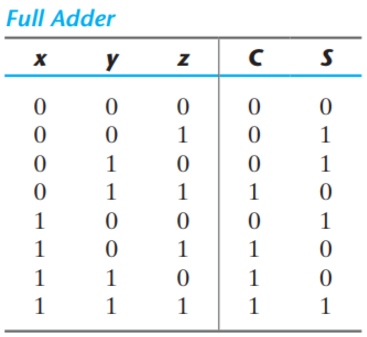
IAT 2 Answer Sheet

Q.1 Design a full adder by constructing the truth table (2) and simplify the output equations (2) (K-map (3) & Verilog module (3))

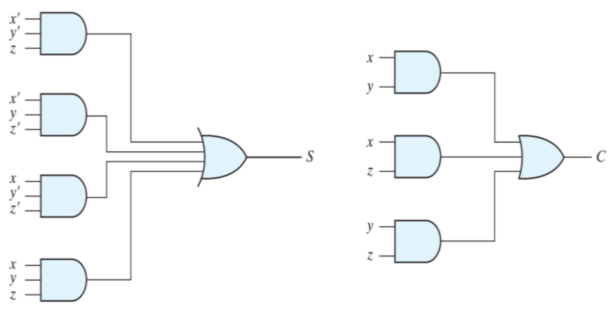
Ans. A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs. We assign symbols x, y and z to the three inputs and S (for sum) and C (for carry) to the outputs. The C output is 1 only when two or three inputs are 1. The S output represents the least significant bit of the sum. The simplified expressions are

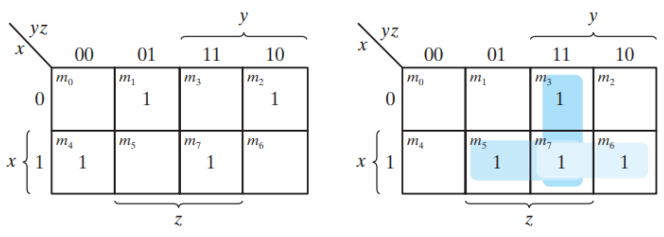
$$S = x'y'z + x'yz' + xy'z' + xyz \quad \textbf{and} \quad C = xy + xz + yz$$



Full Adder

x	y	z	C	S
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





(a) $S = x'y'z + x'yz' + xy'z' + xyz$ (b) $C = xy + xz + yz$

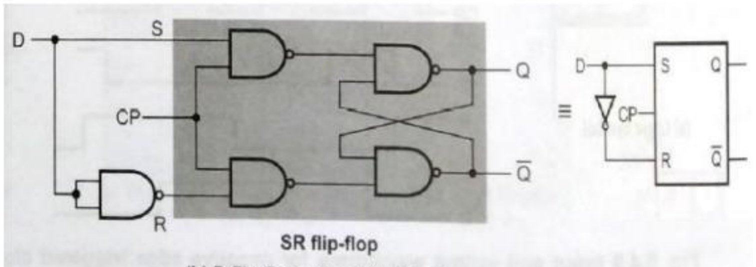
K-Maps for full adder

Verilog Module:

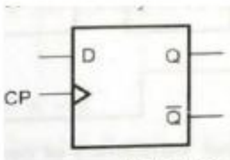
```
module FullAdder(a,b,c,sum,carry);

input a,b,c;

output sum,carry;

assign sum=a^b^c;

assign carry=(a&b)|(b&c)|(c&a);

endmodule
```

Q.2 Explain the D and JK flip flop with logic diagram, function table and equation.

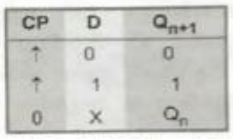
Ans. D flip flop (2+3)



