CMR INSTITUTE OF TECHNOLOGY: HYDERABAD

UGC AUTONOMOUS

B.Tech. III – Semester- I - Mid Term Examinations Key Paper – November – 2023

Digital Logic Design & Computer Organization

(Department of CSE/CSE(AI&ML)/CSE(DS)/AI&ML/CSE(CS))

Note:
1. This question paper contains two parts A and B.
2. Part A is compulsory which carries 5 marks. Answer all questions in Part A.
3. Part B consists of 5 questions. Answer all 5 questions. Each question carries 5 marks
4. Illustrate your answers with NEAT sketches wherever necessary.

PART-A **5 x 1M=5M**

1. a. What is Digital System?

A. A digital system is a system which deals with discrete signal. The input and output of this system is two binary value which is 0 and 1. Examples of digital systems are mobile phones, radio, megaphones and many more.

b. Find the subtraction using 1's complement and 2's complement methods.

(i)11010 – 10011 (ii)11000 – 1011

A. i. 11010 – 10011

2's complement of 10011 is 00110 . Hence

Minued - 11010

2's complement of subtrahend - 00110

Result of addition - 11001

Hence the required result is – 11001.

ii. 11000 – 1011

2's complement of 01011 is 10101 . Hence

Minued - 11000

2's complement of subtrahend - 10101

Result of addition - 101101

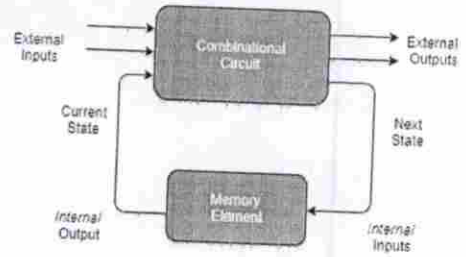
Hence the required result is – 101101.

c. Define: Encoder, Decoder.

A. **Encoder :** An encoder is a device, circuit, or program that converts digital data into an analogue signal.

Decoder: A decoder is a device, circuit, or program that converts an analogue signal into digital data.

d. Draw the block diagram of Sequential Circuit.



A. Figure: Sequential Circuit

e. What is meant by Instruction Code?

A. An instruction code is a group of bits that instruct the computer to perform a specific operation. The operation code of an instruction is a group of bits that define operations such as addition, subtraction, shift, complement, etc.

PART-B

5 x 5M=25M

2. Define: Digital Logic Gates. Explain the different types of logic gates with truth table, logic circuit diagram.

A. The building blocks of a digital circuit are logic gates, which execute numerous logical operations that are required by any digital circuit. These can take two or more inputs but only produce one output.

Types of Logic Gates

A logic gate is a digital gate that allows data to be transferred. Logic gates, use logic to determine whether or not to pass a signal. Logic gates, on the other hand, govern the flow of information based on a set of rules. The following types of logic gates are commonly used:

1. AND
2. OR
3. NOT
4. NOR
5. NAND
6. XOR
7. XNOR

Basic Logic Gates

AND Gate

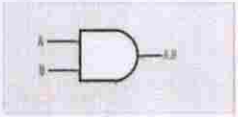
An AND gate has a single output and two or more inputs.

1. When all of the inputs are 1, the output of this gate is 1.
2. The AND gate's Boolean logic is **Y=A.B** if there are two inputs A and B.

An AND gate's symbol and truth table are as follows:

Input		Output
A	B	A AND B
0	0	0
0	1	0

1	0	0
1	1	1

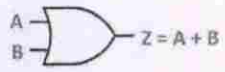


Therefore, in And gate, the output is high when all the inputs are high.

OR Gate
Two or more inputs and one output can be used in an OR gate.

1. The logic of this gate is that if at least one of the inputs is 1, the output will be 1.
2. The OR gate's output will be given by the following mathematical procedure if there are two inputs A and B: Y=A+B

Input		Output
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

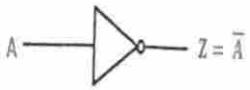


Therefore, in the OR gate, the output is high when any of the inputs is high.

NOT Gate
The NOT gate is a basic one-input, one-output gate.

