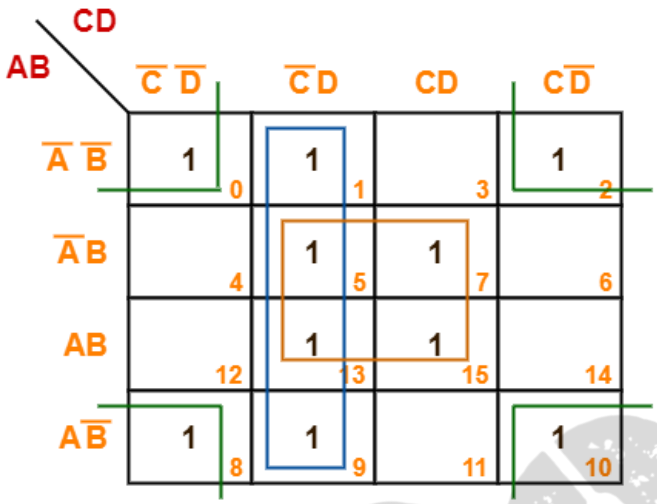
MODULE 1 & 2

1. Minimize the following boolean function-

   F(A, B, C, D) = Σm(0, 1, 2, 5, 7, 8, 9, 10, 13, 15)

   Solution:

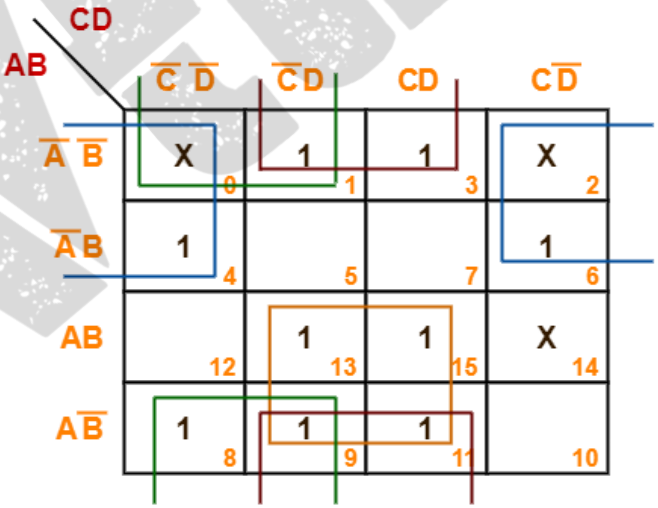


Thus, minimized boolean expression is-

F(A, B, C, D) = BD + C'D + B'D'

2. Minimize the following boolean function

   F(A, B, C, D) = Σm(1, 3, 4, 6, 8, 9, 11, 13, 15) + Σd(0, 2, 14)

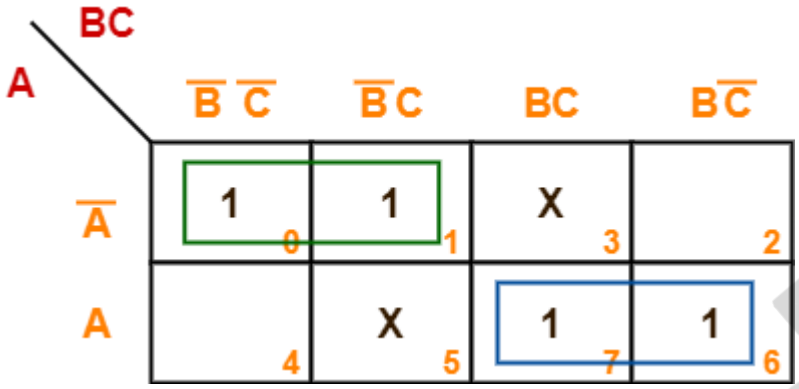


Thus, minimized boolean expression is-

F(A, B, C, D) = AD + B'D + B'C' + A'D'

3. Minimize the following boolean function

F(A, B, C) = Σm(0, 1, 6, 7) + Σd(3, 5)