(b) D flip-flop using NAND gates



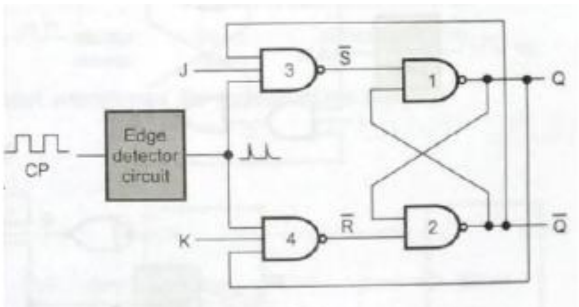
(a) Logic symbol



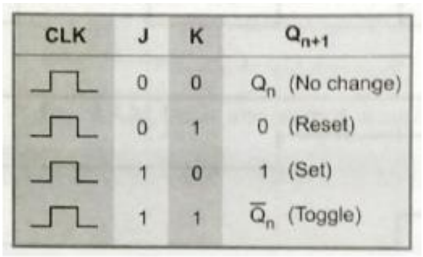
CP	D	Q_{n+1}
↑	0	0
↑	1	1
0	X	Q_n

(b) Truth table of D flip-flop

JK flip flop (2+3)



$\bar{S} = \overline{J\bar{Q}}$ $\bar{R} = \overline{KQ}$



CLK	J	K	Q_{n+1}
⎍	0	0	Q_n (No change)
⎍	0	1	0 (Reset)
⎍	1	0	1 (Set)
⎍	1	1	$\bar{Q}_n$ (Toggle)

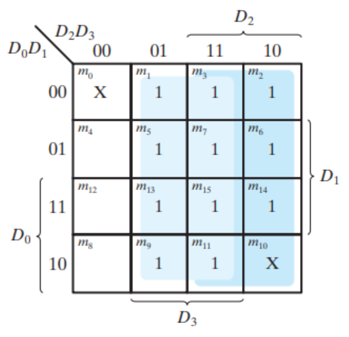
Q.3 a. What is Encoder (1)? Write the compressed truth table (1) for a 4 to 2 line priority encoder with a valid output where the highest priority is given to highest bit position and simplify the same using K-map (1). Design the logic circuit as well (2).

Ans. 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, generate the binary code corresponding to the input value.

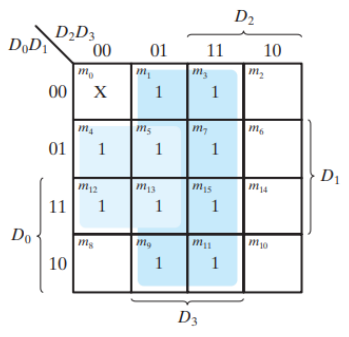
The truth table of a four-input priority encoder is given. In addition to the two outputs x and y. the circuit has a third output designated by V; this is a valid bit indicator that is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0. The other two outputs are not inspected when V equals 0 and are specified as don't-care conditions.

Truth Table of a Priority Encoder

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



$x = D_2 + D_3$



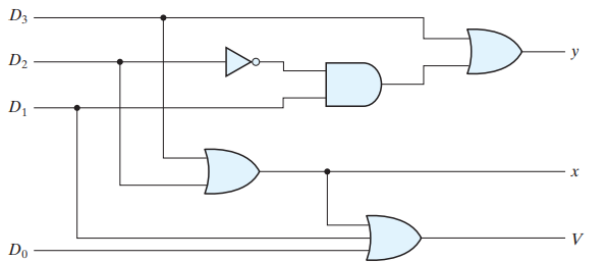
$y = D_3 + D_1D'_2$

The maps for simplifying outputs x and y. The minterms for the two functions are derived.

x = D2 + D3

y = D3 + D1 D 2'

