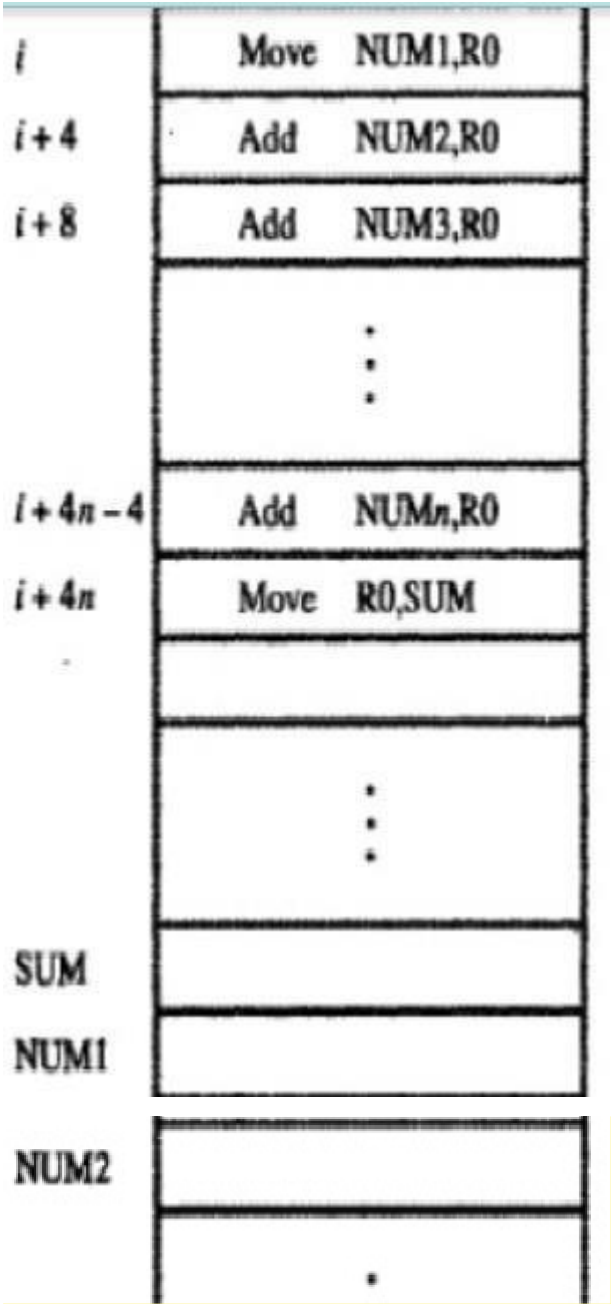


Fig 3.3 A program for C ←[A] + [B]

Program Explanation

- Consider the program for adding a list of n numbers
- The Address of the memory-locations containing the n numbers are symbolically given as NUM1, NUM2…..NUMn.
- Separate Add instruction is used to add each number to the contents of register R0.
- After all the numbers have been added, the result is placed in memory-location SUM.


i
Move NUM1,R0
i + 4
Add NUM2,R0
i + 8
Add NUM3,R0
i + 4n – 4
Add NUMn,R0
i + 4n
Move R0,SUM
SUM
NUM1
NUM2

5. ADDRESSING MODES

5. What is the need for addressing in a computer system? Explain the different addressing modes with suitable examples. (16)Apr 2011, Nov 2011, 2012, 2013,May 2013 (Nov/Dec 2014) (Apr/May 2015) (Nov/Dec 2016) (Or) Explain direct, immediate, relative and indexed addressing modes with examples. Apr/May 2017, 2018, Nov. / Dec. 2018 (Nov/Dec 2019)Nov/Dec 2020.(Or) Identify the addressing mode involved in the instruction XOR r1 r2 + 100 r1 and determine the resultant stored in register R1 if all of its bit were 1'st initially .

Addressing Modes

- The different ways in which the location of an operand is specified in instructions are referred to as addressing modes.
- Different types of addresses involve tradeoffs between instruction length, addressing flexibility and complexity of address calculation.

Overview

The different types of addressing modes are:

- Immediate addressing mode
- Direct or absolute addressing mode
- Indirect addressing mode
- Register addressing mode
- Indexed addressing mode(Displacement)
- Relative addressing mode
- Auto increment
- Auto decrement
- Implied (Stack, and a few others)

Immediate Addressing and Small Operands

- The operand is given explicitly in the instruction.

  Example: **MOVE #200, R0**

- The above statement places the value 200 in the register R0. A common convention is to use the sharp sign (#) in front of the value to indicate that this value is to be used as an immediate operand.
- A great many immediate mode instructions use small operands. (8 bits)
- In 32 or 64 bit machines with variable length instructions space is wasted if immediate operands are required to be the same as the register size.
- Some instruction formats include a bit that allows small operands to be used in immediate instructions.

- ALU will zero-extend or sign-extend the operand to the register size.

Instruction

op-code	**operand**

Immediate

Direct Addressing (Absolute addressing mode)

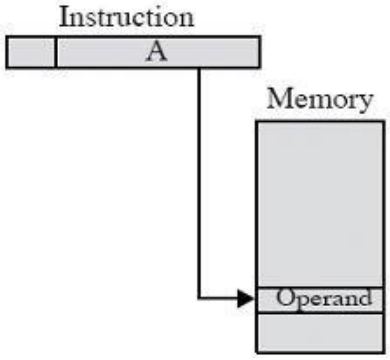
- The operand is in a memory location; the address of this location is given explicitly in the instruction. (In some assembly languages, this mode is called Direct Mode.

  Example: **MOVE LOC, R2**

- This instruction copies the contents of memory location of LOC to register R2.
- Address field contains address of operand.
- Effective address (EA) = address field (A).

  e.g. add ax, count or add ax,[10FC]

- Look in memory at address for operand.
- Single memory reference to access data.
- No additional calculations to work out effective address.



Direct Addressing

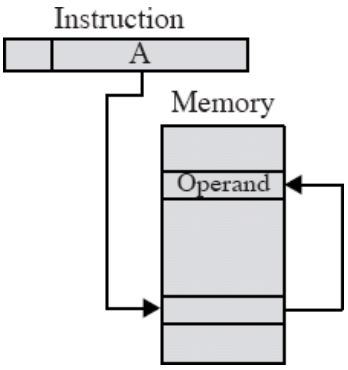
Memory-Indirect Addressing

- The effective address of the operand is the contents of a register or memory location whose address appears in the instruction.

  Example **Add (R2),R0**

- Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2.

- The initialization section of the program loads the counter value n from memory location N into Rl and uses the immediate addressing mode to place the address value NUM 1, which is the address of the first number in the list, into R2.

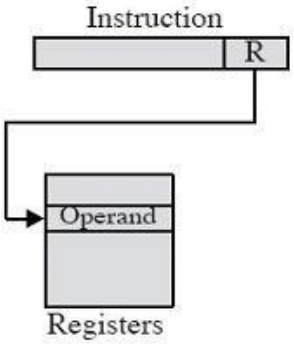


Memory-Indirect Addressing

Register Direct Addressing

- The operand is the contents of a processor register; the name (address) of the register is given in the instruction.

  Example: **MOV R1,R2**
- This instruction copies the contents of register R2 to R1.
- Operand(s) is(are) registers EA = R.
- There are a limited number of registers.
- Therefore a very small address field is needed.
- Shorter instructions are used.
- Instruction fetch is faster when compared to other.
- X86: 3 bits used to specify one of 8 registers.

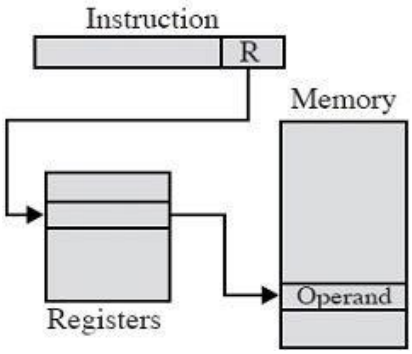


Register Direct Addressing

Register Indirect Addressing

- Similar to memory-indirect addressing; much more common, EA = (R).
- Operand is in memory cell pointed to by contents of register R.
- Large address space (2n).

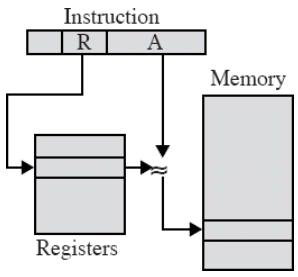
- One fewer memory access than indirect Addressing.



Register Indirect Addressing

Displacement Addressing(Index addressing mode)

- The effective address of the operand is generated by adding a constant value to the contents of a register. The register used may be either a special register provided for this purpose, or, more commonly; it may be anyone of a set of general-purpose registers in the processor.
- In either case, it is referred to as an index register. We indicate the index mode symbolically as **X(Ri).**
- Where X denotes the constant value contained in the instruction and Ri is the name of the register involved. The effective address of the operand is given by **EA = X + [Ri].**
- The contents of the index registers are not changed in the process of generating the effectiveaddress.
    - EA = A + (R)
    - Combines register indirect addressing with direct addressing
    - Address field hold two values
    - A = base value
    - R = register that holds displacement
    - or vice versa



Displacement Addressing

Types of Displacement Addressing

- Relative Addressing

- Base-register addressing
- Indexing

Relative Addressing

- We have defined the Index mode using general-purpose processor registers. A useful version of this mode is obtained if the program counter, PC, is used instead of a general purpose register.
- Then, X(PC) can be used to address a memory location that is X bytes away from the location presently pointed to by the program counter.
- Since the addressed location is identified "**relative**" to the program counter, which always identifies the current execution point in a program, the name Relative mode is associated with this type of addressing.
- EA = X + (PC)

Base-Register Addressing

- A holds displacement.
- R holds pointer to base address.
- R may be explicit or implicit.
- e.g. segment registers in 80x86 are base registers and are involved in all EA computations.
- x86 processors have a wide variety of base addressing formats.

Auto- increment mode

- The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this registers are automatically incremented to point to the next item in a list.
- We denote the Auto increment mode by putting the specified register in parentheses, to show that the contents of the registers are used as the effective address, followed by a plus sign to indicate that these contents are to be incremented after the operand is accessed.
- Thus, the Auto increment mode is written as,

(Ri) +

- As a companion for the Autoincrement mode, another useful mode accesses the items of a list in the reverse order:

Auto- decrement mode

- The contents of a register specified in the instructions are first automatically decremented and is then used as the effective address of the operand.
- We denote the Autodecrement mode by putting the specified register in parentheses, preceded by a minus sign to indicate that the contents of the registers are to be decremented before being used as the effective address. Thus, we write

- (Ri)

Stack Addressing

- Operand is (implicitly) on top of stack.

  e.g. PUSH, POP

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	Ri	EA = Ri
Absolute (Direct)	LOC	EA = LOC
Indirect	(Ri)	EA = [Ri]
	(LOC)	EA = [LOC]
Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri,Rj)	EA = [Ril + [Rj]
Base with index and offset	X(Ri,Rj)	EA = [Ri] + [Rj] + X
Relative	X(PC)	EA = [PC] +X
Autoincrement	(Ri)+	EA = [Ri]; Increment Ri
Autodecrement	-(Ri)	Decrement Ri; EA = [Ri]

EA = effective address Value = a signed number

- X87 is a stack machine so it has instructions such as,

  FADDP; st(1) <- st(1) + st(0); pop stack; result left in st(0)

  FIMUL qword ptr [bx]; st(0) <- st(0) * 64 integer pointed to; by bx

6. **Consider the computer with three instruction classes and CPI measurements as given below and instruction counts for each instruction class for the same program from two different compilers are given. Assume that the computer's clock rate is 4 GHZ. Which code sequence will execute faster according to execution time? (6) (NOV/DEC2014)**

Code from	**CPI for the instruction class**		
	A	**B**	**C**
CPI	1	2	3
Code from	**Instruction count for each class**		
	A	**B**	**C**
Compiler 1	2	1	2
Compiler 2	4	1	1

ANSWER

- Sequence 1 executes 2 + 1 + 2 = 5 instructions. Sequence 2 executes 4 + 1 + 1 = 6 instructions. Therefore, sequence 1 executes fewer instructions. We can use the equation for CPU clock cycles based on instruction count and CPI to find the total number of clock cycles for each sequence:

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{C}_i)$$

- This yields,
- CPU clock cycles1 = (2 × 1) + (1 × 2) + (2 × 3) = 2 + 2 + 6 = 10 cycles
- CPU clock cycles2 = (4 × 1) + (1 × 2) + (1 × 3) = 4 + 2 + 3 = 9 cycles
- So code sequence 2 is faster, even though it executes one extra instruction. Since code sequence 2 takes fewer overall clock cycles but has more instructions, it must have a lower CPI. The CPI values can be computed by,

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$$

$$\text{CPI}_1 = \frac{\text{CPU clock cycles}_1}{\text{Instruction count}_1} = \frac{10}{5} = 2.0$$

$$\text{CPI}_2 = \frac{\text{CPU clock cycles}_2}{\text{Instruction count}_2} = \frac{9}{6} = 1.5$$

6. ENCODING OF MACHINE INSTRUCTION

7. Explain about encoding of machine instruction.

- We have introduced a variety of useful instructions and addressing modes. These instructions specify the actions that must be performed by the processor circuitry to carry out the desired tasks.
- We have often referred to them as machine instructions. Actually, the form in which we have presented the instructions is indicative of the form used in assembly languages, except that we tried to avoid using acronyms for the various operations, which are awkward to memorize and are likely to be specific to a particular commercial processor.
- To be executed in a processor, an instruction must be encoded in a compact binary pattern. Such encoded instructions are properly referred to as machine instructions.
- The instructions that use symbolic names and acronyms are called assembly language instructions, which are converted into the machine instructions using the assembler program.
- We have seen instructions that perform operations such as add, subtract, move, shift, rotate, and branch. These instructions may use operands of different sizes, such as 32-bit and 8-bit numbers or 8-bit ASCII-encoded characters.
- The type of operation that is to be performed and the type of operands used may be specified using an encoded binary pattern referred to as the OP code for the given instruction.
- Suppose that 8 bits are allocated for this purpose, giving 256 possibilities for specifying different instructions. This leaves 24 bits to specify the rest of the required information.
- Let us examine some typical cases. The instruction

Add R1, R2

- Has to specify the registers R1 and R2, in addition to the OP code.
- If the processor has 16 registers, then four bits are needed to identify each register. Additional bits are needed to indicate that the Register addressing mode is used for each operand.
- The instruction

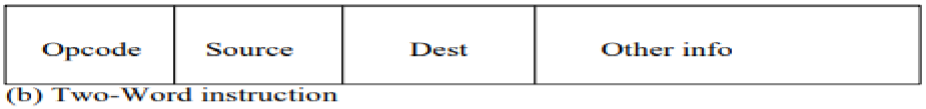
  Move 24(R0), R5
- Requires 16 bits to denote the OP code and the two registers, and some bits to express that the source operand uses the Index addressing mode and that the index value is 24.
- The shift instruction

  LShiftR #2, R0
- And the move instruction

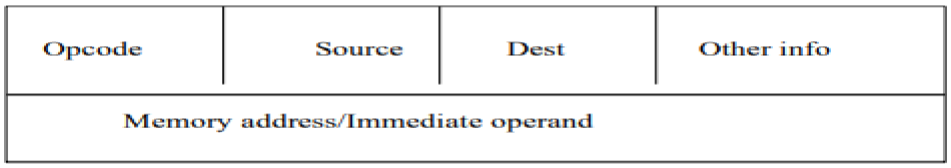
  Move #$3A, R1
- Have to indicate the immediate values 2 and #$3A, respectively, in addition to the 18 bits used to specify the OP code, the addressing modes, and the register.
- This limits the size of the immediate operand to what is expressible in 14 bits
- Consider next the branch instruction

  Branch >0 LOOP
- Again, 8 bits are used for the OP code, leaving 24 bits to specify the branch offset.
- Since the offset is a 2‘s-complement number, the branch target address must be within 223 bytes of the location of the branch instruction.
- To branch to an instruction outside this range, a different addressing mode has to be used, such as Absolute or Register Indirect. Branch instructions that use these modes are usually called Jump instructions.
- In all these examples, the instructions can be encoded in a 32-bit word. Depicts a possible format.
- There is an 8-bit Op-code field and two 7-bit fields for specifying the source and destination operands. The 7-bit field identifies the addressing mode and the register involved (if any).
- The ―Other info‖ field allows us to specify the additional information that may be needed, such as an index value or an immediate operand.
- But, what happens if we want to specify a memory operand using the Absolute addressing mode?
- The instruction Move R2, LOC

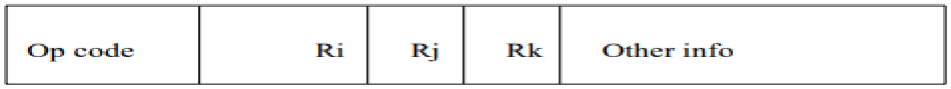
(a) One-word instruction



(b) Two-Word instruction



(c) Three-operand instruction

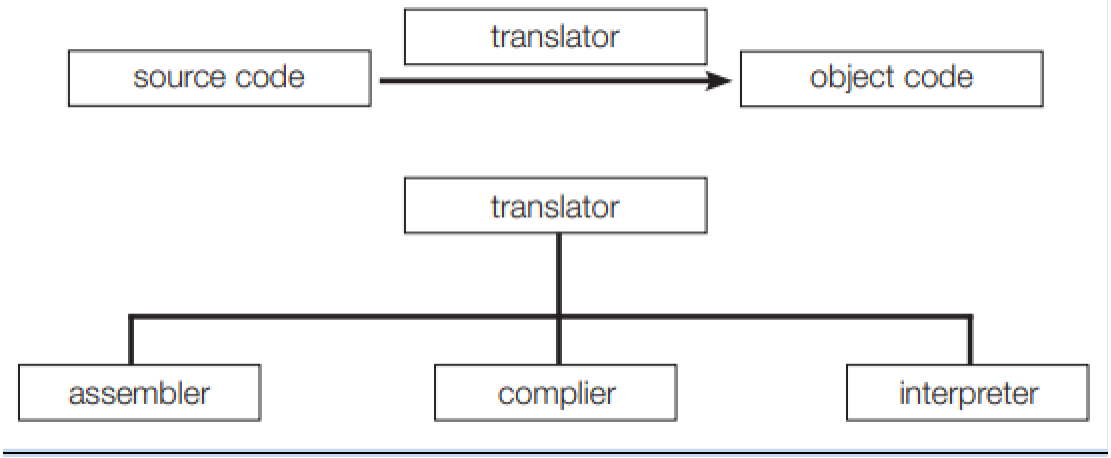


- Requires 18 bits to denote the OP code, the addressing modes, and the register.
- This leaves 14 bits to express the address that corresponds to LOC, which is clearly insufficient. And #$FF000000. R2
- In which case the second word gives a full 32-bit immediate operand. If we want to allow an instruction in which two operands can be specified using the Absolute addressing mode, for example
  Move LOC1, LOC2
- Then it becomes necessary to use an additional words for the 32-bit addresses of the operands. This approach results in instructions of variable length, dependent on the number of operands and the type of addressing modes used.
- Using multiple words, we can implement quite complex instructions, closely resembling operations in high-level programming languages.
- The term complex instruction set computer (CISC) has been used to refer to processors that use instruction sets of this type.
- The restriction that an instruction must occupy only one word has led to a style of computers that have become known as reduced instruction set computer (RISC).
- The RISC approach introduced other restrictions, such as that all manipulation of data must be done on operands that are already in processor registers.
- This restriction means that the above addition would need a two-instruction sequence
  Move (R3), R1 Add R1, R2
- If the Add instruction only has to specify the two registers, it will need just a portion of a 32-bit word. So, we may provide a more powerful instruction that uses three operands
  Add R1, R2, R3
- Which performs the operation R3 -> [R1] + [R2]
- A possible format for such an instruction in shown in fig c. Of course, the processor has to be able to deal with such three-operand instructions.
- In an instruction set where all arithmetic and logical operations use only register operands, the only memory references are made to load/store the operands into/from the processor registers.
- RISC-type instruction sets typically have fewer and less complex instructions than CISC-type sets.

7. INTERACTION BETWEEN ASSEMBLY AND HIGH LEVEL LANGUAGE

8. Explain the concept of interaction between assembly and high level language

- What are _high level languages‘? High level languages are called _high-level‘ because they are closer to human languages and are further removed from machine languages than assembly language.
- There is no one-to-one relationship between the instructions in a high level language and machine language as there is with assembly language.
- List three examples of a high level language. Basic, C, Fortran, Python, Ada etc.
- List three advantages of assembly language over a high level language.
- It requires less memory and execution time.
- It allows hardware-specific complex jobs in an easier way.
- It is suitable for time-critical jobs.
- It is most suitable for writing interrupt service routines and other memory resident programs.
- List three advantages of using a high level language over assembly language.
- Faster program development – it is less time consuming to write and then test the program.
- It is not necessary to remember the registers of the CPU and mnemonic instructions.
- Portability of a program from one machine to other.
- Each assembly language is specific to a particular type of CPU, but most high-level programming languages are generally portable across multiple architectures.
- A compiler reads the whole high level code and translates it into a complete machine code program which is output as a new file and can be saved.
- The biggest advantage of this is that the translation is done once only and as a separate process. The program that is run is already translated into machine code so is much faster in execution.
- The disadvantage is that you cannot change the program without going back to the original source code, editing that and recompiling.
- An interpreter reads the source code one instruction or line at a time, converts this line into machine code and executes it.
- The machine code is then discarded and the next line is read. The advantage of this is it‘s simple and you can interrupt it while it is running, change the program and either continue or start again.
- The disadvantage is that every line has to be translated every time it is executed, even if it is executed many times as the program runs. And because of this interpreters tend to be slow.



TWO MARKS

1. What are the five components of computer system? Apr/May 2017, 2019

The five classic components of computers are input unit, output unit, memory unit, arithmetic & logic unit and control unit.

2. What is cache memory?

The small and fast RAM units are called as caches.

When the execution of an instruction calls for data located in the main memory, the data are fetched and a copy is placed in the cache.

- Later if the same data are required it reads directly from the cache.

3.**What is the function of ALU?**

- Most of the computer operations (arithmetic & logic) are performed in ALU. The data required for the operation is brought by the processor and the operation is performed by the ALU.

4. What is the function of control unit?

- The Control unit is the main part of the computer that coordinates the entire computer operations. Data transfers between the processor and memory controlled by the control unit through timing signal.

5. What are basic operations of a computer memory?

- The basic operations of the memory are READ and WRITE.
  - READ – read the data from input device to memory.
  - WRITE – writes data to the output device.

6. List out the operations of the computer.

The computer accepts the information in the form of programs and data through an input unit and stores it in the memory.

1. Information stored in the memory is fetched under program control into an arithmetic and logic unit where it is processed.
2. Processed information leaves the computer through an output unit.
3. All activities inside the machines are directed by the control unit.

7. What are the main elements of a computer?

- **Processor:** To interpret and execute programs.
- **Memory:** For storing programs and data.
- **Input-output equipment:** For transferring information between the computer and outside world.

8. Define Computer design.

- It is concerned with the hardware design of the computer. Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system.

- Computer design is concerned with the determination of what hardware should be used and how the parts should be connected. This aspect of computer hardware is sometimes referred to as computer implementation.

9. What is instruction set architecture?

- An abstract interface between the hardware and the lowest level software that encompasses all the information necessary to write a machine language program that will run correctly.
- Including instructions, registers, memory access, I/O and so on.

10. State Amdahl's law. Nov / Dec 2014

- Amdahl's Law is used to find the execution time of a program after making the improvement. It can be represented in an equation as follows:

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by improvement}}{Amount\ of\ improvement} + Execution\ time\ unaffected$$

- Hence, Amdahl's Law can be used to estimate performance improvements.

11. Define Stored Programmed Concept.

- Storing program and their data in the same high-speed memory.
- It enables a program to modify its own instructions (such self-modifying Programs have undesirable aspects, however and are rarely used).

12. What are the registers generally contained in the processor?(Nov/Dec-2019)

- MAR – Memory Address Register.
- MDR – Memory Data Register.
- IR – Instruction Register.
- R_0 – Rn – General purpose Register.
- PC – Program Counter.

13. What do you mean by Memory address register (MAR) and Memory dataregister (MDR)?

- The MAR holds the address of the location to be accessed.
- The MDR contains the data to be written into or read out of the addressed location.

14. What is Data path?

- The component of the processor that performs arithmetic operations is called data path.

15. What is elapsed time of computer system?

- The total time to execute the total program is called elapsed time.
- It is affected by the speed of the processor, the disk and the printer.

16. What is processor time of a program?

- The period during which the processor is active is called processor time of a program.

- It depends on the hardware involved in the execution of individual machine instructions.

17. Define clock rate.

- The clock rate is given by,

  R=1/P,

- Where P is the length of one clock. It can be measure as cycles per second (Hertz).

18. What is meant by clock cycle?

- Processor circuit is controlled by a timing signal called a clock.
- The clock defines regular time intervals, called clock cycle.
- To execute the machine instruction the processor divides the action to be performed into sequence of basic steps. Each step can be completed in one clock cycle.

19. Write down the basic performance equation. (Apr/May-2014)(Nov/Dec 2019)

T=N*S/R

Where

T-Processor time

N-Number of machine instructions

S-Number of basic steps needed to execute one machine instruction

R-Clock rate

20. What is meant by addressing mode? List its types. (May/June 2013) Nov/ Dec 2013

The addressing mode is defined as the different ways in which the location or of an operand is specified in an instruction.

The different types of addressing modes are:

1. Immediate addressing mode
2. Register addressing mode
3. Direct or absolute addressing mode
4. Indirect addressing mode
5. Indexed addressing mode
6. Relative addressing mode
7. Auto increment
8. Auto decrement

21. Define Register addressing mode with an example.

- In register addressing mode, the operand is the content of a processor register. The name (address) of the register is given in the instruction.

Effective address (EA) = Ri, Where Ri is a processor register.

22. Define absolute addressing mode with an example.

- In absolute addressing mode, the operand is in a memory location. The addresses of this location are given explicitly in the instruction. This is also called as direct addressing mode.

EA = Loc Where loc is the memory address.

23. What is relative addressing mode with an example? (N/D 2014)

- The Effective address is determined by the index mode using the program counter in place of general purpose register. This mode is used to access the data operands.

EA = X + [PC]

24. What is indirect addressing mode?

- The Effective address of the operand is the contents of a register or memory location whose address appears in the instruction.

EA = [Ri] or EA = [Loc]

25. What is indexed addressing mode?

- The Effective address of the operand is generated by adding a constant value to the contents of a register.

EA = X + [Ri].

26. Define auto increment mode of addressing.

The Effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this registers are automatically incremented to point to the next item in the list.

EA = (Ri) +

27. Define auto decrement mode of addressing.

The contents of a register specified in the instructions are first automatically decremented and are then used as the effective address of the operand.

EA = - (Ri)

28. List the basic instruction types. May / June 2013

The various instruction types are,

- Three address instructions
- Two-address instructions
- Single-address instructions
- Zero-address instructions

29. What is register?

- A small set of high-speed storage devices called registers, which serve as implicit storage locations for operands and results.

30. List the phases, which are included in the each instruction cycle?

- **Fetch:**Fetches instruction from main memory (M).
- **Decode:** Decodes the instruction‘s opcode.
- **Load:** Loads (read) from M any operands needed unless they are already in CPU Registers.
- **Execute**: Executes the instruction via a register-to-register operation using an appropriate functional unit of the CPU such as a fixed–point adder.
- **Store:** Stores (write) the results in M unless they are to be retained in CPU register.

31. What are the types of computer?

- Mini computer
- Micro computers
- Mainframe computers
- Super computers

32. What are the two major steps in processing an instruction? (Or) Write the two steps that are common to implement any type of instruction. Nov. / Dec. 2018

- **Fetch step:** During this step a new instruction is read from the external memory M by the CPU.
- **Execute step:** During this step operations specified by the instructions are executed by the CPU.

33. What are the speedup techniques available to increase the performance of a computer?

- **Cache:** It is a fast accessible memory often placed on the same chip as the CPU. It is used to reduce the average time required to access an instruction or data to a single clock cycle.
- **Pipelining:** Allows the processing of several instructions to be partially overlapped.
- **Super scalar:** Allows processing of several instructions in parallel (full overlapping).

34. What are Timing signals?

- Timing signals are signals that determine when a given action is to take place.
- Data transfers between the processor and the memory are also controlled by the control unit through timing signals.

35. Distinguish between auto increment and auto decrement addressing mode. (May/June 2016)

Auto increment	Auto decrement
1.The effective address of the operand is the contents of the register specified in the instruction. After accessing the operand, the contents of the register are incremented to address the next location.	**1.**The contents of a register specified in the instruction are decremented and then are used as effective address to access a memory location.
2.Auto increment is symbolically represented as $(R_i)+$.**Example:**move(R_2), R_0+	**2.**Auto decrement mode is symbolically represented as $-(R_i)$.**Example:** Move R_1, $-(R_0)$

36. What is an opcode? How many bits are needed to specify 32 distinct operations? (Apr/May 2011)

- An **opcode** is the first byte of an instruction in machine language which tells the hardware what operation needs to be performed with this instruction.
- Every processor/controller has its own set of opcodes defined in its architecture. Opcode is the operation to be performed on data. An opcode is followed by data like address, values etc if needed.
- 5 bits are needed to specify 32 distinct operations.

37. Define word length. (Nov/Dec 2011)

- In computer architecture, a **word** is a unit of data of a defined bit length that can be addressed and moved between storage and the computer processor.
- Address that is divided by 4 is called word.
- The number of bits in the word is called **word length.**
- In longer architected **word length,**the computer processor can do more in a single operation.

38. What are the merits and demerits of single address instructions? (Nov/Dec 2011)

Single address instruction,

Eg**: Add A**

Store A

Add the contents of memory location A to the contents of the accumulator register and place the sum back into accumulator.

39. Explain the disadvantages of using a single type of instruction.

In practice the codes in an instruction (opcode and condition) may be fairly small e.g. 2. to .8 bits. However, if the instruction is to be able to reference large quantities of data then the addresses must be large e.g. 16..32 bits. If the above instruction were to use 6 bits for the opcode, 4 bits for the condition code and 16 bits for each address then it would have to be 90 bits long.

40. What is relative addressing mode? When is it used? (May/June 2012)

- The effective address is determined by the index mode using program counter in place of the general purpose registers.
- This address is commonly used to specify the target address in branch instruction.
  - **Example: JNZ BACK**
    - This instruction causes program executive to go to the branch target location identified by the name BACK, if the branch condition is satisfied.

41. Suppose you wish to run a program P with 8.5 * 109 instructions on a 5 Ghz machine with CPI of 0.8. What is the expected CPU time? (Nov/Dec 2010)

Percentage of elapsed time=(User CPU Time + System CPU Time)/Elapsed Time

Expected CPU time =0.8-0.2*8.5*109=1.36

42. What does the term hertz refer to? (Nov/Dec 2010)

- The hertz abbreviated as Hz.
- It is a unit of frequency.
- It is equal to 1 cycle per second.

43. Mention the registers used for communications between processor and main memory. (May/June 2010)

1) **MAR(Memory Address Register):** The **Memory Address Register (MAR)** is a CPU register that either stores the memory address from which data will be fetched to the CPU or the address to which data will be sent and stored.
2) **MDR (Memory Data Register):**It is the register of a computer's control unit that contains the data to be stored in the computer storage (e.g. RAM), or the data after a fetch from the computer storage. It acts like a buffer and holds anything that is copied from the memory ready for the processor to use it.

44. What is SPEC? Specify the formula for SPEC rating. (May/June 2012)(Apr/May 2014)

- SPEC is a nonprofit consortium of 22 major computer vendors whose common goals are ―to provide the industry with a realistic yardstick to measure the performance of advanced computer systems‖ and to educate consumers about the performance of vendors‘ products.
- SPEC creates, maintains, distributes, and endorses a standardized set of application-oriented programs to be used as benchmarks.

The formula for SPEC rating is as follows:

SPEC rating (ratio) = TR / TC;

where,

TR = Running time of the Reference Computer;

TC = Running time of the Computer under test;

If the SPEC rating = 50 means that the computer under test is 50 times as fast as the ultra sparc 10. This is repeated for all the programs in the SPEC suit, and the geometric mean of the result is computed.

45. Give an example each of zero-address, one-address, two-address and three-address instructions. (Or) Classify the instructions based on the operations they perform and give one example to each category. Apr. / May 2018, Nov. / Dec. 2018

- Zero address- push (Push the value as top of stock)
- One address- INC CL (If carry set, increment CL by one)
- Two address- Add A,B (A→ A+B)
- Three address- Add A,B,C (A→ B+C)

46. Which data structures can be best supported using (a) indirect addressing mode (b) indexed addressing mode?

(a) Indirect addressing mode – Pointer data structure

(b) Indexed addressing mode- Array data structure

47. What are the four basic types of operations that need to be supported by an instructor set?

(i) Data transfer between memory and the processor register.

(ii) Arithmetic and logic operations on Data.

(iii) Program sequencing and control.

(iv) I/O transfer.

48. What are the address-sequencing capabilities required in a control memory?

(i) Incrementing of the control address register.

(ii) Unconditional branch as specified by address field of the micro instruction.

(iii)Conditional branch depending on status bits in registers of computer.

(iv)A facility for sub-routines calls and returns.

49. What are the limitations of assembly language? (M/J 2007)

(i) Assembly language is converted to Machine language using assembler which is time consuming when compared with machine language.

(ii) It is difficult to solve the complex problems.

(iii) A set of symbolic names (mnemonics) and rules has to be followed.

50. A memory byte location contains the pattern 00101100. What does this pattern represent when interpreted as a number? What does it represent as an ASCII Code? (Nov/Dec 2007)

- Interpreted number is 44.
- ASCII code is NULL/idle.

51. What is the information conveyed by addressing modes? (Nov/Dec 2007)

- The information conveyed by addressing mode is to specify the location of an operand in an instruction.

52. Why is the data bus in most microprocessors bi-directional while the address bus is unidirectional? (Apr/May 2008)

- The data bus is bi-directional bus and is used to fetch instruction from memory and to send a command to an I/O device or port. Address is unidirectional to carry memory address while reading from or writing into memory.

53. What is meant by the stored program concept? Discuss. (May/June 2007)

- A set of instruction that performs a task is called a program. Usually the program is stored in the memory. The processor fetches the instructions that take up the program from the memory, one at a time and perform the desired operation.

54. What are the two techniques used to increase the clock rate R?

The two techniques used to increase the clock rate R are:

- The Integrated Circuit (IC) technology can be increased which reduces the time needed to complete a basic step.
- We can reduce the amount of processing done in one basic step.

55. What is Big-Endian and Little-Endian representations? (Nov/Dec 2014)

- **The big-endian** is used when lower byte addresses are used for the more significant bytes(The leftmost bytes) of the word.
- **The little-endian** is used for the opposite ordering, where the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word.

56. What is meant by instructions? (May/June 2016)

- Instructions are command that is governed by the transfer of information within the computer as well as between the computers and its input and output device.

57. What is the use of Instruction register?

- It holds the instructions that are currently being executed.

58. What is the use of MAR?

- It holds the address of the location to be accessed.

59. What is the use of MDR?

- It holds the data to be written into or read out of the addressed location.

60. State the basic performance equation of a computer. (Apr/May 2014)

T= (NxS)/R

Where,

N- Number of instructions

S- Average numbers of steps needed to execute one instruction.

R- Clock rate.

61. What are the two basic operations involving in the memory?

1. Load (Read or fetch)
2. Store (Write)

62. How to measure the performance of the system?

1. Response time
2. Throughput.

63. What is register indirect addressing mode? When is it used? (Nov/Dec 2013)

The effective address of the operand is the contents of a register or memorylocation whose address appears in the instruction. Example **Add (R2),R0**

Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2. The initialization section of the program loads the counter value n from memory location N into Rl and uses the Immediate addressing mode to place the address value NUM 1, which is the address of the first number in the list, into R2.

64. List the eight great ideas invented by computer architects. (Nov/Dec-2015)

- Design for Moore‘s Law
- Use abstraction to simplify design
- Make the common case fast
- Performance viaparallelism
- Performance viapipelining
- Performance viaprediction
- Hierarchy of memories
- Dependabilityvia redundancy

65. Distinguish pipelining from parallelism. (N/D2015)

- Parallelism means we are using more hardware for the executing the desired task. In parallel computing more than one processor are running in parallel. There may be some dedicated hardware running in parallel for doing the specific task.
- Parallelism increases the performance but the area also increases.
- The pipelining is an implementation technique in which multiple instructions are overlapped in execution.
- In case of pipelining the performance and throughput increases at the cost of pipelining registers area.
- In pipelining there are different hazards like data hazards, control hazards etc.

66. Give the formula for CPU execution time for a program.(Nov/Dec 2016)

- A simple formula relates the most basic metrics (clock cycles and clock cycle time) to CPU time:

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

- Alternatively, because clock rate and clock cycle time are inverses,

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

- This formula makes it clear that the hardware designer can improve performance by reducing the number of clock cycles required for a program or the length of the clock cycle.

67. What is an instruction register? (Nov/Dec 2016)

In computing, an instruction register (IR) is the part of a CPU's control unit that holds the instruction currently being executed or decoded.

68. State the need for indirect addressing mode. Give an example. Apr/May 2017

The register or memory location that contains the**address** of the operand is a pointer. When an execution takes place in such **mode**, instruction may be told to go to a specific **address**. Once it's there, instead of finding an operand, it finds an **address** where the operand is located.

In this case the number is usually enclosed with square brackets.

LD Acc, [5];Load the value stored in the memory location pointed to by the operand into the accumulator

Memory location 5 is accessed which contains 3. Memory location 3 is accessed which is 17.

Ax becomes 17.

69. Specify the CPU performance equation. Nov/ Dec 2012

The **performance equation** analyzes execution time as a product of three factors that are relatively independent of each other. The three factors are, in order, known as the instruction count (IC), clocks per instruction (CPI), and clock time (CT). CPI is computed as an effective value.

70. Write the equation for the dynamic power required per transistor. Apr. / May 2018

Energy/transition

$$E_{VDD} = \int_0^\infty iV_{DD}(t)V_{DD}dt = V_{DD}\int_0^\infty C_L\frac{dv_{out}}{dt}dt = C_LV_{DD}\int_0^{V_{DD}} dv_{out} = C_LV_{DD}^2$$

$$\textbf{Power} = \textbf{Energy/transition} * f = C_L * V_{dd}^2 * f$$

f represents the frequency of energy-consuming transitions (0→1)

71. Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.2.Which processor has the highest performance expressed in instructions per second?Nov. / Dec. 2018

Performance = (instructions/sec)

Processor	ClockRate	CPI	Performance
P1	3×10^9	1.5	$3 \times 10^9 / 1.5 = 2 \times 10^9$
P2	2.5×10^9	1.0	$2.5 \times 10^9 / 1.0 = 2.5 \times 10^9$
P3	4×10^9	2.2	$4 \times 10^9 / 2.2 = 1.8 \times 10^9$

P2 has the highest performance

72. Give the MIPS code for the statement f=(g+h)-(i+j). May 2019

Simple arithmetic expression,assignment

int f, g, h, i, j;

f = (g + h) - (i + j);

$s0	(g + h) - (i + j)
$s1	i + j
$s2	h
$s3	i
$s4	j

Assume variables are assigned to $s0, $s1, $s2, $s3, $s4 respectively

add $s0, $s1, $s2 # $s0 = g + h

add $s1, $s3, $s4 # $s1 = i + j

sub $s0, $s0, $s1 # f = (g + h) - (i + j)

73. Define Word Length. Nov/Dec-2019

A word is a unit of data of a defined bit length that can be addressed and moved between storage and the computer processor. ... Typically, an instruction is a word in length, but some architectures support halfword and doubleword-length instructions

74. What is Zero address instruction format? Nov/Dec 2020

A zero-address instruction implies that the absolute address of the operand is held in a special register that is automatically incremented (or decremented) to point to the location of the top of the stack.