1. When the input is 1, the output is 0, and vice versa. A NOT gate is sometimes called an inverter because of its feature.
2. If there is only one input A, the output may be calculated using the Boolean equation Y=A'.

Input	Output
A	Not A
0	1
1	0



A NOT gate, as its truth table shows, reverses the input signal.

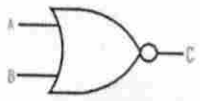
Universal Logic Gates

NOR Gate

A NOR gate, sometimes known as a "NOT-OR" gate, consists of an OR gate followed by a NOT gate.

1. This gate's output is 1 only when all of its inputs are 0. Alternatively, when all of the inputs are low, the output is high.
2. The Boolean statement for the NOR gate is Y=(A+B)' if there are two inputs A and B.

Input		Output
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0



By comparing the truth tables, we can observe that the outputs of the NOR gate are the polar opposite of those of an OR gate. The NOR gate is sometimes known as a universal gate since it may be used to implement the OR, AND, and NOT gates.

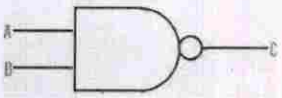
NAND Gate

A NAND gate, sometimes known as a 'NOT-AND' gate, is essentially a Not gate followed by an AND gate.

1. This gate's output is 0 only if none of the inputs is 0. Alternatively, when all of the inputs are not high and at least one is low, the output is high.
2. If there are two inputs A and B, the Boolean expression for the NAND gate is Y=(A.B)'

Input		Output
A	B	A NAND B
0	0	1
0	1	1

1	0	1
1	1	0



By comparing their truth tables, we can observe that their outputs are the polar opposite of an AND gate. The NAND gate is known as a universal gate because it may be used to implement the AND, OR, and NOT gates.

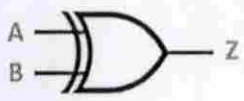
Other Logic Gates

XOR Gate

The Exclusive-OR or 'Ex-OR' gate is a digital logic gate that accepts more than two inputs but only outputs one value.

1. If any of the inputs is 'High,' the output of the XOR Gate is 'High.' If both inputs are 'High,' the output is 'Low.' If both inputs are 'Low,' the output is 'Low.'
2. The Boolean equation for the XOR gate is Y=A'.B+A.B' if there are two inputs A and B.

Input		Output
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



Its outputs are based on OR gate logic, as we can see from the truth table.

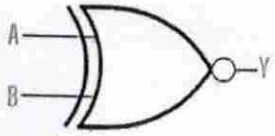
XNOR Gate

The Exclusive-NOR or 'EX-NOR' gate is a digital logic gate that accepts more than two inputs but only outputs one.

1. If both inputs are 'High,' the output of the XNOR Gate is 'High.' If both inputs are 'Low,' the output is 'High.' If one of the inputs is 'Low,' the output is 'Low.'
2. If there are two inputs A and B, then the XNOR gate's Boolean equation is: Y=A.B+A'B'.

Input		Output
A	B	A XNOR B

0	0	1
0	1	0
1	0	0
1	1	1



The truth table shows that its outputs are based on NOR gate logic.

3. State and prove commutative laws, associative laws and distributive law using logic gate and truth table.

A. Commutative Law

If any logical operation of two Boolean variables give the same result irrespective of the order of those two variables, then that logical operation is said to be **Commutative**. The logical OR & logical AND operations of two Boolean variables x & y are shown below

$$A + B = B + A$$

$$A.B = B.A$$

The symbol '+' indicates logical OR operation. Similarly, the symbol '.' indicates logical AND operation and it is optional to represent. Commutative law obeys for logical OR & logical AND operations.

A	B	A+B	A·B
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Table 1

B	A	B+A	B·A
0	0	0	0
1	0	1	0
0	1	1	0
1	1	1	1

Table 2

Associative Law

If a logical operation of any two Boolean variables is performed first and then the same operation is performed with the remaining variable gives the same result, then that logical operation is said to be **Associative**. The logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$$A + (B + C) = (A + B) + C = A + B + C$$

$$A(B.C) = (A.B)C = A \,.\, B \,.\, C$$

A	B	C	BC	A(BC)	ABC	B+C	A+(B+C)	A+B+C
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	1	1
0	1	0	0	0	0	1	1	1
0	1	1	1	0	0	1	1	1
1	0	0	0	0	0	0	1	1
1	0	1	0	0	0	1	1	1
1	1	0	0	0	0	1	1	1
1	1	1	1	1	1	1	1	1

Associative law obeys for logical OR & logical AND operations.