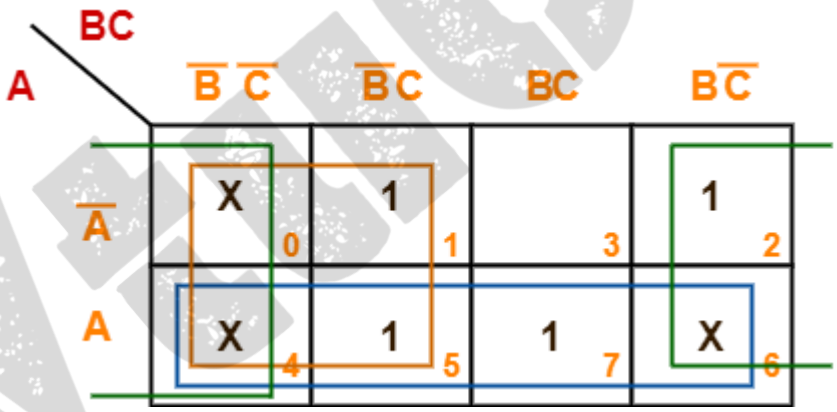


Thus, minimized boolean expression is

F(A, B, C) = AB + A'B'

4. Minimize the following boolean function

F(A, B, C) = Σm(1, 2, 5, 7) + Σd(0, 4, 6)

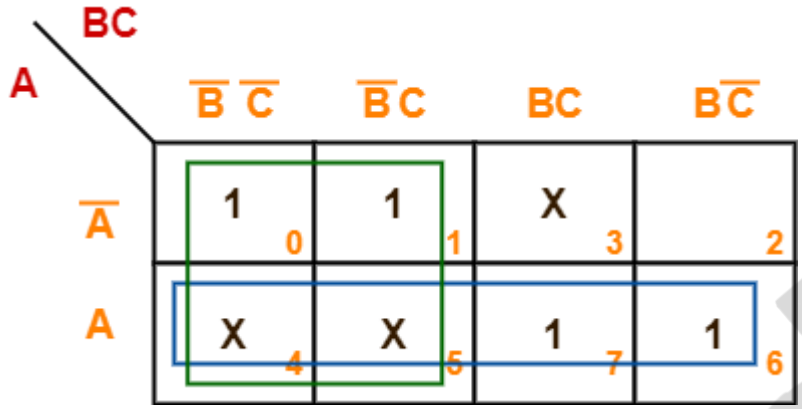


Thus, minimized boolean expression is

F(A, B, C) = A + B' + C'

5. Minimize the following boolean function

F(A, B, C) = Σm(0, 1, 6, 7) + Σd(3, 4, 5)

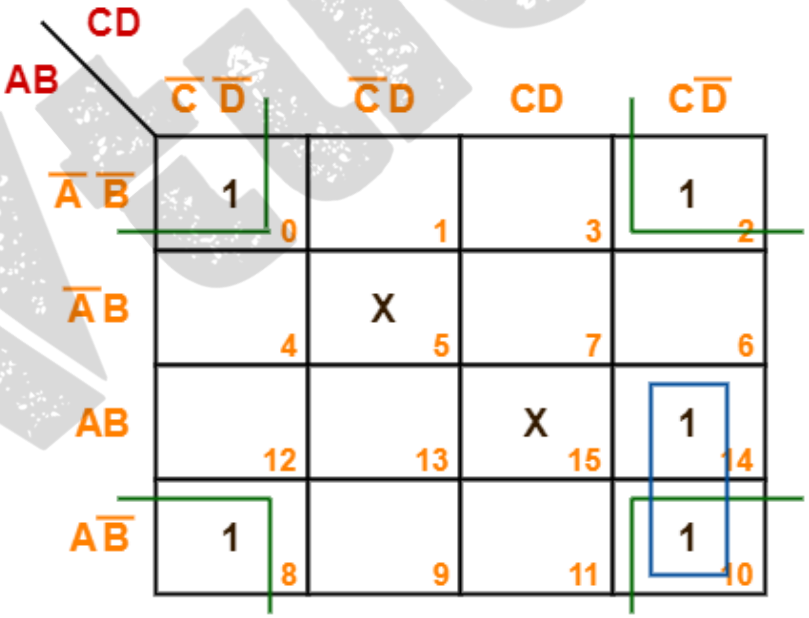


Thus, minimized boolean expression is

F(A, B, C) = A + B'

6. Minimize the following boolean function

F(A, B, C, D) = Σm(0, 2, 8, 10, 14) + Σd(5, 15)