75. What is the impact of frequency of clock signal applied to the microprocessor in the performance of computer? Nov/Dec 2020.

A computer's processor clock speed determines how quickly the central processing unit (CPU) can retrieve and interpret instructions. This helps your computer complete more tasks by getting them done faster. Clock speeds are measured in gigahertz (GHz), with a higher number equating to higher clock speed.

76. List the difference between wall clock time and reponse time.Nov/Dec 2021

Wall clock time is the actual amount of time taken to perform a job. This is equivalent to timing your job with a stopwatch and the measured time to complete your task can be affected by anything else that the system happens to be doing at the time.

User time measures the amount of time the CPU spent running your code. This does not count anything else that might be running, and also does not count CPU time spent in the kernel (such as for file I/O).

CPU time measures the total amount of time the CPU spent running your code or anything requested by your code. This includes kernel time.

The "User time" measurement is probably the most appropriate for measuring the performance of different jobs, since it will be least affected by other things happening on the system.

77. Find the Cycle time of a 450MHz clock frequency.Nov/Dec 2021

Time=1/frequency

$=1/450*10^{6}$

$= 0.00222*10^{-6}$

UNIT IV PROCESSOR

Instruction Execution – Building a Data Path – Designing a Control Unit – Hardwired Control, Micro programmed Control – Pipelining – Data Hazard – Control Hazards.

PART B

1. Briefly explain about Basic MIPS Implementation. Nov / Dec 2015, 2018

A Basic MIPS Implementation:

We will be examining an implementation that includes a subset of the core **MIPS instruction set:(Micro Instruction per Second)**

- **The memory**-reference instructions load word (lw) and store word (sw)
- **The arithmetic**-logical instructions add, sub, AND, OR, and slt
- **The instructions branch** equal (beq) and jump (j), which we add last

- This subset does not include all the integer instructions (for example, shift, multiply, and divide are issuing), nor does it include any floating-point instructions.
- However, the key principles used in creating a data path and designing the control are illustrated.
- The implementation of the remaining instructions is similar. In examining the implementation, we will have the opportunity to see how the instruction set architecture determines many aspects of the implementation, and how the choice of various implementation strategies affects the clock rate and CPI for the computer.
- In addition, most concepts used to implement the MIPS subset in this chapter are the same basic ideas that are used to construct a broad spectrum of computers, from high-performance servers to general-purpose microprocessors to embedded processors.

An Overview of the Implementation

- MIPS instructions, including the integer arithmetic-logical instructions, the memory-reference instructions, and the branch instructions.
- What needs to be done to implement these instructions is the same, independent of the exact class of instruction.

For every instruction, the first two steps are identical:

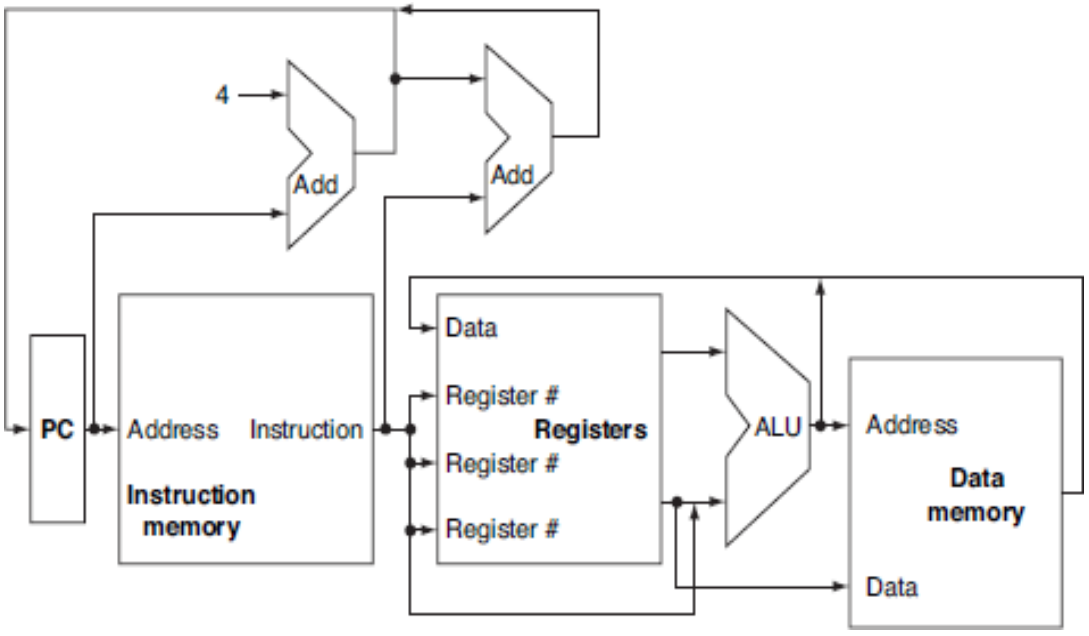
1. Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory.

2. Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, we need to read only one register, but most other instructions require that we read two registers.

- After these two steps, the actions required to complete the instruction depend on the instruction class. Fortunately, for each of the three instruction classes (memory-reference, arithmetic-logical, and branches), the actions are largely the same, independent of the exact instruction.
- The simplicity and regularity of the MIPS instruction set simplifies the implementation by making the execution of many of the instruction classes similar.

For example,

- All instruction classes, except jump, use the arithmetic-logical unit (ALU) after reading the registers.
- The memory-reference instructions use the ALU for an address calculation, the arithmetic-logical instructions for the operation execution, and branches for comparison. After using the ALU, the actions required to complete various instruction classes differ.
- A memory-reference instruction will need to access the memory either to read data for a load or write data for a store.
- An arithmetic-logical or load instruction must write the data from the ALU or memory back into a register. Lastly, for a branch instruction, we may need to change the next instruction address based on the comparison; otherwise, the PC should be incremented by 4 to get the address of the next instruction.



An abstract view of the implementation of the MIPS subset showing the Major functional units and the major connections between them

- All instructions start by using the program counter to supply the instruction address to the instruction memory.
- After the instruction is fetched, the register operands used by an instruction are specified by fields of that instruction.

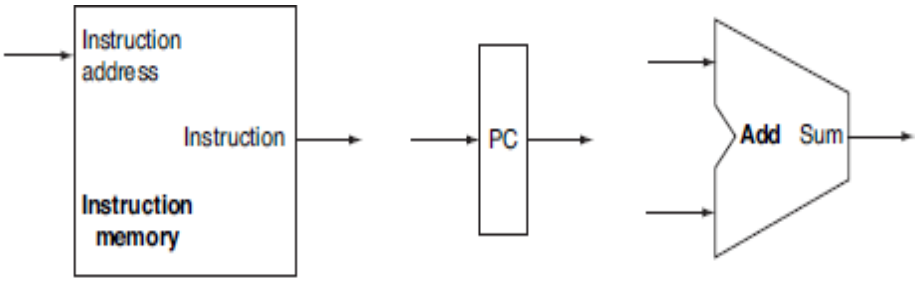
- Once the register operands have been fetched, they can be operated on to compute a memory address (for a load or store), to compute an arithmetic result (for an integer arithmetic-logical instruction), or a compare (for a branch).
- If the instruction is an arithmetic-logical instruction, the result from the ALU must be written to a register. If the operation is a load or store, the ALU result is used as an address to either store a value from the registers or load a value from memory into the registers.
- The result from the ALU or memory is written back into the register file. Branches require the use of the ALU output to determine the next instruction address, which comes either from the ALU (where the PC and branch offset are summed) or from an added that increments the current PC by 4.
- The thick lines interconnecting the functional units represent buses, which consist of multiple signals. The arrows are used to guide the reader in knowing how information flows. Since signal lines may cross, we explicitly show when crossing lines are connected by the presence of a dot where the lines cross.

BUILDING A DATA PATH

2. Give detail description about Building a Data path.(or) Build a suitable Data path for branch instruction. Explain all the blocks with suitable example. Nov/Dec 2021

Data path element: A unit used to operate on or hold data within a processor. In the MIPS implementation, the data path elements include the instruction and data memories, the register file, the ALU and adders.

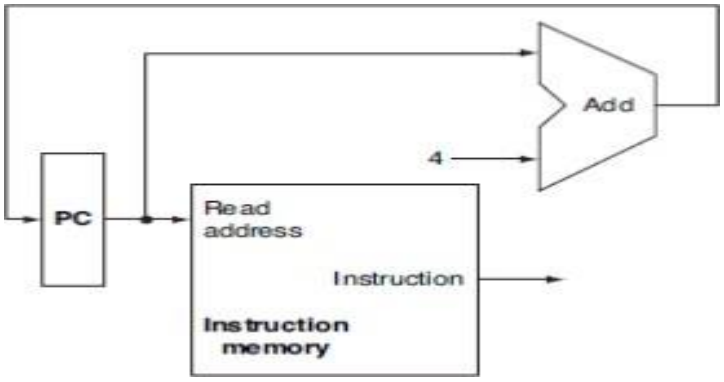
A memory unit to store the instructions of a program and supply instructions given an address. The program counter (PC), is a register that holds the address of the current instruction. We need an adder to increment the PC to the address of the next instruction.



Two state elements are needed to store and access instructions, and an adder is needed to compute the next instruction address.

- The state elements are the instruction memory and the program counter. The instruction memory need only provide read access because the data path does not write instructions.

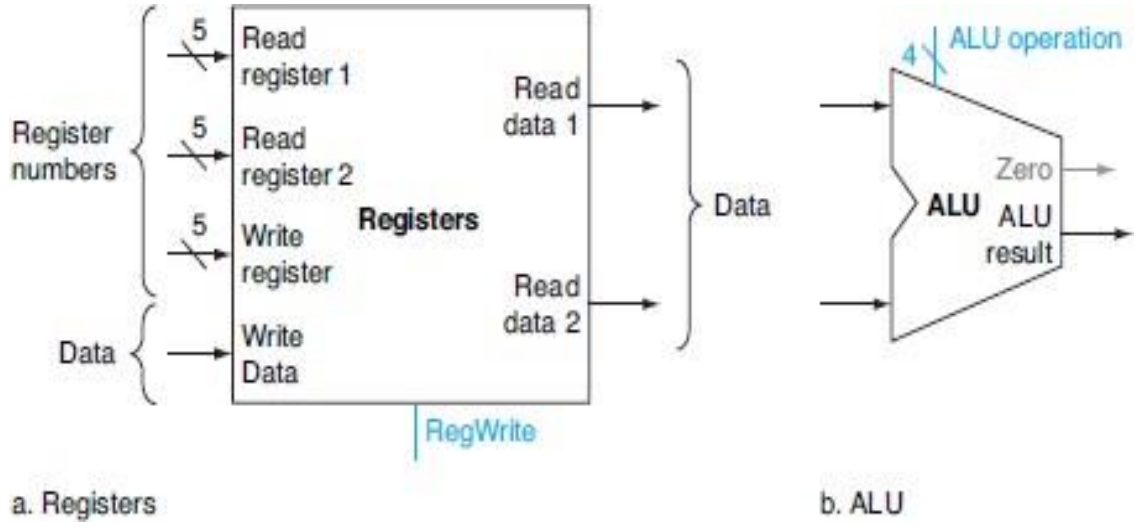
- Since the instruction memory only reads, we treat it as combinational logic: the output at any time reflects the contents of the location specified by the address input, and no read control signal is needed. (We will need to write the instruction memory when we load the program; this is not hard to add, and we ignore it for simplicity.)
- The program counter is a 32‑bit register that is written at the end of every clock cycle and thus does not need a write control signal. The adder is an ALU wired to always add its two 32‑bit inputs and place the sum on its output.
- Simply by wiring the control lines so that the control always specifies an add operation. We will draw such an ALU with the label *Add*, to indicate that it has been permanently made an adder and cannot perform the other ALU functions. To execute any instruction, we must start by fetching the instruction from memory.
- To prepare for executing the next instruction, we must also increment the program counter so that it points at the next instruction, 4 bytes later how to combine the three elements to form a datapath that fetches instructions and increments the PC to obtain the address of the next sequential instruction.
- Now let's consider the R-format instructions. They all read two registers, perform an ALU operation on the contents of the registers, and write the result to a register.
- We call these instructions either ***R-type instructions* or *arithmetic-logical instructions***(since they perform arithmetic or logical operations). This instruction class includes **add, sub, AND, OR, and slt,** Recall that a typical instance of such an instruction is add $t1,$t2,$t3, which reads $t2 and $t3 and writes $t1.
- The processor's 32 general-purpose registers are stored in a structure called a **register file.** A register file is a **collection of registers** in which any register can be read or written by specifying the number of the register in the file. The register file contains the register state of the computer.
- In addition, we will need an ALU to operate on the values read from the registers.
- R-format instructions have three register operands, so we will need to read two data words from the register file and write one data word into the register file for each instruction. For each data word to be read from the registers, input to the register file that specifies the register number to be read and an output from the register file that will carry the value that has been read from the registers.



A portion of the datapath used for fetching instructions and incrementing the program counter. The fetched instruction is used by other parts of the datapath.

To write a data word, we will need two inputs:

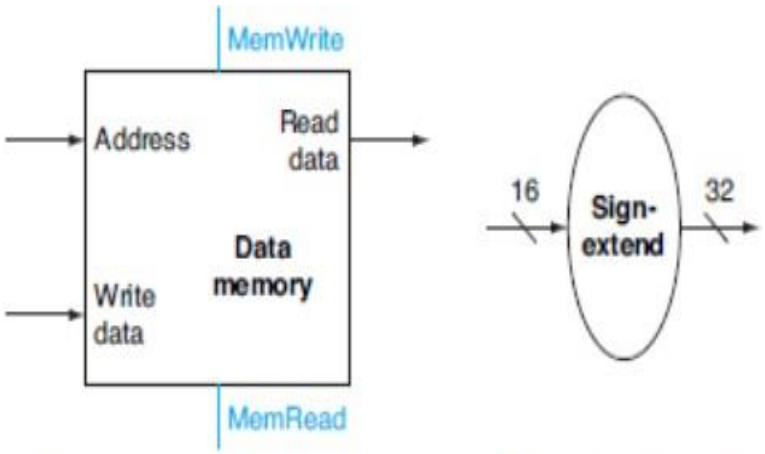
- One to specify the ***register number***to be written and one to supply the ***data*** to be written into the register.
- The register file always outputs the contents of whatever register numbers are on the Read register inputs. Writes, however, are controlled by the write control signal, which must be asserted for a write to occur at the clock edge. We need a total of four inputs (**three for register** numbers and one for **data)** and two outputs (both for data). The register number inputs are 5 bits wide to specify one of 32 registers (32 = 25), whereas the data input and two data output buses are each 32 bits wide.



Register and ALU

- The ALU, which takes two 32‑bit inputs and produces a 32‑bit result, as well as a 1-bit signal if the result is 0. The 4-bit control signal of the ALU.
- **Sign-extend**to increase the size of a data item by replicating the high-order sign bit of the original data item in the high order bits of the larger, destination data item.
- **Sign-extend** the 16‑bit offset field in the instruction to a 32‑bit signed value, and a data memory unit to read from or write to. The data memory must be written on store instructions; hence, data

memory has read and writes control signals, an address input, and an input for the data to be written into memory.



a. Data memory unit b. Sign extension unit

The two units needed to implement loads and stores, in addition to the register file and ALU

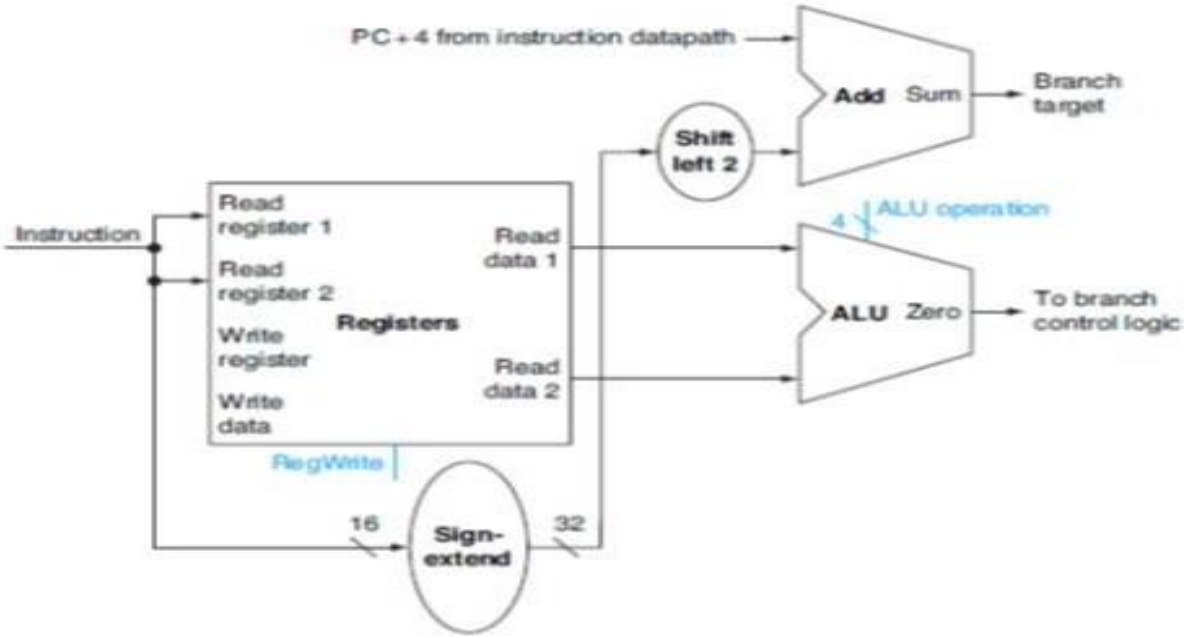
- The beq instruction has three operands, two registers that are compared for equality, and a 16‑bit offset used to compute the **branch target address**relative to the branch instruction address. Its form is beq $t1,$t2,offset. To implement this instruction, we must compute the branch target address by adding the sign-extended offset field of the instruction to the PC.

There are two details in the definition of branch instructions

- The instruction set architecture specifies that the base for the branch address calculation is the address of the instruction following the branch. Since we compute PC + 4 (the address of the next instruction) in the instruction fetch datapath, it is easy to use this value as the base for computing the branch target address.
- The architecture also states that the offset field is shifted left 2 bits so that it is a word offset; this shift increases the effective range of the offset field by a factor of 4.

To deal with the later complication, we will need to shift the offset field by 2.

- **Branch taken.**A branch where the branch condition **is satisfied** and the program counter (PC) becomes the branch target.All unconditional branches are taken branches.
- **Branch not taken or (untaken branch**) .A branch where the branch condition **is false** and the program counter (PC) becomes the address of the instruction that sequentially follows the branch



The datapath for a branch uses the ALU to evaluate the branch condition and a separate adder to compute the branch target as the sum of the incremented PC and the sign-extended, lower 16 bits of the instruction (the branch displacement), shifted left 2 bits.

- The unit labeled ***Shift left 2***is simply a routing of the signals between input and output that adds 00_{two} to the low-order end of the sign-extended offset field; no actual shift hardware is needed, since the amount of the "shift" is constant.
- Since we know that the offset was sign-extended from 16 bits, the shift will throw away only "**sign bits**." Control logic is used to decide whether the incremented PC or branch target should replace the PC, based on the Zero output of the ALU.

DESIGNING A CONTROL UNIT

3. Briefly explain about Control Implementation scheme.

Control Implementation scheme:

Over view:

- The ALU Control:
- Designing the Main Control Unit
- Operation of the Datapath

The ALU Control:

The MIPS ALU defines the **6** following combinations **of four control inputs:**

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Depending on the instruction class, the ALU will need to perform **one of these first five functions.** (NOR is needed for other parts of the MIPS instruction set not found in the subset we are implementing.)

- For load word and store word instructions, we use the ALU to compute the memory address by addition.
- For the R-type instructions, the ALU needs to perform one of the five actions (AND, OR, subtract, add, or set on less than), depending on the value of the 6-bit funct (or function) field in the low-order bits of the instruction.
- For branch equal, the ALU must perform a subtraction.
- We can generate the 4-bit ALU control input using a small control unit that hasinputs the function field of the instruction and a 2-bit control field, which we call ALUOp.
- ALUOp indicates whether the operation to be performed should be add (00) for loads and stores, subtract (01) for beq, or determined by the operation encoded in the funct field (10). The output of the ALU control unit is a 4-bit signal that directly controls the ALU by generating one of the 4-bit combinations shown previously.
- The below table shows how to set the ALU control inputs based on the 2-bit ALUOp control and the 6-bit function code.

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

How the ALU control bits are set depends on the ALUOp control bits and the different function codes for the R-type instruction.

- The opcode, listed in the first column, determines the setting of the ALUOp bits. All the encodings are shown in binary.
- Notice that when the ALUOp code is 00 or 01, the desired ALU action does not depend on the function code field; in this case, we say that we "don't care" about the value of the function code, and the funct field is shown as XXXXXX. When the ALUOp value is 10, then the function code is used to set the ALU control input.

- There are several different ways to implement the mapping from the 2-bit ALUOp field and the 6-bit funct field to the four ALU operation control bits.
- Because only a small number of the 64 possible values of the function field are of interest and the function field is used only when the ALUOp bits equal 10, we can use a small piece of logic that recognizes the subset of possible values and causes the correct setting of the ALU control bits.

As a step in designing this logic, it is useful to create a truth table for the interesting combinations of the function code field and the ALU Op bits.The below**truth table**shows how the 4-bit ALU control is set depending on these **two input fields.**

ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
0	1	X	X	X	X	X	X	0110
1	0	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	0	X	X	0	1	0	0	0000
1	0	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

The truth table for the 4 ALU control bits (called Operation).

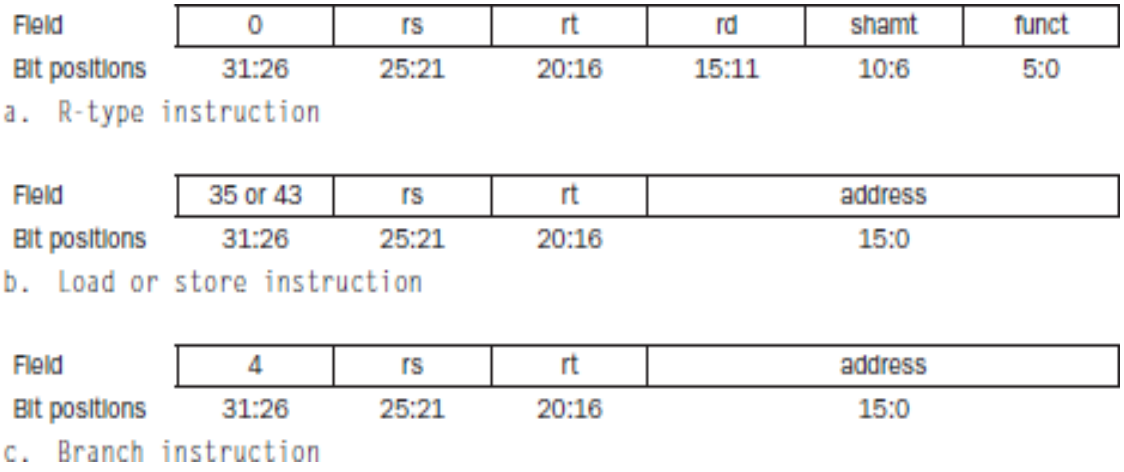
- The inputs are the **ALUOp** and **function code field**. Only the entries for which the ALU control is asserted are shown.
- Some don't-care entries have been added. For example, the ALUOp does not use the encoding 11, so the truth table can contain entries 1X and X1, rather than 10 and 01.
- Note that when the function field is used, the first 2 bits (F5 and F4) of these instructions are always 10, so they are **don't-care terms** and are replaced with XX in the truth table.
- **Don't-care term:**An element of a logical function in which the output does not depend on the values of all the inputs.

4. Give detail description about the Design of Main Control Unit.

Designing the Main Control Unit

To understand how to connect the fields of an instruction to the data path, it is useful to review the formats of the **three instruction classes:**

- The R-type instruction classes,
- Branch instruction classes, and
- Load-store instruction classes



The three instruction classes (R-type, load and store, and branch) use two different instruction formats

The jump instructions use another format, which we will discuss shortly.

(a). Instruction format for R-format instructions, which all have an opcode of 0. These instructions have three register operands: rs, rt, and rd. Fields rs and rt are sources, and rd is the destination. The ALU function is in the funct field and is decoded by the ALU control design in the previous section. The R-type instructions that we implement are add, sub, AND, OR, and slt. The shamt field is used only for shifts; we will ignore it in this chapter.

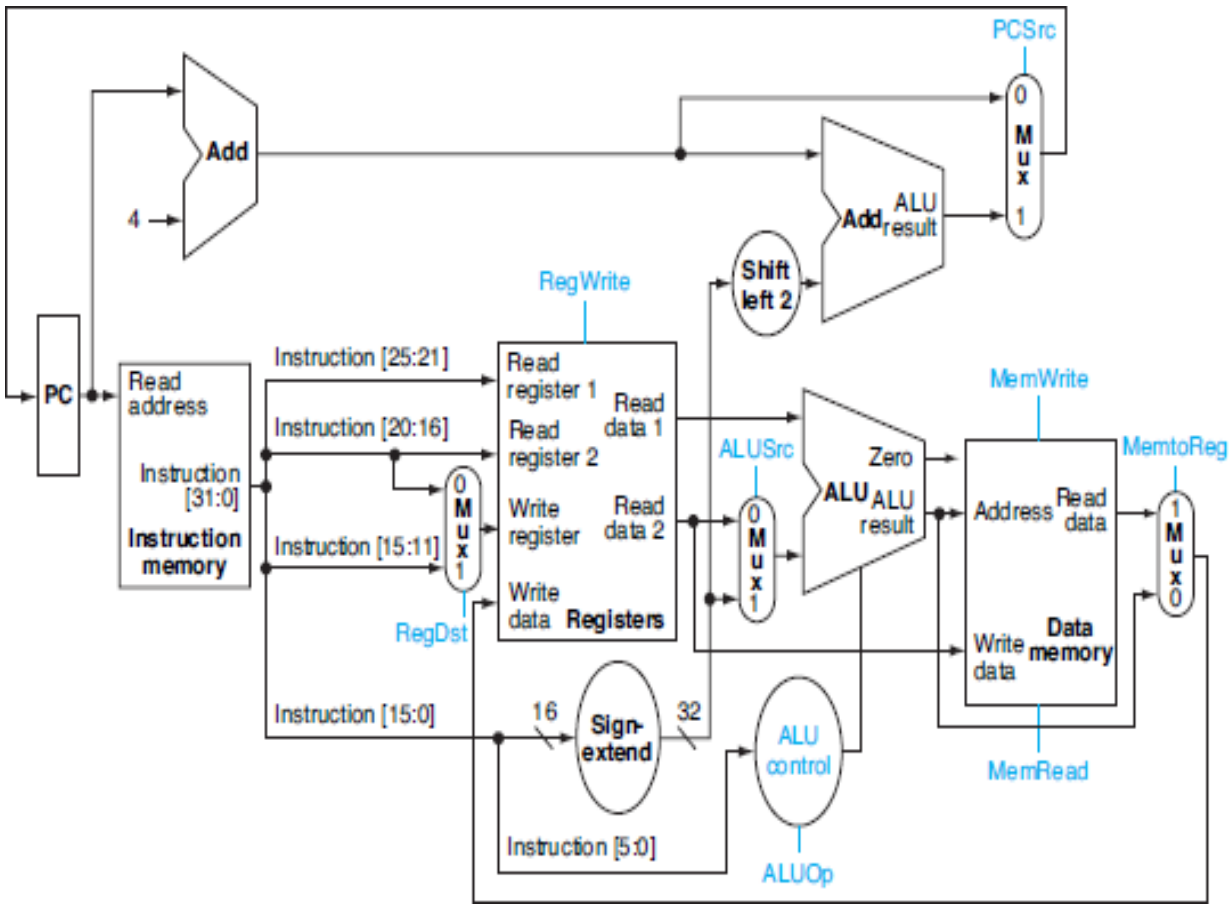
(b). Instruction format for load (opcode = 35ten) and store (opcode = 43ten) instructions. The register rs is the base register that is added to the 16-bit address field to form the memory address. For loads, rt is the destination register for the loaded value. For stores, rt is the source register whose value should be stored into memory.

(c). Instruction format for branch equal (opcode = 4). The registers rs and rt are the source registers that are compared for equality. The 16-bit address field is sign-extended, shifted, and added to the PC+4 to compute the branch target address.

There are several major observations about this instruction format that we will rely on:

- The op field, also called the **opcode**, is always contained in bits 31:26. We will refer to this field as Op[5:0].
- The two registers to be read are always specified by the rs and rt fields, at positions 25:21 and 20:16. This is true for the R-type instructions, branch equal, and store.
- The base register for load and store instructions is always in bit positions 25:21 (rs).
- The 16-bit offset for branch equal, load, and store is always in positions 15:0.
- The destination register is in one of two places. For a load it is in bit positions 20:16 (rt), while for an R-type instruction it is in bit positions 15:11 (rd). Thus, we will need to add a multiplexor to select which field of the instruction is used to indicate the register number to be written.

- Using this information, we can add the instruction labels and extra multiplexor (the Write register number input of the register file) to the simple datapath.
- These additions plus the ALU control block, the write signals forstate elements, the read signal for the data memory, and the control signals for the multiplexors. Since all the multiplexors have two inputs, they each require a single control line.
- Seven single bit control lines plus the 2-bit ALUOp controlsignal. We have already defined how the ALUOp control signal works, and it isuseful to define what the seven other control signals do informally before we determine how to set these control signals during instruction execution.



The data path of all necessary multiplexors and all control lines identified.

- The control lines are shown in color. The ALU control block has also been added. The PC does not require a write control, since it is written once at the end of every clock cycle; the branch control logic determines whether it is written with the incremented PC or the branch target address.

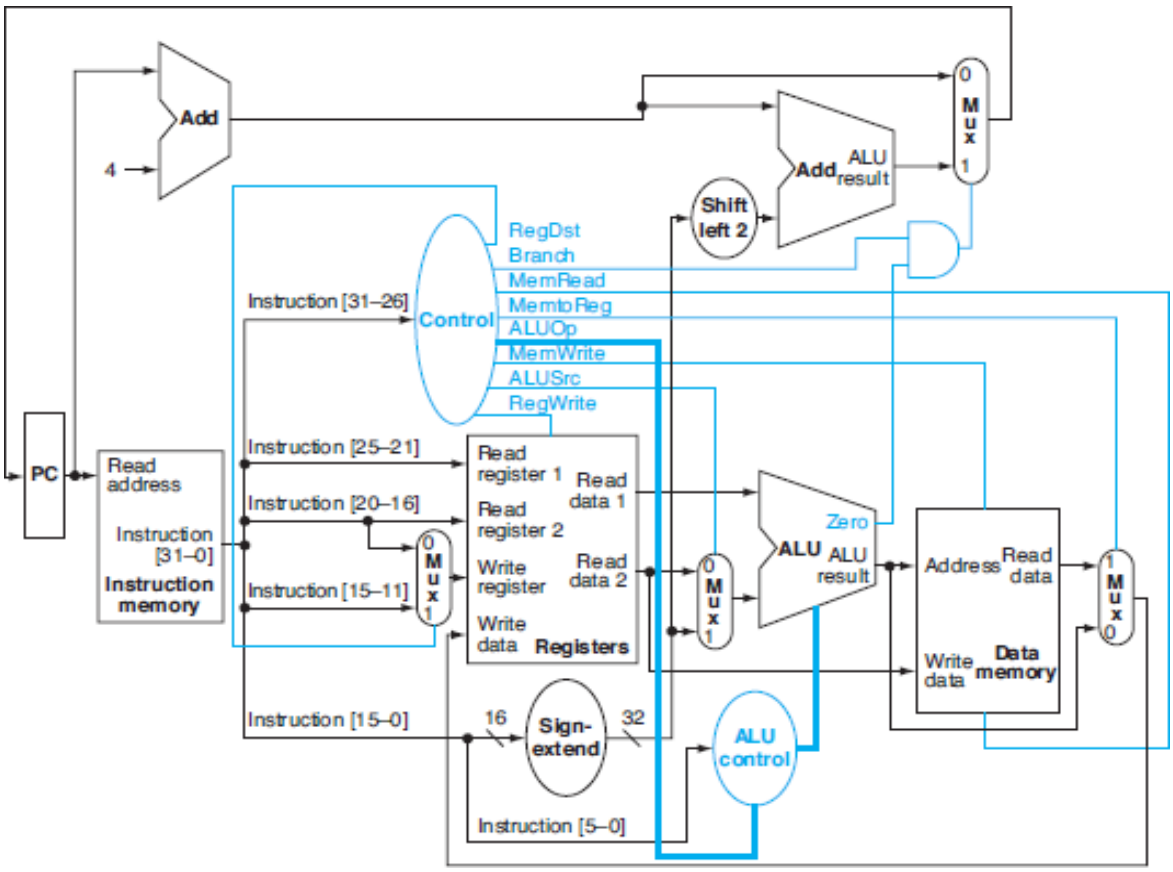
Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

The effect of each of the seven control signals.

- When the 1-bit control to a two-way multiplexor is asserted, the multiplexor selects the input corresponding to 1. Otherwise, if the control is disserted, the multiplexor selects the 0 input.
- Remember that the state elements all have the clock as an implicit input and that the clock is used in controlling writes. Gating the clock externally to a state element can create timing problems.

5. Briefly explain about Operation of the Data path with neat diagram. Apr. / May 2018,Nov/Dec2020. Nov/Dec 2021

Operation of the Data path:



The simple datapath with the control unit.

- The input to the control unit is the 6-bit opcode field from the instruction.
- The outputs of the control unit consist of three 1-bit signals that are used to control multiplexers (RegDst, ALUSrc, and MemtoReg), three signals for controlling reads and writes in the register file and data memory (RegWrite, MemRead, and MemWrite), a 1-bit signal used in determining whether to possibly branch (Branch), and a 2-bit control signal for the ALU (ALUOp).
- An AND gate is used to combine the branch control signal and the Zero output from the ALU.
- The AND gate output controls the selection of the next PC. Notice that PCSrc is now a derived signal, rather than one coming directly from the control unit. Thus, we drop the signal name in subsequent figures.
- The operation of the datapath for an R-type instruction, such as add $t1,$t2,$t3. Although everything occurs in one clock cycle, we can think of four steps to execute the instruction;

These steps are ordered by the flow of information:

1. The instruction is fetched, and the PC is incremented.

2. Two registers, $t2 and $t3, are read from the register file; also, the main control unit computes the setting of the control lines during this step.

3. The ALU operates on the data read from the register file, using the function code (bits 5:0, which is the funct field, of the instruction) to generate the ALU function.

4. The result from the ALU is written into the register file using bits 15:11 of the instruction to select the destination register ($t1). Similarly, we can illustrate the execution of a load word, such as

```
lw $t1, offset($t2)
```

The active functional units and asserted control lines for a load. We can think of a load instruction as operating in

Five steps (similar to the R-type executed in four):

1. An instruction is fetched from the instruction memory, and the PC is incremented.

2. A register ($t2) value is read from the register file.

3. The ALU computes the sum of the value read from the register file and the Sign-extended, lower16 bits of the instruction (offset).

4. The sum from the ALU is used as the address for the data memory.

5. The data from the memory unit is written into the register file; the register destination is givenby bits 20:16 of the instruction ($t1).

The data path in operation for a load instruction:

- The control lines, data path units, and connections that are active are highlighted.
- A store instruction would operate very similarly. The main difference would be that the memory control would indicate a write rather than a read, the second register value read would be used

for the data to store, and the operation of writing the data memory value to the register file would not occur.

The data path in operation for a branch-on-equal instruction.

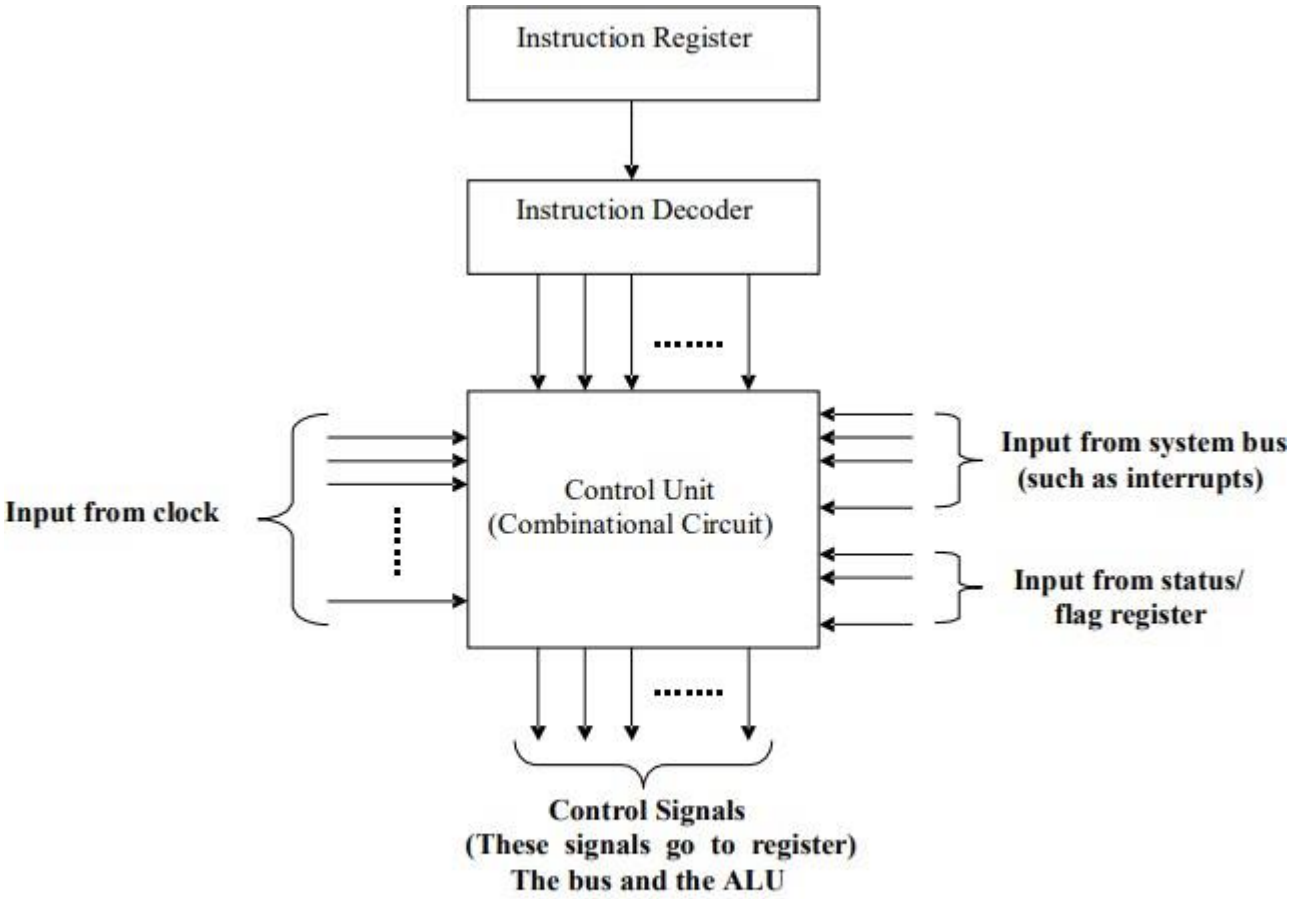
- Finally, we can show the operation of the branch-on-equal instruction, such as **beq $t1,$t2,offset** in the same fashion.
- It operates much like an R-format instruction, but the ALU output is used to determine whether the PC is written with PC + 4 or the branch target address.