Distributive Law

If any logical operation can be distributed to all the terms present in the Boolean function, then that logical operation is said to be **Distributive**. The distribution of logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$$X(Y + Z) = X.Y + X.Z$$

$$X + (Y.Z) = (X + Y).(X + Z)$$

X	Y	Z	Y + Z	XY	XZ	X(Y + Z)	XY + XZ
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	0	1	1	1
1	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1

Distributive law obeys for logical OR and logical AND operations.

These are the Basic laws of Boolean algebra. We can verify these laws easily, by substituting the Boolean variables with '0' or '1'.

4. Explain in detail about floating –point representation with an example.

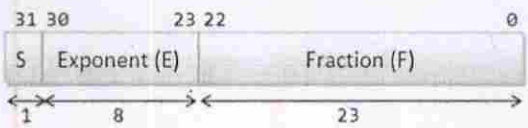
A. A floating-point number is typically expressed in the scientific notation, with a *fraction* (F), and an *exponent* (E) of a certain *radix* (r), in the form of F×r^E. Decimal numbers use radix of 10 (F×10^E); while binary numbers use radix of 2 (F×2^E).

Representation of floating point number is not unique. For example, the number 55.66 can be represented as 5.566×10^1, 0.5566×10^2, 0.05566×10^3, and so on. The fractional part can be *normalized*. In the normalized form, there is only a single non-zero digit before the radix point. For example, decimal number 123.4567 can be normalized as 1.234567×10^2; binary number 1010.1011B can be normalized as 1.0101011B×2^3.

IEEE-754 32-bit Single-Precision Floating-Point Numbers

In 32-bit single-precision floating-point representation:

- The most significant bit is the *sign bit* (S), with 0 for positive numbers and 1 for negative numbers.
- The following 8 bits represent *exponent* (E).
- The remaining 23 bits represents *fraction* (F).



32-bit Single-Precision Floating-point Number

Normalized Form

Let's illustrate with an example, suppose that the 32-bit pattern is 1 1000 0001 011 0000 0000 0000 0000 0000, with:

- S = 1
- E = 1000 0001
- F = 011 0000 0000 0000 0000 0000

In the *normalized form*, the actual fraction is normalized with an implicit leading 1 in the form of 1.F. In this example, the actual fraction is 1.011 0000 0000 0000 0000 0000

The sign bit represents the sign of the number, with S=0 for positive and S=1 for negative number. In this example with S=1, this is a negative number, i.e., -1.375D.

In normalized form, the actual exponent is E-127 (so-called excess-127 or bias-127). This is because we need to represent both positive and negative exponent. With an 8-bit E, ranging from 0 to 255, the excess-127 scheme could provide actual exponent of -127 to 128. In this example, E-127=129-127=2D.

The IEEE 754 standard specifies two precisions for floating-point numbers. Single precision numbers have 32 bits − 1 for the sign, 8 for the exponent, and 23 for the significand. The significand also includes an implied 1 to the left of its radix point.

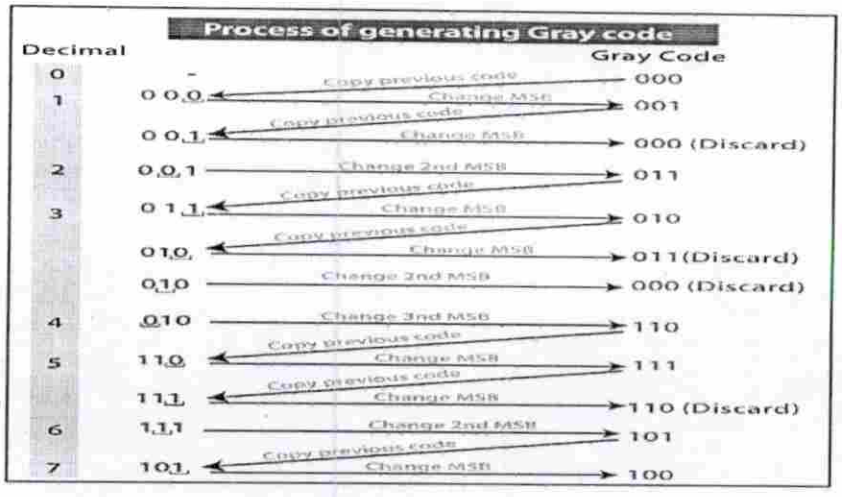
Example representation of +19.5 (=10011.1 or 1.00111×2^4 in binary) in single-precision format. The leading 1 is not included in the significand, its presence is implicit in this standard. Exponent 4 is represented as 1000 0011, or 131, due to the addition of the bias of 127.