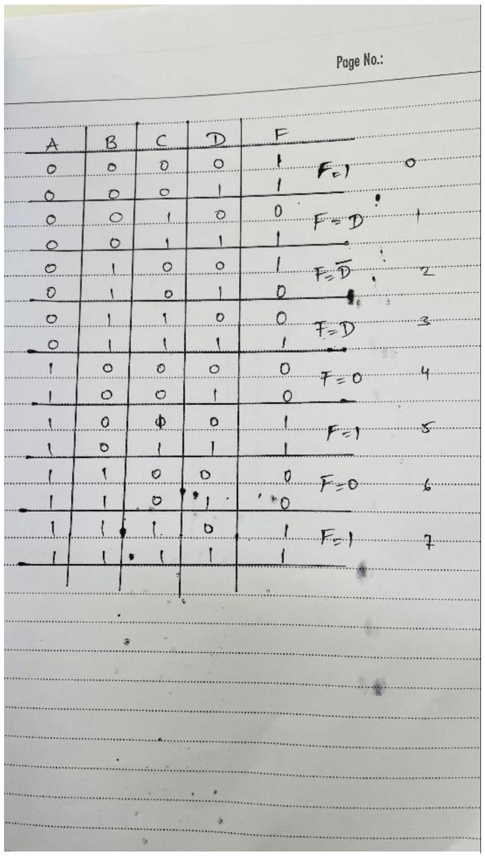
V = D0 + D1 + D2 + D3



Four-input priority encoder

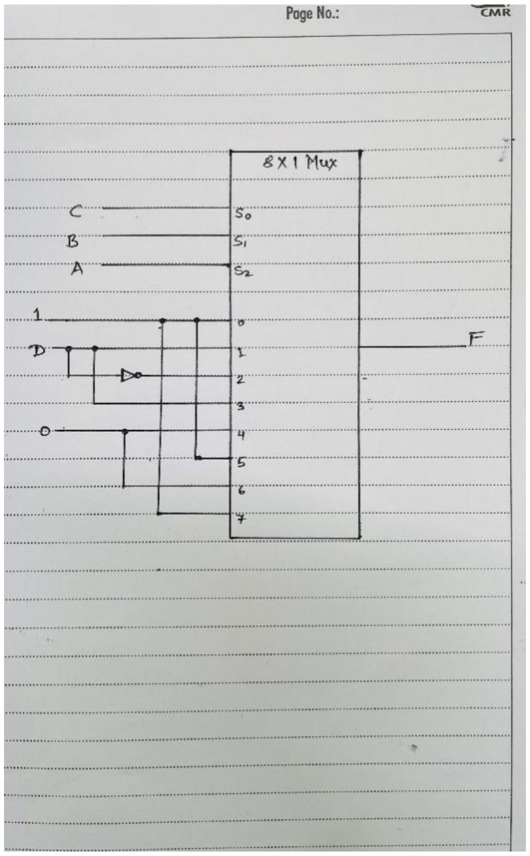
b. Implement the function using 8:1 MUX, F(a,b,c,d) = Σm(0,1,3,4,7,10,11,14,15) (2+3)

Ans.



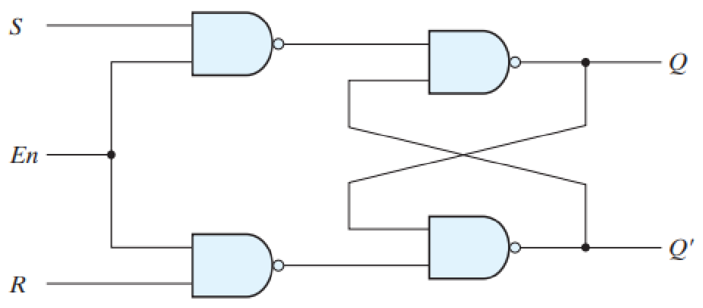
Page No.:

A	B	C	D	F		
0	0	0	0	1	F=1	0
0	0	0	1	1		
0	0	1	0	0	F=D	1
0	0	1	1	1		
0	1	0	0	1	$F=\overline{D}$	2
0	1	0	1	0		
0	1	1	0	0	F=D	3
0	1	1	1	1		
1	0	0	0	0	F=0	4
1	0	0	1	0		
1	0	ϕ	0	1	F=1	5
1	0	1	1	1		
1	1	0	0	0	F=0	6
1	1	0	1	0		
1	1	1	0	1	F=1	7
1	1	1	1	1		

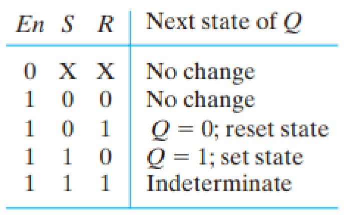


Q.4 a. With logic diagram (2) and truth table (3) explain the operation of a SR latch.

Ans. An SR latch with a control input consists of the basic SR latch and two additional NAND gates. The control input En acts as an enable signal for the other two inputs. The outputs of the NAND gates stay at the logic-1 level as long as the enable signal remains at 0. This is the quiescent condition for the SR latch. When the enable input goes to 1, information from the S or R input is allowed to affect the latch.