The four steps for execution:

1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. Two registers, $t1 and $t2, are read from the register file.
3. The ALU performs subtract operation on the data values read from the register file. The value of PC+4is added to the sign-extended, lower 16 bits of the instruction (offset) shifted left by two; Result is inthe branch target address.
4. The Zero result from the ALU is used to decide which adder result to store into the PC.

HARDWIRED CONTROL AND MICRO PROGRAMMED CONTROL

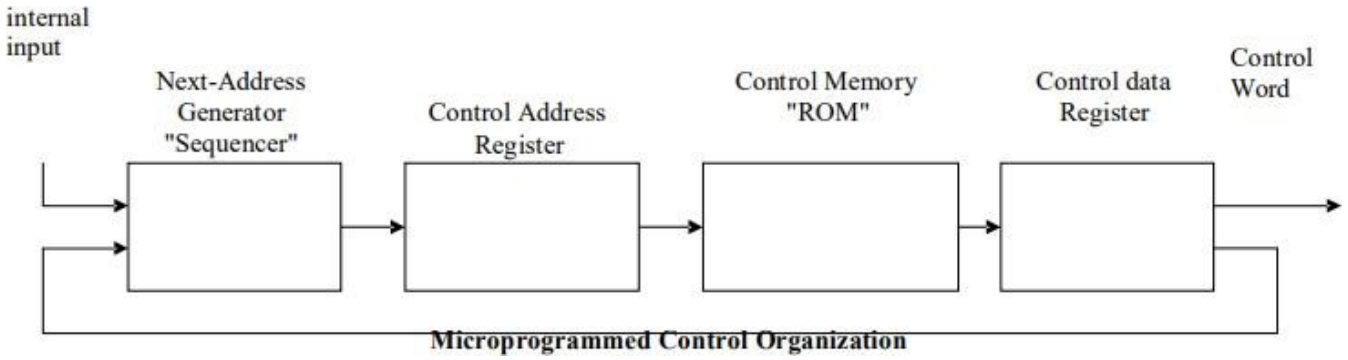
- Hardwired and Micro programmed Control For each instruction, the control unit causes the CPU to execute a sequence of steps correctly.
- In reality, there must be control signals to assert lines on various digital components to make things happen.
- For example, when we perform an Add instruction in assembly language, we assume the addition takes place because the control signals for the ALU are set to "add" and the result is put into the AC.
- The ALU has various control lines that determine which operation to perform. The question we need to answer is, "How do these control lines actually become asserted?" We can take one of two approaches to ensure control lines are set properly.
- The first approach is to physically connect all of the control lines to the actual machine instructions. The instructions are divided up into fields, and different bits in the instruction are combined through various digital logic components to drive the control lines.
- This is called hardwired control, and is illustrated in figure



Hardwired Control Organization

- The control unit is implemented using hardware (for example: NAND gates, flip-flops, and counters).We need a special digital circuit that uses , as inputs, the bits from the Opcode field in our instructions, bits from the flag (or status) register, signals from the bus, and signals from the clock.
- It should produce, as outputs, the control signals to drive the various components in the computer. The advantage of hardwired control is that is very fast.
- The disadvantage is that the instruction set and the control logic are directly tied together by special circuits that are complex and difficult to design or modify.
- If someone designs a hardwired computer and later decides to extend the instruction set, the physical components in the computer must be changed.
- This is prohibitively expensive, because not only must new chips be fabricated but also the old ones must be located and replaced. Microprogramming is a second alternative for designing control unit of digital computer (uses software for control).
- A control unit whose binary control variables are stored in memory is called a micro programmed control unit. The control variables at any given time can be represented by a string of 1's and 0's called a control word (which can be programmed to perform various operations on the component of the system).

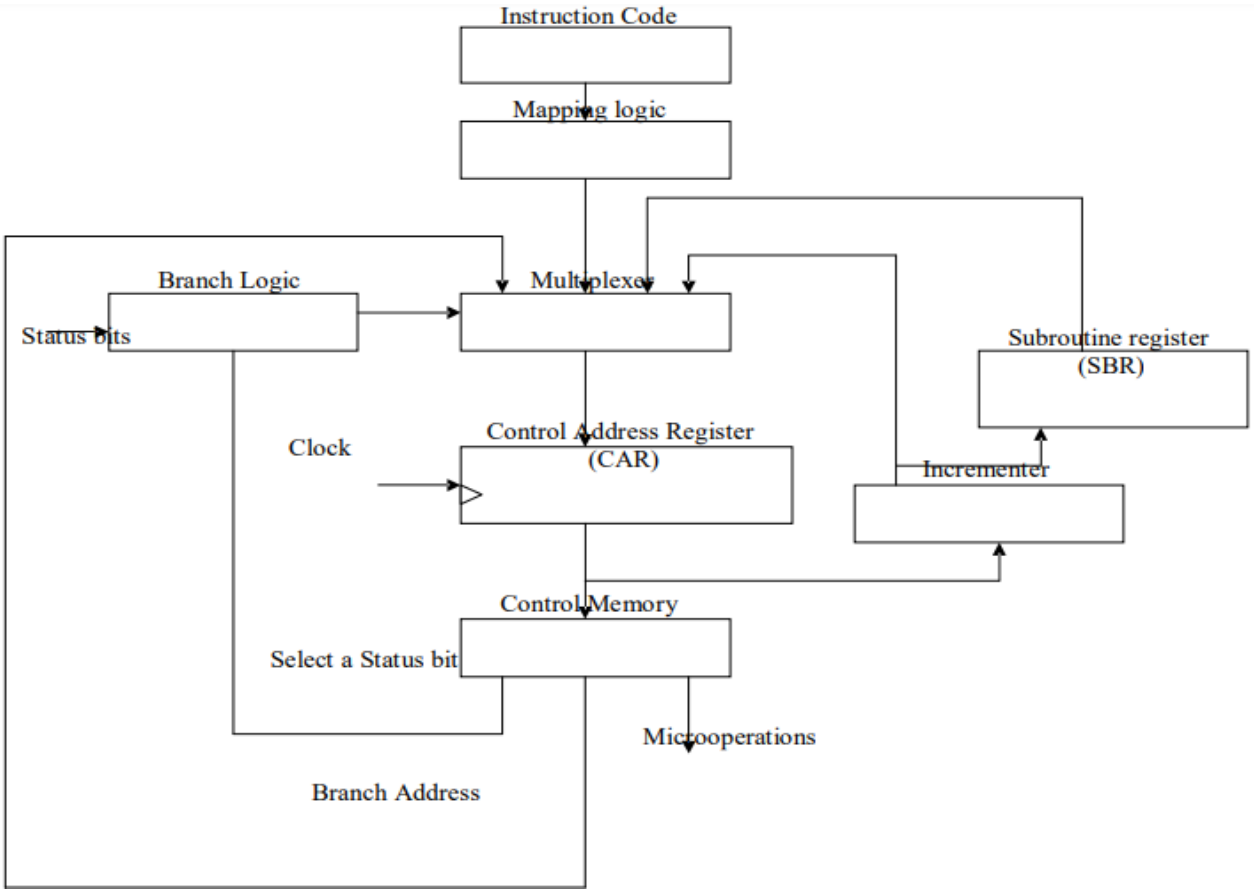
- Each word in control memory contains within it a microinstruction. The microinstruction specifies one or more micro operations for the system. A sequence of microinstructions constitutes a micro program.
- A memory that is part of a control unit is referred to as a control memory. A more advanced development known as dynamic microprogramming permits a micro program to be loaded initially from an auxiliary memory such as a magnetic disk.
- Control units that use dynamic microprogramming employ a writable control memory; this type of memory can be used for writing (to change the micro program) but is used mostly for reading.
- The general configuration of a micro programmed control unit is demonstrated in the block diagram of Figure.
- The control memory is assumed to be a ROM, within which all control information is permanently stored.



Microprogrammed Control Organization

- The control memory address register specifies the address of the microinstruction and the control data register holds the microinstruction read from memory the microinstruction contains a control word that specifies one or more micro operations for the data processor.
- Once these operations are executed, the control must determine the next address. The location of the next microinstruction may be the one next in sequence, or it may be locate somewhere else in the control memory. For this reason it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction.
- The next address may also be a function of external input conditions. While the micro operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.
- The next address generator is sometimes called a micro program sequencer, as it determines the address sequence that is read from control memory, the address of the next microinstruction can be specified several ways, depending on the sequencer inputs.
- Typical functions of a micro program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address or loading an initial address to start the control operations.

- The main advantages of the micro programmed control are the fact that once the hardware configuration is established; there should be no need for further hardware or wiring changes.
- If we want to establish are different control sequence for the system, all we need to do is specify different set microinstructions for control memory.
- The hardware configuration should not be changed for different operations; the only thing that must be changed is the micro program residing in control memory.
- Microinstructions are stored in control memory in groups, with each group specifying routine. Each computer instruction has micro program routine in control memory to generate the micro operations that execute the instruction.
- The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be to branch from one routine to another. The address sequencing capabilities required in a control memory are:
    1. Incrementing of the control address register.
    2. Unconditional branch or conditional branch, depending on status bit conditions.
    3. A mapping process from the bits of the instruction to an address for control memory.
    4. A facility for subroutine call and return.
- Figure shows a block diagram of control memory and the associated hardware needed for selecting the next microinstruction address.
- The microinstruction in control memory contains a set of bits to initiate micro operations in computer registers and other bits to specify the method by which the address is obtained.
- The diagram shows four different paths from which the control address register (CAR) receives the address.
- The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence.
- Branching is achieved by specifying the branch address in one of the fields of the microinstruction.
- Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.
- An external address is transferred into control memory via a mapping logic circuit. The return address for a subroutine is stored in a special register whose value is then used when the micro program wishes to return from the subroutine.



Selection address for control memory

PIPELINING

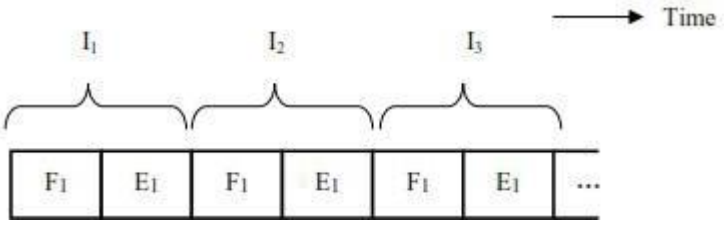
6. Explain a 4-stage instruction pipeline. Explain the issues affecting pipeline performance. (Or) Discus the basic concepts of pipelining. (Apr/May2012) (May/June2013)Nov / Dec 2015, 2016,Nov/Dec 2020.

OVERVIEW:

- **Role of cache memory**
- **Pipelining Performance**

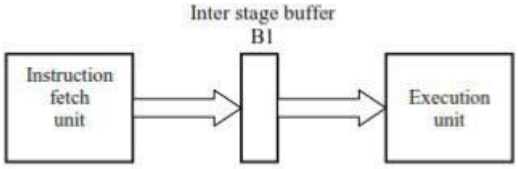
- Pipelining is an implementation technique in which multiple instructions are overlapped in execution pipelining is key to make processor fast.
- A pipeline can be visualized as a collection of processing segments through which binary information follows.
- In computer architecture Pipelining means executing machine instructions concurrently. The pipelining is used in modern computers to achieve high performance. The speed of execution of programs is influenced by **many factors.**
- One way to improve performance is to use faster circuit technology to build the processor and the main memory.
- Another possibility is to arrange the hardware so that more than one operation can be performed at the same time

- In this way, the number of operations performed per second is increased even though the elapsed time needed to perform anyone operation is not changed.
- Pipelining is a particularly effective way of organizing concurrent activity in a computer system.
- The basic idea is very simple. It is frequently encountered in manufacturing plants, where pipelining is commonly known as an assembly-line operation.
- The processor executes a program by fetching and executing instructions, one after the other. Let Fi and Ei refer to the fetch and execute steps for instruction Ii.
- An execution of a program consists of a sequence of fetch and execute steps as shown below.



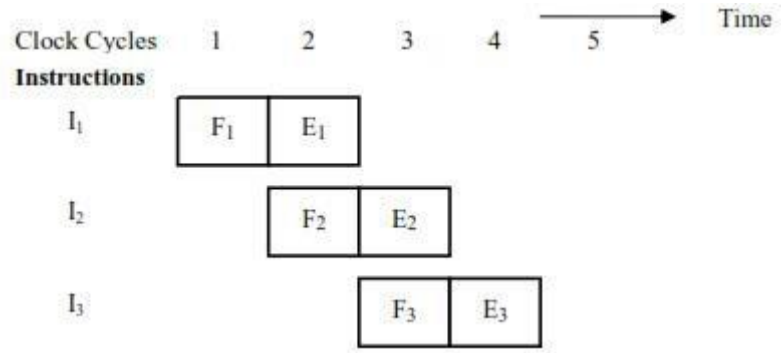
Sequential executions of instructions.

- Now consider a computer that has two separate hardware units, one for fetching instructions and another for executing them, as shown below. The instruction fetched by the fetch unit is deposited in an intermediate storage buffer B1.
- This buffer is needed to enable the execution unit to execute the instruction while the fetch unit is fetching the next instruction.
- The results of execution are deposited in the destination location specified by the instruction. The data can be operated by the instructions are inside the block labeled **"Execution unit".**



Hardware organization of pipelining.

- The computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can each be completed in one clock cycle.
- In the first clock cycle, the fetch unit fetches an instruction I1 (step F1) and stores it in buffer Bl at the end of the clock cycle.
- In the second clock cycle, the instruction fetch unit proceeds with the fetch operation for instruction I2 (step F2). Meanwhile, the execution unit performs the operation specified by instruction I1, which is available to it in buffer Bl (step E1).
- By the end of the second clock cycle, the execution of instruction I1 is completed and instruction I2 is available. Instruction I2 is stored in B1, replacing I1, which is no longer needed. Step E2 is performed by the execution unit during the third clock cycle, while instruction I3 is being fetched by the fetch unit. In this manner, both the fetch and execute units are kept busy all the time.

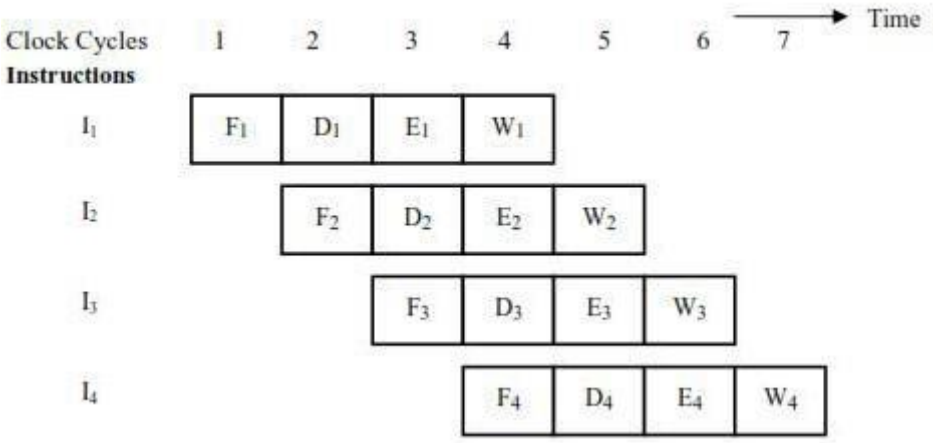


Pipelined executions of instructions (Instructions Pipelining)

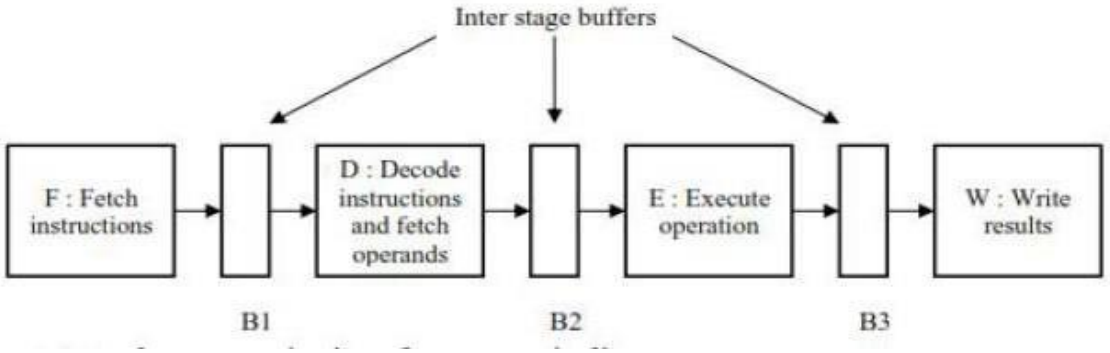
A pipelined processor may process each instruction in **four steps, as follows:**

- **F Fetch:** read the instruction from the memory.
- **D Decode:** decode the instruction and fetch the source operand(s).
- **E Execute:** perform the operation specified by the instruction.
- **W Write:** store the result in the destination location.

The sequence of events for this case is shown below. Four instructions are in progress at any given time. This means that four distinct hardware units are needed. These units must be capable of performing their tasks simultaneously and without interfering with one another. Information is passed from one unit to the next through a storage buffer.



Instruction execution divided into four steps.



Hardware organization of a 4- stage pipeline

For example, during clock cycle 4, the information in the buffers is as follows:

- Buffer B1 holds instruction I3, which was fetched in cycle 3 and is being decoded by the Instruction-decoding unit.
- Buffer B2 holds both the source operands for instruction I2 and the specification of the operation to be performed.
- Buffer B3 holds the results produced by the execution unit and the destination information for instruction 11.

ROLE OF CACHE MEMORY:

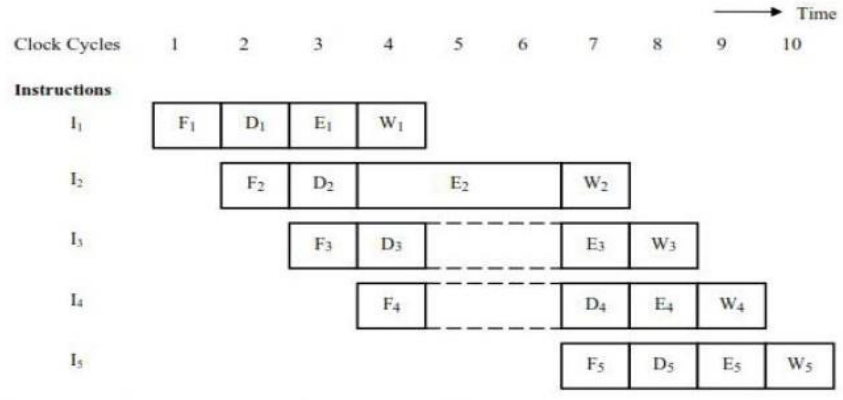
- Each stage in a pipeline is expected to complete its operation in one clock cycle. Hence, the clock period should be sufficiently long to complete the task being performed in any stage.
- Pipelining is most effective in **improving performance** if the tasks being performed in different stages require about the same amount of time.
- The use of cache memories solves the memory access problem. In particular, when a cache is included on the same chip as the processor, access time to the cache is usually the same as the time needed to perform other basic operations inside the processor.
- This makes it possible to divide instruction fetching and processing into steps that are more or less equal in duration. Each of these steps is performed by a different pipeline stage, and the clock period is chosen to correspond to the longest one.

7. Explain about Pipeline Performance. (Or) How to measure the performance of a pipeline? (Or)List the key aspects in gaining the performance in pipelined systems.(Apr/May2010) or Explain the difference types of pipeline hazards with suitable examples. (16 Marks) Apr / May 2015, 2016, Nov. / Dec. 2018 (Nov/Dec 2019)Nov/Dec 2020.Nov/Dec 2021

PIPELINE PERFORMANCE:

The pipelined processor completes the processing of one instruction in each clock cycle, which means that the rate of instruction processing is four times that of sequential operation.

- The potential increase in performance resulting from pipelining is proportional to the number of pipeline stages.
- However, this increase would be achieved only if pipelined operation as depicted could be sustained without interruption throughout program execution. Unfortunately, this is not the case.



Effect of an execution operation taking more than one clock cycle

Performance measures:

The various performance measures of pipelining are

- Throughput
- CPI
- Speedup
- Dependencies
- Hazards

Throughput

- The number of instruction executed per secondCPI(clock cycle per instruction)
- The CPI and MIPS can be related by the equation

  CPI=f/MIPS

  Where F is the clock frequency in MHz

Speedup

- Speedup is defined by

  S(m)=T(1)/T (m)

- Where T (m) is the execution time for some target workload on an m-stage pipeline and T(1) is the execution time for same target workload on a non-pipelined**Processor.**

Dependencies

- If the output of any stage interferes the execution of other stage then dependencies exists. There are two types of dependencies. They are
  1. **Control dependency**
  2. **Data dependency**

8. Give detail description about Pipelined data path and control. (Nov/Dec2014)(Apr/May2012) (Or) Discuss the modified data path to accommodate pipelined executions with a diagram. Apr/May 2016, 2017, 2018Nov/Dec 2021

Pipelined data path and control

Consider the three-bus structure suitable for pipelined execution with a slight modification to support a 4-stage pipeline. Several important changes are

- There are ***separate instruction and data caches***that use separate address and data connections to the processor. This requires two versions of the MAR register, IMAR for accessing tile instruction cache and DMAR for accessing the data cache.
- The PC is connected***directly***to the IMAR, so that the contents of the PC can be transferred to IMAR at the same time that an independent ALU operation is taking place.
- The data address in DMAR can be obtained ***directly from the register file or from the ALU***to support the register indirect and indexed addressing modes.

- Separate MDR registers are provided for ***read and write operations*.** Data can be transferred directly between these registers and the register file during load and store operations without the need to pass through the ALU.
- ***Buffer registers***have been introduced at the inputs and output of the ALU. These are registers SRCl, SRC2, and RSLT. Forwarding connections may be added if desired.
- The instruction register has been replaced with an ***instruction queue***, which is loaded from the instruction cache.
- The output of the instruction decoder is connected to the***control signal pipelin*e**. This pipeline holds the control signals in buffers B2 and B3.

The following operations can be performed independently in the process,

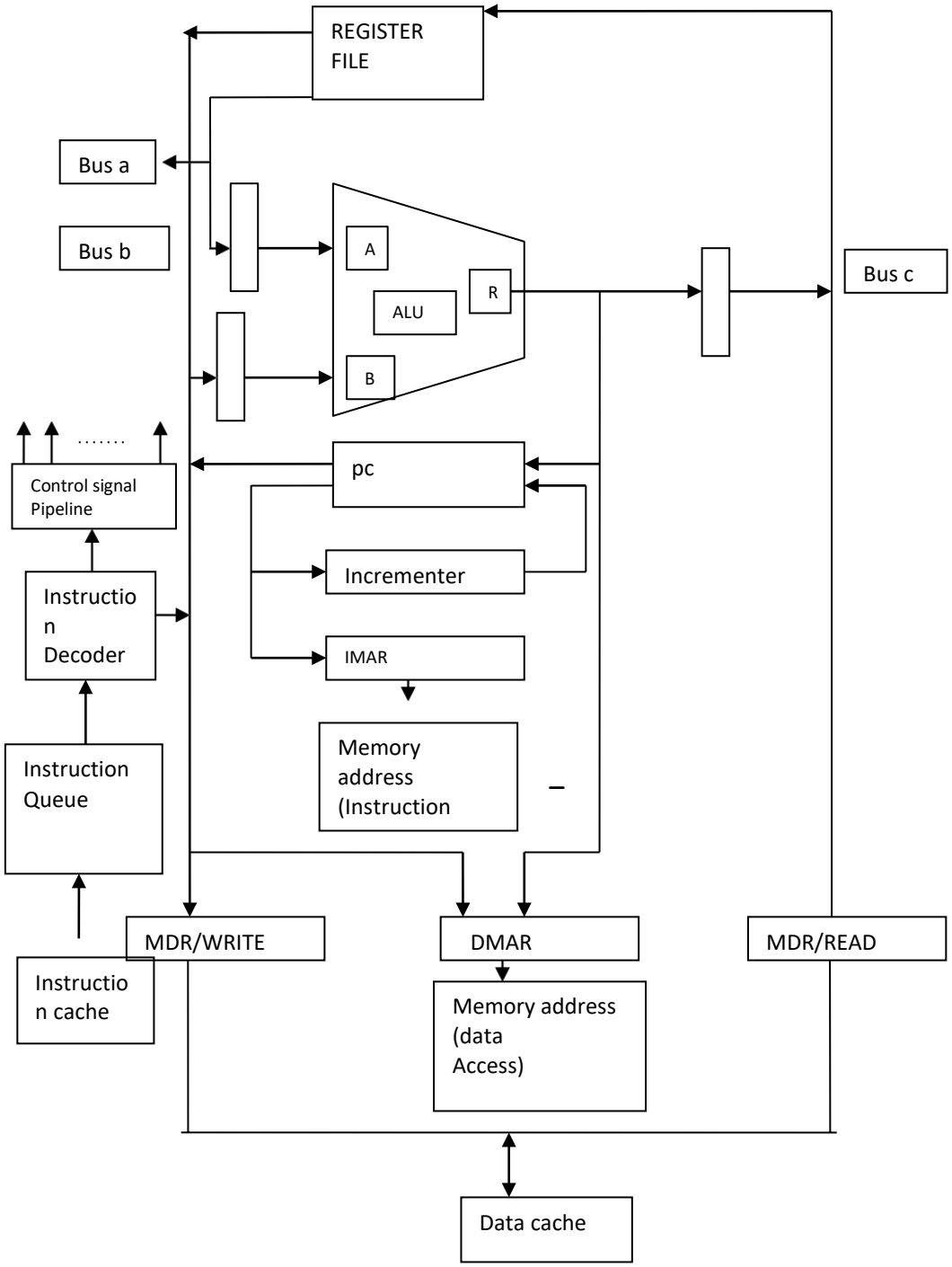
- Reading an instruction from the instruction cache
- Incrementing the pc
- Decoding the instruction
- Reading from or writing into the data cache.
- Reading the contents of up to two registers from the register file.
- Writing in to one register in the register file
- Performing an ALU operation.

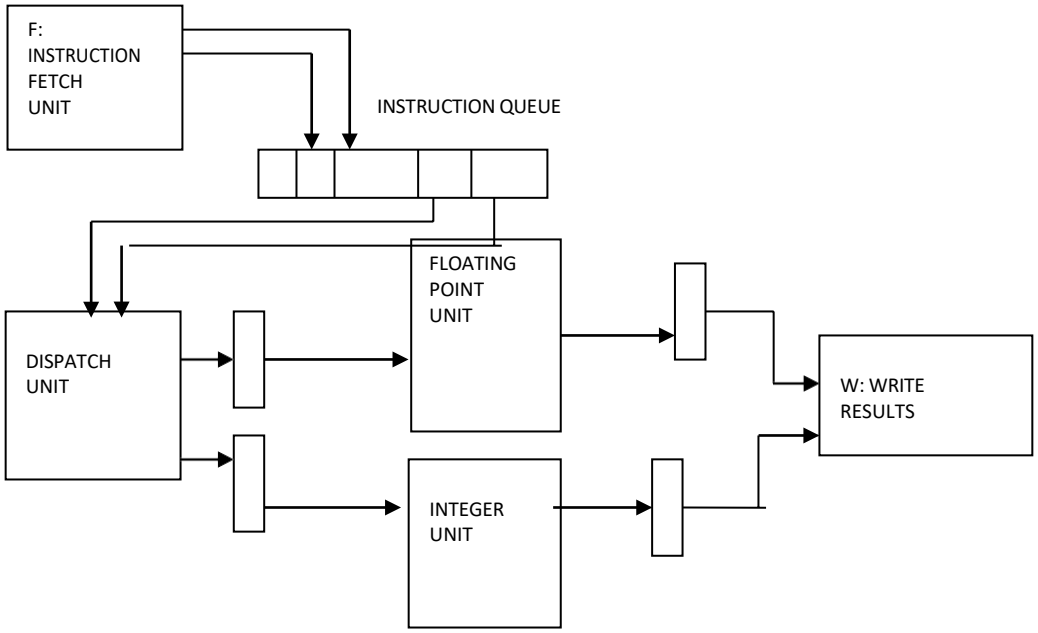
The structure provides the flexibility required to implement the four-stage pipeline.

For example: I1, I2, I3, I4

Be the sequence of four instructions.

- Write the result of instruction I1 into the register file.
- Read the operands of instruction I2 from the register file.
- Decode instruction I3
- Fetch instruction I4 and increment the PC.


REGISTER FILE
Bus a
Bus b
A
ALU
R
B
Bus c
.....
Control signal Pipeline
pc
Incrementer
IMAR
Instructio n Decoder
Memory address (Instruction
Instruction Queue
MDR/WRITE
DMAR
MDR/READ
Instructio n cache
Memory address (data Access)
Data cache



Use of instruction queue in hardware organization

Advantages of Pipelining:

- The cycle time of the processor is reduced; increasing the instruction throughput. Pipelining doesn't reduce the time it takes to complete an instruction; instead it increases the number of instructions that can be processed simultaneously ("at once") and reduces the delay between completed instructions (called 'throughput').
  The more pipeline stages a processor has, the more instructions it can process "at once" and the less of a delay there is between completed instructions. Every predominant general purpose microprocessor manufactured today uses at least 2 stages of pipeline up to 30 or 40 stages.
- If pipelining is used, the CPU Arithmetic logic unit can be designed faster, but more complex.
- Pipelining in theory increases performance over an un-pipelined core by a factor of the number of stages (assuming the clock frequency also increases by the same factor) and the code is ideal for pipeline execution.
- Pipelined CPUs generally work at a higher clock frequency than the RAM clock frequency, (as of 2008 technologies, RAMs work at a low frequencies compared to CPUs frequencies) increasing computers overall performance.

Disadvantages of Pipelining:

Pipelining has many disadvantages though there are a lot of techniques used by CPUs and compilersdesigners to overcome most of them; the following is a list of common drawbacks:

- The design of a non-pipelined processor is simpler and cheaper to manufacture, non-pipelined processor executes only a single instruction at a time.
- This prevents branch delays (in Pipelining, every branch is delayed) as well as problems when serial instructions being executed concurrently.

- In pipelined processor, insertion of flip flops between modules increases the instruction latency compared to a non-pipelining processor.
- A non-pipelined processor will have a defined instruction throughput. The performance of a pipelined processor is much harder to predict and may vary widely for different programs.
- Many designs include pipelines as long as 7, 10, 20, 31 and even more stages; a disadvantage of a long pipeline is when a program branches, the entire pipeline must be flushed (cleared).
- The higher throughput of pipelines falls short when the executed code contains many branches: the processor cannot know in advance where to read the next instruction, and must wait for the branch instruction to finish, leaving the pipeline behind it empty.
- This disadvantage can be reduced by predicting whether the conditional branch instruction will branch based on previous activity.
- After the branch is resolved, the next instruction has to travel all the way through the pipeline before its result becomes available and the processor resumes "working" again.
- In such extreme cases, the performance of a pipelined processor could be worse than non-pipelined processor.
- Unfortunately, not all instructions are independent. In a simple pipeline, completing an instruction may require 5 stages. To operate at full performance, this pipeline will need to run 4 subsequent independent instructions while the first is completing.
- Any of those 4 instructions might depend on the output of the first instruction, causing the pipeline control logic to wait and insert a stall or wasted clock cycle into the pipeline until the dependency is resolved.
- Fortunately, techniques such as forwarding can significantly reduce the cases where stalling is required.
- Self-modifying programs may fail to execute properly on a pipelined architecture when the instructions being modified are near the instructions being executed.
- This can be caused by the instructions may already being in the Prefetch Input Queue, so the modification may not take effect for the upcoming execution of instructions. Instruction caches make the problem even worse.
- **Hazards**: When a programmer (or compiler) writes assembly code, they generally assume that each instruction is executed before the next instruction is being executed.
- When this assumption is not validated by pipelining it causes a program to behave incorrectly, the situation is known as a **hazard**. Various techniques for resolving hazards or working around such as forwarding and delaying (by inserting a stall or a wasted clock cycle) exist.

HAZARD

- Any condition that causes the pipeline to stall is called a ***hazard***.

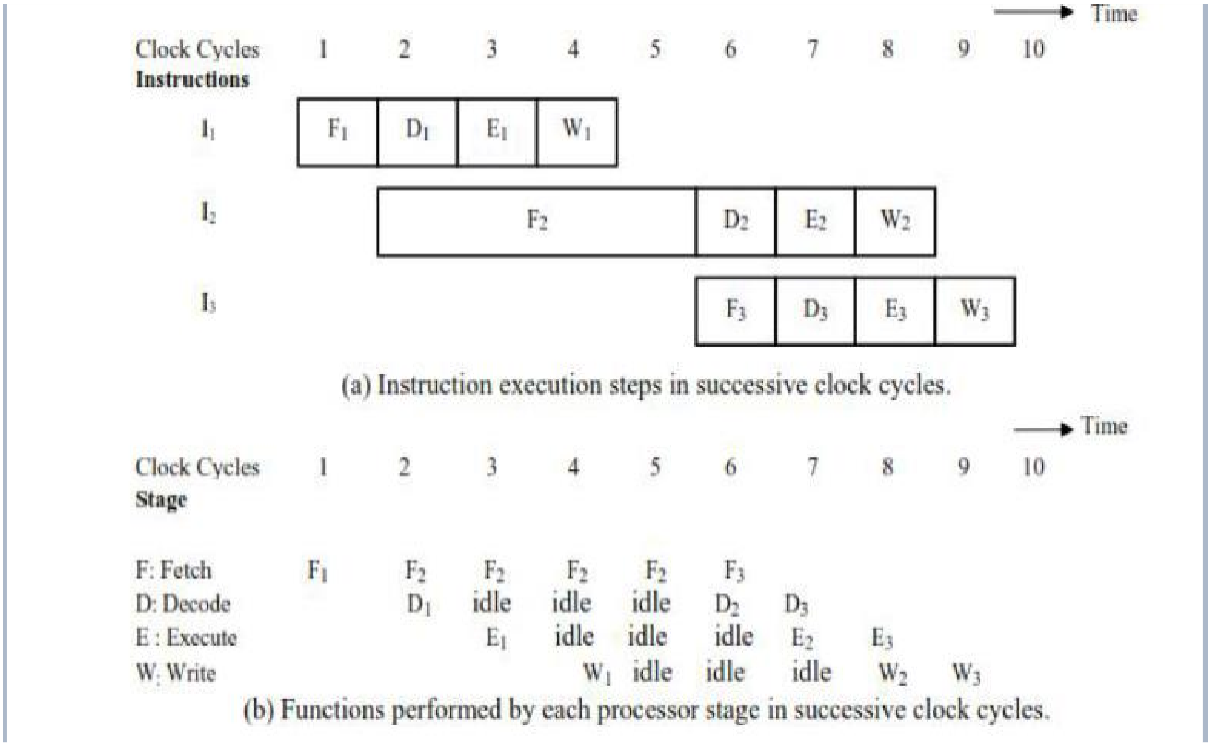
There are three type of hazard

I. Data Hazards

II. Control/instruction hazards

III. Structural Hazard

- The operation specified in instruction I2 requires three cycles to complete, from cycle 4 through cycle 6. Thus, in cycles 5 and 6, the Write stage must be told to do nothing, because it has no data to work with. Meanwhile, the information in buffer B2 must remain intact until the Execute stage has completed its operation.
- This means that stage 2 and, in turn, stage1 are blocked from accepting new instructions because the information in B1 cannot be overwritten.
- Thus, steps D4 and F5 must be postponed as shown below. Pipelined operation is said to have been stalled for two clock cycles. Normal pipelined operation resumes in cycle 7.



Pipeline stall caused by a cache miss in F2

9. Briefly explain about how to handle the Data hazard.(Nov/Dec2012, 2014)(Apr/May2015) Or What is a data hazard? How do you overcome it? Discuss its side effects.(Apr/May 2014) Or Describe operand forwarding in a pipeline processor with a diagram. (6) Apr/ May 2017 Nov/Dec 2020.

Handling Data hazard

- Operand forwarding
- Handling data hazards by introducing NOP (software method)

- When stalls are required

DATA HAZARD

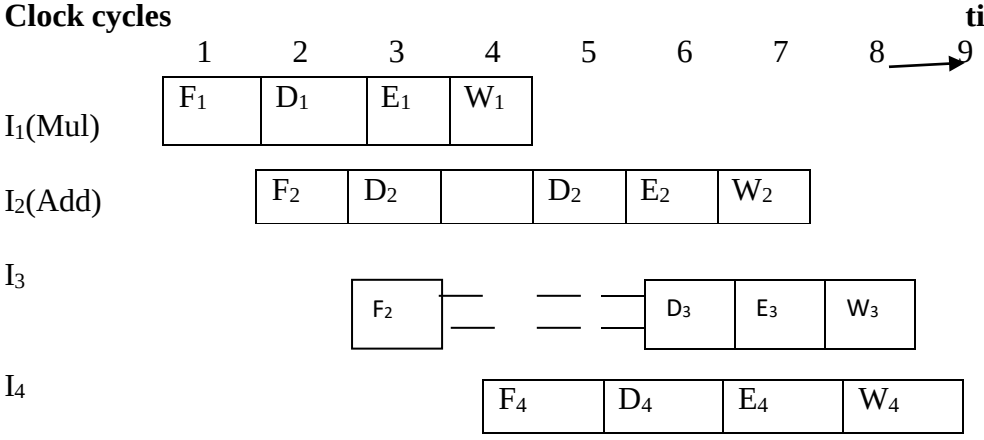
- When either the source or the destination operands of an instruction are not available at the time expected in the pipeline and as a result pipeline is stalled, we say such a situation is a **data hazard.**
- Consider a program with two instructions I1 followed by I2; when this program is executed in a pipeline; the execution of these two instructions can be performed concurrently. Such case, the result of I1 may not be available for the execution of I2. If the result of I2 is dependent on the result of I1, we may get incorrect result if both are executed concurrently. For example; assume A=8 in the following two operations.

I_1:A<- A + 6
I_2:B<- A X 2

- When these two operations are performed in the order given, the result is B=28. But if they are performed concurrently, the value of A used in computing B would be the original value 8; leads to an incorrect result. In the case data used in the I_2 depend on the result of I1. The hazard due to such situation is called data **hazard** or **data dependent hazard.**
- To avoid incorrect result we have to execute dependent instruction one after the other.
- The data dependency arises when the destination of one instruction is used as a source in the nest instruction. For example the two instruction

Mul	R_2, R_3, R_4
Add	R_5, R_4, R_6

- The contents of register R2 are multiplied by the contents of R_3 and the result is stored in R_4, which in turn is one of the two source operands of the Add instruction. Assume that the multiply operation takes one clock cycle to complete; execution would proceed as shown in figure below.