Double precision numbers use 64 bits − 1 for the sign, 11 for the exponent, and 52 for the significand. As in single precision, the significand has an implied leading 1 for most values. The exponent has a bias of 1023 and a range value from -1022 to +1023. The smallest and largest exponent values, -1023 and +1024 are reserved for special numbers. Example representation of +19.5 in double precision format. The exponent is stored as 4 + bias, or 4 + 1023 = 1027, for this value.

5. Illustrate Gray code and its application.

A. The **Gray Code** is a sequence of binary number systems, which is also known as **reflected binary code**. The reason for calling this code as reflected binary code is the first N/2 values compared with those of the last N/2 values in reverse order. In this code, two consecutive values are differed by one bit of binary digits. Gray codes are used in the general sequence of hardware-generated binary numbers. These numbers cause ambiguities or errors when the transition from one number to its successive is done. This code simply solves this problem by changing only one bit when the transition is between numbers is done.

The gray code is a very light weighted code because it doesn't depend on the value of the digit specified by the position. This code is also called a cyclic variable code as the transition of one value to its successive value carries a change of one bit only.



Gray Code Table

Decimal Number	Binary Number	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

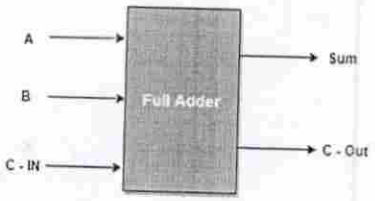
Application:

- The Gray code is used in the transmission of digital signals, as it minimizes the occurrence of errors.
- The Gray code is preferred over the straight binary code in angle-measuring devices. Use of the Gray code almost eliminates the possibility of an angle misread, which is likely if the angle is represented in straight binary. The cyclic property of gray code is a plus in this application.
- The Gray code is used for labelling the axes of Karnaugh maps, a graphical technique used for the minimization of Boolean expressions.
- The use of Gray codes to address program memory in computers minimizes power consumption. This is due to fewer address lines changing state with advances in the program counter.
- Gray codes are also very useful in genetic algorithms since mutations in the code allow for mostly incremental changes. However, occasionally a one-bit change can result in a big leap, thus leading to new properties.

6. Construct the logic circuit of a full adder and give its truth table.

A. Full Adder is the adder that adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM. The C-OUT is also known as the majority 1's detector, whose output goes high when more than one input is high. A

full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to another. we use a full adder because when a carry-in bit is available, another 1-bit adder must be used since a 1-bit half-adder does not take a carry-in bit. A 1-bit full adder adds three operands and generates 2-bit results.

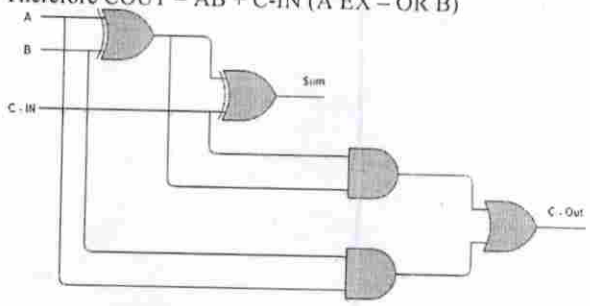


Full Adder Truth Table:

Inputs			Outputs	
A	B	C - IN	Sum	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