(a) Logic diagram



En	*S*	*R*	Next state of *Q*
0	X	X	No change
1	0	0	No change
1	0	1	*Q* = 0; reset state
1	1	0	*Q* = 1; set state
1	1	1	Indeterminate

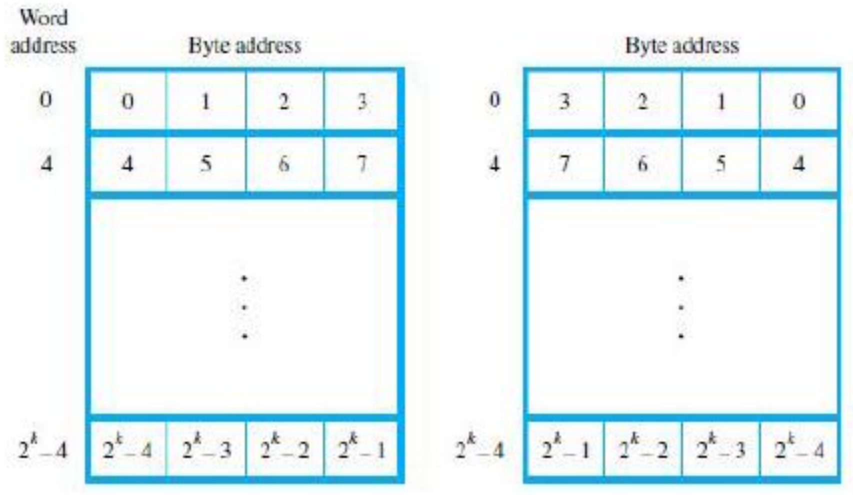
(b) Function table

b. Analyze Big –endian (2) and little –endian (2) methods of byte addressing with Example (1).

Ans. Big-Endian: lower byte addresses are used for the most significant bytes of the word.

Little-Endian: opposite ordering. Lower byte addresses are used for the less significant bytes of the word.

In both cases, byte-addresses 0, 4, 8.....are taken as the addresses of successive words in the memory.



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

Example: Consider a 32-bit integer (in hex): 0 **x** 12345678 which consists of 4 bytes: 12, 34, 56, and78.

• Hence this integer will occupy 4 bytes in memory.

• Assume, we store it at memory address starting 1000.

On little-endian, memory will look like

Address	Value
1000	78
1001	56
1002	34
1003	12

On big-endian, memory will look like

Address	Value
1000	12
1001	34
1002	56
1003	78

Q.5 a. For the following processor, obtain the performance

Clock rate = 800MHz

No. of instructions executed = 1000

Average no. of steps needed/machine instruction = 20

Ans. (2+3)

$$T = \frac{N \times S}{R} = (1000*20)/800 * 10^{6} = 25 \text{ micro sec or } 25*10^{-6} \text{ sec}$$

b. Explain operational Concepts of a computer.

Ans. An Instruction consists of 2parts, 1) Operation code (Op code) and 2) Operands.

• The data/ operands are stored in memory. The individual instruction is brought from the memory to the processor. Then, the processor performs the specified operation. Let us see a typical instruction

ADD LOCA, R0

• The same instruction can be realized using 2instructions as: Load LOCA, R1 Add R1, R0

The following are the steps to execute the instruction:

Step1: Fetch the instruction from main-memory into the processor.

Step2: Fetch the operand at location LOCA from main-memory into the register R1.

Step3: Add the content of Register R1 and the contents of register R0.

Step4: Store the result in R0.

Q.6 a. What is an Addressing Modes (1) and explain any Four (4).

Ans. The term addressing modes refers to how the operand of an instruction is specified. Information contained in the instruction code is the value of the operand or the address of the operand. Following are the main addressing modes that are used on various platforms and architectures.