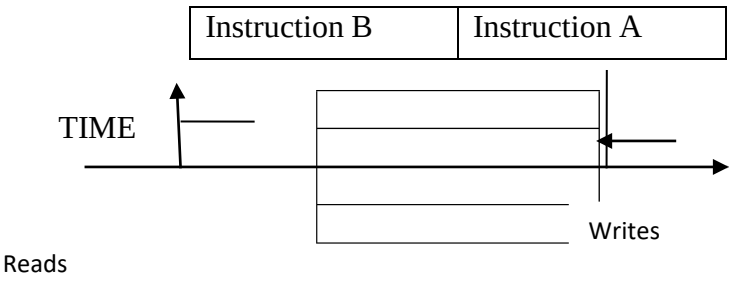
Pipeline stalled by data dependency between D2 and w1

- In the above figure the decode unit decodes the Add instruction in cycle 3, it realizes that R_4 is used as a source operand.
- Hence, the D step of that instruction cannot be completed until the step W of multiply instruction has been completed.

- Step D_2 must be delayed to clock cycle 5, and is shown as step D_{2A} in the figure. Instruction I_3 is fetched in cycle 3, but its decoding must be delayed because step D_3 cannot precede D_2.
- Hence, pipelined execution is stalled for two cycles.

Operand Forwarding:

- Consider instruction A and B as illustrated in the figure, B tries to read a register before A has written if and gets the old value.

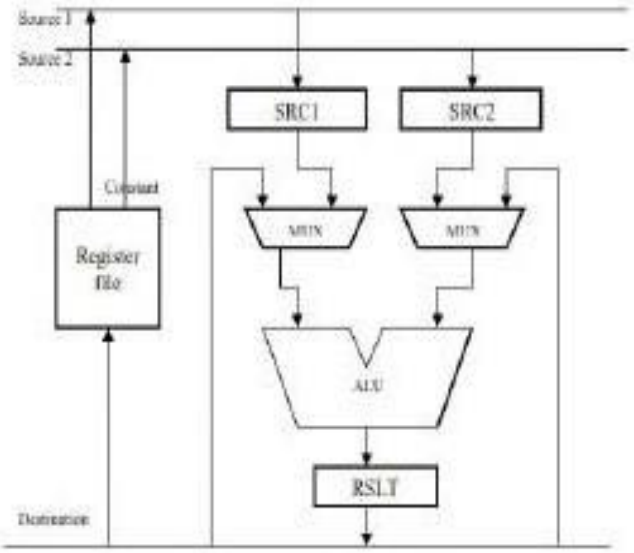


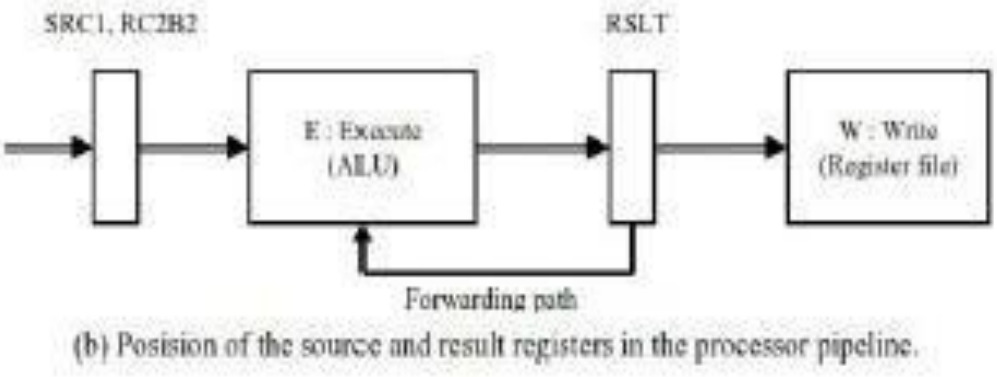
- This is quite common and called **read after write data hazard.** This situation is solved with a simple hardware technique called **operand forwarding.**

Example:

Add $ s0, $ t0, $ t1
Sub $ t2, $ s0, $ t3

(A) Forwarding Datapath





(b) Posision of the source and result registers in the processor pipeline.

Operand forwarding in a pipeline processor

Figure shows a part of the processor data path involving the ALU and the register file. This arrangement is similar to the three bus structure except that registers SRC1, SRC2 and RSLT have been added.

- These registers constitute inter stage buffers needed for pipelined operation. The two multiplexers connected at the inputs to the ALU allow the data on the destination bus to be selected instead of the contents of either the SRC1 or SRC2 register.
- After decoding instruction I_2 and detecting the data depending, a decision is made to use data forwarding. Register R_2 is read and loaded in register. SRC1 in clock cycle 3.
- In the next cycle, the product produced by instruction I, is available in register RSLT and because of the forwarding connection it can be used in steep E_2. Hence execution of I_2 proceeds without interruption.

Handling Data Hazard in Software:

- Another way to avoid data dependencies is to use software.
- In the software approach compiler can introduce two cycle delay needed between instruction I_4 and I_2 in figure by NOP(no operation) instruction as follows:

I_1: Mul R_2,R_3,R_4

NOP

NOP

I_2: Add R_5,R_4,R_6

- In the responsibility for detecting such dependencies is left entirely to the software the compiler must insert NOP instruction to obtain a correct result. This possibility illustrates the close link between the compiler and the hardware.

Side Effects:

- When a location other one explicitly named in an instruction as a destination operand is affected, the instruction is said to have a side effect.
- An instruction that uses an auto increment or auto decrement addressing mode is an example.
- In addition to storing a new data in its destination location, the instruction changes the contents of a source register used to access one of its operands.
- For example, a stack instructions, such as push and pop, produce similar side effects because they implicitly use the auto increment and auto decrement addressing modes
    - Another possible side effect involves the condition code flags, which are used by instructions such as conditional branches and add with carry.

      Add R_1,R_3

      Add with carry R_2,R_4.
    - An implicit dependency exists between these two instructions through the carry flag. This flag is set by the first instruction and used in the second instruction, which performs the operation.

      R_4<-[R_2] + [R_4] + carry.

- Instructions that have side effects give rise to multiple data dependencies, which lead to a substantial increase in the complexity of the hardware or software needed to resolve them. For this reason, instructions designed for execution on pipelined hardware should have few side effects.
- Ideally, only the content of the destination location, either a register or a memory location, should be affected by any given instruction. Side effects, such as setting the condition code flags or updating the contents of an address pointer, should be kept to a minimum.

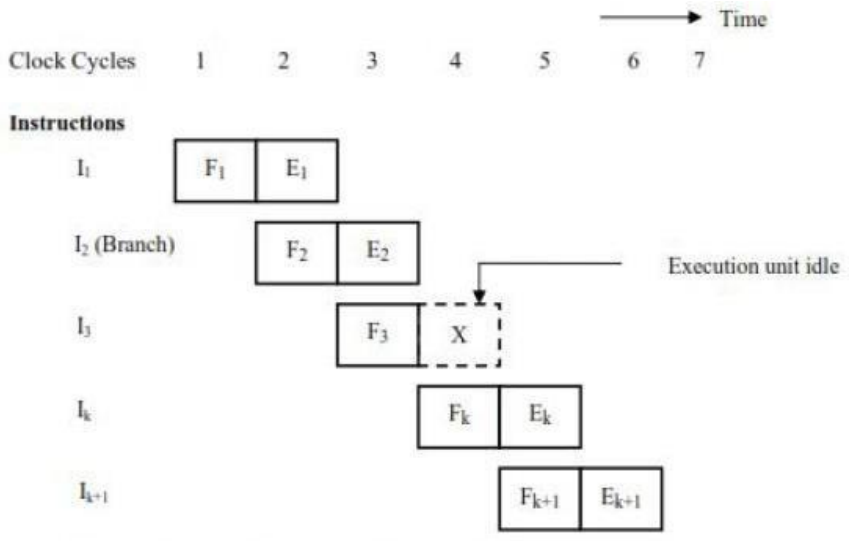
Over View:

- Unconditional Branch
- Conditional Branch
- Branch Prediction

- This type of hazard arises from pipeline of branch and other instructions that change the contents of PC. (i.e) Trying to make a decision based on the results of instruction while others are executing.

Unconditional Branch:

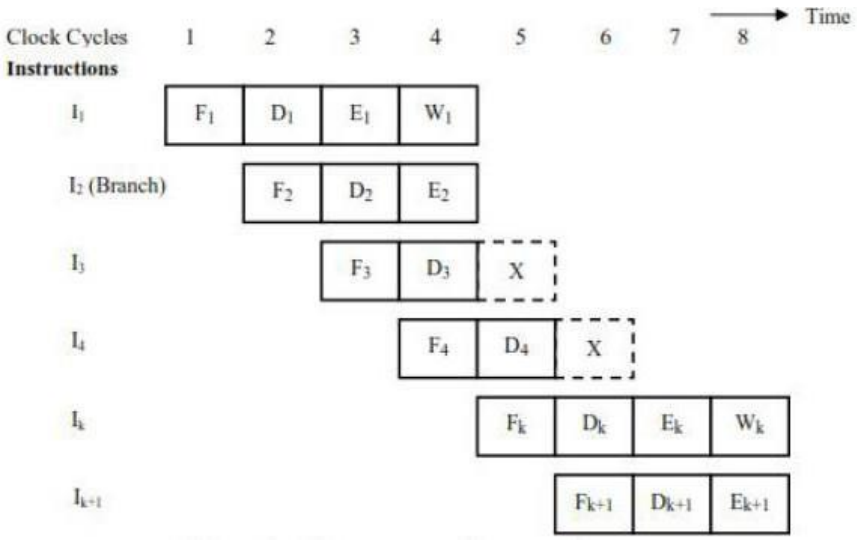
The belowfigure shows a sequence of instructions being executed in a two-stage pipeline instruction I_1 to I_3 are stored at consecutive memory address and instruction I_2 is a branch instruction.



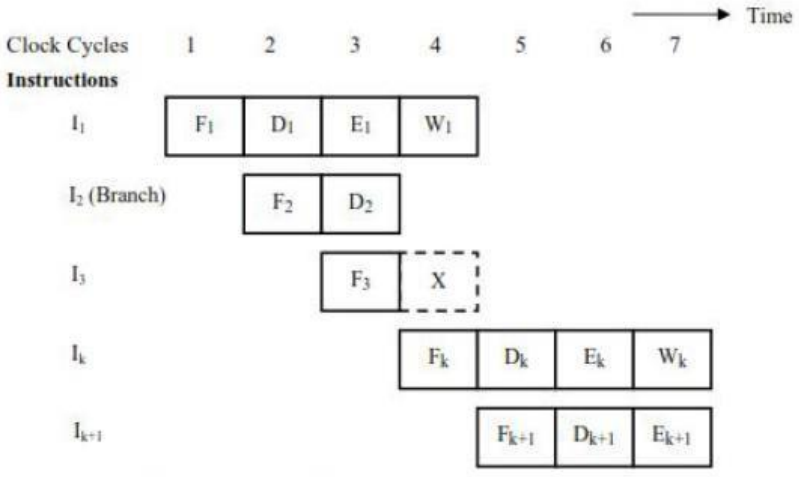
An idle cycle caused by a branch instruction

- If branch is taken as shown in figure, then the PC value is not known till the end of I_2. Next three instructions are fetched even though they are not required. Hence they have to be flushed after branch is taken and new set of instructions have to be fetched from the branch address.

- In figure, clock cycle 3, the fetch operation for instruction I_3 is in progress at the same time the branch instruction is being decoded. In clock cycle 4, the processor must discard I_3,which has been incorrectly fetched, and fetch instruction I_k. Thus the pipeline is stalled for one clock cycle.
- The time lost as a result of branch instruction is often referred to as the **branchpenalty.**
- The branch penalties can be reduced by proper scheduling through compiler technique. The basic idea behind these techniques is to fill the 'delay slot' with some useful instruction which in most cases will be executed.
- For longer pipeline, the branch penalty may be higher. Reducing the branch penalty requires the branch address to be computed earlier in the pipeline.
- The instruction fetch unit has dedicated hardware to identify a branch instruction and compute branch target address as quickly as possible after an instruction is fetched.
- With these additional hardware both these tasks can be performed in step D_2, leading to the sequence of events shown in figure. In this case the branch penalty is only one clock cycle.



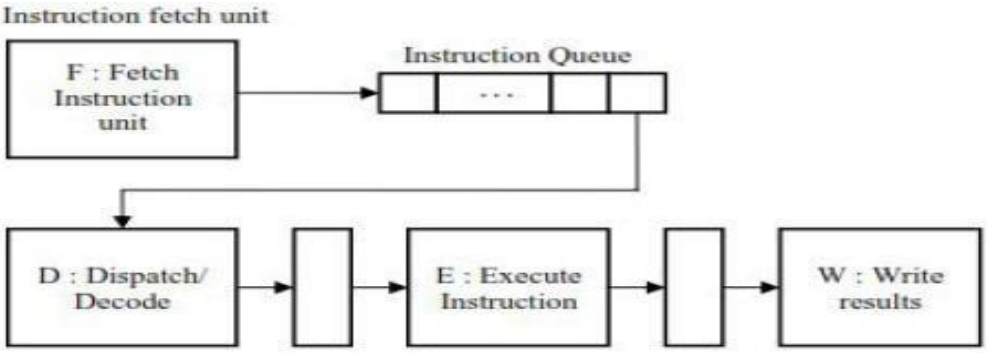
Branch address computed in execute stage



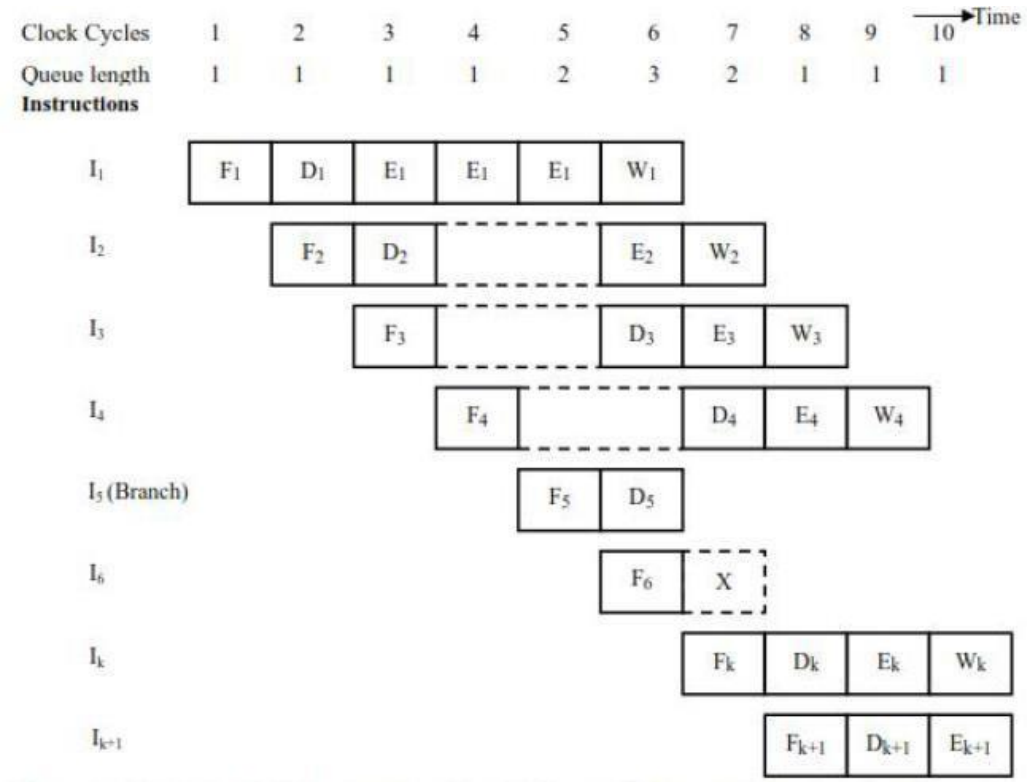
Branch address computed in decode stage

Instruction Queue and Pre fetching:

- The **Fetch unit** may contain instruction queue to store the instruction before they are needed to avoid interruption.
- Another unit called **dispatch unit** takes instruction from the front of the queue and sends them to the section unit. The dispatch unit also performs the decoding function.



Hardware organization of instruction queue



Branch timing in the presence of an instruction queue. Branch target address in computed D stage

- The fetch unit must have sufficient decoding and processing capability to recognize and execute branch instruction.
- The fetch unit always keeps the instruction queue filled at all times.
- Fetch unit continues to fetch instructions and add them to the queue.
- Similarly if there is a delay in fetching instructions, the dispatch unit continues to issue instruction from the instruction queue.
- Every fetch operation adds one instruction to the queue and each dispatch operation reduces queue length by one. Hence queue length remains same for first four clock cycle.

- Instruction I_5 is a branch instruction Its target instruction, I_k , is fetched in cycle 7, and instruction I_6 is discarded. The branch instruction would normally cause a stall in cycle 7 as a result of discarding instruction I_6. Instead, instruction I_4 is dispatched from queue to the decoding stage. After discarding I6, the queue length drops to 1 in cycle 8. The queue length will be at this value until another stall is encountered.
- The sequence of instruction completions instruction I_1,I_2,I_3,I_4 and I_k complete execution in successive clock cycle. Hence the branch instruction does not increase the overall execution time.
- This is because the instruction fetch unit has executed branch instruction concurrently with the execution of other instruction. This technique is referred to as **branch folding.**
- Branch folding occurs only if at the time a branch instruction encountered, at least one instruction is available in the queue other than the branch instruction.

11. Explain about Conditional Branches: (Apr/May2014)

Conditional Branches:

- The conditional branching is a major factor that affects the performance of instruction pipelining. When a conditional branch is executed if may or may not change the PC.
- If a branch changes the PC to its target address, it is a taken branch, if it falls through, it is not taken. The decision to branch cannot be taken until the execution of that instruction has been completed.

Delayed Branch:

- The location following the branch instruction is called branch delay slot. There may be more than one branch delay slot depending on the time it takes to execute the branch instruction.
- The instruction in the delay slot is always fetched at least partially executed before the branch decision is made and the branch target address is completed.
- A technique called **delayed branching** can minimize the penalty caused by conditional branch instruction.
- The instruction in the delay slot is always fetched. Therefore, arrange the instructions which are fully executed, whether or not the branch is taken. Place the useful instructions in the delay slot.If no useful instructions available; fill the slot with NOP instructions.

EXAMPLE:

LOOP	shift-Left	R_1
	Decrement	R_2
	Branch = 0	LOOP
NEXT	Add	R_1, R_2

(a). Original program loop

LOOP	Decrement	R_2

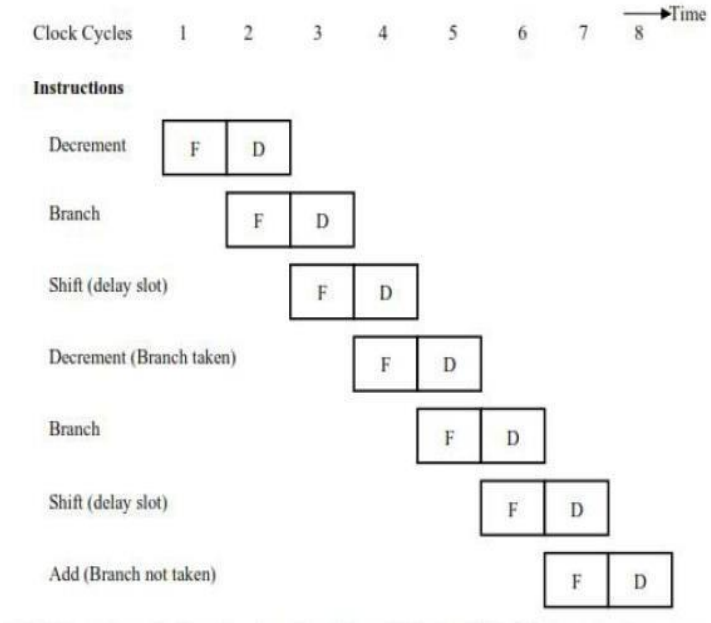
	Branch = 0	LOOP
	shift-Left	R_1
NEXT	Add	R_1, R_2

(b). Reordering instructions

Reordering of instructions for a delayed branch

- Register R_2 is used as counter to determine the number of times contents of R_1 are shifted left. For a processor with one delay slot, the instructions can be recorded a above. For a processor with one delay slot, the instructions can be reordered as shown in above figure(b).
- The shift instruction is fetched while branch instruction is being executed.
- After evaluating the branch condition, the processor fetches the instruction at LOOP or at NEXT, depending on whether the branch condition is true or false respectively. In either case, it completes the execution of the shift instructions.

The sequence of events during the last two passes in the loop is illustrated in figure.



Execution timing showing the delay slot being filled during two passes through the loop

- Pipelined operation is not interrupted at any time, and there are no idle cycles. Branching takes place one instruction later than where branch instruction appears in the sequence, hence named "delayed branch".

12. Explain about Branch prediction Algorithm. Nov / Dec 2016

Branch prediction:

Over view:

- Speculative execution
- Static prediction
- Dynamic Branch Prediction

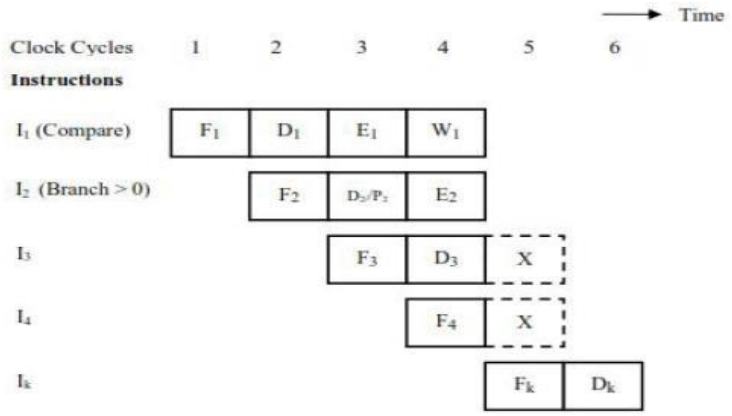
Branch prediction

- Prediction techniques can be used to check whether a branch will be valid or not valid. The simplest form of branch prediction is to assume that the branch will not take place and to continue to fetch instructions in sequential address order. Until the branch condition is evaluated, instruction execution along the predicted path must be done on a speculative basis.

Speculative execution means that instructions are executed before the processor is certain that they are in the correct execution sequence.

The below figure illustrate the incorrectly predicted branch.

- Figure shows a compare instruction followed by Branch > 0 instruction. In cycle 3 the branch prediction takes; the fetch unit predicts that branch will not be taken and it continues to fetch instruction I_4 as I_3 enters the Decode Stage.
- The result of compare operation is available at the ends of cycle 3. The branch condition is evaluated in cycle 4. At this point, the instruction fetch unit realizes that the prediction was incorrect and the two instructions in the execution pipe are purged.
- A new instruction I_k is fetched from the branch target address in clock cycle 5. We will examine prediction schemes static and dynamic prediction.



Timing when branch decision has been incorrectly predicted as not taken

Static prediction

- Static prediction is usually carried out by the compiler and it is static because the prediction is already known even before the program is executed.

Dynamic Branch Prediction: (May/June2013)

- Dynamic prediction in which the prediction **may change depending on execution** history.

Algorithm:

- If the branch taken recently, the next time if the same branch is executed, it is likely that the branch is take.

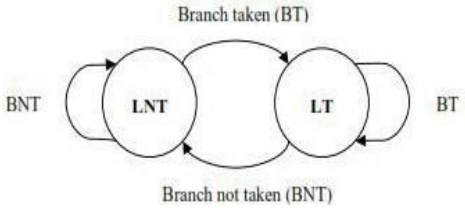
  State 1:LT: Branch is likely to be take.

  State 2:LNT:Branch is likely not to be take.

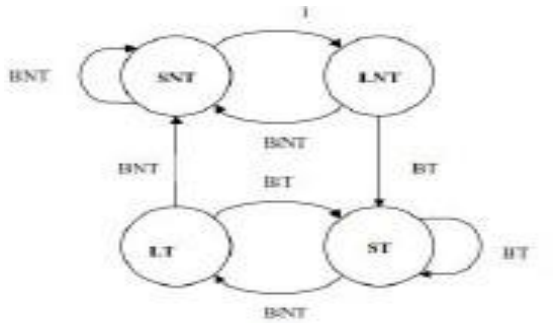
The algorithm is stated in state LNT when the branch is executed.

- If the branch is taken, the machine moves to LT. Otherwise it remains in state LNT.

- The branch is predicted as taken if the corresponding state machine is in state LT, otherwise it is predicted as not take.
- The branch is predicted as taken if the corresponding state machine is in state LT, otherwise it is predicted as not take.



A 2-State machine representation of branch-prediction



A 4-State machine representation of branch-prediction

Algorithm:

- An algorithm that uses 4 states, thus requiring two bits of history information for each branch instruction is shown in figure. The four states are:

ST : Strongly likely to be taken

LT : Likely to be taken.

LNT : Likely not to be taken

SNT : Strongly likely not to be taken.

STEP 1: Assume that the state of algorithm is initially set to LNT.

STEP 2: If the branch is actually taken change to ST, otherwise it is changed to SNT

STEP 3: When a branch instruction is encountered, the branch will be taken if the state is either LT or ST and it begins to fetch instructions at the branch target address. Otherwise, it continues to fetch instructions in sequential address order.

- When in state SNR, the instruction fetch unit predicts that the branch will not be taken.
- If the branch is actually taken, that is if the prediction is incorrect, the state changes to LNT.

The state information used in dynamic branch prediction algorithm requires 2 bits for 4 states and may be kept by the processes in a variety of ways,

- Use look-up table, which is accessed using low-order part of the branch of instruction address.
- Store as tag bits associated with branch instruction in the instruction cache.

PART-A

1. What is MIPS and write its instruction set?

MIPS is a reduced instruction set **computer** (RISC) instruction set **architecture** (ISA) developed by **MIPS** Technologies (formerly **MIPS Computer** Systems). The early **MIPS architectures** were 32-bit, with 64-bit versions added later.

MIPS instruction set:(Micro Instruction per Second)

- **The memory**-reference instructions load word (lw) and store word (sw)
- **The arithmetic**-logical instructions add, sub, AND, OR, and slt
- **The instructions-**branchequal (beq) and jump (j), which we add last.

2. What are R-type instructions? (Apr/May 2015)Nov/Dec 2020

MIPS fields are given names to make them easier to discuss:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Here is the meaning of each name of the fields in MIPS instructions:

- **op:**Basic operation of the instruction, traditionally called the **opcode**.
- **rs:**The first register source operand.
- **rt:**The second register source operand.
- **rd:**The register destination operand. It gets the result of the operation.
- **shamt:**Shift amount.
- **funct:** Function. This field, often called the *function code,* selects the specific variant of the operation in the op field.

3. Define Branch target address.

- The address specified in a branch, which becomes the new program counter (PC) if the branch is taken. In the MIPS architecture the branch target is given by the sum of the offset field of the instruction and the address of the instruction following the branch.

4. Define the terms Data path element, CPU Data path and Data path cycle? Nov / Dec 2016, Apr. / May 2018 Nov/Dec 2020.

- A unit used to operate on or hold data within a processor. In the MIPS implementation, the data path elements include the instruction and data memories, the register file, the ALU and adders.
- The path that data follows within the CPU, along buses from registers to ALU and back is called the **CPU Datapath**.
- Everything a computer does, whether playing an MPEG file, or a video game, is, in the end, essentially a sequence of very primitive operations whereby data is moved from registers to the ALU, operations are performed on that data, and then the result is returned to the registers. A single round of Registers -> ALU -> Registers is called a **CPU Datapath Cycle**.

5. When will the instruction have die effect?

- Sometime an instruction changes the contents of a register other than the destination. An instruction that uses an auto increment or auto decrement addressing mode is an example.
- Add with Carry R2, R4
- This instruction will take the carry value present in the condition code register. So it refers the register which is not represented in the instruction

6. Define branch penalty.

- The time lost as a result of a branch instruction is often referred to as the branch penalty. This will cause the pipeline to stall. So we can reduce branch penalty by calculating the branch address in early stage.

7. What is the use of instruction queue in pipeline?

- Many processors can fetch the instruction before they are needed and put them in queue is called instruction queue. This instruction queue can store several instructions.

8. Define dispatch unit.

- It is mainly used in pipeline concept. It takes the instruction from the front of the instruction queue and sends them to the execute unit for execution.

9. What is meant by branch folding and what is the condition to implement it?

- The instruction fetch unit has executed the branch instruction concurrently with in the execution of other instructions is called branch folding.
- This occurs only if at the time of branch is encountered at least one instruction is available in the queue than the branch instruction.

10. What is meant by delay branch slot?

- A location following branch instruction is called as branch delay slot. There may be more than one branch delay slot, depending on the execution time.
- The instruction in the delay slot is always fetched and at least partially executes before the branch decision is made.

11. Define delayed branching.

- It is a technique by using it we can handle the delay branch slot instructions. We can place some useful instruction in the branch delay slot and execute these instruction s when the processor is executing the branch instruction.
- If there is no useful instruction in the program we can simply place NOP instruction in delay slot. This technique will minimize the branch penalty.

12. Define branch prediction.Nov / Dec 2015

It is a technique used for reducing branch penalty associated with the condition branches. Assume that the branch will not take place and to continue the fetch instructions in sequential address order until the branch condition is evaluated.

13. What are the two types of branch prediction technique available? (May/June 2009)

The two types of branch prediction techniques are

- Static branch prediction
- Dynamic branch prediction

14. Define static and dynamic branch prediction.

- The branch prediction decision is always the same for every time a given instruction is executed.This is known as static branch prediction.
- Another approach in which the prediction may change depending on execution history is called dynamic branch prediction.

15. List the two states in the dynamic branch prediction.

- LT : Branch is likely to be taken.
- LNT : Branch is likely not to be taken.

16.List out the four stages in the branch prediction algorithm.

- ST :Strongly likely to be taken
- LT :Likely to be taken
- LNT :Likely not to be taken
- SNT :Strongly not to be taken

17. Define Register renaming. (Nov/Dec 2009)

- When temporary register holds the contents of the permanent register, the name of permanent register is given to that temporary register is called as **register renaming**.
- **For example**, if I2 uses R4 as a destination register, then the temporary register used in step TW2 is also referred as R4 during cycles 6 and 7 that temporary register used only for instructions that follow I2 in program order.
- For example, if I1 needs to read R4 in cycle 6 or 7, it has to access R4 though it contains unmodified data be I2.

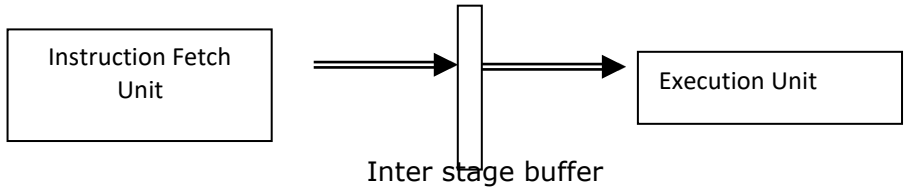
18. What is pipelining and what are the advantages of pipelining? (Apr/May 2010) Nov / Dec 2013

- Pipelining is process of handling the instruction concurrently.
- The pipelining processor executes a program by one after another.

Advantages: May / June 2016

- Pipelining improves the overall throughput of an instruction set processor.
- It is applied to design of complex data path units such as multiplexers and floating points adders.

19. Draw the hardware organization of two-stage pipeline.



20. Name the four stages of pipelining. (Or)What are the steps in pipelining processor?Nov/Dec 2020.

Fetch	:	Read the instruction from the memory.
Decode	:	Decode the instruction and fetch the source operands.
Execute	:	Perform the operation specified by the instruction
Write	:	Store the result in the destination location.

21. Write short notes on instruction pipelining.

- The various cycles involved in the instruction cycle.
- These fetch, decode and execute cycles for several instructions are performed simultaneously to reduce overall processing time.
- This process is referred as **instruction pipelining.**

22. What is the role of cache in pipelining? (Or) What is the need to use the cache memory in pipelining concept?(Nov/Dec 2011)

- Each stage in a pipeline is expected to complete its operation in one clock cycle. But the accessing time of the main memory is high.
- So it will take more than one clock cycle to complete its operation. So we are using cache memory for pipelining concept.
- The accessing speed of the cache memory is very high.

23. What is meant by bubbles in pipeline? Or what is meant by pipeline bubble? Nov / Dec 2016

- Any condition that causes the pipeline to be idle is known as pipeline stall. This is also known as bubble in the pipeline. Once the bubble is created as a result of a delay, a bubble moves down stream until it reaches the last unit.

24. What are the major characteristics of pipeline?

- Pipelining cannot be implemented on a single task, as it works of splitting multiple tasks into a number of subtasks and operating on them simultaneously.
- The speedup or efficiency is achieved by using a pipeline depends on the number of pipe stages and the number of available tasks that can be subdivided.

25. Give the features of the addressing mode suitable for pipelining. (Apr/May 2014)

- They access operand from memory in only one cycle.
- Only load and store instruction are provided to access memory.
- The addressing modes used do not have side effects.(When a location other than one explicitly named in an instruction as the destination operand is a affected, the instruction is said to have a side effect).
- Three basic addressing modes used do not have these features are register, register indirect and index. The first two require bus address computation. In the index mode, the address can be computed in one cycle, whether the index value is given in the instruction or in registration.

26. What are the disadvantages of increasing the number of stages in pipelined processing?(Apr/May 2011) (Or) What would be the effect,if we increase the number of pipelining stages? (Nov/Dec 2011)

Speedup:

Speedup is defined by

S(m)=T(1)/T (m)

- **Where T (m)** is the execution time for some target workload on an m-stage pipeline and **T(1)** is the execution time for same target workload on a non-pipelined Processor.

27. What is the ideal CPI of a pipelined processor?

The ideal CPI on a pipelined processor is almost always 1. Hence, we can compute the pipelined CPI:

CPI pipelined = Ideal CPI + Pipeline stall clock cycles per instruction = 1 + Pipeline stall clock cycles per instruction

28. How can memory access be made faster in a pipelined operation? Which hazards can be reduced by faster memory access? (Apr/May 2010)

The goal in controlling a pipelined CPU is maximize its performance with respect to the target workloads.

Performance measures:

The various performance measures of pipelining are,

- Throughput
- CPI
- Speedup
- Dependencies
- Hazards

The following Hazards can be reduced by faster memory access:

- Structural hazards
- Data or Data dependent hazards
- Instruction or control hazards

29. Write down the expression for speedup factor in a pipelined architecture. May 2013

- The speedup for pipeline computer is

  $S=(K+n-1)t_p$

  Where,

  k-number of segments in a pipeline.

  n-number of instruction to be executed.

  t_p- cycle time.

30. Define Hazard and State different types of hazards that occur in pipeline.Nov / Dec 2015, Apr / May 2017, May 2019 Nov/Dec 2020.

In the domain of central processing unit (CPU) design, **hazards** are problems with the instruction pipeline in CPU microarchitectures when the next instruction cannot execute in the following clock cycle, and can potentially lead to incorrect computation results.

The various pipeline hazards are:

- Structural hazards
- Data or Data dependent hazards
- Instruction or control hazards

31. What is structural hazard?(Nov/Dec 2008) (Apr /May 2014)

- When two instructions require the use of a given hardware resource at the same time this hazard will occur. The most common case of this hazard is memory access.

32. What is data hazard in pipelining? (Nov/Dec 2007, 2008)

- A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in pipeline. As a result some operation has be delayed and the pipeline stalls.
- Arise when an instruction depends on the results of a previous instruction in a way that is exposed by overlapping of instruction in pipeline

33. What are instruction hazards (or) control hazards?

- They arise while pipelining branch and other instructions that change the contents of program counter.
- The simplest way to handle these hazards is to stall the pipeline stalling of the pipeline allows few instructions to proceed on completion while stopping the execution of those which results in hazards.

34. How can we eliminate the delay in data hazard?

- In pipelining the data can be executed after the completion of the fetch operation. The data are available at the output of the ALU once the execute stage completes.
- Hence the delay can be reduced if we arrange for the result of fetch instruction to be forwarded directly for use in next step. This is known as operand forwarding.

35. How can we eliminate data hazard using software?

- The data dependencies can be handled with the software. The compiler can be used for this purpose. The compiler can introduce the two cycle delays needed between instruction I1 and I2 by inserting NOP (no operation)

  I_1: MUL R2, R3, R4
  NOP
  NOP
  I_2: ADD R5, R4, R6

36. List the techniques used for overcoming hazard.

- Data forwarding
- Adding sufficient hardware
- Stalling instructions
- Document to find instruction in wrong order.

37. What are the techniques used to present control hazard?

- Scheduling instruction in delay slots
- Loop unrolling
- Conditional execution
- Speculation (by both compiler and CPU).

38. List the types of data hazards.

i. RAW (Read After Write)
ii. WAW (Write After Write)
iii. WAR (Write After Read)
iv. RAR (Read After Read)

39. Define stall.

Idle periods are called stalls. They are also often referred to as bubbles in the pipeline.

40. Give 2 examples for instruction hazard.

- Cache miss
- Hazard in pipeline.

41. A =5 A<-3+A A<-4+A What hazard does the above two instructions create when executed concurrently? (Apr/May 2011)

If these operations are performed in the order given, the result is 32. But, if they were performed concurrently, the value is 5. So output is wrong.

42. What is meant by speculative execution? (Apr/May 2012) Or what is the need for speculation? (Nov/Dec 2014), May 2019

- A technique allows a superscalar processor to keep its functional units as busy as possible by executing instructions before it is known that they will be needed.
- The Intel P6 uses speculative execution.

43. What is meant by hazard in pipelining? Define data and control hazards. (May/June 2013) (Apr/May 2012)

- The idle periods in any of the pipeline stage due to the various dependency relationships among instructions are said to be stalls.
- **A data hazard** is any condition in which either the source or the destination operands of an instruction are not available at the time expected in pipeline.
- As a result some operation has be delayed and the pipeline stalls arise when an instruction depends on the results of a previous instruction in a way that is exposed by overlapping of instruction in pipeline.

  Types

  1. RAW **2.** WAW **3.** WAR
- **Control hazards** arise while pipelining branch and other instructions that change the contents of program counter. The simplest way to handle these hazards is to stall the pipeline stalling of the pipeline allows few instructions to proceed to completion while stopping the execution of those which results in hazards

44. Why is branch prediction algorithm needed? Distinguish between static and dynamic branch prediction. (May/June 2009) Or Differentiate between the static and dynamic techniques. (May/June 2013)

Branch Prediction has become essential to getting good performance from scalar instruction streams.

– Underlying algorithm has regularities.

- Data that is being operated on has regularities.
- Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems.

S.NO.	STATIC BRANCH PREDICTION	DYNAMIC BRANCH PREDICTION
1.	Branch can be predicted based on branch codes type statistically.	It used recent branch history during program execution,information is stored in buffer called branch target buffer(BTB).
2.	It may not produce accurate result every time.	Processing of conditional branches with zero delay.