Logical Expression for SUM: = A' B' C-IN + A' B C-IN' + A B' C-IN' + A B C-IN = C-IN (A' B' + A B) + C-IN' (A' B + A B') = C-IN XOR (A XOR B) = (1,2,4,7)
Logical Expression for C-OUT: = A' B C-IN + A B' C-IN + A B C-IN' + A B C-IN = A B + B C-IN + A C-IN = (3,5,6,7)

Another form in which C-OUT can be implemented: = A B + A C-IN + B C-IN (A + A') = A B C-IN + A B + A C-IN + A' B C-IN = A B (1 +C-IN) + A C-IN + A' B C-IN = A B + A C-IN + A' B C-IN = A B + A C-IN (B + B') + A' B C-IN = A B C-IN + A B + A B' C-IN + A' B C-IN = A B (C-IN + 1) + A B' C-IN + A' B C-IN = A B + A B' C-IN + A' B C-IN = AB + C-IN (A' B + A B')
Therefore COUT = AB + C-IN (A EX – OR B)



7. Explain about J-K flip-flop with a characteristic and excitation table.

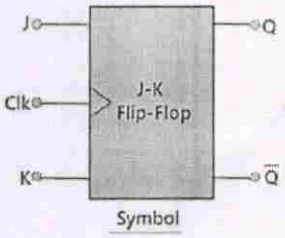
A. The SR Flip Flop or Set-Reset flip flop has lots of advantages. But, it has the following switching problems:

- When Set 'S' and Reset 'R' inputs are set to 0, this condition is always avoided.
- When the Set or Reset input changes their state while the enable input is 1, the incorrect latching action occurs.
- The JK Flip Flop removes these two drawbacks of SR Flip Flop.
- The JK flip flop is one of the most used flip flops in digital circuits. The JK flip flop is a universal flip flop having two inputs 'J' and 'K'. In SR flip flop, the 'S' and 'R' are the shortened abbreviated letters for Set and Reset, but J and K are not. The J and K are themselves autonomous letters which are chosen to distinguish the flip flop design from other types.
- The JK flip flop work in the same way as the SR flip flop work. The JK flip flop has 'J' and 'K' flip flop instead of 'S' and 'R'. The only difference between JK flip flop and SR flip flop is that when both inputs of SR flip flop is set to 1, the

circuit produces the invalid states as outputs, but in case of JK flip flop, there are no invalid states even if both 'J' and 'K' flip flops are set to 1.

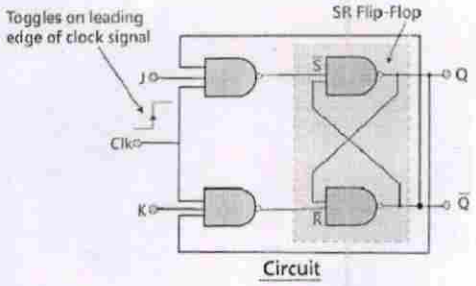
- The JK Flip Flop is a gated SR flip-flop having the addition of a clock input circuitry. The invalid or illegal output condition occurs when both of the inputs are set to 1 and are prevented by the addition of a clock input circuit. So, the JK flip-flop has four possible input combinations, i.e., 1, 0, "no change" and "toggle". The symbol of JK flip flop is the same as **SR Bistable Latch** except for the addition of a clock input.

Block Diagram:



Symbol

Circuit Diagram:



Circuit

In SR flip flop, both the inputs 'S' and 'R' are replaced by two inputs J and K. It means the J and K input equates to S and R, respectively.

The two 2-input AND gates are replaced by two 3-input NAND gates. The third input of each gate is connected to the outputs at Q and Q'. The cross-coupling of the SR flip-flop permits the previous invalid condition of (S = "1", R = "1") to be used to produce the "toggle action" as the two inputs are now interlocked.

If the circuit is "set", the J input is interrupted from the "0" position of Q' through the lower NAND gate. If the circuit is "RESET", K input is interrupted from 0 positions of Q through the upper NAND gate. Since Q and Q' are always different, we can use them to control the input. When both inputs 'J' and 'K' are set to 1, the JK toggles the flip flop as per the given truth table.

Truth Table:

Truth Table

J	K	CLK	Q
0	0	↑	Q_0 (no change)
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q}_0$ (toggles)

When both of the inputs of JK flip flop are set to 1 and clock input is also pulse "High" then from the SET state to a RESET state, the circuit will be toggled. The JK flip flop work as a T-type toggle flip flop when both of its inputs are set to 1.