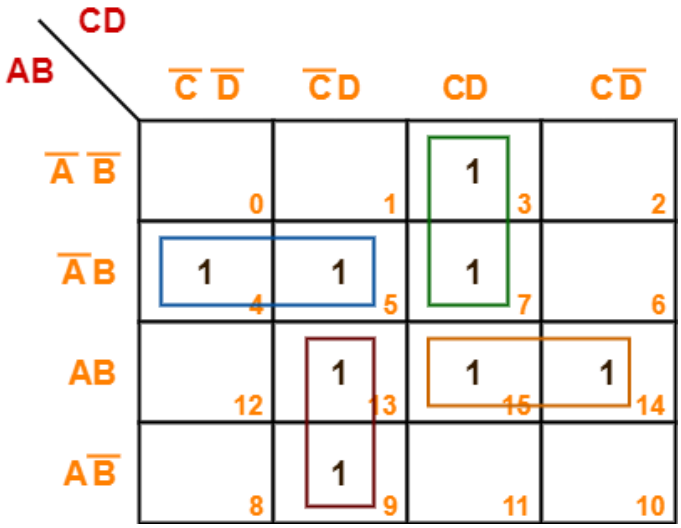


Thus, minimized boolean expression is

F(A, B, C, D) = ACD' + B'D'

7. Minimize the following boolean function-

F(A, B, C, D) = Σm(3, 4, 5, 7, 9, 13, 14, 15)

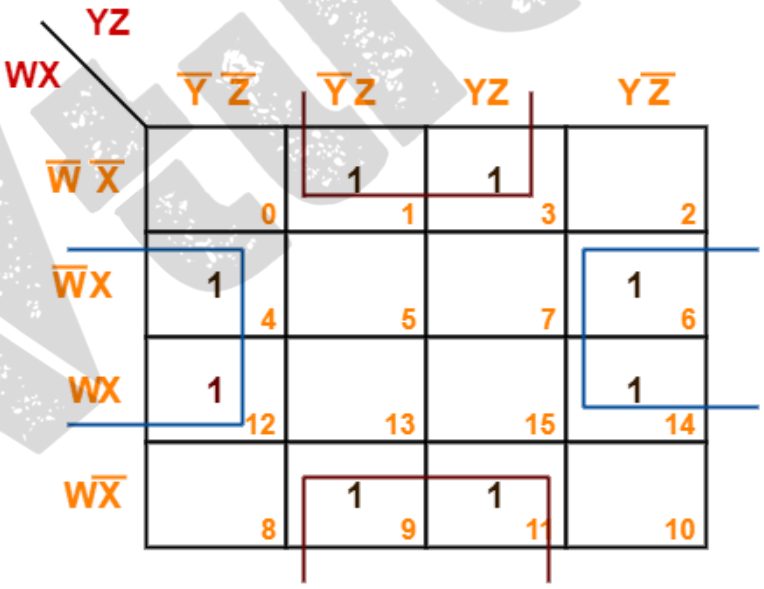


Thus, minimized boolean expression is

F(A, B, C, D) = A'BC' + A'CD + AC'D + ABC

8. Minimize the following boolean function

F(W, X, Y, Z) = Σm(1, 3, 4, 6, 9, 11, 12, 14)

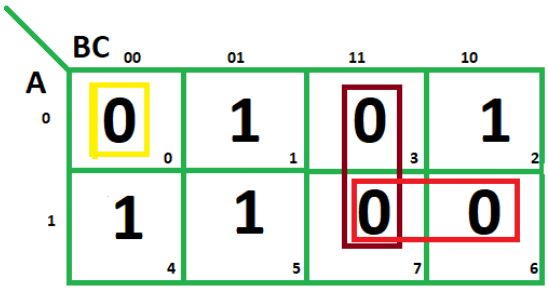


Thus, minimized boolean expression is-

F(W, X, Y, Z) = X ⊕ Z

9. Minimize the following boolean function

F(A,B,C)=π(0,3,6,7)