45. What is Branch Target Buffer?

Branch Target Buffer (BTB): A hardware mechanism that aims at reducing the stall cycles resulting from correctly predicted taken branches to zero cycles.

46. Define program counter (PC).

- The register containing the address of the instruction in the program being executed.

47. What are Sign-extend?

- To increase the size of a data item by replicating the high-order sign bit of the original data item in the high order bits of the larger, destination data item.

48. Define Register file.

- A state element that consists of a set of registers that can be read and written by supplying a register number to be accessed.

49. What is a Don't-care term?

- An element of a logical function in which then output does not depend on the values of all the inputs.

50. Define Forwarding.

A method of resolving a data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer visible registers or memory.Also called **bypassing.**

51. What is a branch prediction buffer? (Apr/May 2015)

A small memory that is indexed by the lower portion of the address of the branch instruction and that contains one or more bits indicating whether the branch was recently taken or not. It is also called branch **history table**.

52. What is an Exception? Nov / Dec 2014, May / June 2016, Apr. / May 2018, Nov. / Dec. 2018

Exceptions and interrupts are unexpected events that disrupt the normal flow of instruction execution. An **exception** is an unexpected event from within the processor. An interrupt is an unexpected event from outside the processor. We have to implement **exception** and interrupt handling in our multi cycle CPU design.

53. Give one example for MIPS exception. Apr. / May 2018, Nov. / Dec. 2018

Exceptions in MIPS

stage	Problem exceptions occurring
IF	Page fault on IF, misaligned memory access, memory protection violation
ID	Undefined or illegal opcode
EX	Arithmetic exception
MEM	Page fault on data fetch, misaligned memory access, memory protection violation
WB	None

54. What is precise and imprecise exception? (Apr/May 2009)(Nov/Dec 2019)

- A **precise exception** is one in which all instruction prior to the faulting instruction are complete and instruction following the instruction, including the faulting instruction do not change the state of the machine. (Or)
- If the execution occurs during an instruction, all subsequent instructions that may have been executed are discarded. This is called **precise exception.**
- If one instruction causes an exception and succeeding instructions are permitted to complete execution, then the processor is said to have **imprecise exception.**

55. Define edge triggered clocking. May 2019(Nov/Dec 2019)

A falling **edge** is the high to low transition. It is also known as the negative **edge**. When a circuit is falling **edge**-**triggered**, it becomes active when the **clock** signal goes from high to low, and ignores the low-to-high transition. A leading **edge** is an event that is **triggered** on the front **edge** of a pulse.

56.What is Instruction Level Parallelism? (Dec 2012, Dec 2013, May 2015, May 2016)

- The technique which is used to overlap the execution of instructions and improve performance is called ILP.

57. What are the approaches to exploit ILP? (Dec 2012, Dec 2015)

The two separable approaches to exploit ILP are,

- Dynamic or Hardware Intensive Approach
- Static or Compiler Intensive Approach.

58. What is Loop Level Parallelism?

- Loop level parallelism is a way to increase the amount of parallelism available among instructions is to exploit parallelism among iterations of loop.

59. Give the methods to enhance performance of ILP.

To obtain substantial performance enhancements, the ILP across multiple basic blocks are exploited using

- Loop Level Parallelism
- Vector Instructions

60. Define Dynamic Scheduling. (May 2013) (Or) Explain the idea behind dynamic scheduling. (Nov/Dec 2016)

- Dynamic scheduling is a technique in which the hardware rearranges the instruction execution to reduce the stalls while maintaining data flow and exception behavior.

61. List the drawbacks of Dynamic Scheduling.

- The complexity of the tomasulo scheme.
- Each reservation station must contain an associative buffer.
- The performance can be limited by the single CDB.

62. List the advantages of Dynamic Scheduling. (May 2012)

- It handles dependences that are unknown at compile time.
- It simplifies the compiler.
- It allows code compiled for one pipeline to run efficiently on a different pipeline
- Uses speculation techniques to improve the performance.

63. Differentiate Static and Dynamic Scheduling.

Static Scheduling	Dynamic Scheduling
• The data hazard that prevents a new instruction issue in the next cycle was resolved using a technique called data forwarding • And also by compiler scheduling that separated the dependent instruction this is called as static scheduling.	• The CPU rearranges the instructions to reduce stalls while preserving dependences. • It uses hardware based mechanism to rearrange instruction execution order to reduce stalls at runtime. • It enables handling cases where dependences are unknown at compile time.

64. Define Dynamic Scheduling using Tomasulo's algorithm.

- **Tomasulo's algorithm** is a computer architecture hardware **algorithm** for **dynamic scheduling** of instructions that allows out-of-order execution and enables more efficient use of multiple execution units.

65.What is Branch Prediction?

- In computer architecture, a **branch predictor** is a digital circuit that tries to guess which way a **branch** (e.g. an if–then–else structure) will go before this is known definitively.
- The purpose of the branch predictor is to improve the flow in the instruction pipeline.

66. What are the types of branch prediction?

There are two types of branch prediction. They are,

- Dynamic Branch Prediction & Static Branch Prediction

67. What is meant by dynamic branch prediction? [May 2019]

- ***Branch prediction*** is used to overcome the fetch limitation imposed by control hazards in order to expose instruction-level parallelism (ILP).

- It is the key ingredient to pipelined and superscalar architectures that mask instruction execution latencies by exploiting (ILP).

68. What is Branch Prediction Buffer? (May 2014)

- Branch prediction buffer is a small memory indexed by the lower portion of the address of the branch instruction.
- The memory contains a bit that says whether the branch was recently taken or not.

69. What are the things present in Dynamic Branch Prediction?

It uses two things they are,

- Branch Prediction Buffer &Branch History Table

70. Define Correlating Branch Prediction.

- Branch prediction that uses the behavior of other branches to make a prediction is called correlating branch prediction.

71. List the five levels of branch prediction. (May 2013)

- Perfect
- Tournament Based Branch Predictor
- Standard Two Bit Branch Predictor with 512 - 2 Bit Entries
- Profile Based
- None

72. What is Reservation Station?

- In Tomasulo's scheme, register renaming is provided by reservation station.
- The basic idea is that the reservation station fetches and buffers an operand as soon as it is available, eliminating the need to get the operand from a register.

73. What is ROB?

- ROB stands for Reorder Buffer.
- It supplies operands in the interval between completion of instruction execution and instruction commit. ROB is similar to the store buffer in Tomasulo's algorithm.

74. What are the four fields involved in ROB?

ROB contains four fields,

- Instruction Type
- Destination Field
- Value Field
- Ready Field

75. What is Imprecise Exception?

- An exception is imprecise if the processor state when an exception is raised does not look exactly as if the instructions were executed sequentially in strict program order.

76. What are the two possibilities of imprecise exceptions?

- If the pipeline has already completed instructions that are later in program order then that instruction will cause exception.

- If the pipeline has not yet completed instructions that are earlier in program order then that instructions will cause exception.

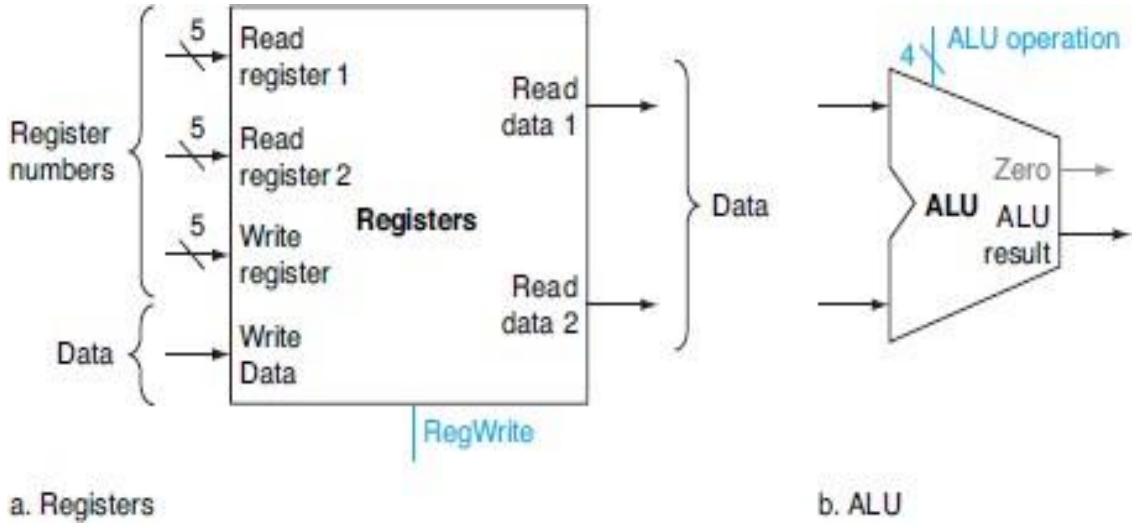
77. What is Register Renaming?

- Renaming of register operand is called register renaming.
- It can be either done statically by the compiler or dynamically by the hardware.

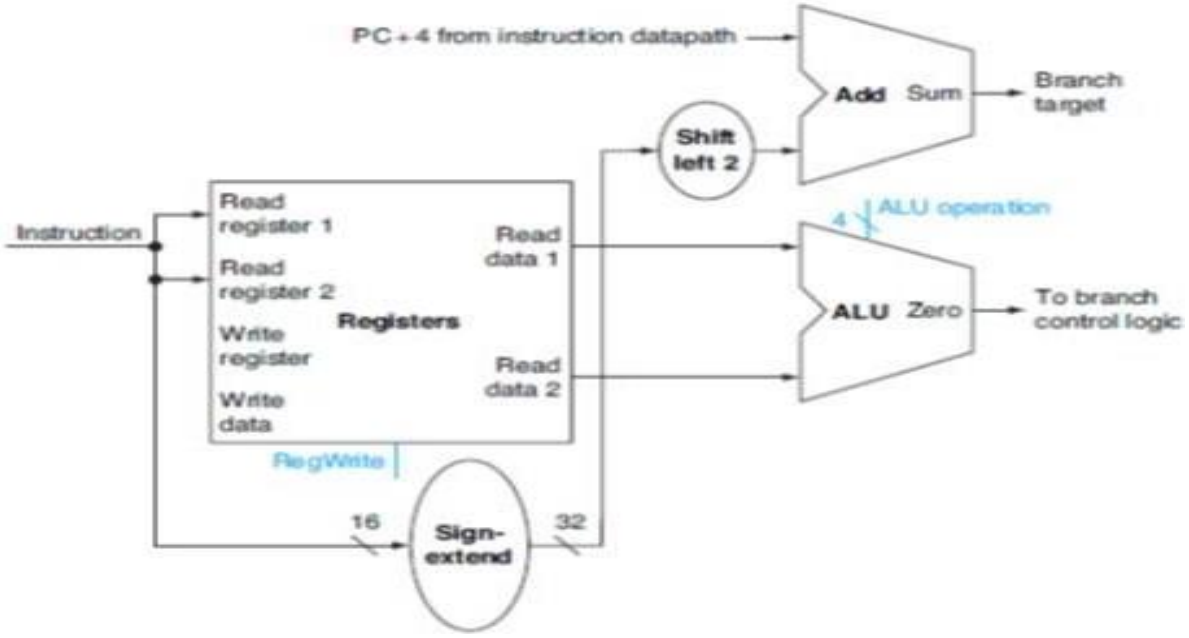
78. Difference between Static and Dynamic Branch Prediction? (May 2011)

Static Branch Prediction	Dynamic Branch Prediction
• Static branch prediction is usually carried out by the complier. • It is static because the prediction is already known even before the program is executed.	• It uses the run time behavior of branch to make more accurate prediction. • Information about the outcome of previous occurrences of a given branch is used to predict the current occurrences.

79. In a datapath diagram, what is the size of ALUop Control signal. Nov/Dec 2021



80. How PCSrc Signal generated in a datapath diagram? Nov/Dec 2021



UNIT V MEMORY AND I/O

Memory Concepts and Hierarchy – Memory Management – Cache Memories: Mapping and Replacement Techniques – Virtual Memory – DMA – I/O – Accessing I/O: Parallel and Serial Interface – Interrupt I/O – Interconnection Standards: USB, SATA

MEMORY CONCEPTS AND HIERARCHY

1. Explain about the memory concepts and hierarchy.

What is Memory?

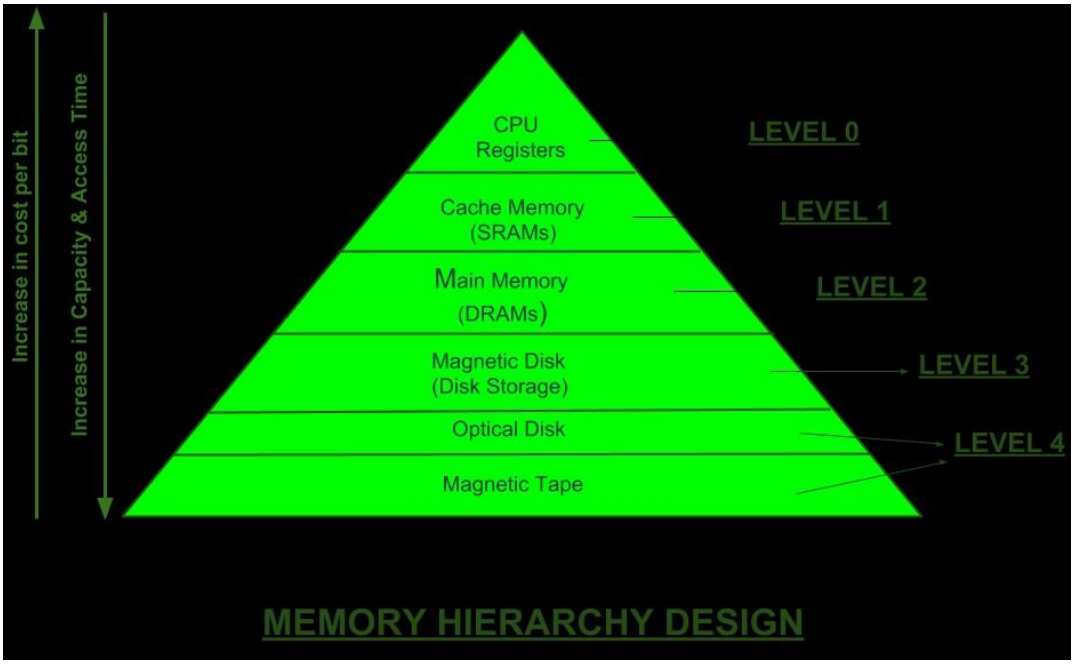
Computer memory is just like the human brain. It is used to store data/information and instructions. It is a data storage unit or a data storage device where data is to be processed and instructions required for processing are stored. It can store both the input and output can be stored here.

Characteristics of Main Memory:

- It is faster computer memory as compare to secondary memory.
- It is semiconductor memories.
- It is usually a volatile memory.
- It is the main memory of the computer.
- A computer system cannot run without primary memory.

Memory Hierarchy

In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references. The figure below clearly demonstrates the different levels of memory hierarchy :



This Memory Hierarchy Design is divided into 2 main types:

1. **External Memory or Secondary Memory –** Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.
2. **Internal Memory or Primary Memory –** Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

We can infer the following characteristics of Memory Hierarchy Design from above figure:

1. **Capacity:**
   It is the global volume of information the mehmory can store. As we move from top to bottom in the Hierarchy, the capacity increases.
2. **Access Time:** It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.
3. **Performance:**
   Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.
4. **Cost per bit:** As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

In general, memory is of three types:

- Primary memory
- Secondary memory
- Cache memory

Now we discuss each type of memory one by one in detail:

1. Primary Memory: It is also known as the main memory of the computer system. It is used to store data and programs or instructions during computer operations. It uses semiconductor technology and hence is commonly called semiconductor memory. Primary memory is of two types:

(i) RAM (Random Access Memory): It is a volatile memory. Volatile memory stores information based on the power supply. If the power supply fails/ interrupted/stopped, all the data & information on this memory will be lost. RAM is used for booting up or start the computer. It temporarily stores programs/ data which has to be executed by the processor. RAM is of two types:

- **S RAM (Static RAM):** It uses transistors and the circuits of this memory are capable of retaining their state as long as the power is applied. This memory consists of the number of flip flops with each flip flop storing 1 bit. It has less access time and hence, it is faster.

- **D RAM (Dynamic RAM):** It uses capacitors and transistors and stores the data as a charge on the capacitors. They contain thousands of memory cells. It needs refreshing of charge on capacitor after a few milliseconds. This memory is slower than S RAM.

(ii) ROM (Read Only Memory): It is a non-volatile memory. Non-volatile memory stores information even when there is a power supply failed/ interrupted/stopped. ROM is used to store information that is used to operate the system. As its name refers to read-only memory, we can only read the programs and data that is stored on it. It contains some electronic fuses that can be programmed for a piece of specific information. The information stored in the ROM in binary format. It is also known as permanent memory. ROM is of four types:

- **MROM (Masked ROM):** Hard-wired devices with a pre-programmed collection of data or instructions were the first ROMs. Masked ROMs are a type of low-cost ROM that works in this way.
- **PROM (Programmable Read Only Memory):** This read-only memory is modifiable once by the user. The user purchases a blank PROM and uses a PROM program to put the required contents into the PROM. Its content can‘t be erased once written.
- **EPROM (Erasable Programmable Read Only Memory):** It is an extension to PROM where you can erase the content of ROM by exposing it to Ultraviolet rays for nearly 40 minutes.
- **EEPROM (Electrically Erasable Programmable Read Only Memory):** Here the written contents can be erased electrically. You can delete and re-programme EEPROM up to 10,000 times. Erasing and programming take very little time, i.e., nearly 4 -10 ms (milliseconds). Any area in an EEPROM can be wiped and programmed selectively.

2. Secondary Memory: It is also known as auxiliary memory and backup memory. It is a non-volatile memory and used to store a large amount of data or information. The data or information stored in secondary memory is permanent, and it is slower than primary memory. A CPU cannot access secondary memory directly. The data/information from the auxiliary memory is first transferred to the main memory, and then the CPU can access it.

Characteristics of Secondary Memory:

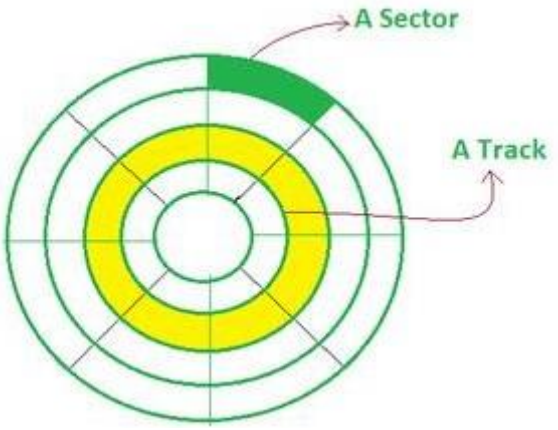
- It is a slow memory but reusable.
- It is a reliable and non-volatile memory.
- It is cheaper than primary memory.
- The storage capacity of secondary memory is large.
- A computer system can run without secondary memory.
- In secondary memory, data is stored permanently even when the power is off.

Types of secondary memory:

(i) Magnetic Tapes: Magnetic tape is a long, narrow strip of plastic film with a thin, magnetic coating on it that is used for magnetic recording. Bits are recorded on tape as magnetic patches called RECORDS that run along many tracks. Typically, 7 or 9 bits are recorded concurrently. Each track has one read/write

head, which allows data to be recorded and read as a sequence of characters. It can be stopped, started moving forward or backward, or rewound.

(ii) Magnetic Disks: A magnetic disc is a circular metal or a plastic plate and these plates are coated with magnetic material. The disc is used on both sides. Bits are stored in magnetized surfaces in locations called tracks that run in concentric rings. Sectors are typically used to break tracks into pieces.



Hard discs are discs that are permanently attached and cannot be removed by a single user.

(iii) Optical Disks: It‘s a laser-based storage medium that can be written to and read. It is reasonably priced and has a long lifespan. The optical disc can be taken out of the computer by occasional users. Types of Optical Disks :

(a) CD – ROM:

- It‘s called Compact Disk. Only read from memory.
- Information is written to the disc by using a controlled laser beam to burn pits on the disc surface.
- It has a highly reflecting surface, which is usually aluminum.
- The diameter of the disc is 5.25 inches.
- 16000 tracks per inch is the track density.
- The capacity of a CD-ROM is 600 MB, with each sector storing 2048 bytes of data.
- The data transfer rate is about 4800KB/sec. & the new access time is around 80 milliseconds.

(b) WORM-(WRITE ONCE READ MANY):

- A user can only write data once.
- The information is written on the disc using a laser beam.
- It is possible to read the written data as many times as desired.
- They keep lasting records of information but access time is high.
- It is possible to rewrite updated or new data to another part of the disc.
- Data that has already been written cannot be changed.
- Usual size – 5.25 inch or 3.5 inch diameter.
- The usual capacity of 5.25 inch disk is 650 MB,5.2GB etc.

(c) DVDs:

- The term –DVD‖ stands for –Digital Versatile/Video Disc,‖ and there are two sorts of DVDs: (i)DVDR (writable) and (ii) DVDRW (Re-Writable)
- *DVD-ROMS (Digital Versatile Discs):* These are read-only memory (ROM) discs that can be used in a variety of ways. When compared to CD-ROMs, they can store a lot more data. It has a thick polycarbonate plastic layer that serves as a foundation for the other layers. It‘s an optical memory that can read and write data.
- *DVD-R:* It is a writable optical disc that can be used just once. It‘s a DVD that can be recorded. It‘s a lot like WORM. DVD-ROMs have capacities ranging from 4.7 to 17 GB. The capacity of 3.5 inch disk is 1.3 GB.

3. Cache Memory: It is a type of high-speed semiconductor memory that can help the CPU run faster. Between the CPU and the main memory, it serves as a buffer. It is used to store the data and programs that the CPU uses the most frequently.

Advantages of cache memory:

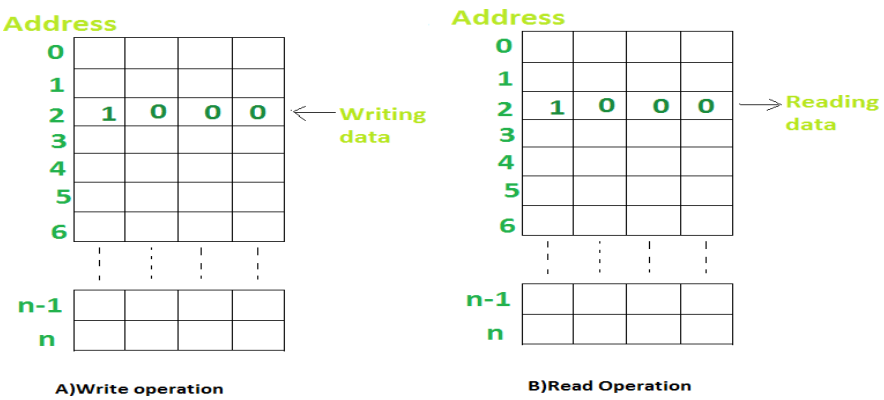
- It is faster than the main memory.
- When compared to the main memory, it takes less time to access it.
- It keeps the programs that can be run in a short amount of time.
- It stores data in temporary use.

Disadvantages of cache memory:

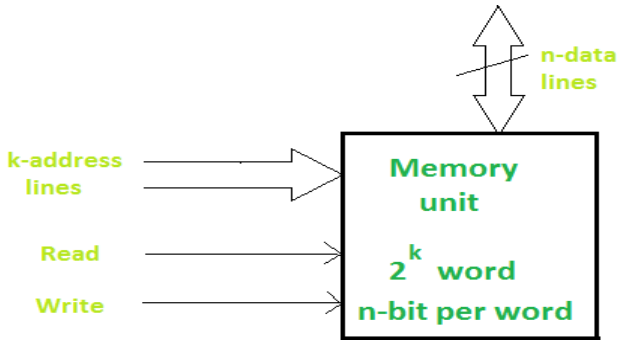
- Because of the semiconductors used, it is very expensive.
- The size of the cache (amount of data it can store) is usually small.

Memory unit:

- Memories are made up of registers.
- Each register in the memory is one storage location.
- The storage location is also called a memory location. Memory locations are identified using **Address**.
- The total number of bits a memory can store is its **capacity**.
- A storage element is called a **Cell**.
- Each register is made up of a storage element in which one bit of data is stored.
- The data in a memory are stored and retrieved by the process called **writing** and **reading** respectively.



- A **word** is a group of bits where a memory unit stores binary information.
- A word with a group of 8 bits is called a **byte**.
- A memory unit consists of data lines, address selection lines, and control lines that specify the direction of transfer. The block diagram of a memory unit is shown below:



- Data lines provide the information to be stored in memory.
- The control inputs specify the direct transfer.
- The k-address lines specify the word chosen.

When there are k address lines, 2^k memory words can be accessed.

Following are some important memory units:

- **Bit (Binary Units):** bit is a logical representation of the electric state. It can be 1 or 0.
- **Nibble:** it means the group of 4 bits.
- **Byte:** a byte is a group of 8 bits.
- **Word:** it is a fixed number of bits; it is different from computer to computer, but the same for each device. Compute store information in the form of words.

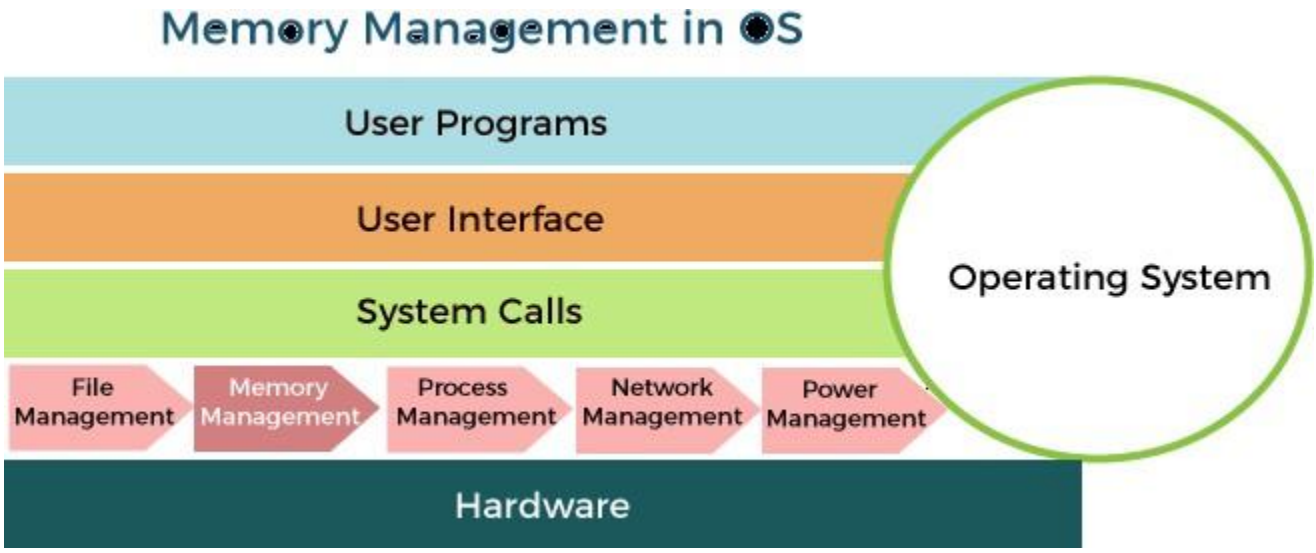
Following are conversations of units:

- **Kilobyte (kb):** 1kb = 1024 byte
- **Megabyte (mb):** 1mb = 1024 kb
- **Gigabyte (gb):** 1gb = 1024 mb
- **Terabyte (tb):** 1tb = 1024 gb
- **Petabyte (pb):** 1pb = 1024 tb

MEMORY MANAGEMENT

2. What do you mean by memory management?

Memory is the important part of the computer that is used to store the data. Its management is critical to the computer system because the amount of main memory available in a computer system is very limited. At any time, many processes are competing for it. Moreover, to increase performance, several processes are executed simultaneously. For this, we must keep several processes in the main memory, so it is even more important to manage them effectively.



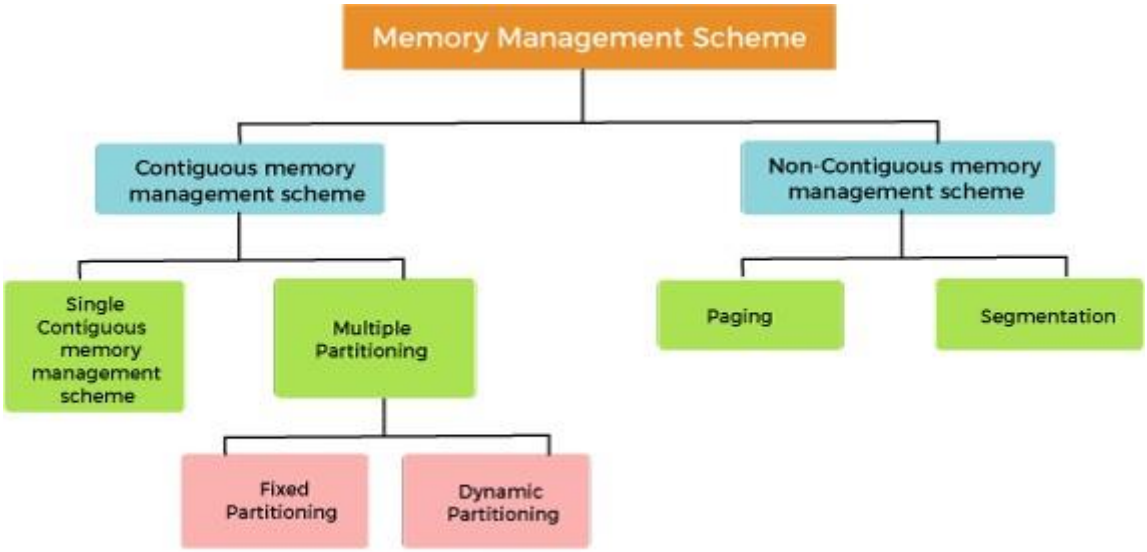
Following are the important roles in a computer system:

- Memory manager is used to keep track of the status of memory locations, whether it is free or allocated. It addresses primary memory by providing abstractions so that software perceives a large memory is allocated to it.
- Memory manager permits computers with a small amount of main memory to execute programs larger than the size or amount of available memory. It does this by moving information back and forth between primary memory and secondary memory by using the concept of swapping.
- The memory manager is responsible for protecting the memory allocated to each process from being corrupted by another process. If this is not ensured, then the system may exhibit unpredictable behavior.
- Memory managers should enable sharing of memory space between processes. Thus, two programs can reside at the same memory location although at different times.

Memory management Techniques:

The Memory management Techniques can be classified into following main categories:

- Contiguous memory management schemes
- Non-Contiguous memory management schemes



Classification of memory management schemes

Contiguous memory management schemes:

In a Contiguous memory management scheme, each program occupies a single contiguous block of storage locations, i.e., a set of memory locations with consecutive addresses.

Single contiguous memory management schemes:

The Single contiguous memory management scheme is the simplest memory management scheme used in the earliest generation of computer systems. In this scheme, the main memory is divided into two contiguous areas or partitions. The operating systems reside permanently in one partition, generally at the lower memory, and the user process is loaded into the other partition.

Advantages of Single contiguous memory management schemes:

- Simple to implement.
- Easy to manage and design.
- In a Single contiguous memory management scheme, once a process is loaded, it is given full processor's time, and no other processor will interrupt it.

Disadvantages of Single contiguous memory management schemes:

- Wastage of memory space due to unused memory as the process is unlikely to use all the available memory space.
- The CPU remains idle, waiting for the disk to load the binary image into the main memory.
- It can not be executed if the program is too large to fit the entire available main memory space.
- It does not support multiprogramming, i.e., it cannot handle multiple programs simultaneously.

Multiple Partitioning:

The single Contiguous memory management scheme is inefficient as it limits computers to execute only one program at a time resulting in wastage in memory space and CPU time. The problem of inefficient CPU use can be overcome using multiprogramming that allows more than one program to run concurrently. To switch between two processes, the operating systems need to load both processes into the main memory. The

operating system needs to divide the available main memory into multiple parts to load multiple processes into the main memory. Thus multiple processes can reside in the main memory simultaneously.

- Fixed Partitioning
- Dynamic Partitioning

Fixed Partitioning

The main memory is divided into several fixed-sized partitions in a fixed partition memory management scheme or static partitioning. These partitions can be of the same size or different sizes. Each partition can hold a single process. The number of partitions determines the degree of multiprogramming, i.e., the maximum number of processes in memory. These partitions are made at the time of system generation and remain fixed after that.

Advantages of Fixed Partitioning memory management schemes:

- Simple to implement.
- Easy to manage and design.

Disadvantages of Fixed Partitioning memory management schemes:

- This scheme suffers from internal fragmentation.
- The number of partitions is specified at the time of system generation.

Dynamic Partitioning

The dynamic partitioning was designed to overcome the problems of a fixed partitioning scheme. In a dynamic partitioning scheme, each process occupies only as much memory as they require when loaded for processing. Requested processes are allocated memory until the entire physical memory is exhausted or the remaining space is insufficient to hold the requesting process. In this scheme the partitions used are of variable size, and the number of partitions is not defined at the system generation time.

Advantages of Dynamic Partitioning memory management schemes:

- Simple to implement.
- Easy to manage and design.

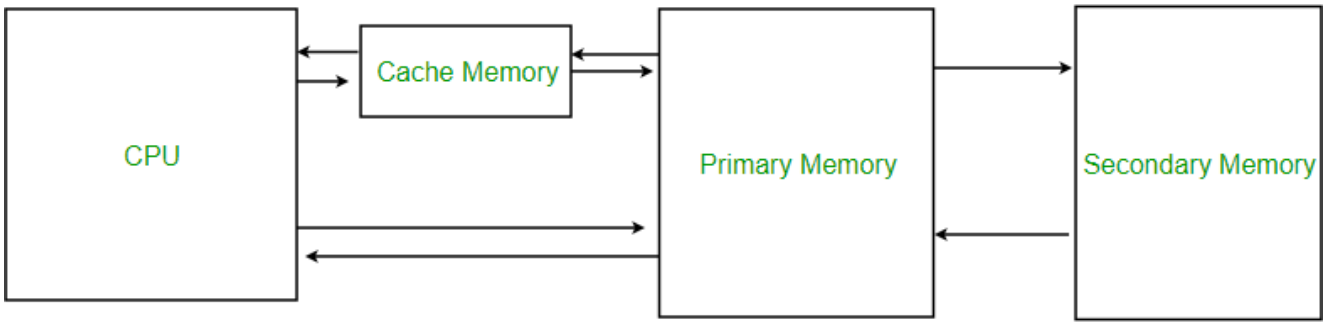
Disadvantages of Dynamic Partitioning memory management schemes:

- This scheme also suffers from internal fragmentation.
- The number of partitions is specified at the time of system segmentation.

Non-Contiguous memory management schemes:

In a Non-Contiguous memory management scheme, the program is divided into different blocks and loaded at different portions of the memory that need not necessarily be adjacent to one another. This scheme can be classified depending upon the size of blocks and whether the blocks reside in the main memory or not.

Cache memory



Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.

Levels of memory:

- **Level 1 or Register –**
  It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.
- **Level 2 or Cache memory –**
  It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory –**
  It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory –**
  It is external memory which is not as fast as main memory but data stays permanently in this memory.

Cache Performance:

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Hit ratio = hit / (hit + miss) = no. of hits/total accesses

We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

Cache Measures

- **Cache:** Cache is small, fast storage used to improve average access time to slow memory. It applied whenever buffering is employed to reuse commonly occurring items, i.e. file caches, name caches, and so on.
- **Cache Hit:** CPU finds a requested data item in the cache.
- **Cache Miss:** The item in not in the cache at access.
- Block is a fixed size collection of data, retrieved from memory and placed into the cache.
- **Advantage of Temporal Locality:** If access data from slower memory, move it to faster memory. If data in faster memory is unused recently, move it to slower memory.
- **Advantage of Spatial Locality:** If need to move a word from slower to faster memory, move adjacent words at same time.
- **Hit Rate (Hit Ratio):** Fraction of accesses that are hits at a given level of the hierarchy.
- **Hit Time:** Time required accessing a level of the hierarchy, including time to determine whether access is a hit or miss.
- **Miss Rate (Miss Ratio):** Fraction of accesses that are misses at a given level.
- **Miss Penalty:** Extra time required to fetch a block into some level from the next level down.
- The address space is usually broken into fixed size blocks, called pages. At each time, each page resides either in main memory or on disk.
- Average memory access time is a useful measure to evaluate the performance of a memory-hierarchy configuration.

  Average Memory Access Time = Memory Hit Time + Memory Miss Rate x Miss Penalty

Cache Mapping

3. Discuss in detail about various cache mapping techniques.

- When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.
  - ✓ If the processor finds that the memory location is in the cache, a cache hit has occurred and data is read from cache
  - ✓ If the processor does not find the memory location in the cache, a cache miss has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, and then the request is fulfilled from the contents of the cache.
- The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Hit Ratio = Hit / (Hit + Miss) = No. of Hits / Total Accesses

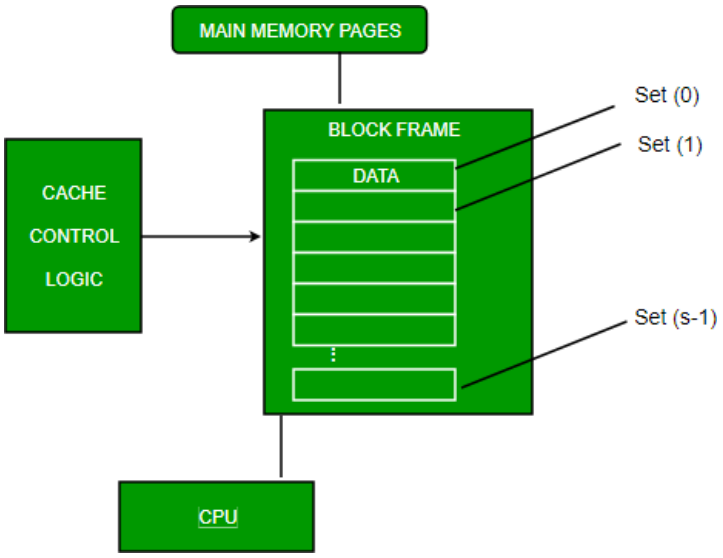
- We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty and reduce the time to hit in the cache.

Cache Mapping

- Cache memory mapping is the way in which we map or organize data in cache memory, this is done for efficiently storing the data which then helps in easy retrieval of the same.
- The three different types of mapping used for the purpose of cache memory are as follow,
  - ✓ Direct Mapping
  - ✓ Associative Mapping
  - ✓ Set-Associative Mapping

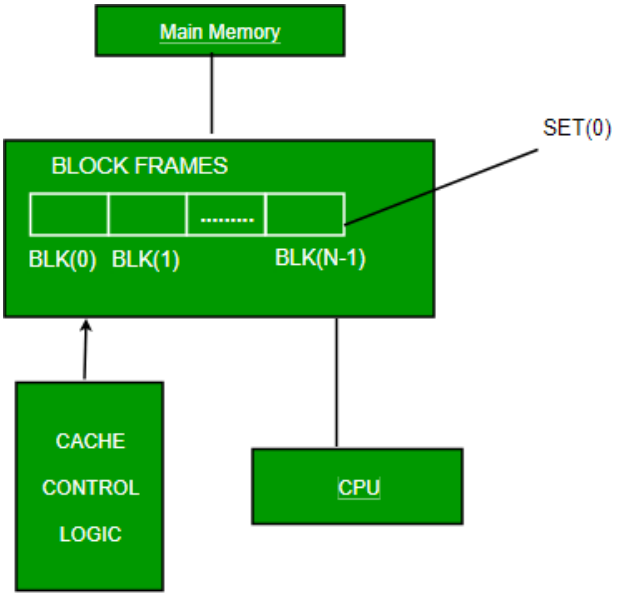
Direct Mapping:

- In direct mapping, assigned each memory block to a specific line in the cache.
- If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed.
- An address space is split into two parts index field and tag field.
- The cache is used to store the tag field whereas the rest is stored in the main memory.
- Direct mapping`s performance is directly proportional to the Hit ratio.