The JK flip flop is an improved clocked SR flip flop. But it still suffers from the **"race"** problem. This problem occurs when the state of the output Q is changed before the clock input's timing pulse has time to go **"Off"**. We have to keep short timing plus period (T) for avoiding this period.

Characteristic table:

J	K	Q_n	Q_{n+1}	State
0	0	0	0	Q_n (Hold)
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggel
1	1	1	0	

Excitation table:

Q n	Q n+1	J	K
0 ➡	0	0	X
1 ➡	0	X	1
0 ➡	1	1	X
1 ➡	1	X	0

8. Distinguish combinational and sequential circuits.

A.

COMPARISON BETWEEN COMBINATIONAL AND SEQUENTIAL CKT

COMBINATIONAL CKT	SEQUENTIAL CKT
1. Its output is determined by the present values of its input only.	1. Its output is determined by the present values of the input as well as past values of the output.
2. It does not have a memory.	2. It has a memory and a feedback path from output to input.
3. It does not have a feedback path from output to input.	3. Its operation can be described by truth table and timing diagram.
4. Its operation can be described by a truth table.	4. It may or may not have clock signal but most sequential ckt have a clock signal.
5. It does not have a clock signal .	5. Its ckt is more complex than that of combinational ckts.
6. Its action does not depend on clock transition.	6. It is built using basic gates and combinational logic ckts.
7. Its ckt is simpler than that of sequential logic ckt.	7. Ex:- flip flops, counters,shift registers.
8. It built using basic gates i.e NOT,AND,OR,NAND,NOR,XOR,XNOR etc.	
9. Ex:- Adder,Subtractor, multiplexer,demux,decoder,comparator.	

9. Demonstrate the design steps of synchronous counters with suitable examples.

A. Design of Synchronous Counter

A synchronous counter is a type of counter in which all the flip flops are triggered simultaneously by the same clock pulse. The systematic procedure of designing a synchronous counter is explained below.

Step (1) – Determine the number of flip-flops required

Firstly, analyze the problem description and determine the number of flip-flops required to implement the synchronous counter. If n is the required number of flip flops, then the smallest value of n is such that the number of states N is less than or equal to 2^n.

Step (2) – Draw the state diagram

Secondly, draw the state diagram that shows all the possible states. The state diagram is basically a graphical way of representing the sequence of states through which the counter progresses. In the state diagram, we can also include the case in which the counter goes to a particular state from the invalid states on the next clock pulse.

Step (3) – Selection of flip-flops and excitation table

In the third step, select a particular type of flip flop to be used to implement the counter and draw its excitation table. The excitation table is one that gives the information about the present states, next states, and required excitations of the flip flop.

Step (4) – Obtain the minimized expression for excitations

Now, obtain the minimal expressions for the excitations of the flip flops by using any minimization technique such as K-maps.

Step (5) – Draw the logic diagram

Finally, draw the logic circuit diagram according to the minimal expression obtained in the step 4.

In this way, we can design a synchronous counter

Synchronous counter may suffer from the problem of lock-out, which means they may not be self-starting. A self-starting counter is a type of synchronous counter that will enter to its proper sequence of states regardless of its initial state. We can make a counter self-starting by designing it so that it enters to a particular state whenever it goes to an invalid state.

Now, let us take an example to understand the procedure of designing a synchronous counter.

Example

Design a synchronous counter using D flip flops that goes through states 0, 1, 2, 4, 0. The unused states must always go to zero on the next clock pulse.

Solution

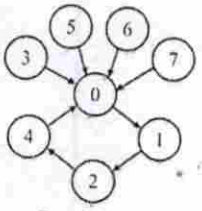
This synchronous counter is designed as per the following steps –

Step 1 – Number of flip flops required –

This synchronous counter has four stable states, i.e. 0 (000), 1 (001), 2 (010), 4 (100). But we require three flip flops because it counts 4 (100) as well. Since three flip-flops can count eight states. Thus, the remaining four states, i.e. 3 (011), 5 (101), 6 (110), and 7 (111) are unused states. As per the problem statement, the unused states must go to 0 (000) after the next clock pulse. Therefore, there are no don't care states.