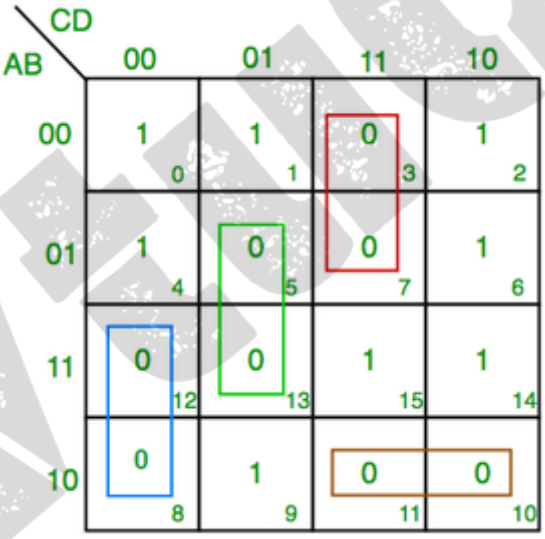


Thus, minimized boolean expression is-

(A' + B') (B' + C') (A + B + C)

10. Minimize the following boolean function

F(A,B,C,D)= π (3,5,7,8,10,11,12,13)

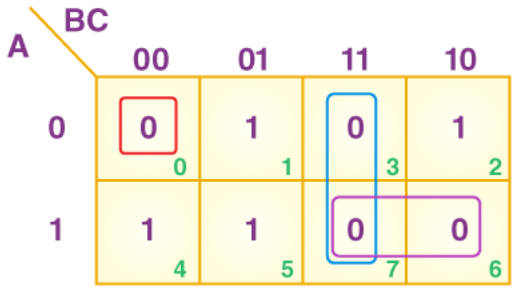


Thus, minimized boolean expression is-

(C+D'+B').(C'+D'+A).(A'+C+D).(A'+B+C')

11. Minimize the following boolean function

F (P, Q, R) = π (0,3,6,7)

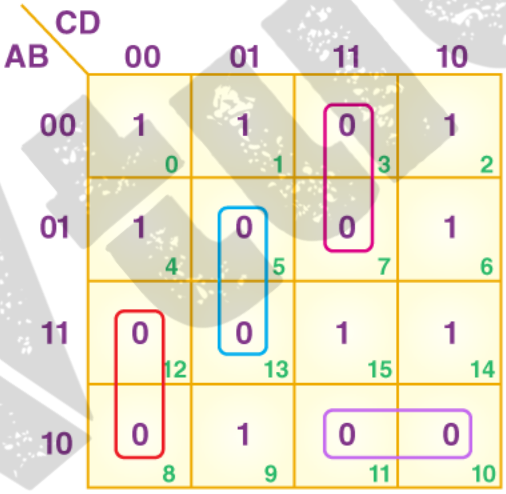


Thus, minimized boolean expression is-

$$(A' + B')\,(A' + C')\,(A + B + C)$$

12. Minimize the following boolean function

F (A, B, C, D) = π (3, 5, 7, 8, 10, 11, 12, 13)



Thus, minimized boolean expression is-

$$(C + D' + B').(C' + D' + A).(A' + C + D).(A' + B + C')$$

13. Write Verilog code for the following digital circuits.

a) AND gate

b) NOT gate

AND gate

```
       //AND gate using Structural modeling
module and_gate_s(a,b,y);
input a,b;
output y;

and(y,a,b);

endmodule



//AND gate using data flow modeling
module and_gate_d(a,b,y);
input a,b;
output y;

assign y = a & b;

endmodule

//AND gate using behavioural modeling
module nAND_gate_b(a,b,y);
input a;
output y;

always @ (a,b)
y = a & b;

endmodule
```

NOT gate

```
//NOT gate using Structural modeling
module not_gate_s(a,y);
input a;
output y;

not(y,a);

endmodule
```

```
//NOT gate using data flow modeling
module not_gate_d(a,y);
input a;
output y;

assign y = ~a;

endmodule
```

```
//NOT gate using behavioural modeling
module not_gate_b(a,y);
input a;
output reg y;

always @ (a)
y = ~a;

endmodule
```

14. **Develop** Verilog code for the following combinational logic circuts using **Structural** and **Dataflow** description.