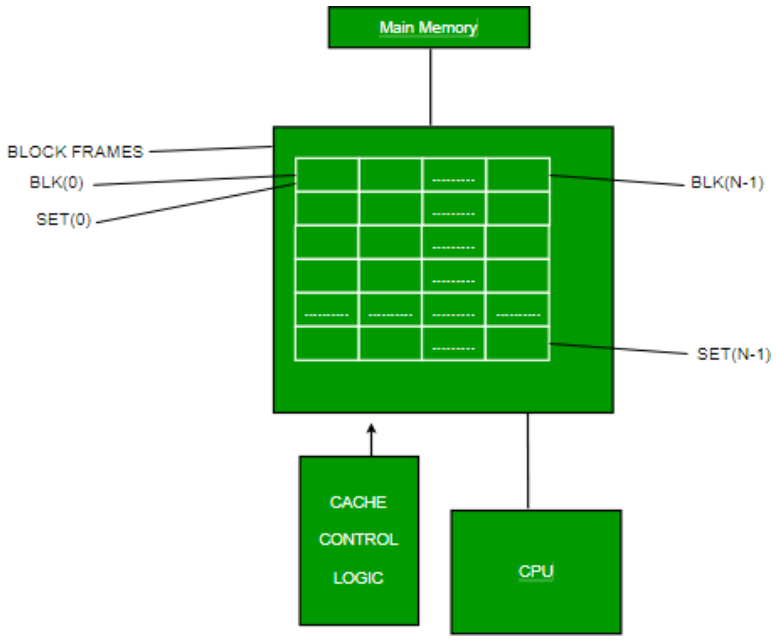
Associative Mapping:

- In this type of mapping, the associative memory is used to store content and addresses both of the memory word. Any block can go into any line of the cache.
- This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits.
- This enables the placement of the any word at any place in the cache memory.
- It is considered to be the fastest and the most flexible mapping form.



Set-Associative Mapping:

- This form of mapping is an enhanced form of the direct mapping where the drawbacks of direct mapping are removed.
- Set associative addresses the problem of possible thrashing in the direct mapping method.
- It does this by saying that instead of having exactly one line that a block can map to in the cache; we will group a few lines together creating a ***set***.
- Then a block in memory can map to any one of the lines of a specific set.
- Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address.
- Set associative cache mapping combines the best of direct and associative cache mapping techniques.



Uses of Cache

- Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.

- The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

Types of Cache

- **Primary Cache –** A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
- **Secondary Cache –** secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.

Locality of Reference

- Since size of cache memory is less as compared to main memory.
- So to check which part of main memory should be given priority and loaded in cache is decided based on locality of reference.

Types of Locality of Reference

- **Spatial Locality of reference** – this says that there is chance that element will be present in the close proximity to the reference point and next time if again searched then more close proximity to the point of reference.
- **Temporal Locality of reference** – In this Least recently used algorithm will be used. Whenever there is page fault occurs within word will not only load word in main memory but complete page fault will be loaded because spatial locality of reference rule says that if you are referring any word next word will be referred in its register that‘s why we load complete page table so complete block will be loaded.

Cache replacement Techniques:

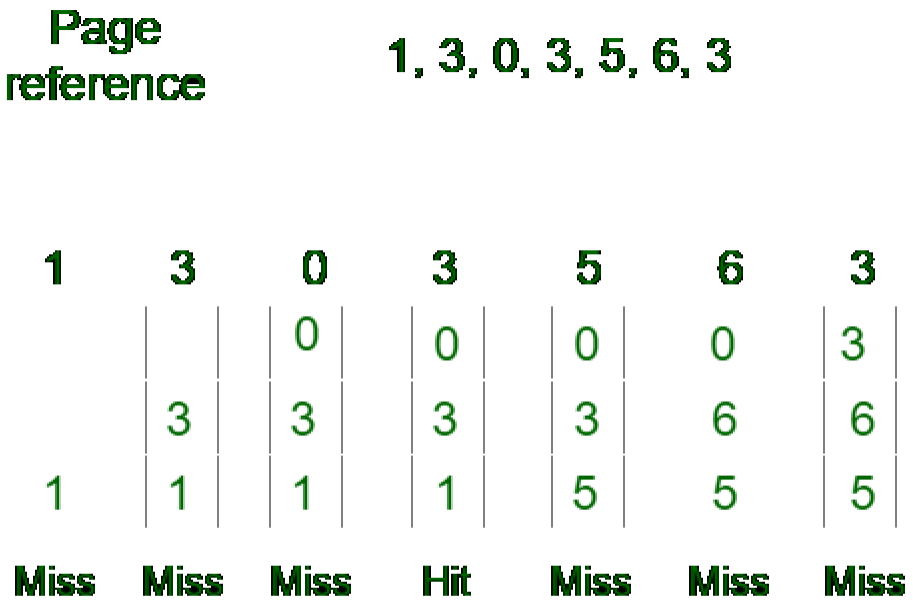
In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

Page Fault: A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

Page Replacement Algorithms:

1. First In First Out (FIFO): This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example 1: Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames. Find the number of page faults.

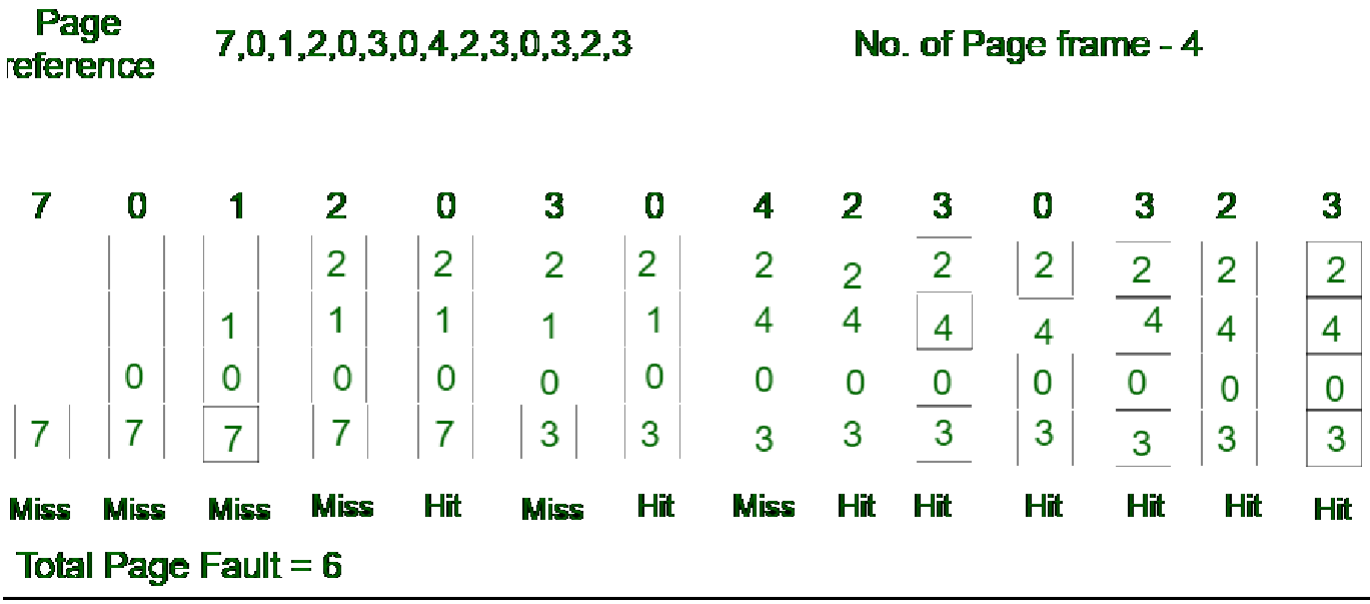


Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.** when 3 comes, it is already in memory so —> **0 Page Faults.** Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.** 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.** Finally, when 3 come it is not available so it replaces 0 **1 page fault.**

Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference strings 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4, and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10-page faults.

2. Optimal Page replacement: In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frame. Find number of page fault.



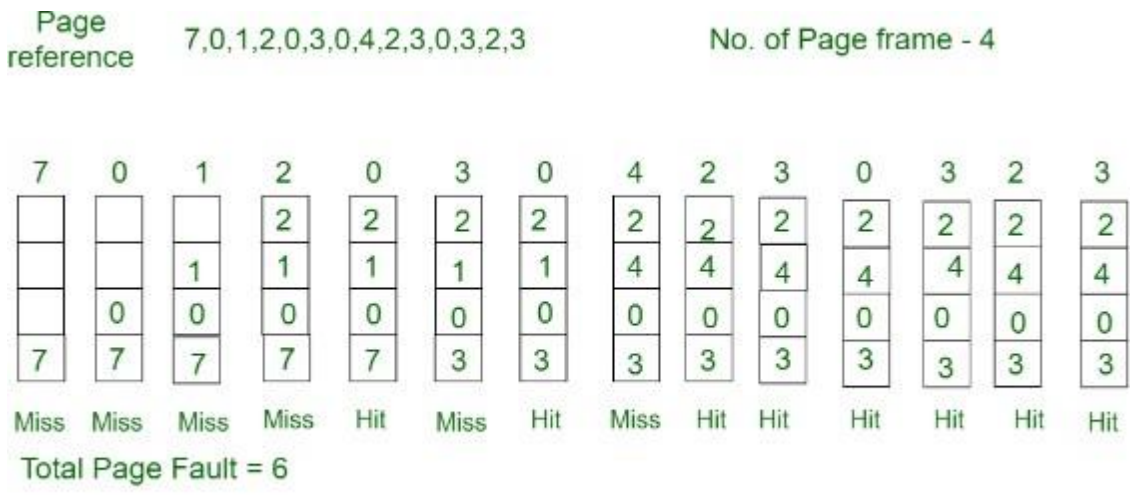
Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults** 0 is already there so —> **0 Page fault.** when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.** 0 is already there so —> **0 Page fault.** 4 will takes place of 1 —> **1 Page Fault.**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

3. Least Recently Used: In this algorithm, page will be replaced which is least recently used.

Example-3: Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames. Find number of page faults.



Here LRU has same number of page fault as optimal but it may differ according to question.

Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults** 0 is already their so —> **0 Page fault.** when 3 came it will take the place of 7 because it is least recently

used —> **1 Page fault** 0 is already in memory so —> **0 Page fault**. 4 will takes place of 1 —> **1 Page Fault** Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

4. Most Recently Used (MRU): In this algorithm, page will be replaced which has been used recently. Belady's anomaly can occur in this algorithm.

4. Explain in detail about virtual memory with an example.(Nov/Dec 2019) (Nov/Dec 2021) or ***Discuss the concept of virtual memory and explain how a virtual memory system is implemented, pointing out the hardware and software support. (Nov/Dec 2017)Nov/Dec 2020.***

VIRTUAL MEMORY

- Virtual memory divides physical memory into blocks (called page or segment) and allocates them to different processes.
- With virtual memory, the CPU produces virtual addresses that are translated by a combination of HW and SW to physical addresses, which accesses main memory.
- The process is called memory mapping or address translation.
- Today, the two memory-hierarchy levels controlled by virtual memory are DRAMs and magnetic disks.
- Virtual Memory manages the two levels of the memory hierarchy represented by main memory and secondary storage.
- Figure below shows the mapping of virtual memory to physical memory for a program with four pages.

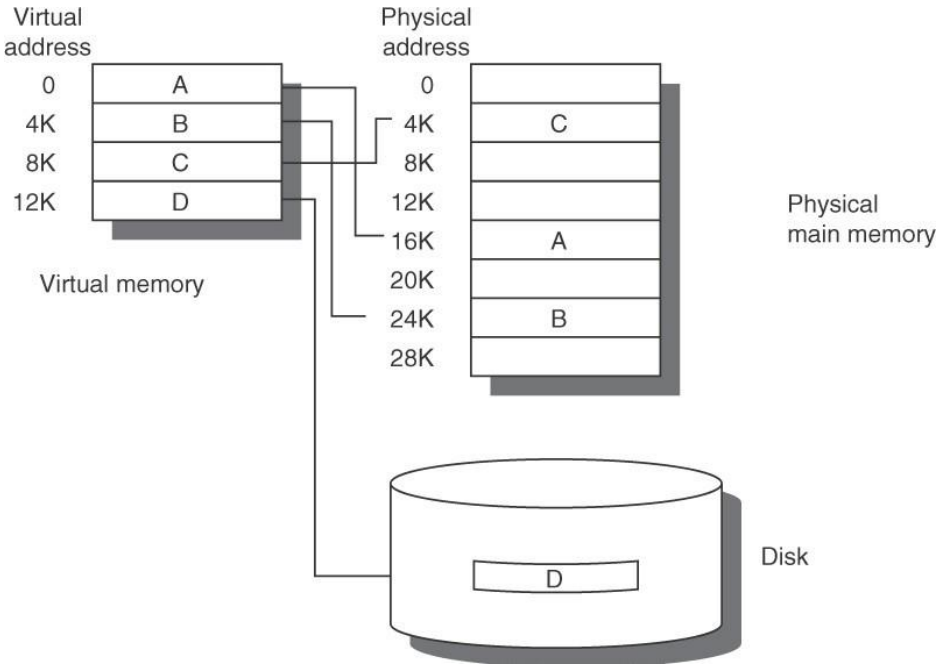
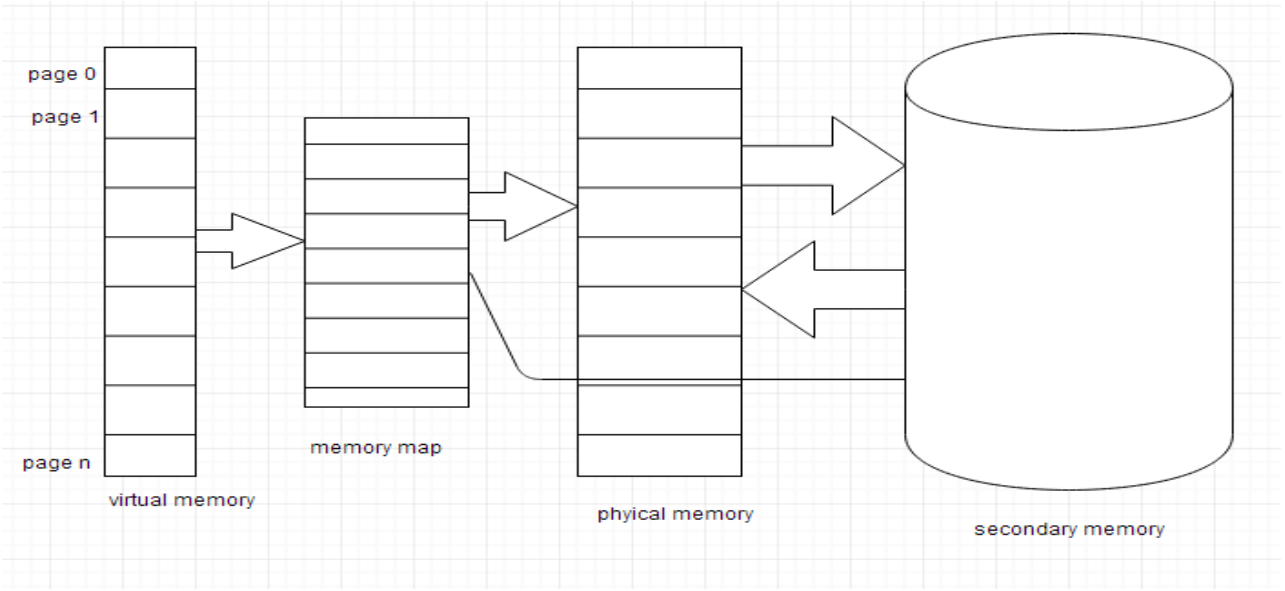


Figure: Virtual Memory Space

- Virtual memory is the separation of logical memory from physical memory.

- This separation provides large virtual memory for programmers when only small physical memory is available.
- Virtual memory is a memory management capability of an OS that uses hardware and software to allow a computer to compensate for physical memory shortages by temporarily transferring data from random access memory (RAM) to disk storage.
- Virtual address space is increased using active memory in RAM and inactive memory in hard disk drives (HDDs) to form contiguous addresses that hold both the application and its data.
- Computers have a finite amount of RAM so memory can run out, especially when multiple programs run at the same time.
- A system using virtual memory can load larger programs or multiple programs running at the same time, allowing each one to operate as if it has infinite memory and without having to purchase more RAM.
- As part of the process of copying virtual memory into physical memory, the OS divides memory into page files or swap files that contain a fixed number of addresses.
- Each page is stored on a disk and when the page is needed, the OS copies it from the disk to main memory and translates the virtual addresses into real addresses.



Pros and Cons of using Virtual Memory

- Among the primary benefits of virtual memory is its ability to handle twice as many addresses as main memory.
- It uses software to consume more memory by using the HDD as temporary storage while memory management units translate virtual memory addresses to physical addresses via the central processing unit.
- Programs use virtual addresses to store instructions and data; when a program is executed, the virtual addresses are converted into actual memory addresses.

5. Discuss the concept of Programmed I/O. Discuss about Programmed I/Os associated with computers. (Apr/May 2018)

Programmed I/O

- If I/O operations are completely controlled by the CPU, the computer is said to be using **programmed I/O**. In this case, the CPU executes programs that initiate, direct and terminate the I/O operations.
- If a part of the main memory address space is assigned to I/O ports, then such systems are called as **Memory-Mapped I/O systems**.
- In **I/O-mapped** I/O systems, the memory and I/O address space are separate. Similarly the control lines used for activating memory and I/O devices are also different. Two sets of control lines are available. READ M and WRITE M are related with memory and READ I/O and WRITE I/O are related with I/O devices.

S.No.	Parameter	Memory-mapped I/O	I/O-mapped I/O
1.	Address space	Memory and I/O devices share the entire address space	Memory and I/O devices have separate address space
2.	Hardware	No additional hardware required	Additional hardware required
3.	Implementation	Easy to implement	Difficult to implement
4.	Address	Same address cannot be used to refer both memory and I/O device.	Same address can be used to refer both memory and I/O device.
5.	Control lines	Memory control lines are used to control I/O devices.	Different set of control lines are used to control memory and I/O.
6.	Control lines used	The control lines are: READ, WRITE	The control lines are: READ M, WRITE M, READ I/O, WRITE I/O

I/O instructions

Two I/O instructions are used to implement programmed I/O.

- **IN:** The instruction IN X causes a word to be transferred from I/O port X to the accumulator register A.
- **OUT:** The instruction OUT X transfer a word from the accumulator register A to the I/O port X.

Limitations of programmed I/O

The programmed I/O method has two limitations:

- The speed of the CPU is reduced due to low speed I/O devices.
- Most of the CPU time is wasted

6. Describe the DMA controller in a computer system with a neat block diagram. Explain mechanism Direct Memory Access. (Nov/Dec2012, 2013, 2015, 2016) (Apr / May 2016, 2017, Nov /Dec 2011) (Nov/Dec 2018).With a neat sketch explain the working principle of DMA. (Apr/May 2019)

DIRECT MEMORY ACCESS

- A special control unit may be provided to allow the transfer of large block of data at high speed directly between the external device and main memory, without continuous intervention by the processor. This approach is called DMA.
- DMA transfers are performed by a control circuit called the **DMA Controller.**

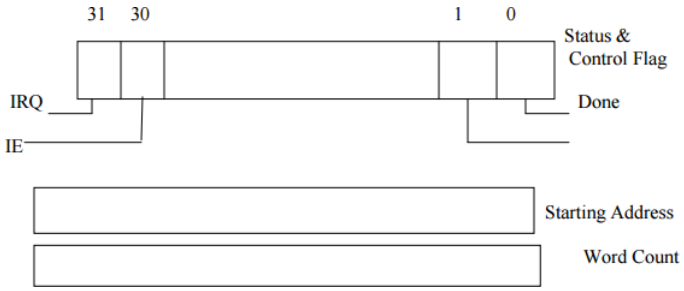
To initiate the transfer of a block of words, the processor sends,

i) Starting address

ii) Number of words in the block

iii) Direction of transfer.

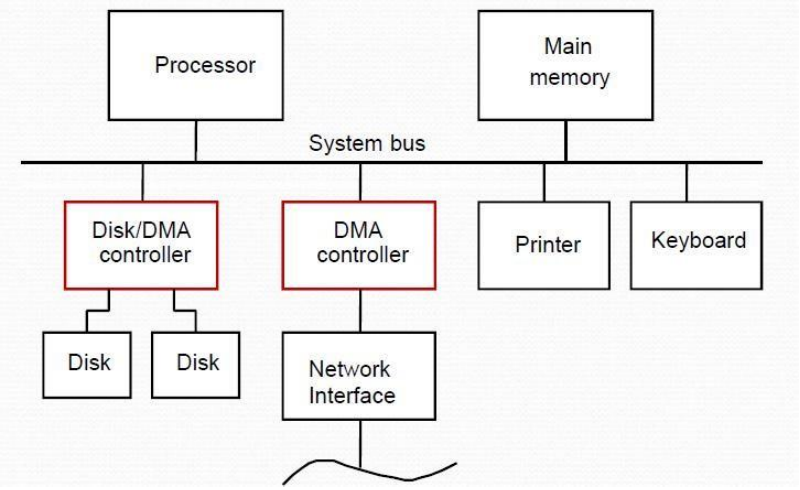
- When a block of data is transferred , the DMA controller increment the memory address for successive words and keep track of number of words and it also informs the processor by raising an interrupt signal.



- While DMA control is taking place, the program requested the transfer cannot continue and the processor can be used to execute another program.
- After DMA transfer is completed, the processor returns to the program that requested the transfer.

R/W->Determines the direction of transfer

- When **R/W =1**, DMA controller read data from memory to I/O device.
- **R/W =0**, DMA controller perform write operation.
- **Done Flag=1**, the controller has completed transferring a block of data and is ready to receive another command.
- **IE=1,** it causes the controller to raise an interrupt (interrupt Enabled) after it has completed transferring the block of data.
- **IRQ=1,** it indicates that the controller has requested an interrupt.



- A DMA controller connects a high speed network to the computer bus, and the disk controller for two disks also has DMA capability and it provides two DMA channels.
- To start a DMA transfer of a block of data from main memory to one of the disks, the program write‘s the address and the word count information into the registers of the corresponding channel of the disk controller.
- When DMA transfer is completed, it will be recorded in status and control registers of the DMA channel (ie) Done bit=IRQ=IE=1.

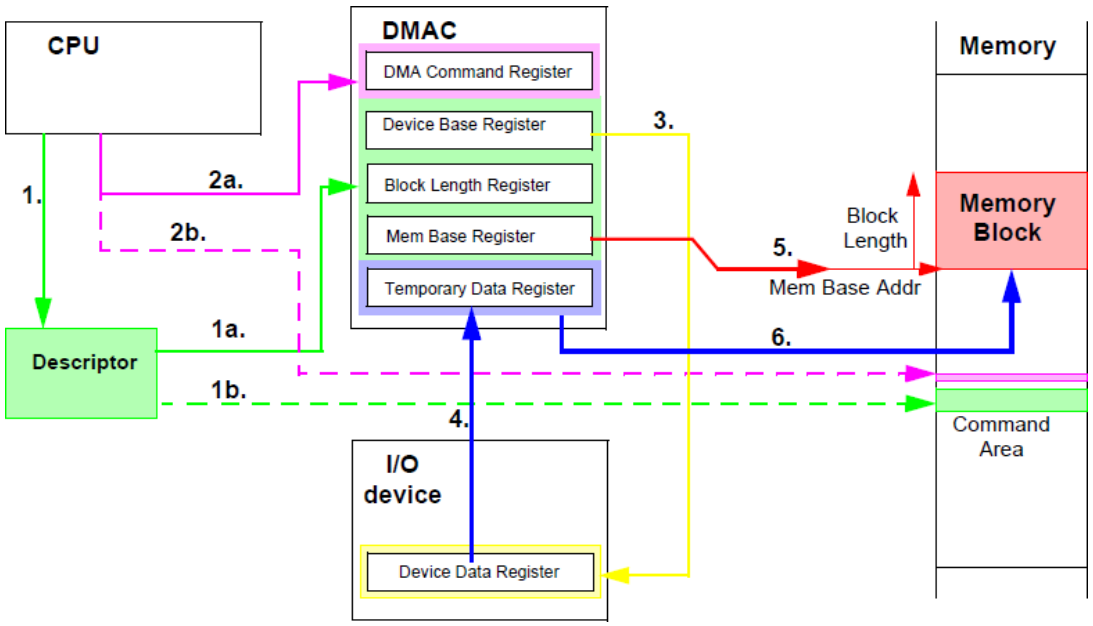
DMA Operations: May 2009

A lot of different operating modes exist for DMACs. The simplest one is the single block transfer copying a block of data from a device to memory. For the more complex operations please refer to the literature.

Here, only a short list of operating modes is given:

- Single block transfer
- Chained block transfers
- Linked block transfers
- Fly-by transfers

All these operations normally access the block of data in a linear sequence. Nevertheless, there are more usefull access functions possible, as there are: constant stride, constant stride with offset, incremental stride.



Execution of a DMA-operation (single block transfer)

- The CPU prepares the DMA-operation by the construction of a descriptor (1), containing all necessary information for the DMAC to independently perform the DMA-operation (offload engine for data transfer).
- It initializes the operation by writing a command to a register in the DMAC (2a) or to a special assigned memory area (command area), where the DMAC can poll for the command and/or the descriptor (2b). Then the DMAC addresses the device data register (3) and read the data into a temporary data register (4).
- In another bus transfer cycle, it addresses the memory block (5) and writes the data from the temporary data register to the memory block (6).

Cycle Stealing:

- Requests by DMA devices for using the bus are having higher priority than processor requests.
- Top priority is given to high speed peripherals such as, Disk High speed Network Interface and Graphics display device.
- Since the processor originates most memory access cycles, the DMA controller can be said to steal the memory cycles from the processor. This interviewing technique is called **Cycle stealing.**
- **Burst Mode:** The DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as **Burst/Block Mode.**
- **Bus Master:** The device that is allowed to initiate data transfers on the bus at any given time is called the **bus master.**

BUS ARBITRATION:

7. **Explain in detail about the Bus Arbitration techniques in DMA. (Nov 2011, 2012, 2014) Apr / May 2011, 2017, May 2013**

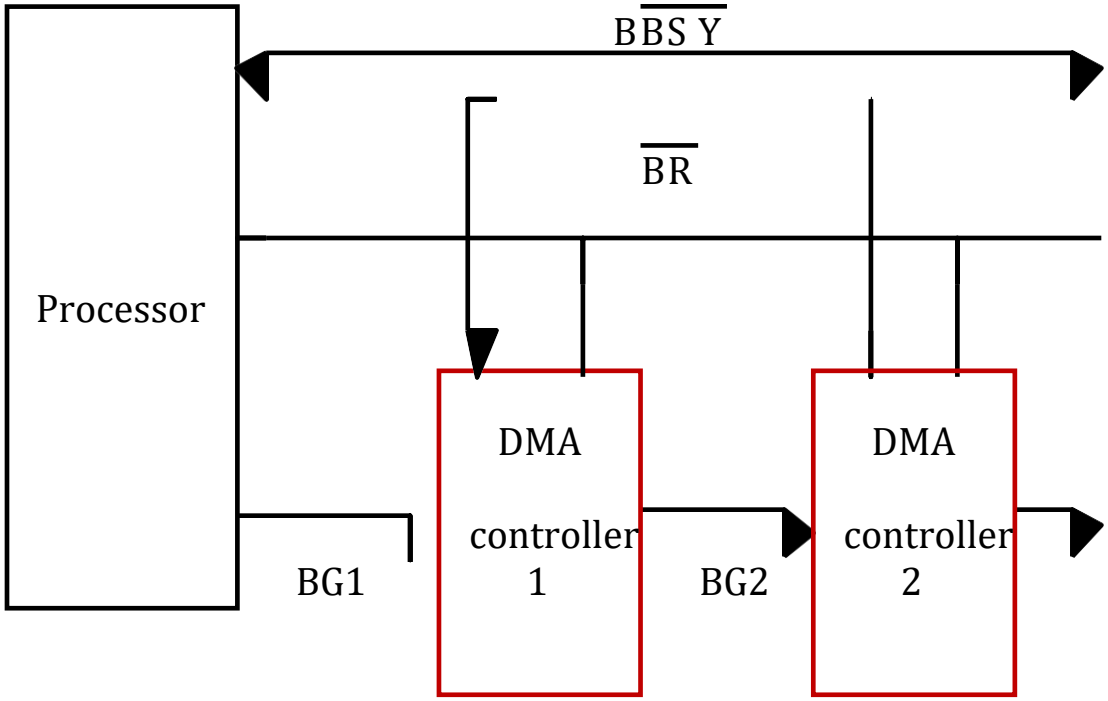
- **Bus Arbitration:** It is the process by which the next device to become the bus master is selected and the bus mastership is transferred to it.

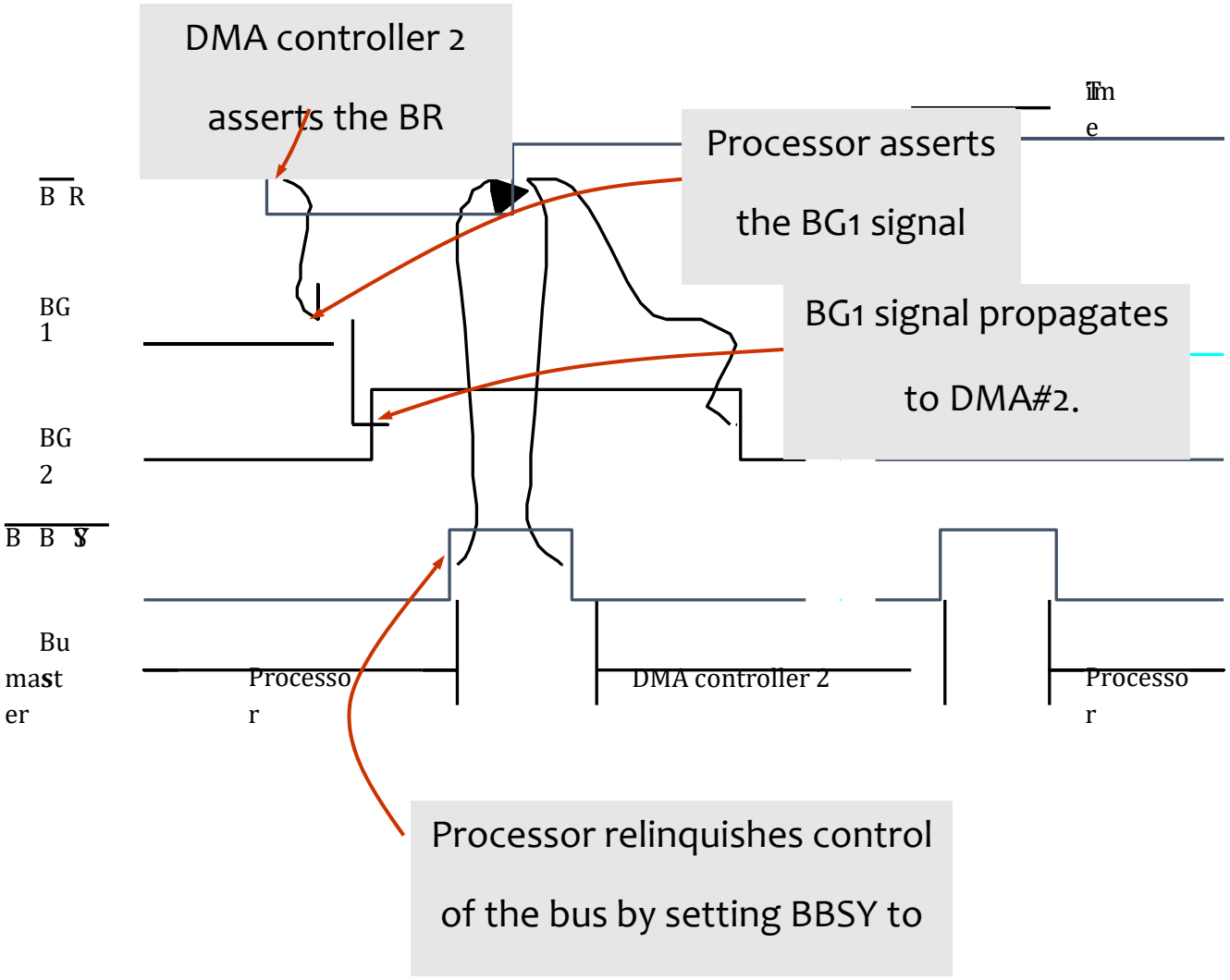
Types: There are 2 approaches to bus arbitration. They are,

- Centralized arbitration (A single bus arbiter performs arbitration)
- Distributed arbitration (all devices participate in the selection of next bus master).

Centralized Arbitration:

- Here the processor is the bus master and it may grants bus mastership to one of its DMA controller.
- A DMA controller indicates that it needs to become the bus master by activating the Bus Request line (BR) which is an open drain line.
- The signal on BR is the logical OR of the bus request from all devices connected to it.When BR is activated the processor activates the Bus Grant Signal (BGI) and indicated the DMA controller that they may use the bus when it becomes free.
- This signal is connected to all devices using a daisy chain arrangement.
- If DMA requests the bus, it blocks the propagation of Grant Signal to other devices and it indicates to all devices that it is using the bus by activating open collector line, Bus Busy (BBSY).



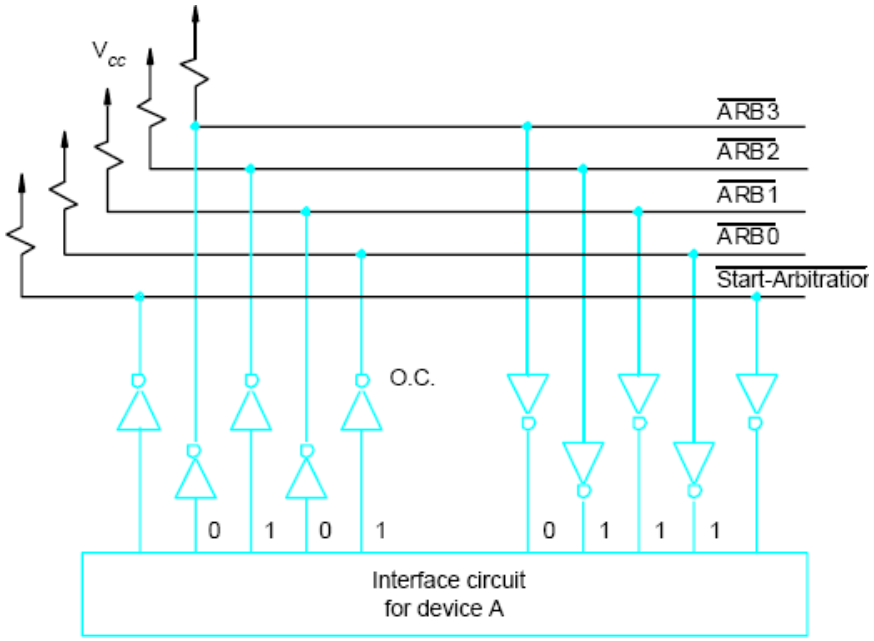


Sequence of signals during transfer of bus mastership for the devices

- The timing diagram shows the sequence of events for the devices connected to the processor is shown.
- DMA controller 2 requests and acquires bus mastership and later releases the bus.
- During its tenture as bus master, it may perform one or more data transfer.
- After it releases the bus, the processor resources bus mastership.

Distributed Arbitration

- It means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process.



- Each device on the bus is assigned a 4 bit id. When one or more devices request the bus, they assert the Start-Arbitration signal & place their 4 bit ID number on four open collector lines, ARB0 to ARB3.
- A winner is selected as a result of the interaction among the signals transmitted over these lines.
- The net outcome is that the code on the four lines represents the request that has the highest ID number.
- The drivers are of open collector type. Hence, if the i/p to one driver is equal to 1, the i/p to another driver connected to the same bus line is equal to _0‘ (ie. bus is in low-voltage state).
- **Eg:** Assume two devices A & B have their ID 5 (0101), 6(0110) and their code is 0111.
- Each device compares the pattern on the arbitration line to its own ID starting from MSB.
- If it detects a difference at any bit position, it disables the drivers at that bit position. It does this by placing _0‘ at the i/p of these drivers.
- _A‘ detects a difference in line ARB1; hence it disables the drivers on lines ARB1 & ARB0. This causes the pattern on the arbitration line to change to 0110 which means that _B‘ has won the contention.

INPUT DEVICES:

8. Explain in detail about input devices with an example.

Input Devices

The Input Devices are the hardware that is used to transfer transfers input to the computer. The data can be in the form of text, graphics, sound, and text. Output device display data from the memory of the computer.

Output can be text, numeric data, line, polygon, and other objects.

These Devices include:

1. Keyboard

2. Mouse
3. Trackball
4. Spaceball
5. Joystick
6. Light Pen
7. Digitizer
8. Touch Panels
9. Voice Recognition
10. Image Scanner

Keyboard:

The most commonly used input device is a keyboard. The data is entered by pressing the set of keys. All keys are labeled. A keyboard with 101 keys is called a QWERTY keyboard.

The keyboard has alphabetic as well as numeric keys. Some special keys are also available.

1. **Numeric Keys:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
2. **Alphabetic keys:** a to z (lower case), A to Z (upper case)
3. **Special Control keys:** Ctrl, Shift, Alt
4. **Special Symbol Keys:** ; , " ? @ ~ ? :
5. **Cursor Control Keys:** ↑ → ← ↓
6. **Function Keys:** F1 F2 F3... F9.
7. **Numeric Keyboard:** It is on the right-hand side of the keyboard and used for fast entry of numeric data.

Function of Keyboard:

1. Alphanumeric Keyboards are used in CAD. (Computer Aided Drafting)
2. Keyboards are available with special features line screen co-ordinates entry, Menu selection or graphics functions, etc.
3. Special purpose keyboards are available having buttons, dials, and switches. Dials are used to enter scalar values. Dials also enter real numbers. Buttons and switches are used to enter predefined function values.

Advantage:

1. Suitable for entering numeric data.
2. Function keys are a fast and effective method of using commands, with fewer errors.