1. Register Addressing Mode
2. Immediate Addressing Mode
3. Direct (or Absolute) Addressing Mode
4. Indirect Addressing Mode
5. Index Addressing Mode

6. Relative Addressing Mode
7. Auto increment Addressing Mode
8. Auto decrement Addressing Mode.

Register Addressing Mode

- The operand is the content of a processor register. Register name is specified in the instruction.
- Effective Address of the Operand: Register name specified in the instruction
- Operator, operand
- Move R1, R2
- ADD R0, R1

Direct(or Absolute) Addressing Mode

- The operand is a Memory location. The address of the memory location is given in the instruction explicitly.
- Effective Address of the Operand: Address of the memory location given directly in the instruction
- Mov LOC R1
- ADD NUM1, R0

Immediate Addressing Mode

- The operand is given explicitly in the instruction
- Effective Address of the Operand: Operand value given in the instruction
- Mov #400, R1

Indirect Addressing Mode

- Here neither the operands nor the addresses are given explicitly. The instruction provides the information from which the address of the operand is determined i.e., the instruction provides effective address of the operand using register or memory location. The indirection is denoted by () sign around register or memory.
- Effective Address of the Operand: (Ri) or (LOCA) is the contents of a register or the memory location whose address appears in the instruction

Index Addressing Mode

- The effective address of the operand is generated by adding a constant value to the contents of a register specified in the instruction. The register in this case is called as Index register.
- The operation is indicated as X(Ri).
- Effective Address of the Operand: X+Ri where X is a constant value (signed integer) and Ri is the index register.

Relative Addressing Mode

- In this mode the content of the program counter is added to the address part of the instruction to obtain the effective address.
- Effective Address : X+PC where X is a constant value (signed integer) and PC is the contents of the program counter.
- Program counter is responsible to carry out the contents of the operands

Autoincrement Addressing Mode

- This is indirect mode with a modification. The effective address of the operand is the contents of a pointer register specified in the instruction. After accessing the operand, the contents of this pointer register is incremented automatically to point to the next entity.
- The mode is denoted by (Ri)+, where Ri is the pointer register.
- The + sign indicates that Ri is incremented after the operation.
- The increment operation is depending on the size of the accessed operand. Thus, the increment value is 1 for byte-size operands, 2 for word-size (16-bit) operands and 4 for long-word (32-bit) operands.
- This mode is useful when operands are stored consecutively in memory i.e., for array manipulation
- **contents of the register**

(Ri)+ ->EA

Autodecrement Addressing Mode

- This mode is useful to access an array in the reverse order. The value of the pointer register specified in the instruction is decremented first and this value is used as the effective address of the operand.
- The mode is denoted by -(Ri), where Ri is the pointer register.
- The - sign indicates that Ri is decremented before accessing the operand.
- The decrement operation is depending on the size of the accessed operand. Thus, the decrement value is 1 for byte-size operands, 2 for word-size (16-bit) operands and 4 for long-word (32-bit) operands.
- This two modes (Autoincrement and Autodecrement) are useful to implement a data structure called Stack.
- **-(Ri) ->EA**

b. Explain Basic Instruction Types.

Ans. Instruction Set Categories based on the Operands explicitly specified in the instruction.

1. Three-address or 3-Operand instructions
2. Two-address or 2-Operand instructions
3. One-address or 1-Operand instructions
4. Zero-address or 0-Operand instructions

Three-address or 3-Operand instructions

- Three-address instruction can be represented symbolically **ADD A, B, C**
- A and B are called the source operands, C is called destination operand, and ADD is the operation to be performed on the operands.
- A general instruction of this type has the format

Operation	**Source Operand1**	**Source Operand2**	**Destination Operand**

Two-address or 2-Operand instructions

- Two-address instruction can be represented symbolically **ADD A, B**
- A and B are called the source operands, B is called destination operand, and ADD is the operation to be performed on the operands.
- A general instruction of this type has the format

Operation	**Source Operand**	**Source/Destination Operand**

One-address or 1-Operand instructions

- Only one Operand will be specified in the instructions.
- Accumulator Register will be used as second Operand.
- Example: **ADD A**
- Acc <- [Acc]+[A] Meaning : Add the contents of accumulator with the memory location A ; And Store the result in the accumulator register
- General Format:

Operation	**Source/Destination Operand**

Zero-address or 0-Operand instructions

- Locations of the operands are defined implicitly.
- Stack will be used to store operands.
- Example: **ADD** Top two elements of the stack will popped and sum of the popped numbers will pushed on to stack
- General Format:

Operation