a) 2x4 Decoder **b) 4x1 Multiplexer**

2x4 Decoder

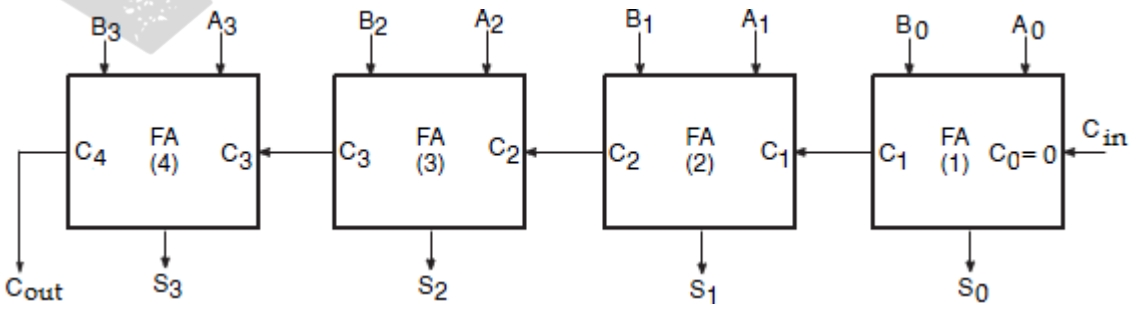
```
module decoder_2_4(a,b,w,x,y,z);

output w,x,y,z;
input a,b;

assign w = (~a) & (~b);
assign x = (~a) & b;
assign y = a & (~b);
assign z = a & b;
end module
```

4x1 Multiplexer

```
module m41(out, i0, i1, i2, i3, s0, s1);
output out;
input i0, i1, i2, i3, s0, s1;
assign y0 = (i0 & (~s0) & (~s1));
assign y1 = (i1 & (~s0) & s1);
assign y2 = (i2 & s0 & (~s1));
assign y3 = (i3 & s0 & s1);
assign out = (y0 | y1 | y2 | y3);
end module
```

15. Explain Binary Adder (Parallel Adder) with a neat diagram

The 4-bit binary adder using full adder circuits is capable of adding two 4-bit numbers resulting in a 4-bit sum and a carry output as shown in figure below



Since all the bits of augend and addend are fed into the adder circuits simultaneously and the additions in each position are taking place at the same time, this circuit is known as parallel adder.

Let the 4-bit words to be added be represented by,
$A_3A_2A_1A_0$= 1111 and $B_3B_2B_1B_0$= 0011

Significant place		4	3	2	1
Input carry		1	1	1	0
Augend word A :		1	1	1	1
Addend word B :		0	0	1	1
	1	0	0	1	0 ← Sum
	↑				
	Output Carry				

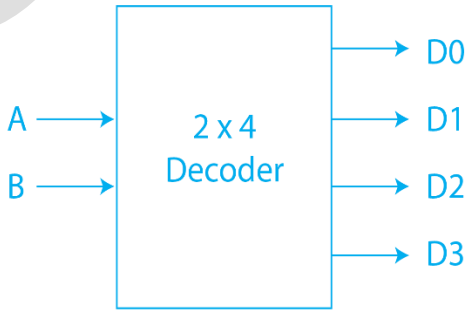
The bits are added with full adders, starting from the least significant position, to form the sum it and carry bit. The input carry C_0 in the least significant position must be 0. The carry output of the lower order stage is connected to the carry input of the next higher order stage. Hence this type of adder is called ripple-carry adder.

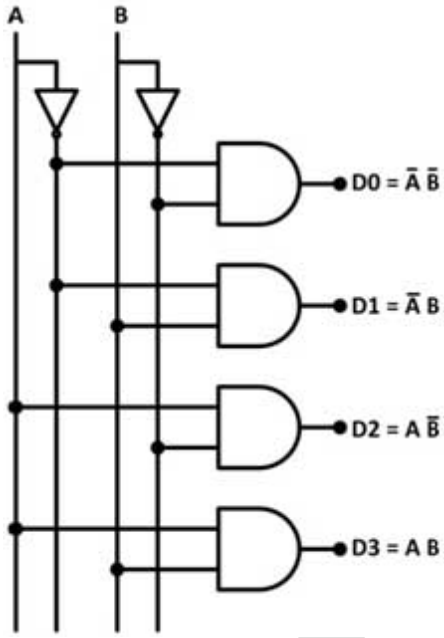
In the least significant stage, A_0, B_0 and C_0 (which is 0) are added resulting in sum S_0 and carry C_1. This carry C_1 becomes the carry input to the second stage. Similarly in the second stage, A_1, B_1 and C_1 are added resulting in sum S_1 and carry C_2, in the third stage, A_2, B_2 and C_2 are added resulting in sum S_2 and carry C_3, in the third stage, A_3, B_3 and C_3 are added resulting in sum S_3 and C_4, which is the output carry.

Thus the circuit results in a sum ($S_3S_2S_1S_0$) and a carry output (C_{out}).