Disadvantage:

1. Keyboard is not suitable for graphics input.

Mouse:

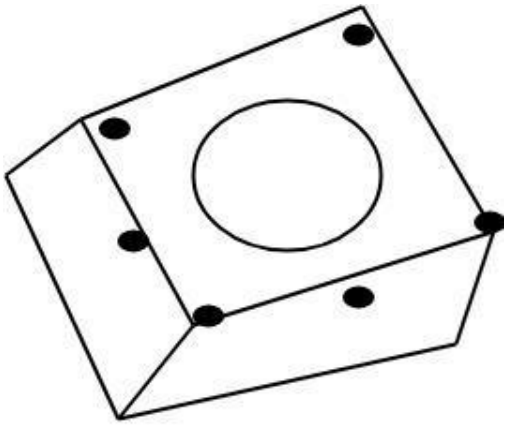
A Mouse is a pointing device and used to position the pointer on the screen. It is a small palm size box. There are two or three depression switches on the top. The movement of the mouse along the x-axis helps in the horizontal movement of the cursor and the movement along the y-axis helps in the vertical movement of the cursor on the screen. The mouse cannot be used to enter text. Therefore, they are used in conjunction with a keyboard.

Advantage:

1. Easy to use
2. Not very expensive

Trackball

It is a pointing device. It is similar to a mouse. This is mainly used in notebook or laptop computer, instead of a mouse. This is a ball which is half inserted, and by changing fingers on the ball, the pointer can be moved.



TrackBall

Advantage:

1. Trackball is stationary, so it does not require much space to use it.
2. Compact Size

Spaceball:

It is similar to trackball, but it can move in six directions where trackball can move in two directions only. The movement is recorded by the strain gauge. Strain gauge is applied with pressure. It can be pushed and pulled in various directions. The ball has a diameter around 7.5 cm. The ball is mounted in the base using rollers. One-third of the ball is an inside box, the rest is outside.

Applications:

1. It is used for three-dimensional positioning of the object.
2. It is used to select various functions in the field of virtual reality.
3. It is applicable in CAD applications.
4. Animation is also done using spaceball.

5. It is used in the area of simulation and modeling.

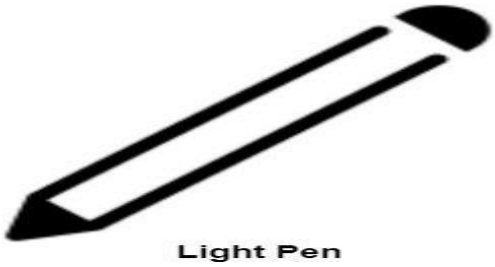
Joystick:

A Joystick is also a pointing device which is used to change cursor position on a monitor screen. Joystick is a stick having a spherical ball as its both lower and upper ends as shown in fig. The lower spherical ball moves in a socket. The joystick can be changed in all four directions. The function of a joystick is similar to that of the mouse. It is mainly used in Computer Aided Designing (CAD) and playing computer games.

When the wave signals are interrupted by some contact with the screen, that located is recorded. Touch screens have long been used in military applications.

Light Pen

Light Pen (similar to the pen) is a pointing device which is used to select a displayed menu item or draw pictures on the monitor screen. It consists of a photocell and an optical system placed in a small tube. When its tip is moved over the monitor screen, and pen button is pressed, its photocell sensing element detects the screen location and sends the corresponding signals to the CPU.



Uses:

1. Light Pens can be used as input coordinate positions by providing necessary arrangements.
2. If background color or intensity, a light pen can be used as a locator.
3. It is used as a standard pick device with many graphics system.
4. It can be used as stroke input devices.
5. It can be used as valuators

Digitizers:

The digitizer is an operator input device, which contains a large, smooth board (the appearance is similar to the mechanical drawing board) & an electronic tracking device, which can be changed over the surface to follow existing lines. The electronic tracking device contains a switch for the user to record the desire x & y

coordinate positions. The coordinates can be entered into the computer memory or stored or an off-line storage medium such as magnetic tape.

Digitizer

Advantages:

1. Drawing can easily be changed.
2. It provides the capability of interactive graphics.

Disadvantages:

1. Costly
2. Suitable only for applications which required high-resolution graphics.

Touch Panels:

Touch Panels is a type of display screen that has a touch-sensitive transparent panel covering the screen. A touch screen registers input when a finger or other object comes in contact with the screen.

When the wave signals are interrupted by some contact with the screen, that located is recorded. Touch screens have long been used in military applications.

OUTPUT DEVICES:

9. Explain in detail about Output devices with an example.

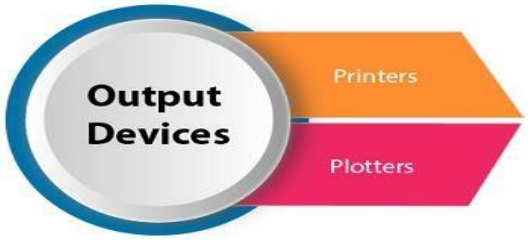
It is an electromechanical device, which accepts data from a computer and translates them into form understand by users.

Following are Output Devices:

1. Printers
2. Plotters

Printers:

Printer is the most important output device, which is used to print data on paper.



Types of Printers:

1. Impact Printers: The printers that print the characters by striking against the ribbon and onto the papers are known as Impact Printers.

These Printers are of two types:

1. Character Printers
2. Line Printers

2. Non-Impact Printers: The printers that print the characters without striking against the ribbon and onto the papers are called Non-Impact Printers. These printers print a complete page at a time, therefore, also known as Page Printers.

Page Printers are of two types:

1. Laser Printers
2. Inkjet Printers

Laser Printers:

These are non-impact page printers. They use laser lights to produces the dots needed to form the characters to be printed on a page & hence the name laser printers.

The output is generated in the following steps:

Step1: The bits of data sent by processing unit act as triggers to turn the laser beam on & off.

Step2: The output device has a drum which is cleared & is given a positive electric charge. To print a page the modulated laser beam passing from the laser scans back & forth the surface of the drum. The positive electric charge on the drum is stored on just those parts of the drum surface which are exposed to the laser

beam create the difference in electric which charges on the exposed drum surface.
Step3: The laser exposed parts of the drum attract an ink powder known as toner.

Step4: The attracted ink powder is transferred to paper.

Step5: The ink particles are permanently fixed to the paper by using either heat or pressure technique.

Step6: The drum rotates back to the cleaner where a rubber blade cleans off the excess ink & prepares the drum to print the next page.

Liquid Crystal Displays (LCDs)

LCD Display Monitor

The flat-panel display refers to a class of video devices that have reduced volume, weight and power requirement in comparison to the CRT. You can hang them on walls or wear them on your wrists. Current uses of flat-panel displays include calculators, video games, monitors, laptop computer, and graphics display.

The flat-panel display is divided into two categories −

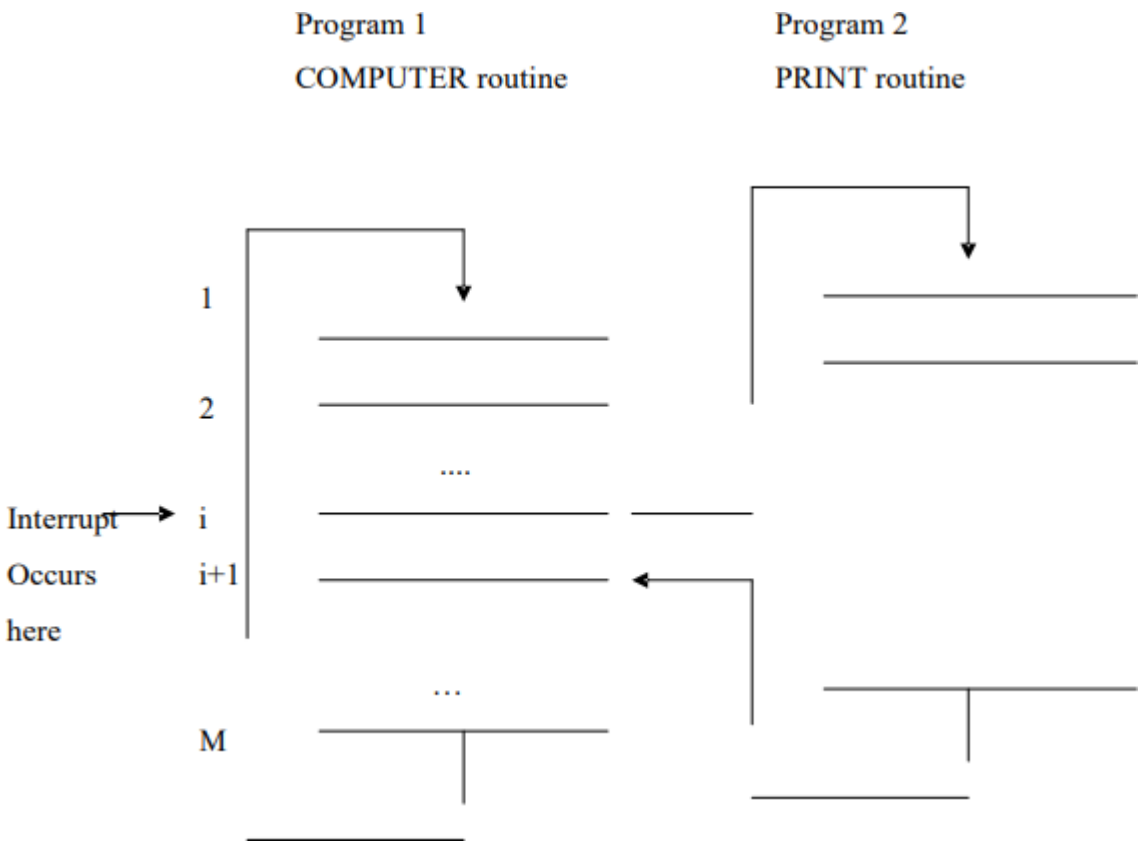
- **Emissive Displays** − Emissive displays are devices that convert electrical energy into light. For example, plasma panel and LED (Light-Emitting Diodes).
- **Non-Emissive Displays** − Non-emissive displays use optical effects to convert sunlight or light from some other source into graphics patterns. For example, LCD (Liquid-Crystal Device).

ACCESSING I/O

10. How to accessing I/O devices to a computer?

- A simple arrangement to connect I/O devices to a computer is to use a single bus arrangement. The bus enables all the devices connected to it to exchange information.
- Typically, it consists of three sets of lines used to carry address, data, and control signals. Each I/O device is assigned a unique set of addresses.
- When the processor places a particular address on the address line, the device that recognizes this address responds to the commands issued on the control lines.
- The processor requests either a read or a write operation, and the requested data are transferred over the data lines, when I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.
- With memory-mapped I/O, any machine instruction that can access memory can be used to transfer data to or from an I/O device.
- For example, if DATAIN is the address of the input buffer associated with the keyboard, the instruction Move DATAIN, R0 Reads the data from DATAIN and stores them into processor register R0.

- Similarly, the instruction Move R0, DATAOUT Sends the contents of register R0 to location DATAOUT, which may be the output data buffer of a display unit or a printer.
- Most computer systems use memory-mapped I/O. some processors have special In and Out instructions to perform I/O transfers.
- When building a computer system based on these processors, the designer had the option of connecting I/O devices to use the special I/O address space or simply incorporating them as part of the memory address space.
- The I/O devices examine the low-order bits of the address bus to determine whether they should respond. The hardware required to connect an I/O device to the bus.
- The address decoder enables the device to recognize its address when this address appears on the address lines.
- The data register holds the data being transferred to or from the processor. The status register contains information relevant to the operation of the I/O device.
- Both the data and status registers are connected to the data bus and assigned unique addresses.
- The address decoder, the data and status registers, and the control circuitry required to coordinate I/O transfers constitute the device‘s interface circuit.
- I/O devices operate at speeds that are vastly different from that of the processor. When a human operator is entering characters at a keyboard, the processor is capable of executing millions of instructions between successive character entries.
- An instruction that reads a character from the keyboard should be executed only when a character is available in the input buffer of the keyboard interface.
- Also, we must make sure that an input character is read only once. This example illustrates program-controlled I/O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device.
- We say that the processor polls the device. There are two other commonly used mechanisms for implementing I/O operations: interrupts and direct memory access.
- In the case of interrupts, synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation.
- Direct memory access is a technique used for high-speed I/O devices. It involves having the device interface transfer data directly to or from the memory, without continuous involvement by the processor.
- The routine executed in response to an interrupt request is called the interrupt service routine, which is the PRINT routine in our example.
- Interrupts bear considerable resemblance to subroutine calls. Assume that an interrupt request arrives during execution of instruction i in figure 1



- The processor first completes execution of instruction i. Then, it loads the program counter with the address of the first instruction of the interrupt-service routine.
- For the time being, let us assume that this address is hardwired in the processor. After execution of the interrupt-service routine, the processor has to come back to instruction i +1.
- Therefore, when an interrupt occurs, the current contents of the PC, which point to instruction i+1, must be put in temporary storage in a known location.
- A Return-from interrupt instruction at the end of the interrupt-service routine reloads the PC from the temporary storage location, causing execution to resume at instruction i +1.
- In many processors, the return address is saved on the processor stack. We should note that as part of handling interrupts, the processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal.
- This may be accomplished by means of a special control signal on the bus. An interrupt-acknowledge signal.
- The execution of an instruction in the interrupt-service routine that accesses a status or data register in the device interface implicitly informs that device that its interrupt request has been recognized.
- So far, treatment of an interrupt-service routine is very similar to that of a subroutine. An important departure from this similarity should be noted.
- A subroutine performs a function required by the program from which it is called. However, the interrupt-service routine may not have anything in common with the program being executed at the time the interrupt request is received.

- In fact, the two programs often belong to different users. Therefore, before starting execution of the interrupt-service routine, any information that may be altered during the execution of that routine must be saved.
- This information must be restored before execution of the interrupt program is resumed. In this way, the original program can continue execution without being affected in any way by the interruption, except for the time delay.
- The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine.
- The task of saving and restoring information can be done automatically by the processor or by program instructions.
- Most modern processors save only the minimum amount of information needed to maintain the registers involves memory transfers that increase the total execution time, and hence represent execution overhead.
- Saving registers also increase the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine. This delay is called interrupt latency.

SERIAL AND PARALLEL INTERFACE

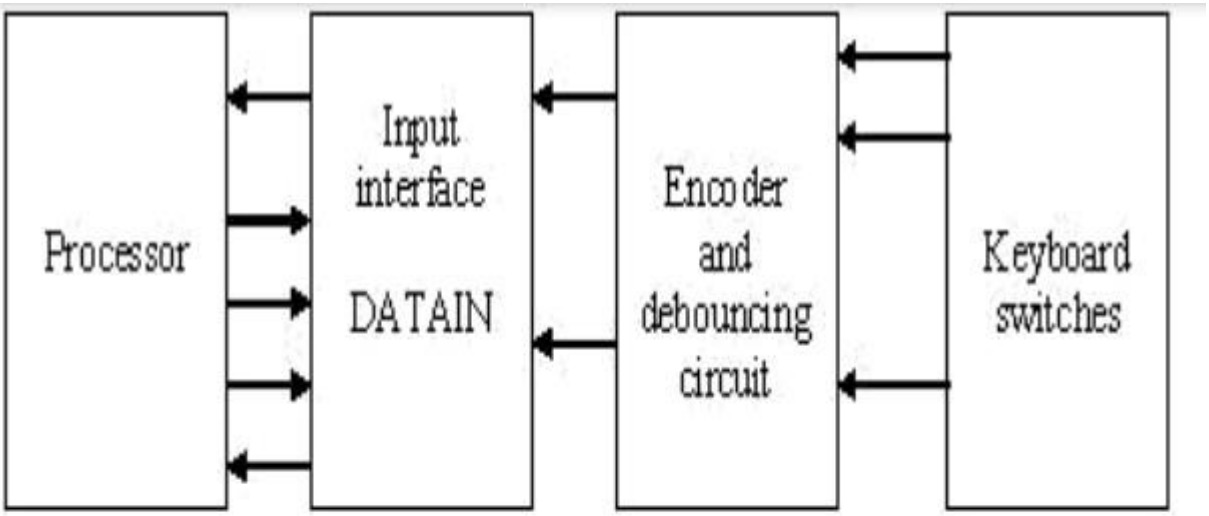
11. Explain about serial and parallel interface.

- An I/O interface consists of the circuitry required to connect an I/O device to a computer bus. On one side of the interface, we have bus signals.
- On the other side, we have a data path with its associated controls to transfer data between the interface and the I/O device – port.
- We have two types: Serial port and Parallel port A parallel port transfers data in the form of a number of bits (8 or 16) simultaneously to or from the device.
- A serial port transmits and receives data one bit at a time. Communication with the bus is the same for both formats.
- The conversion from the parallel to the serial format, and vice versa, takes place inside the interface circuit.
- In parallel port, the connection between the device and the computer uses a multiple-pin connector and a cable with as many wires.
- This arrangement is suitable for devices that are physically close to the computer. In serial port, it is much more convenient and cost-effective where longer cables are needed.
- Typically, the functions of an I/O interface are:
  - ✓ Provides a storage buffer for at least one word of data
  - ✓ Contains status flags that can be accessed by the processor to determine whether the buffer is full or empty
  - ✓ Contains address-decoding circuitry to determine when it is being addressed by the processor
  - ✓ Generates the appropriate timing signals required by the bus control scheme

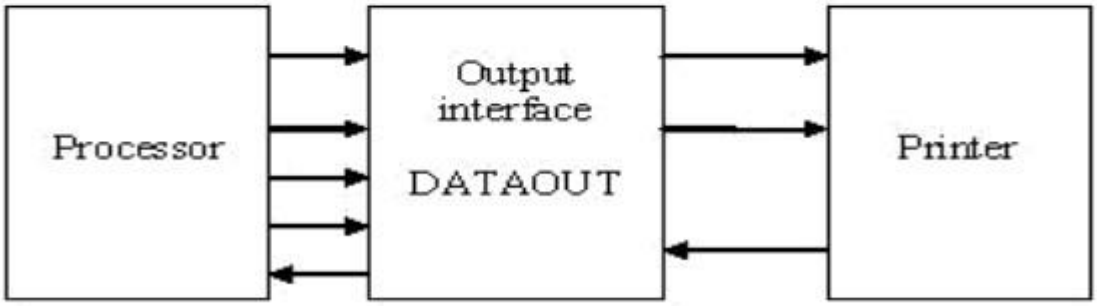
- ✓ Performs any format conversion that may be necessary to transfer data between the bus and the I/O device, such as parallel-serial conversion in the case of a serial port.

Parallel Port

- The hardware components needed for connecting a keyboard to a processor Consider the circuit of input interface which encompasses (as shown in below figure): –Status flag, SIN –R/~W –Master-ready –Address decoder.
- A detailed figure showing the input interface circuit is presented in figure.
- Now, consider the circuit for the status flag (figure 4.30). An edge-triggered D flip-flop is used along with read-data and master-ready signals



Keyboard to processor connection



Printer to processor connection

- The hardware components needed for connecting a printer to a processor are: the circuit of output interface, and –Slave-ready –R/~W –Master-ready –Address decoder –Handshake control.
- The input and output interfaces can be combined into a single interface.
- The general purpose parallel interface circuit that can be configured in a variety of ways.
- For increased flexibility, the circuit makes it possible for some lines to serve as inputs and some lines to serve as outputs, under program control.

Serial Port

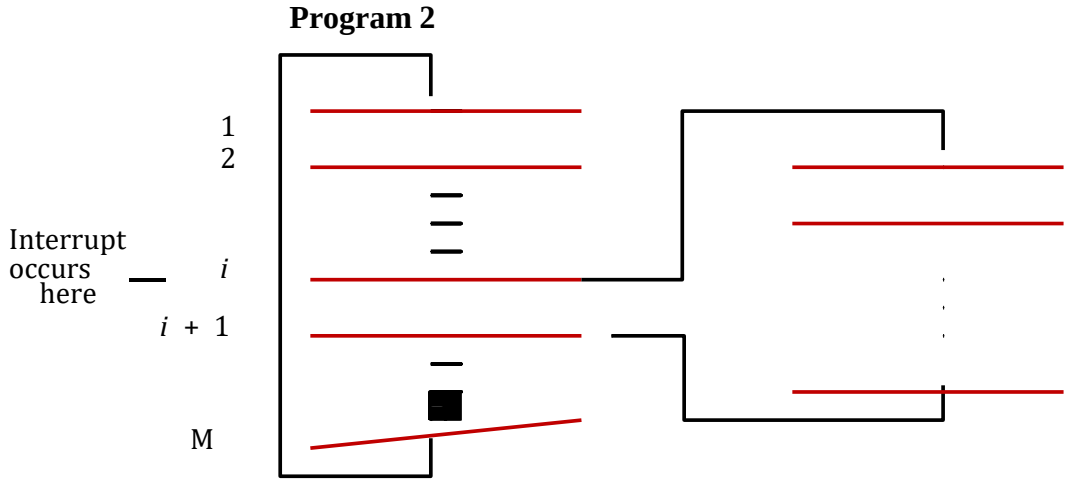
- A serial interface circuit involves – Chip and register select, Status and control, Output shift register, DATAOUT, DATAIN, Input shift register and Serial input/output

INTERRUPT

12. What is an interrupt? Explain the different types of interrupts and the different ways of handling the interrupts. Explain Interrupt Handling. (Nov/Dec 2016) (Nov/Dec 2018)Nov/Dec 2020.

INTERRUPTS:

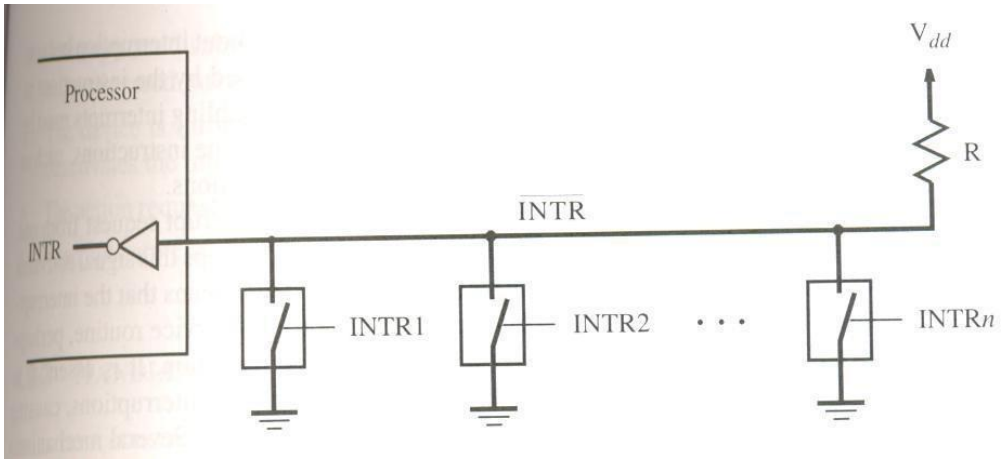
- An interrupt is an event that causes the execution of one program to be suspended and the execution of another program to begin.
- In program-controlled I/O, when the processor continuously monitors the status of the device, the processor will not perform any function.
- An alternate approach would be for the I/O device to alert the processor when it becomes ready. The Interrupt request line will send a hardware signal called the **interrupt signal** to the processor. On receiving this signal, the processor will perform the useful function during the waiting period.
- The routine executed in response to an interrupt request is called **Interrupt Service Routine**. The interrupt resembles the subroutine calls.



- The processor first completes the execution of instruction i. Then it loads the PC(Program Counter) with the address of the first instruction of the ISR.
- After the execution of ISR, the processor has to come back to instruction i + 1
- Therefore, when an interrupt occurs, the current contents of PC which point to i +1 is put in temporary storage in a known location.
- A return from interrupt instruction at the end of ISR reloads the PC from that temporary storage location, causing the execution to resume at instruction i+1.
- When the processor is handling the interrupts, it must inform the device that its request has been recognized so that it removes its interrupt requests signal.
- This may be accomplished by a special control signal called the **interrupt acknowledge signal.**
- The task of saving and restoring the information can be done automatically by the processor.
- The processor saves only the contents of program counter & status register (ie) it saves only the minimal amount of information to maintain the integrity of the program execution.

- Saving registers also increases the delay between the time an interrupt request is received and the start of the execution of the ISR. This delay is called the **Interrupt Latency**.
- Generally, the long interrupt latency in unacceptable. The concept of interrupts is used in Operating System and in Control Applications, where processing of certain routines must be accurately timed relative to external events. This application is also called as **real-time processing.**

Interrupt Hardware



- A single interrupt request line may be used to serve _n' devices.
- All devices are connected to the line via switches to ground. To request an interrupt, a device closes its associated switch, the voltage on INTR line drops to 0(zero).
- If all the interrupt request signals (INTR1 to INTRn) are inactive, all switches are open and the voltage on INTR line is equal to Vdd.
- When a device requests an interrupts, the value of INTR is the logical OR of the requests from individual devices. (ie)INTR = INTR1+..... +INTRn.
- INTR->It is used to name the INTR signal on common line it is active in the low voltage state.
- **Open collector** or **Open drain** is used to drive INTR line.
- The Output of the Open collector (or) Open drain control is equal to a switch to the ground that is open when gates input is in _0' state and closed when the gates input is in _1' state.
- Resistor _R' is called **a pull-up resistor** because it pulls the line voltage up to the high voltage state when the switches are open.

Enabling and Disabling Interrupts

- The arrival of an interrupt request from an external device causes the processor to suspend the execution of one program & start the execution of another because the interrupt may alter the sequence of events to be executed.
- INTR is active during the execution of **Interrupt Service Routine**.
- There are 3 mechanisms to solve the problem of infinite loop which occurs due to successive interruptions of active INTR signals.
- The following are the typical scenario.

- The device raises an interrupt request.
- The processor interrupts the program currently being executed.
- Interrupts are disabled by changing the control bits is PS (Processor Status register)
- The device is informed that its request has been recognized & in response, it deactivates the INTR signal.
- The actions are enabled & execution of the interrupted program is resumed.

Edge-triggered

- The processor has a special interrupt request line for which the interrupt handling circuit responds only to the leading edge of the signal. Such a line said to **be edge-triggered.**

Handling Multiple Devices:

- When several devices requests interrupt at the same time, it raises some questions. They are.
  - How can the processor recognize the device requesting an interrupt?
  - Given that the different devices are likely to require different ISR, how can the processor obtain the starting address of the appropriate routines in each case?
  - Should a device be allowed to interrupt the processor while another interrupt is being serviced?
  - How should two or more simultaneous interrupt requests be handled?

Polling Scheme:

- If two devices have activated the interrupt request line, the ISR for the selected device (first device) will be completed & then the second request can be serviced.
- The simplest way to identify the interrupting device is to have the ISR polls all the encountered with the IRQ bit set is the device to be serviced.
- IRQ (Interrupt Request) -> when a device raises an interrupt requests, the status register IRQ is set to 1.

Merit:

- It is easy to implement.

Demerit:

- The time spent for interrogating the IRQ bits of all the devices that may not be requesting any service.

Vectored Interrupt: Nov / Dec 2011, 2012

- Here the device requesting an interrupt may identify itself to the processor by sending a special code over the bus & then the processor start executing the ISR.
- The code supplied by the processor indicates the starting address of the ISR for the device.
- The code length ranges from 4 to 8 bits. The location pointed to by the interrupting device is used to store the staring address to ISR.
- The processor reads this address, called the interrupt vector & loads into PC.
- The interrupt vector also includes a new value for the Processor Status Register.

- When the processor is ready to receive the interrupt vector code, it activate the interrupt acknowledge (INTA) line.

Interrupt Nesting: Multiple Priority Scheme:

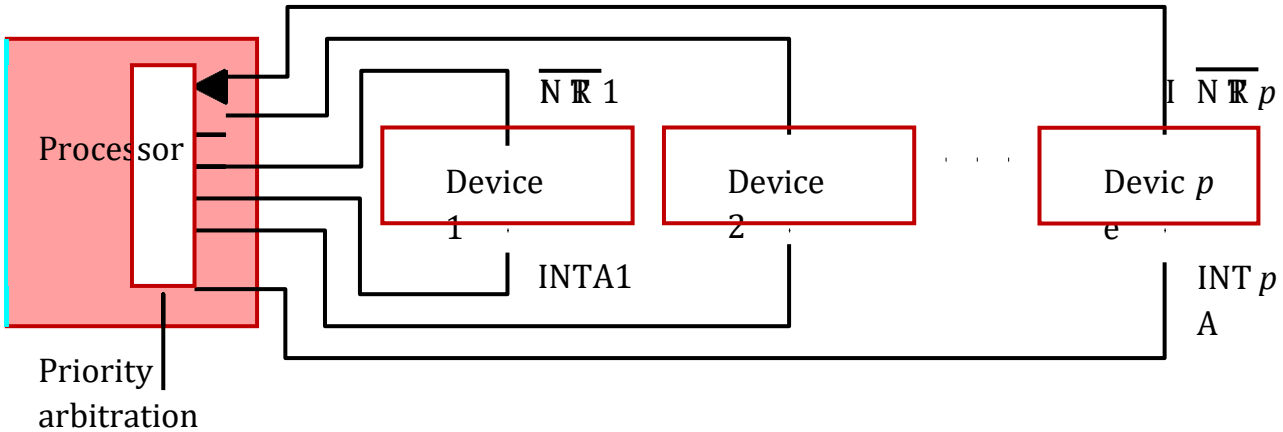
- In multiple level priority schemes, we assign a priority level to the processor that can be changed under program control.
- The priority level of the processor is the priority of the program that is currently being executed.
- The processor accepts interrupts only from devices that have priorities higher than its own.
- At the time the execution of an ISR for some device is started, the priority of the processor is raised to that of the device.
- The action disables interrupts from devices at the same level of priority or lower.

Privileged Instruction:

- The processor priority is usually encoded in a few bits of the Processor Status word.
- It can also be changed by program instruction & then it is writing into PS. These instructions are called **privileged instruction.**
- This can be executed only when the processor is in supervisor mode.
- The processor is in supervisor mode only when executing OS routines. It switches to the user mode before beginning to execute application program.

Privileged Exception:

- User program cannot accidently or intentionally change the priority of the processor & disrupts the system operation.
- An attempt to execute a privileged instruction while in user mode, leads to a special type of interrupt called the privileged exception.

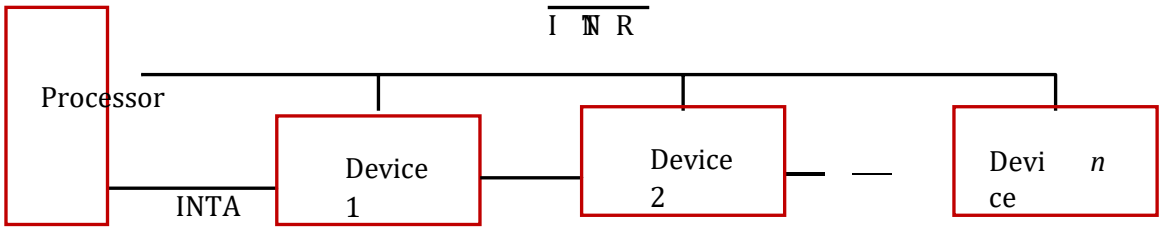


Implementation of Interrupt Priority using individual Interrupt request acknowledge lines

- Each of the interrupt request line is assigned a different priority level.
- Interrupt request received over these lines are sent to a priority arbitration circuit in the processor.
- A request is accepted only if it has a higher priority level than that currently assigned to the processor.

Simultaneous Requests:

Daisy Chain:



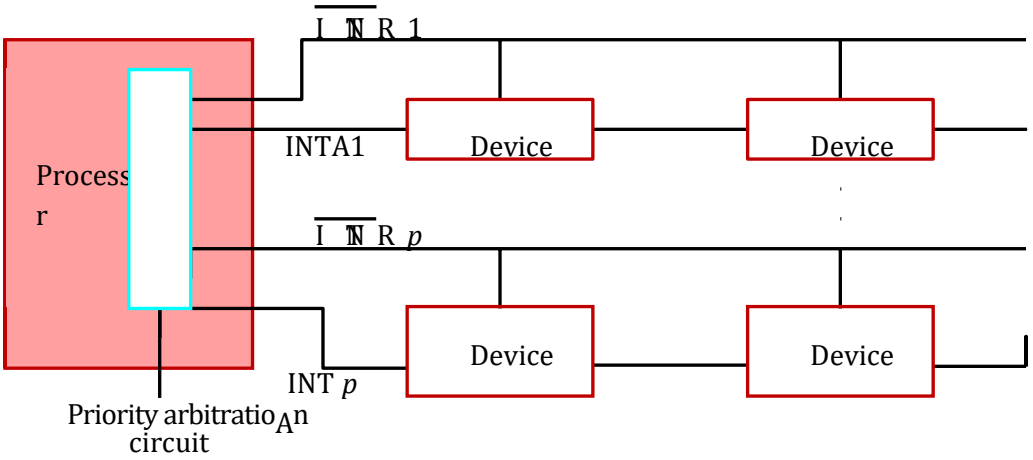
- The interrupt request line INTR is common to all devices.
- The interrupt acknowledge line INTA is connected in a daisy chain fashion such that INTA signal propagates serially through the devices.
- When several devices raise an interrupt request, the INTR is activated & the processor responds by setting INTA line to 1. This signal is received by device.
- Device1 passes the signal on to device2 only if it does not require any service.
- If devices1 has a pending request for interrupt blocks that INTA signal & proceeds to put its identification code on the data lines. Therefore, the device that is electrically closest to the processor has the highest priority.

Merits:

- It requires fewer wires than the individual connections.

Arrangement of Priority Groups:

- Here the devices are organized in groups & each group is connected at a different priority level. Within a group, devices are connected in a daisy chain.



- At the devices end, an interrupt enable bit in a control register determines whether the device is allowed to generate an interrupt requests.
- At the processor end, either an interrupt enable bit in the PS (Processor Status) or a priority structure determines whether a given interrupt requests will be accepted.

Initiating the Interrupt Process:

- Load the starting address of ISR in location INTVEC (vectored interrupt).

- Load the address LINE in a memory location PNTR. The ISR will use this location as a pointer to store i/o characters in the memory.
- Enable the keyboard interrupts by setting bit 2 in register CONTROL to 1.
- Enable interrupts in the processor by setting to 1, the IE bit in the processor status register PS.

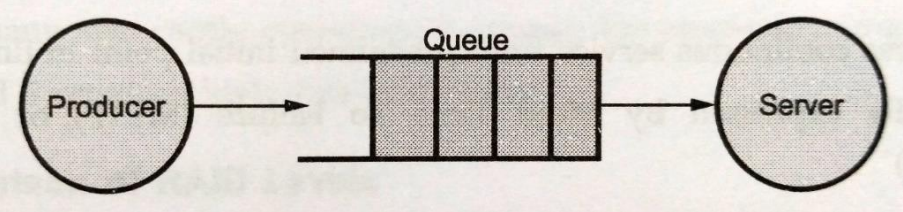
Exception of ISR:

- Read the input characters from the keyboard input data register. This will cause the interface circuits to remove its interrupt requests.
- Store the characters in a memory location pointed to by PNTR & increment PNTR.
- When the end of line is reached, disable keyboard interrupt & inform program main.
- Return from interrupt.

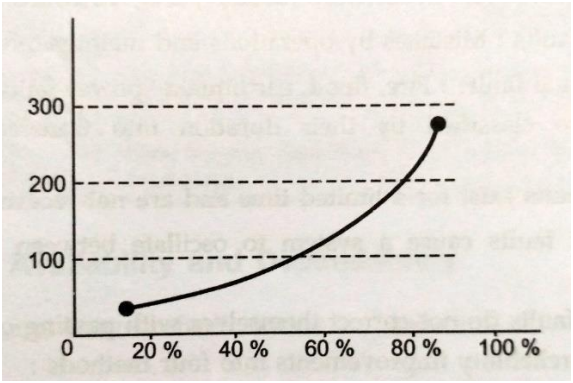
I/OPerformanceMeasures

13. Explain various ways to List and explain various I/O performance measures. (April/May 2017) (Or)Explain in detail about I/O performance measures with an example. (April/May 2014,2015)

- Measures used to quantify I/O performance attempt to measure a more diverse range of properties than the case of CPU performance.
- The traditional measures of performance/ namely response time and throughput/ also apply to I/O. I/O throughput is sometimes called I/O bandwidth/ and response time is sometimes called latency.
- Fig. shows the traditional producer-server model of response time.



- The producer creates tasks to be performed and places them in a buffer; the server takes tasks from the first-in-first-out buffer and performs them.
- Response time and throughput are non-linear. From a transaction server model:
  1.Throughput is maximized when the queue is never empty;
  2.Response time is minimized when die queue is empty.
- Figure below shows throughput versus response time for a typical storage system. Consider two interactive computing environments/ one keyboard driven/ one graphical.



- Computing interaction or transaction time is divided into three components,
    1. **Entry Time :** Time for user to make a request;
    2. **System Response Time :** Time between request and response;
    3. **Think Time :** Time between system response and next request
- The sum of these three parts is called the transaction time. Several studies report that user productivity is inversely proportional to transaction time; transactions per hour are a measure of the work completed per hour by the user.
- System response time is naturally the shortest duration. Does this minimize the impact of response time?
- Effect of system response time on user 'thinking' time.
    1. Any reduction to response time has more than a linear reduction on total transaction time.
    2. Users need less time to think when given a faster response;
    3. Possible to attach an economic benefit to response time and throughput;
    4. In order to maintain user interest, response times need to be < 1.0 second.

Problem: [May 2019]

Suppose a processor sends 80 disks I/Os per second, these requests are exponentially distributed, and the average service time of an older disk is 25 ms.

Answer the following questions:

1. On average, how utilized is the disk?

2. What is the average time spent in the queue?

3.What is the average response time for a disk request, including the queuing time and disk service time?

Average number of Arriving tasks / second is 80 ms.

Average disk time to service a task is 25ms = (0 .025 sec).

The server utilization is then

Service utilization = Arrival rate x Time $_{\text{Server}}$

= 80 x 0.025 = 2.0

Since the service times are exponentially distributed, we can use the simplifiedformula for the average time spent waiting in line:

$$Time_{Queue} = Time_{Server} \text{ X } \frac{server\ Utilization}{(1 - Server\ Utilization)}$$

$$= 25\ ms \text{ X } \frac{2}{1-2}$$

= 50 ms (consider +ve value)

The average response time is

Time system = Time $_{\text{queue}}$ + Time $_{\text{Server}}$

= 50 ms + 25 ms

= 75 ms

Thus, on average we spend 75% of our time waiting in the queue!

Universal Serial Bus (USB)

14. Explain about USB.

- The **universal serial bus (USB)** is a standard interface for connecting a wide range of devices to the computer such as keyboard, mouse, smart phones, speakers, cameras etc.
- The USB was introduced for commercial use in the year 1995 at that time it has a data transfer speed of 12 megabits/s.
- With some improvement, a modified USB 2 was introduced which is also called a high speed USB that transfers data at 480 megabits/s.
- With the evolution of I/O devices that require high speed data transfer also leads to the development of USB 3 which is also referred to as Super speed USB which transfers data at 5 gigabits/s.
- The recent version of USB can transfer data up to 20 gigabits/s.