Step 2 – Draw the state diagram –

The state diagram of the 0, 1, 2, 4, 0, … counter is drawn as shown in the following figure



State Diagram

Step 3 – Chose the type of flip flop and write the excitation table –

The D flip flop is to be used and the excitation table of the counter using D flip flop is written below.

Present State			**Next State**			**Required Excitation**		
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	D_3	D_2	D_1
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	1	0	0	1	0	0
0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0

Step 4 – Derive the minimal expression –

From the excitation table, we can see that there is no minimization is possible. Hence, the expressions for the excitations can be directly written from the excitation table itself as follows –

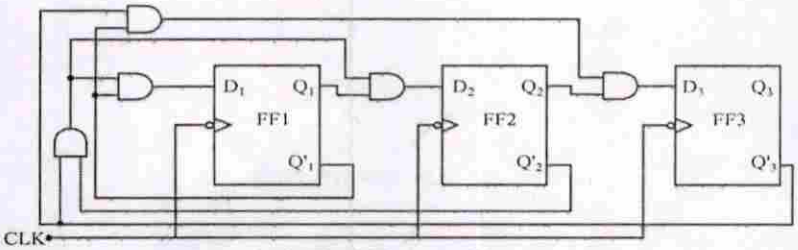
D1=Q3'Q2'Q1'

D2=Q3'Q2'Q1'

D3 =Q3'Q2'Q1'

Step 5 – Draw the logic circuit diagram –

The logic circuit diagram of the counter 0, 1, 2, 4, 0,… as per the expressions is shown in the figure below.

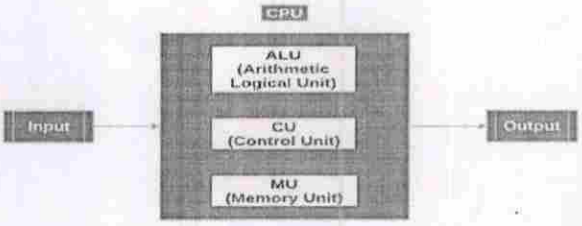


Logic Circuit Diagram of Counter

10. Explain about functional units of basic computer.

A. **Computer:** A computer is a combination of **hardware and software** resources which integrate together and provides various functionalities to the user. Hardware are the physical components of a computer like the processor, memory devices, monitor, keyboard etc. while software is the set of programs or instructions that are required by the hardware resources to function properly.

There are a few basic components that aids the working-cycle of a computer i.e. the Input- Process- Output Cycle and these are called as the functional components of a computer. It needs certain input, processes that input and produces the desired output. The input unit takes the input, the central processing unit does the processing of data and the output unit produces the output. The memory unit holds the data and instructions during the processing.



- **Input Unit :**The input unit consists of input devices that are attached to the computer. These devices take input and convert it into binary language that the computer understands. Some of the common input devices are keyboard, mouse, joystick, scanner etc.
- **Central Processing Unit (CPU) :** Once the information is entered into the computer by the input device, the processor processes it. The CPU is called the brain of the computer because it is the control center of the computer. It first fetches instructions from memory and then interprets them so as to know what is to be done. If required, data is fetched from memory or input device. Thereafter CPU executes or performs the required computation and then either stores the output or displays on the output device. The CPU has three main components which are responsible for different functions – Arithmetic Logic Unit (ALU), Control Unit (CU) and Memory registers
- **Arithmetic and Logic Unit (ALU) :** The ALU, as its name suggests performs mathematical calculations and takes logical decisions. Arithmetic calculations include addition, subtraction, multiplication and division. Logical decisions involve comparison of two data items to see which one is larger or smaller or equal.
- **Control Unit :** The Control unit coordinates and controls the data flow in and out of CPU and also controls all the operations of ALU, memory registers and also input/output units. It is also responsible for carrying out all the instructions stored in the program. It decodes the fetched instruction, interprets it and sends control signals to input/output devices until the required operation is done properly by ALU and memory.