16. What is Decoder? Explain 2 x 4 decoder with a neat diagram.

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.





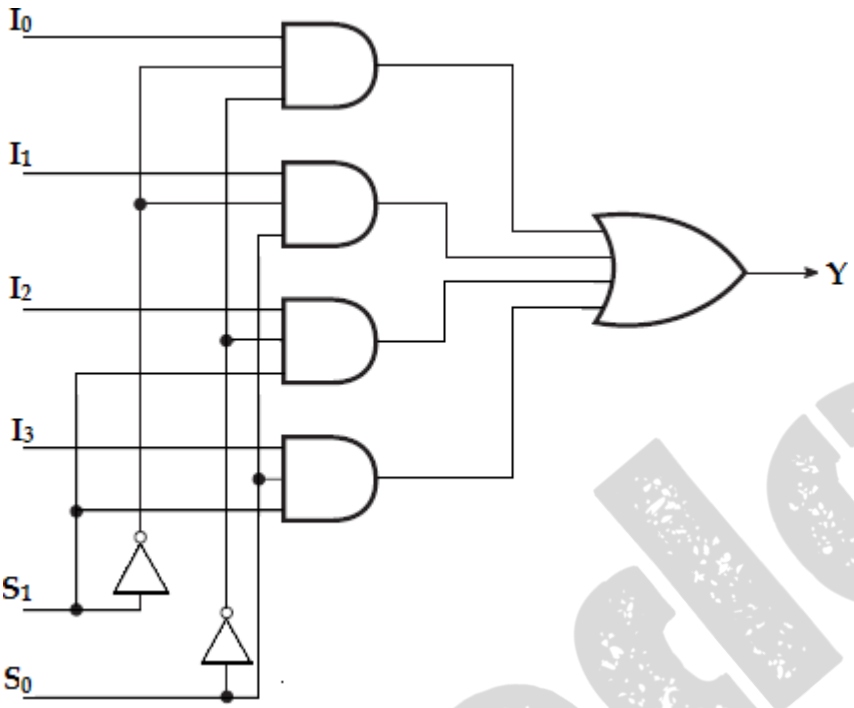
Inputs		Outputs				X
A	B	d_0	d_1	d_2	d_3	
0	0	1	0	0	0	0
0	1	0	1	0	0	1
1	0	0	0	1	0	2
1	1	0	0	0	1	3

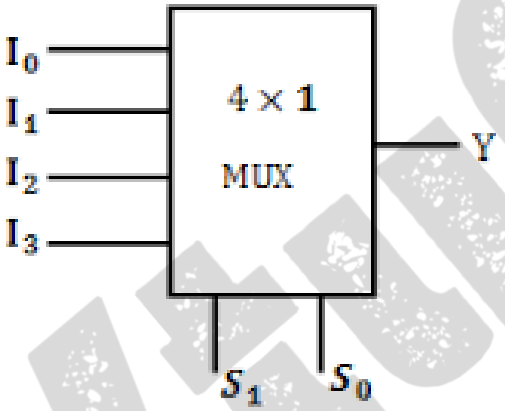
Here the 2 inputs are decoded into 4 outputs, each output representing one of the minterms of the two input variables.

The output Y_0 is active, ie., D_0= 1 when inputs A= B= 0,
D_1 is active when inputs, A= 0 and B= 1,
D_2 is active, when input A= 1 and B= 0,
D_3 is active, when inputs A= B= 1.

17. What is Multiplexer? Explain 4 : 1 Multiplexer with a neat diagram.

A ***multiplexer*** or *MUX*, is a combinational circuit with more than one input line, one output line and more than one selection line.
Each of the four inputs I_0 through I_3, is applied to one input of AND gate.
Selection lines S_1 and S_0 are decoded to select a particular AND gate. The outputs of the AND gate are applied to a single OR gate that provides the 1-line output.





Truth table

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

To demonstrate the circuit operation, consider the case when S_1S_0= 10. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2. The other three AND gates have atleast one input equal to 0, which makes their outputs equal to 0. The OR output is now equal to the value of I_2, providing a path from the selected input to the output.

The data output is equal to I_0 only if S_1= 0 and S_0= 0; Y= I_0S_1'S_0'.

The data output is equal to I_1 only if S_1= 0 and S_0= 1; Y= I_1S_1'S_0.