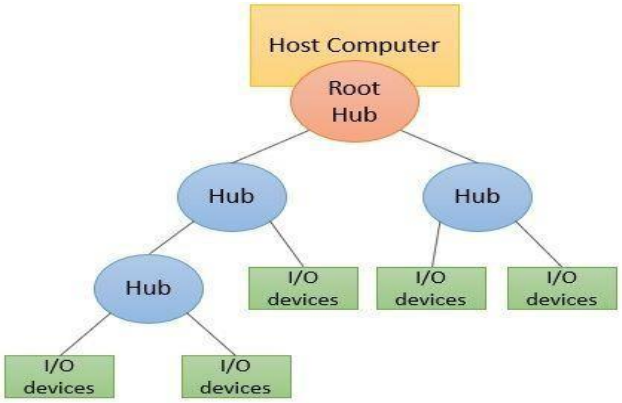
Content: Universal Serial Bus (USB)

1. Key Objectives
2. USB Architecture
3. Isochronous Traffic on USB
4. Types of USB Connectors
5. Electrical Characteristics of USB

Key Objectives of Universal Serial Bus

- Before getting into the details of the universal serial bus we will discuss some of the key objectives that are taken into account while designing a USB.
- The developed USB must be simple and a low-cost interconnection system that should be easy to use.
- The developed USB must be compatible with all new I/O devices, their bit rates, internet connections and audio, video application.
- The USB must support a plug-and-play mode of operation.
- The USB must support low power implementation.
- The USB must also provide support for legacy hardware and software.

USB Architecture



Universal Serial Bus Tree Structure

- When multiple I/O devices are connected to the computer through USB they all are organized in a tree structure.
- Each I/O device makes a point-to-point connection and transfers data using the serial transmission format we have discussed serial transmission in our previous content _interface circuit‘.
- As we know a tree structure has a **root, nodes** and **leaves.**
- The tree structure connecting I/O devices to the computer using USB has nodes which are also referred to as a **hub**.
- Hub is the inter-mediatory connecting point between the I/O devices and the computer.
- Every tree has a root here; it is referred to as the **root hub** which connects the entire tree to the hosting computer.
- The leaves of the tree here are nothing but the I/O devices such as a mouse, keyboard, camera, and speaker.
- The USB works on the principle of polling.
- In **polling**, the processor keeps on checking whether the I/O device is ready for data transfer or not.
- So, the devices do not have to inform the processor about any of their statuses.
- It is the processor‘s responsibility to keep a check. This makes the USB simple and low cost.
- Whenever a new device is connected to the hub it is addressed as 0.
- Now at a regular interval the host computer polls all the hubs to get their status which lets the host know of I/O devices that are either detached from the system or are attached to the system.
- When the host becomes aware of the new device it gets to know about the capabilities of the device by reading the information present in the special memory of the device‘s USB interface.
- So that the host can use the appropriate device driver to communicate with the device.
- The host then assigns an address to this new device, this address is written to the register of the device interface register.
- With this mechanism, USB serves plug-and-play capability.

- The **plug and play** feature let the host recognize the existence of the new I/O device automatically when the device is plugged in.
- The host software determines the capabilities of the I/O devices and if it has any special requirement.
- The USB is **hot-pluggable** which means the I/O device can be attached or removed from the host system without performing any restart or shutdown.
- That means your system can keep running while the I/O device is plugged or removed.

Isochronous Traffic on USB

- USB also supports the isochronous traffic where the data is transferred at a fixed timed interval, where the intervals are regular and of very short time.
- The isochronous data transmission is comparatively faster than asynchronous and synchronous data transfer.
- To accommodate the isochronous traffic, the root hub sends a sequence of bits over the USB tree this indicates the start of isochronous data and after this sequence of bits, the actual data is transmitted.
- As USB support the isochronous data transmission the audio-video signals are transferred in a precisely timely manner.

Types of USB Connectors

- The USB has different types of ports and connectors.
- Usually, the upstream port and connector are always the USB type A the downstream port and connector differ depending on the type of device connected.

USB Type A:

- This is the standard connector that can be found at one end of the USB cable and is also known as upstream.
- It has a flat structure and has four connecting lines as you can see in the image below.

USB Type B:

- This is an older standard cable and was used to connect the peripheral devices also referred to as downstream.
- It is approximately a square as you can see in the image below.
- This is now been replaced by the newer versions.

Mini USB:

- This type of USB is compatible with mobile devices.
- This type of USB is now superseded your micro-USB still you will get it on some devices.

Micro USB:

- This type of USB is found on newer mobile devices. It has a compact 5 pin design.

USB Type C:

- This type of USB is used for transferring both data and power to the attached peripheral or I/O device.
- The USB C does not have a fixed orientation as it is reversible i.e. you can plug it upside down or in reverse.

USB 3.0 Micro B:

- This USB is a super speed USB. This USB is used for a device that requires high-speed data transfer.
- You can find this kind of USB on portable hard drives.

Electrical Characteristics of USB

- The standard USB has four lines of connection among which two carry power (one carry +5 V and one is for Ground).
- The other two lines of connection are for data transfer.
- USB also supply power to connected I/O device that requires very low power.
- Transferring of data over USB can be divided into two categories i.e., transferring data at low speed and transferring data at high speed.
- The low-speed transmission uses **single-ended signaling** where varying high voltage is transmitted over one of the two data lines to represent the signal bit 0 or 1.
- The other data line is connected to the reference voltage i.e., ground.
- The single-ended signaling is prone to noise.
- The high-speed data transmission uses the approach **differential signaling**.
- Here, the signal is transmitted over the two data lines that are twisted together.
- Here both the data lines are involved in carrying the signal no ground wire is required.
- The differential signaling is not prone to noise and uses low voltages as compared to single-ended transmission.
- So, this is all about the universal serial bus which connects the I/O devices to the host computer. We have seen how it works and how many versions of USB we have.

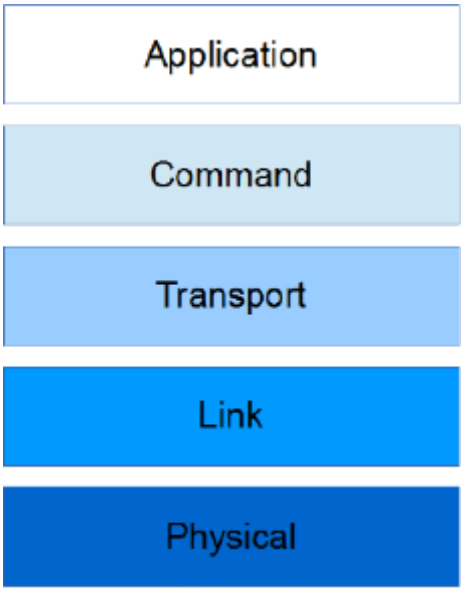
SATA

15. Explain about SATA.

- Serial ATA is a peripheral interface created in 2003 to replace Parallel ATA, also known as IDE.

- Hard drive speeds were getting faster, and would soon outpace the capabilities of the older standard—the fastest PATA speed achieved was 133MB/s, while SATA began at 150MB/s and was designed with future performance in mind.
- Also, newer silicon technologies used lower voltages than PATA's 5V minimum.
- The ribbon cables used for PATA were also a problem; they were wide and blocked air flow, had a short maximum length restriction, and required many pins and signal lines.
- SATA has a number of features that make it superior to Parallel ATA. The signaling voltages are low and the cables and connectors are very small.
- SATA has outpaced hard drive performance, so the interface is not a bottleneck in a system.
- It also has a number of new features, including hot-plug support. SATA is a point-to-point architecture, where each SATA link contains only two devices: a SATA host (typically a computer) and the storage device.
- If a system requires multiple storage devices, each SATA link is maintained separately. This simplifies the protocol and allows each storage device to utilize the full capabilities of the bus simultaneously, unlike in the PATA architecture where the bus is shared.
- To ease the transition to the new standard, SATA maintains backward compatibility with PATA.
- To do this, the Host Bus Adapter (HBA) maintains a set of shadow registers that mimic the registers used by PATA. The disk also maintains a set of these registers.
- When a register value is changed, the register set is sent across the serial line to keep both sets of registers synchronized.
- This allows for the software drivers to be agnostic about the interface being used.
- Serial ATA is a peripheral interface created in 2003 to replace Parallel ATA, also known as IDE.
- Hard drive speeds were getting faster, and would soon outpace the capabilities of the older standard—the fastest PATA speed achieved was 133MB/s, while SATA began at 150MB/s and was designed with future performance in mind.
- Also, newer silicon technologies used lower voltages than PATA's 5V minimum. The ribbon cables used for PATA were also a problem; they were wide and blocked air flow, had a short maximum length restriction, and required many pins and signal lines.
- SATA has a number of features that make it superior to Parallel ATA.
- The signaling voltages are low and the cables and connectors are very small. SATA has outpaced hard drive performance, so the interface is not a bottleneck in a system. It also has a number of new features, including hot-plug support.
- SATA is a point-to-point architecture, where each SATA link contains only two devices: a SATA host (typically a computer) and the storage device.
- If a system requires multiple storage devices, each SATA link is maintained separately. This simplifies the protocol and allows each storage device to utilize the full capabilities of the bus simultaneously, unlike in the PATA architecture where the bus is shared.

- To ease the transition to the new standard, SATA maintains backward compatibility with PATA.
- To do this, the Host Bus Adapter (HBA) maintains a set of shadow registers that mimic the registers used by PATA.
- The disk also maintains a set of these registers. When a register value is changed, the register set is sent across the serial line to keep both sets of registers synchronized.
- This allows for the software drivers to be agnostic about the interface being used.



Physical Layer:

- The physical layer is the lowest layer of the SATA protocol stack. It handles the electrical signal being sent across the cable.
- The physical layer also handles some other important aspects, such as resets and speed negotiation.
- SATA uses low-voltage differential signaling (LVDS). Instead of sending 1's and 0's relative to a common ground, the data being sent is based on the difference in voltage between two conductors sending data.
- In other words, there is a TX+ and a TX- signal. A logic 1 corresponds to a high TX+ and a low TX-; and vice versa for a logic 0. SATA uses a ±125mV voltage swing.

Link Layer

- The link layer is the next layer and is directly above the physical layer. This layer is responsible for encapsulating data payloads and manages the protocol for sending and receiving them.
- A data payload that is sent is called a Frame Information Structure (FIS). The link layer also provides some other services for ensuring data integrity, handling flow control, and reducing EMI.
- The host and the disk each have their own transmit pair in a SATA cable, and theoretically data could be sent in both directions simultaneously.
- However, this does not occur. Instead, the receiver sends ―backchannel‖ information to the sender that indicates the status of the transfer in progress.

- For instance, if an error were to be detected mid- transmission, such as a disparity error, the receiver could notify the sender of this.

Transport Layer

- The transport layer is responsible for constructing, delivering, and receiving Frame Information Structures.
- It defines the format of each FIS and the valid sequence of FISes that can exchanged.
- The first byte of each FIS defines the type. The second byte contains type- dependent control fields.
- The following table lists some of the types of FISes that are defined, and the value of their type field.

FIS Type	Type Value
Register – Host to Device	0x27
Register – Device to Host	0x34
Data	0x46
DMA Activate	0x39

Table 4- FIS Types

2 Marks
Question Bank

1. What is Memory?

- Memory is a device used to store the data and instructions required for any operation.

2. What is the secondary memory?

- Secondary memory is where programs and data are kept on a long-term basis. Common secondary storage devices are the hard disk and optical disks. The hard disk has enormous storage capacity compared to main memory. The hard disk is usually contained inside the case of a computer.

3. What are some examples of secondary storage device?

- Some other examples of secondary storage technologies are flash memory (e.g. USB flash drives or keys), floppy disks, magnetic tape, paper tape, punched cards, standalone RAM disks, and Iomega Zip drives.

4. What are the characteristics of a secondary storage device?

- Characteristics of a secondary storage devices are,
- Capacity
- Speed
- Portability
- Durability
- Reliability

5. What are the three main categories of secondary storage?

Currently the most common forms of secondary storage device are:

- Floppy disks
- Hard disks
- Optical Disks
- Magnetic Tapes
- Solid State Devices

6. What is Bandwidth?

- The maximum amount of information that can be transferred to or from the memory per unit time is called bandwidth.

7. Define a Cache.

- It is a small fast intermediate memory between the processor and the main memory.

8. What is Cache Memory?

- Cache memory is a very high speed memory that is placed between the CPU and primary or main memory.

- It is used to reduce the average time to access data from the main memory.
- The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations.
- Most CPUs have different independent caches, including instruction and data.

9. Give the mapping techniques of cache.

The three different types of mapping techniques used for the purpose of cache memory are as follow,

- ✓ Direct Mapping
- ✓ Associative Mapping
- ✓ Set-Associative Mapping

10. What is Write Stall?

- When the processor must wait for writes to complete during write through, the processor caches is said to write stall.

11. Define Mapping Functions.

- The correspondence of memory blocks in cache with the memory blocks in the main memory is defined as mapping functions.

12. What is Address Translation?

- The conversion of virtual address to physical address is termed as address translation.

13. What is the transfer time?

- The time it takes to transmit or move data from one place to another.
- It is the time interval between starting the transfer and the completion of the transfer.

(Or)

- Transfer time is the time it takes to transfer a block of bits, typically a sector under the read / write head.

14. What is latency and seek time?

- **Seek Time** is measured defines the amount of time it takes a hard drive's read/write head to find the physical location of a piece of data on the disk.
- **Latency** is the average time for the sector being accessed to rotate into position under a head, after a completed seek.

15. What is the clock cycle time?

- The speed of a computer processor, or CPU, is determined by the clock cycle, which is the amount of time between two pulses of an oscillator.
- Computer processors can execute one or more instructions per clock cycle, depending on the type of processor.

16. What is Access Time?

- The time a program or device takes to locate a single piece of information and make it available to the computer for processing. *DRAM (dynamic random access memory)* chips for personal computers have access times of 50 to 150 nanoseconds (billionths of a second).

17. What is meant by disk fragmentation?

- Fragmentation refers to the condition of a disk in which files are divided into pieces scattered around the disk.
- It occurs naturally when you use a disk frequently, creating, deleting, and modifying files.
- At some point, the operating system needs to store parts of a file in noncontiguous clusters.

18. What is the average access time for a hard disk?

- Disk access times are measured in milliseconds (thousandths of a second), often abbreviated as ms.
- Fast hard disk drivesfor personal computers boast access times of about 9 to 15 milliseconds.
- Note that this is about 200 times slower than average DRAM.

19. What is rotational latency time?

- The amount of time it takes for the desired sector of a disk (i.e., the sector from which data is to be read or written) to rotate under the read-write heads of the disk drive.
- It is also called as rotational delay.

20. What is meant by disk latency?

- Disk latency refers to the time delay between a request for data and the return of the data. It sounds like a simple thing, but this time can be critical to the performance of a system.

21. Define Page Fault.

- If the processor access for the particular page in main memory and if the page is not present there then it is known as page fault.

22. Define a Cache Unit.

- When the CPU refers to memory and finds a required word in cache it is termed as cache hit.

23. Define Hit Ratio.(Nov/Dec 2019)Nov/Dec 2021

- The ratio of the number of hits divided by the total CPU references to memory is the hit ratio.

24. Define a Miss.Nov/Dec 2021

- When the CPU refers to memory and if the required word is not found in cache it is termed as miss.

25. What is meant by memory stall cycles? (M 2016)

- The number of cycles during which the CPU is stalled waiting for a memory access is called memory stall cycles.

26. What is Miss Penalty?

- The number of stall cycles depends on both the number of misses and the cost per miss, which is called the miss penalty.

27. Write the formula to calculate average memory access time. (Or) Write the formula to measure average memory access time for memory hierarchy performance. (Nov / Dec 2018)

- Average memory access time = Hit time + Miss rate x Miss penalty

28. What is a miss in a cache? (Or) What does Cache Miss mean? (Or) Define Cache Miss. (Nov/Dec 2010, April/May 2018)

- Cache miss is a state where the data requested for processing by a component or application is not found in the cache memory.
- It causes execution delays by requiring the program or application to fetch the data from other cache levels or the main memory.

29. Define Cache Hit. (Or) What does Cache Hit mean? (April/May 2018)

- A cache hit is a state in which data requested for processing by a component or application is found in the cache memory.
- It is a faster means of delivering data to the processor, as the cache already contains the requested data.

30. Differentiate between Cache Miss and Cache Hit.

- The difference between the two is the data requested by a component or application in a cache memory being found or not.
- In a cache miss, the data is not found so the execution is delayed because the component or application tries to fetch the data from main memory or other cache levels.
- In a cache hit, the data is found in the cache memory making it faster to process.
- The **cache *hit*** is when you look something up in a cache and it *was* storing the item and is able to satisfy the query.

31. What is miss penalty for a cache?

- Cache is a small high-speed memory. Stores data from some frequently used addresses (of main memory). Processor loads data from M and copies intocache.
- This results in extra delay, called miss penalty.
- Hit ratio = percentage of memory accesses satisfied by the cache.

32. What is miss rate in cache?

- The fraction or percentage of accesses that result in a hit is called the hit rate.
- The fraction or percentage of accesses that result in a miss is called the miss rate.
- It follows that hit rate + miss rate = 1.0 (100%).
- The difference between lower level access time and cache access time is called the miss penalty.

33. What is hit time in cache?

- AMAT's three parameters hit time (or hit latency), miss rate, and miss penalty provide a quick analysis of memory systems.

- Hit latency (H) is the time to hit in thecache. Miss rate (MR) is the frequency of cache misses, while average miss penalty (AMP) is the cost of a cache miss in terms of time.

34. How is cache memory measured?

- The CPU cache is a piece of hardware which reduces the access time to the data in the memory by keeping some part of the frequently used data of the main memory in itself.
- It is smaller and faster than the main memory.

35. What is the memory cycle time?

- Cycle time is the time, usually measured in nanosecond s, between the start of one random access memory (RAM) access to the time when the next access can be started.
- Access time is sometimes used as a synonym (although IBM deprecates it).

36. What is the memory access time?

- Memory access time is how long it takes for a character in memory to be transferred to or from the CPU.
- In a PC or Mac, fast RAM chips have an access time of 70 nanoseconds (ns) or less.
- SDRAM chips have a burst mode that obtains the second and subsequent characters in 10 ns or less.

37. What is the data transfer rate?

- The speed with which data can be transmitted from one device to another.
- Data rates are often measured in megabits (million bits) or megabytes (million bytes) per second.
- These are usually abbreviated as Mbps and MBps, respectively. Another term for data transfer rate is throughput.

38. What does access time measure?

- The total time it takes the computer to read data from a storage device such as computer memory, hard drive, CD-ROM or other mechanism.
- Computer access time is commonly measured in nanoseconds or milliseconds and the lower the access the time the better.

39. List the method to improve the cache performance.

Improving the cache performance following methods are used:

- Reduce the miss rate.
- Reduce the miss penalty.
- Reduce the time to hit in the cache.

40. What is Split Transactions?

- With multiple masters, a bus can offer higher bandwidth by using packets, as opposed to holding the bus for the full transaction. This technique is called split transactions.

41. What is Cylinder?

- Cylinder is used to refer to all the tracks under the arms at a given points on all surfaces.

42. What is Synchronous Bus?

- Synchronous bus includes a clock in the control lines and a fixed protocol for sending address and data relative to the clock.

43. Explain difference between latency and throughput.

- Latency is defined as the time required processing a single instruction, while throughput is defined as the number of instructions processes per second.

44. What is called Pages?

- The address space is usually broken into fixed-size blocks, called pages. Each page resides either in main memory or on disk.

45. What are the techniques to reduce hit time?

The techniques to reduce hit time are:

- Small and simple cache: Direct mapped.
- Avoid address translation during indexing of the cache.
- Pipelined cache access.
- Trace cache.

46. What are the categories of cache miss? (April/May 2013) (Or) Point out one simple technique used to reduce each of the three "C" misses in cache memories. (Nov/Dec 2017)

- Categories of cache misses are,
  - ✓ Compulsory
  - ✓ Capacity
  - ✓ Conflict

47. How the conflicts misses are divided? (Nov/Dec 2016)

Four divisions of conflict misses are:

- **Eight way:** Conflict misses due to going from fully associative to eight way associative.
- **Four way:**Conflict misses due to going from eight way associative to four way associative.
- **Two way:** Conflict misses due to going from four way associative to two way associative.
- **One way:** Conflict misses due to going from two way associative to one way associative.

48. What is Sequence Recorded?

- The sequence recorded on the magnetic medics is a sector number, a gap, the information for that sector including error correction cede, a gap, the sector number of the next sector and so on.

49. Write the formula to calculate the CPU execution time.

- CPU execution time = (CPU clock cycles + Memory stall cycles) x Clock cycle time.

50. Write the formula to calculate the CPU time.

- CPU time = (CPU execution clock cycles + Memory stall clock cycles) x Clock cycle time.

51. What is RAID? (Nov/Dec 2011, April/May 2015, 2017)

- RAID stands forRedundant Array of Independent Disks.
- It is also called as redundant array of inexpensive disks.

- It is a way of storing the same data in different places on multiple hard disks.

(Or)

- **RAID** is a storage technology that combines multiple disk drive components into a logical unit for the purposes of data redundancy and performance improvement.
- Data is distributed across the drives in one of several ways, referred to as RAID levels, depending on the specific level of redundancy and performance required.

52. Explain the terms availability and dependability. (Nov/Dec 2017)

- Availability is a measure of the service accomplishment with respect to the alternation between the two states of accomplishment and interruption.
- Dependability is the quality of delivered service such that reliance can justifiable be placed on this service.

53. What are the differences and similarities between SCSI and IDE? (April/May 2017)

Parameters	IDE	SCSI
Cost	IDE is a much cheaper solution	SCSI is often more expensive to implement and support
Expansion	It allows 2 two devices per channel.	It is capable of supporting up to 7 or 15 devices.
Ease	IDE is commonly a much easier product to setup than SCSI.	Configuring SCSI can be more difficult for most users when compared to IDE.
CPU	IDE devices cannot communicate independently from the CPU.	SCSI devices can communicate independently from the CPU over the SCSI bus.

54. Why does DRAM generally have much larger capacities than SRAMs constructed in the same fabrication technology? (Nov/Dec 2016)

- DRAM bit cells require only two devices (capacitor and transistor) while SRAM bit cells typically requires six transistors.
- This makes the bit cells of the DRAM much smaller than the bit cells of the SRAM, allowing the DRAM to store more data in the same amount of chip space.

55. What are the measures of I/O performance? (Nov/Dec 2013)

I/O Performance Measures

- Measures used to quantify I/O performance attempt to measure a more diverse range of properties than the case of CPU performance.
- The traditional measures of performance namely response time and throughput also apply to I/O. I/O throughput is sometimes called I/O bandwidth and response time is sometimes called latency.

56. What are the types of storage devices? (Nov/Dec 2016, April/May 2017)

- Physical components or materials on which data is stored are called storage media.
- Hardware components that read/write to storage media are called storage devices. A floppy disk drive is a storage device.
- Two main categories of storage technology used today are magnetic storage and optical storage. Storage devices hold data, even when the computer is turned off. The physical material that actually holds data is called storage medium.

57. What do you mean by Memory Interleaving?

- Interleaved memory is a design made to compensate for the relatively slow speed of dynamic random-access memory (DRAM) or core memory, by spreading memory addresses evenly across memory banks.

58. What are the factors responsible for the maximum I/O bus performance? (April/May 2005)

Factors to be considered for maximum I/O bus performance are:

- Latency &Bandwidth

59. What are the two major advantages and disadvantages of the bus? (May/June 2007)

Advantages:

- It is easy to set-up and extend bus network.
- Cable length required for this topology is the least compared to other networks. Bus topology costs very less.
- Bus topology costs very less.

Disadvantages:

- There is a limit on central cable length and number of nodes that can be connected
- Dependency on central cable in this topology has its disadvantages. If the main cable (i.e. bus) encounters some problem, whole network breaks down.

60. Is the RISC processor is necessary? Why? (May/June 2007)

- Although RISC was indeed able to scale up in performance quite quickly and cheaply and it is necessary for building block of high performance parallel processors.

61. Define the terms cache miss and cache hit. (Nov/Dec 2011, May/June 2013)

- A *cache miss*, generally, is when something is looked up in the cache and is not found.
- The cache did not contain the item being looked up. The *cache hit* is when you look something up in a cache and it *was* storing the item and is able to satisfy the query.

62. Compare software and hardware RAID.

Software RAID	Hardware RAID
1. A simple way to describe software RAID is that the RAID task runs on the CPU of your computer system. 2. It is implemented in Pure Software model –	1. A hardware RAID solution has its own processor and memory to run the RAID application. In this implementation, the RAID system is an independent small computer system

Operating System Software and hybrid model-Assisted Software RAID.	dedicated to the RAID application, offloading this task from the host system. 2. Hardware RAID can be implemented in a variety of ways: • as a discrete RAID Controller Card, or • as integrated hardware based on RAID-on-Chip technology

63. Explain the need to implement memory as a hierarchy. (April/May 2017)

- A "memory hierarchy" in computer storage distinguishes each level in the "hierarchy" by response time.
- Each level of the hierarchy is of higher speed and lower latency, and is of smaller size, than lower levels.
- The levels in the memory hierarchy not only differ in speed and cost. They are performing different roles.

64. What is a register in memory?

- A register is a very small amount of very fast memory that is built into the CPU (central processing unit) in order to speed up its operations by providing quick access to commonly used values.
- Registers are the top of the memory hierarchy and are the fastest way for the system to manipulate data.

65. What is the storage hierarchy?

- A storage device hierarchy consists of a group of storage devices that have different costs for storing data, different amounts of data stored, and different speeds of accessing the data.
- Level 0, including DFSMShsm-managed storage devices at the highest level of the hierarchy, contains data directly accessible to you.

66. What is Cache Optimization?

- The idea behind this approach is to hide both the low main memory bandwidth and the latency of main memory accesses which is slow in contrast to the floating-point performance of the CPUs.

67. List the six basic optimization techniques of cache. (Nov/Dec 2016) (Or) List the basic six cache optimizations for improving cache performance. (Nov / Dec 2018)

- The six basic optimization techniques of cache are,
  - ✓ Larger block size to reduce miss rate
  - ✓ Bigger caches to reduce miss rate
  - ✓ Higher associativity to reduce miss rate
  - ✓ Multilevel caches to reduce miss penalty
  - ✓ Giving priority to read misses over writes to reduce miss penalty
  - ✓ Avoiding address translation during indexing of the cache to reduce hit time

68. Difference between Volatile and Non-volatile Memory. (Or) Outline the difference between volatile and non-volatile memory. (April/May 2018)

Volatile Memory (RAM)	Non-volatile Memory (ROM)
→ It is a volatile memory. → Contents are stored temporarily. → Cost is very high. → Small storage capacity. → Processing speed is high.	→ It is a non-volatile memory. → Contents are stored permanently. → Cost Effective. → High storage capacity. → Processing speed is low.

69. What is a Cache Performance?

- Cache Performance - Average memory access time is a useful measure to evaluate the performance of a memory-hierarchy configuration.

70. What is hit and miss ratio?

- The hit ratio is the fraction of accesses which are a hit.
- The miss ratio is the fraction of accesses which are a miss. It holds that. miss rate = 1 − hit rate.
- The (hit/miss) latency (AKA access time) is the time it takes to fetch the data in case of a hit/miss.

71. Differentiate between SRAM and DRAM.

- SRAM (Static RAM) and DRAM (Dynamic RAM) holds data but in a different ways.
- DRAM requires the data to be refreshed periodically in order to retain the data.
- SRAM does not need to be refreshed as the transistors inside would continue to hold the data as long as the power supply is not cut off.
- DRAM memory slower and less desirable than SRAM.

72. Differentiate between throughput and response time. [May 2019]

Throughput Time	Response Time
1. This is the time difference between submission of a request until the response begins to be received. 2.The response time should be as low as possible so that a large number of interactive users receive an acceptable response time.	1. The number of processes that are completed per unit time is called the throughput. 2. It is desirable to maximize CPU utilization and throughput and to minimize turnaround time and response time.

73. Suppose a processor sends 80 disk I/Os per second, these requests are exponentially distributed, and the average service time of an older disk is 25 ms. Nov/Dec 2020.

Answer the following questions:

1. On average, how utilized is the disk?

2. What is the average time spent in the queue?

3.What is the average response time for a disk request, including the queuing time and disk service time?

Average number of Arriving tasks / second is 80 ms.

Average disk time to service a task is 25ms = (0 .025 sec).

The server utilization is then

Service utilization = Arrival rate x Time $_{\text{Server}}$

= 80 x 0.025 = 2.0

Since the service times are exponentially distributed, we can use the simplifiedformula for the average time spent waiting in line:

$$Time_{Queue} = Time_{Server} \text{ X } \frac{server\ Utilization}{(1 - Server\ Utilization)}$$

$$= 25\ ms \text{ X } \frac{2}{1-2}$$

= 50 ms (consider +ve value)

The average response time is

Time system = Time $_{\text{queue}}$ + Time $_{\text{Server}}$

= 50 ms + 25 ms

= 75 ms

Thus, on average we spend 75% of our time waiting in the queue!

74. What is IO mapped input output?

- A memory reference instruction activated the READ M (or) WRITE M control line anddoes not affect the IO device. Separate IO instruction is required to activate the READ IO and WRITE IO lines,which cause a word to be transferred between the address port and the CPU.
- The memory and IO address space are kept separate.

75. Specify the three types of the DMA transfer techniques?

- Single transfer mode (cyclestealing mode)
- Block Transfer Mode (Burst Mode)
- Demand Transfer Mode
- Cascade Mode

76. Name any three of the standard I/O interface.

- SCSI (small computer system interface), bus standards
- Back plane bus standards
- IEEE 796 bus (multibus signals)
- NUBUS & IEEE 488 bus standard

77. What is an I/O channel?

- An I/O channel is actually a special purpose processor; also called peripheral processor.
- The main processor initiates a transfer by passing the required information in the input output channel. The channel then takes over and controls the actual transfer of data.

78. Why program controlled I/O is unsuitable for high-speed data transfer?

- In program controlled I/O considerable overhead is incurred. Because several program instructions have to be executed for each data word transferred between the external devices and MM.
- Many high speed peripheral; devices have a synchronous modes of operation. That is data transfer is controlled by a clock of fixed frequency, independent of the CPU.

79. What is the function of I/O interface?

- The function is to coordinate the transfer of data between the CPU and external devices.

80. Name some of the IO devices.

- Video terminals
- Video displays
- Alphanumeric displays
- Graphics displays
- Flat panel displays
- Printers
- Plotters

81. Define interface.

- The word interface refers to the boundary between two circuits or devices

82. What is programmed I/O?

- Data transfer to and from peripherals may be handled using this mode. Programmed I/O operations are the result of I/O instructions written in the computer program.

83. What are the limitations of programmed I/O? (Nov / Dec 2011)

The main limitation of programmed I/O and interrupt driven I/O is given below:

Programmed I/O

- It used only in some low-end microcomputers.
- It has single input and single output instruction.
- Each instructions selects one I/O device (by number) and transfers a single character (byte)
- Example: microprocessor controlled video terminal.
- Four registers: input status and character, output status and character.

Interrupt-driven I/O

- Primary disadvantage of programmed I/O is that CPU spends most of its time in a tight loop waiting for the device to become ready. This is called busy waiting.
- With interrupt-driven I/O, the CPU starts the device and tells it to generate an interrupt when it is finished.
- Done by setting interrupt-enable bit in status register.
- Still requires an interrupt for every character read or written.
- Interrupting a running process is an expensive business (requires saving context).

- Requires extra hardware (DMA controller chip).

84. Differentiate Programmed I/O and Interrupt I/O. (Nov / Dec 2014)

Programmed I/O	Interrupt I/O
In programmed I/O, processor has to check each I/O device in sequence and in effect ‗ask' each one if it needs communication with the processor. This checking is achieved by continuous polling cycle and hence processor cannot execute other instructions in sequence.	External asynchronous input is used to tell the processor that I/O device needs its service and hence processor does not have to check whether I/O device needs it service or not.
During polling processor is busy and therefore, has serious and decremental effect on system throughput.	In Interrupt driven I/O, the processor is allowed to execute its instructions in sequence and only stop to service I/O device when it is told to do so by the device itself. This increases system throughput.
It is implemented without interrupt hardware support.	It is implemented using interrupt hardware support.
It does not depend on interrupt status.	Interrupt must be enabled to process Interrupt driven I/O.
It does not need initialization of stack.	It needs initialization of stack.
System throughput decreases as number of I/O devices connected in the system increases.	System throughput does not depend on number of I/O devices connected in the system.

85. Differentiate between memory-mapped I/O and I/O mapped I/O. (Apr / May 2011)

S.No.	Parameter	Memory-mapped I/O	I/O-mapped I/O
1.	Address space	Memory and I/O devices share the entire address space	Memory and I/O devices have separate address space
2.	Hardware	No additional hardware required	Additional hardware required
3.	Implementation	Easy to implement	Difficult to implement
4.	Address	Same address cannot be used to refer both memory and I/O device.	Same address can be used to refer both memory and I/O device.
5.	Control lines	Memory control lines are used to control I/O devices.	Different set of control lines are used to control memory and I/O.
6.	Control lines used	The control lines are: READ, WRITE	The control lines are: READ M, WRITE M, READ I/O, WRITE I/O

86. What is DMA?

- A special control unit may be provided to enable transfer a block of data directly betweenan external device and memory without contiguous intervention by the CPU. This approach is called DMA. (OR)

- A direct memory access (DMA) is an operation in which data is copied (transported) from one resource to another resource in a computer system without the involvement of the CPU.

87. What are MAR and MBR?

- MAR – Memory address register holds the address of the data to be transferred.
- MBR – Memory buffer register contains the data to be transferred to or from the main memory.

88. Define bus arbitration. List out its types. (May / June 2009)

Bus Arbitration: It is the process by which the next device to become the bus master is selected and the bus mastership is transferred to it.

Types: There are 2 approaches to bus arbitration. They are,

- Centralized arbitration (A single bus arbiter performs arbitration)
- Distributed arbitration (all devices participate in the selection of next bus master).

89. Define memory interleaving. (Apr / May 2017)

Memory interleaving is the technique used to increase the throughput. The core idea is to split the memory system into independent banks, which can answer read or write requests independents in parallel.

90. What is an interrupt?

- An interrupt is an event that causes the execution of one program to be suspended and another program to be executed.

91. What are the uses of interrupts?

- Recovery from errors
- Debugging
- Communication between programs
- Use of interrupts in operating system

92. Define vectored interrupts.

- In order to reduce the overhead involved in the polling process, a device requesting an interrupt may identify itself directly to the CPU. Then, the CPU can immediately start executing the corresponding interrupt-service routine. The term vectored interrupts refers to all interrupt handling schemes base on this approach.

93. What are the steps taken when an interrupt occurs?

Source of the interrupt

- The memory address of the required ISP
- The program counter & CPU information saved in subroutine
- Transfer control back to the interrupted program

94. Summarize the sequence of events involved in handling an interrupt request from a single device. (Apr / May 2017) Write the sequence of operations carried out by a processor when interrupted by a peripheral device connected to it. (Apr/May 2018)

Let us summarize the sequence of events involved in handling an interrupt request from a single device.

Assuming that interrupts are enabled, the following is a typical scenario.

1. The device raises an interrupt request.

2. The processor interrupts the program currently being executed.

3. Interrupts are disabled by changing the control bits in the PS (except in the case of edge-triggered interrupts).

4. The device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal. 5. The action requested by the interrupt is performed by the interrupt-service routine. 6. Interrupts are enabled and execution of the interrupted program is resumed.

95. Point out how DMA can improve I/O speed. (May / June 2015)

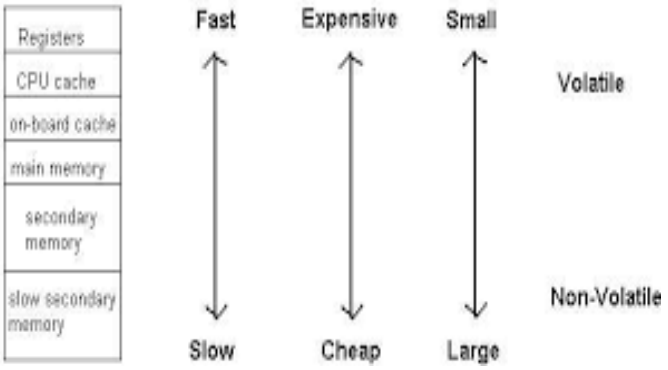
DMA module controls exchange of data between main memory and the I/O device. Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred. In this case DMA improves the I/O speed of the system.

96. Define IO Processor.

The IOP attaches to the system I/O bus and one or more input/output adapters (IOAs). The IOP processes instructions from the system and works with the IOAs to control the I/O devices.

97. Draw the Memory hierarchy in a typical computer system. (Nov/Dec 2018)(Or)

Draw the basic structure of a memory hierarchy. (Apr/May 2019)



98. What is meant by Memory-mapped I/O? (Nov/Dec 2018)

Memory mapped I/O is a way to exchange data and instructions between a CPU and peripheral devices attached to it. **Memory mapped** IO is one where the processor and the IO device share the same **memory**location(**memory**),i.e.,the processor and IO devices are**mapped** using the **memory** address.

99. What is the use of DMA Controller? (Apr/May 2018)

- A special control unit may be provided to allow the transfer of large block of data at high speed directly between the external device and main memory, without continuous intervention by the processor. This approach is called DMA.
- DMA transfers are performed by a control circuit called the **DMA Controller.**

100. Define Bus Structures.

In computer **architecture**, a **bus** (a contraction of the Latin omnibus) is a communication system that transfers data between components inside a computer, or between computers. This expression covers all related hardware components (wire, optical fiber, etc.) and software, including communication protocols.

101. What is the meaning of USB?(Nov/Dec 2019)

A Universal Serial Bus (**USB**) is a common interface that enables communication between devices and a host controller such as a personal computer (PC). It connects peripheral devices such as digital cameras, mice, keyboards, printers, scanners, media devices, external hard drives and flash drives.

102. what is baud rate?Nov/Dec 2020.

The baud rate is the rate at which information is transferred in a communication channel. Baud rate is commonly used when discussing electronics that use serial communication. In the serial port context, "9600 baud" means that the serial port is capable of transferring a maximum of 9600 bits per second.

At baud rates above 76,800, the cable length will need to be reduced. The higher the baud rate, the more sensitive the cable becomes to the quality of installation, due to how much of the wire is untwisted around each device.

103. In memory organization,what is temporal locality? Nov/Dec 2021

Temporal locality means current data or instruction that is being fetched may be needed soon. When CPU accesses the current main memory location for reading required data or instruction, it also gets stored in the cache memory which is based on the fact that same data or instruction may be needed in near future**.**

104. How many total bits are required for a direct-mapped cache with 16KB of data and 4 word blocks, assuming a 32-bit address? (Apr/May 2019)

16 KB = 16384 (2^{14}) bytes = 4096 (2^{12}) words

Block size of 4 (2^2) words = 16 bytes (2^4) = 1024

(2^{10}) blocks with 4 x 32 = 128 bits of data

So, n = 10

m = 2

$= 2^{10}$x (4 x 32 + (32 –10 –2 - 2) + 1)

$= 2^{10}$x 147

= 18.4 KB

105. Define SATA.

Serial ATA is a peripheral interface created in 2003 to replace Parallel ATA, also known as IDE. Hard drive speeds were getting faster, and would soon outpace the capabilities of the older standard—the

fastest PATA speed achieved was 133MB/s, while SATA began at 150MB/s and was designed with future performance in mind.

106. Explain about USB.

- The **universal serial bus (USB)** is a standard interface for connecting a wide range of devices to the computer such as keyboard, mouse, smart phones, speakers, cameras etc.
- The USB was introduced for commercial use in the year 1995 at that time it has a data transfer speed of 12 megabits/s.