- **Memory Registers :** A register is a temporary unit of memory in the CPU. These are used to store the data which is directly used by the processor. Registers can be of different sizes(16 bit, 32 bit, 64 bit and so on) and each register inside the CPU has a specific function like storing data, storing an instruction, storing address of a location in memory etc. The user registers can be used by an assembly language programmer for storing operands, intermediate results etc. Accumulator (ACC) is the main register in the ALU and contains one of the operands of an operation to be performed in the ALU.
- **Memory :** Memory attached to the CPU is used for storage of data and instructions and is called internal memory The internal memory is divided into many storage locations, each of which can store data or instructions. Each memory location is of the same size and has an address. With the help of the address, the computer can read any memory location easily without having to search the entire memory. When a program is executed, it's data is copied to the internal memory and is stored in the memory till the end of the execution. The internal memory is also called the Primary memory or Main memory. This memory is also called as RAM, i.e. Random Access Memory. The time of access of data is independent of its location in memory, therefore this memory is also called Random Access memory (RAM).
- **Output Unit :** The output unit consists of output devices that are attached with the computer. It converts the binary data coming from CPU to human understandable form. The common output devices are monitor, printer, plotter etc.

11.Explain the different types of Instruction Codes.

A. Computer organization refers to the way in which the components of a computer system are organized and interconnected to perform specific tasks. One of the most fundamental aspects of computer organization is the set of basic computer instructions that the system can execute.

Basic computer instructions are the elementary operations that a computer system can perform. These instructions are typically divided into three categories: data movement instructions, arithmetic and logic instructions, and control instructions.

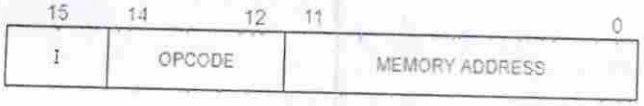
Data movement instructions are used to move data between different parts of the computer system. These instructions include load and store instructions, which move data between memory and the CPU, and input/output (I/O) instructions, which move data between the CPU and external devices.

Arithmetic and logic instructions are used to perform mathematical operations and logical operations on data stored in the system. These instructions include add, subtract, multiply, and divide instructions, as well as logic instructions such as AND, OR, and NOT.

Control instructions are used to control the flow of instructions within the computer system. These instructions include branch instructions, which transfer control to different parts of the program based on specified conditions, and jump instructions, which transfer control to a specified memory location.

The basic computer has 16-bit instruction register (IR) which can denote either memory reference or register reference or input-output instruction.

1. **Memory Reference –** These instructions refer to memory address as an operand. The other operand is always accumulator. Specifies 12-bit address, 3-bit opcode (other than 111) and 1-bit addressing mode for direct and indirect addressing.

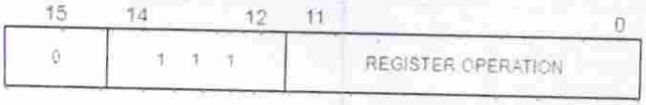


1. **Example –** IR register contains = 0001XXXXXXXXXXX, i.e. ADD after fetching and decoding of instruction we find out that it is a memory reference instruction for ADD operation.

Hence, DR ← M[AR]

AC ← AC + DR, SC ← 0

1. **Register Reference –** These instructions perform operations on registers rather than memory addresses. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 0 (differentiates it from input/output instructions). The rest 12 bits specify register operation.



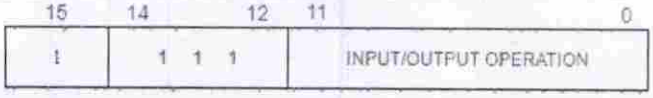
1. **Example –** IR register contains = 0111001000000000, i.e. CMA after fetch and decode cycle we find out that it is a register reference instruction for complement accumulator.

Hence, AC ← ~AC

BASIC COMPUTER INSTRUCTIONS

Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

1. **Input/Output** – These instructions are for communication between computer and outside environment. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 1 (differentiates it from register reference instructions). The rest 12 bits specify I/O operation.



Example – IR register contains = 1111100000000000, i.e. INP after fetch and decode cycle we find out that it is an input/output instruction for inputing character. Hence, INPUT character from peripheral device.

Faculty Signature

1.P.Divya(Course Coordinator)- 23/11/23

2.Swapna-

3.V.V.Bhavani- 23/11/23

4. Vinod- 23/11/23

5. Prince- 23/11/23