The data output is equal to I_2 only if S_1= 1 and S_0= 0; Y= $I_2S_1S_0$'.

The data output is equal to I_3 only if S_1= 1 and S_0= 1; Y= $I_3S_1S_0$.

When these terms are ORed, the total expression for the data output is,

Y= I_0S_1'S_0'+ I_1S_1'S_0 +$I_2S_1S_0$'+ $I_3S_1S_0$.

18. Implement the following boolean function using 4: 1 multiplexer,
F (A, B, C) = Σm (1, 3, 5, 6).

Solution:
Variables, n= 3 (A, B, C)
Select lines= n-1 = 2 (**S_1, S_0**)
2_{n-1} to MUX i.e., 2_2 to 1 = 4 to 1 MUX
Input lines= $2_{n-1} = 2_2 = 4$ (**D_0, D_1, D_2, D_3**)

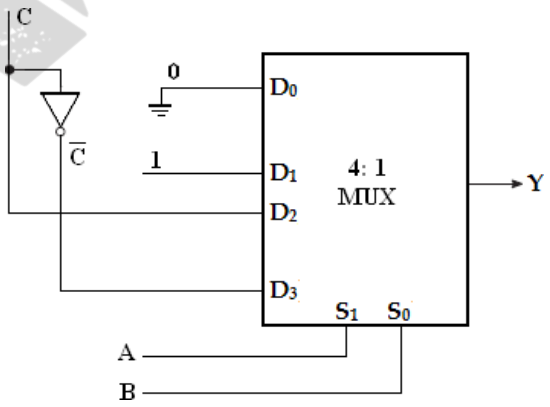
Implementation table:
Apply variables A and B to the select lines. The procedures for implementing the function are:

i. List the input of the multiplexer
ii. List under them all the minterms in two rows as shown below.
The first half of the minterms is associated with A' and the second half with A. The given function is implemented by circling the minterms of the function and applying the following rules to find the values for the inputs of the multiplexer.

1. If both the minterms in the column are not circled, apply 0 to the corresponding input.
2. If both the minterms in the column are circled, apply 1 to the corresponding input.
3. If the bottom minterm is circled and the top is not circled, apply C to the input.
4. If the top minterm is circled and the bottom is not circled, apply C' to the input.

	D_0	D_1	D_2	D_3
$\overline{C}$	0	(1)	2	(3)
C	4	(5)	(6)	7
	0	**1**	C	$\overline{C}$

Multiplexer Implementation:



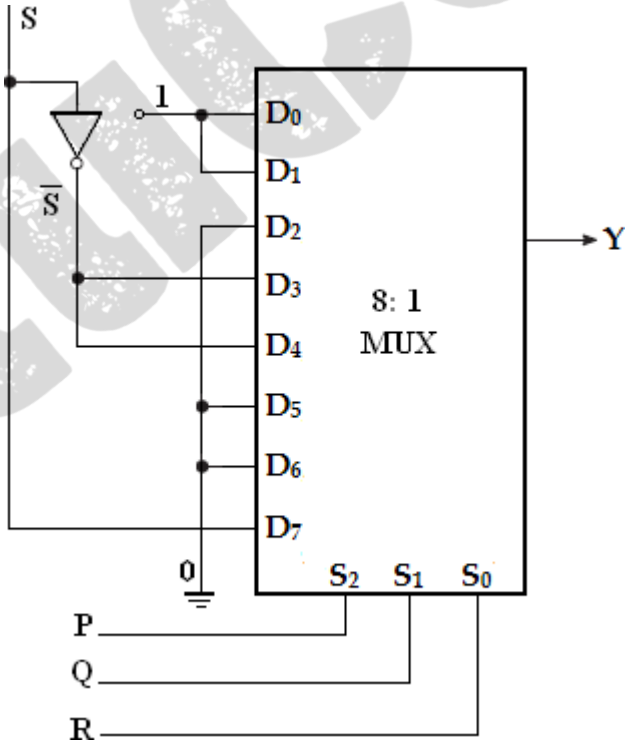
19. F(P, Q, R, S)= Σm (0, 1, 3, 4, 8, 9, 15)

Solution:
Variables, n= 4 (P, Q, R, S)
Select lines= n-1 = 3 (**S_2, S_1, S_0**)
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX
Input lines= $2^{n-1} = 2^3 = 8$ (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

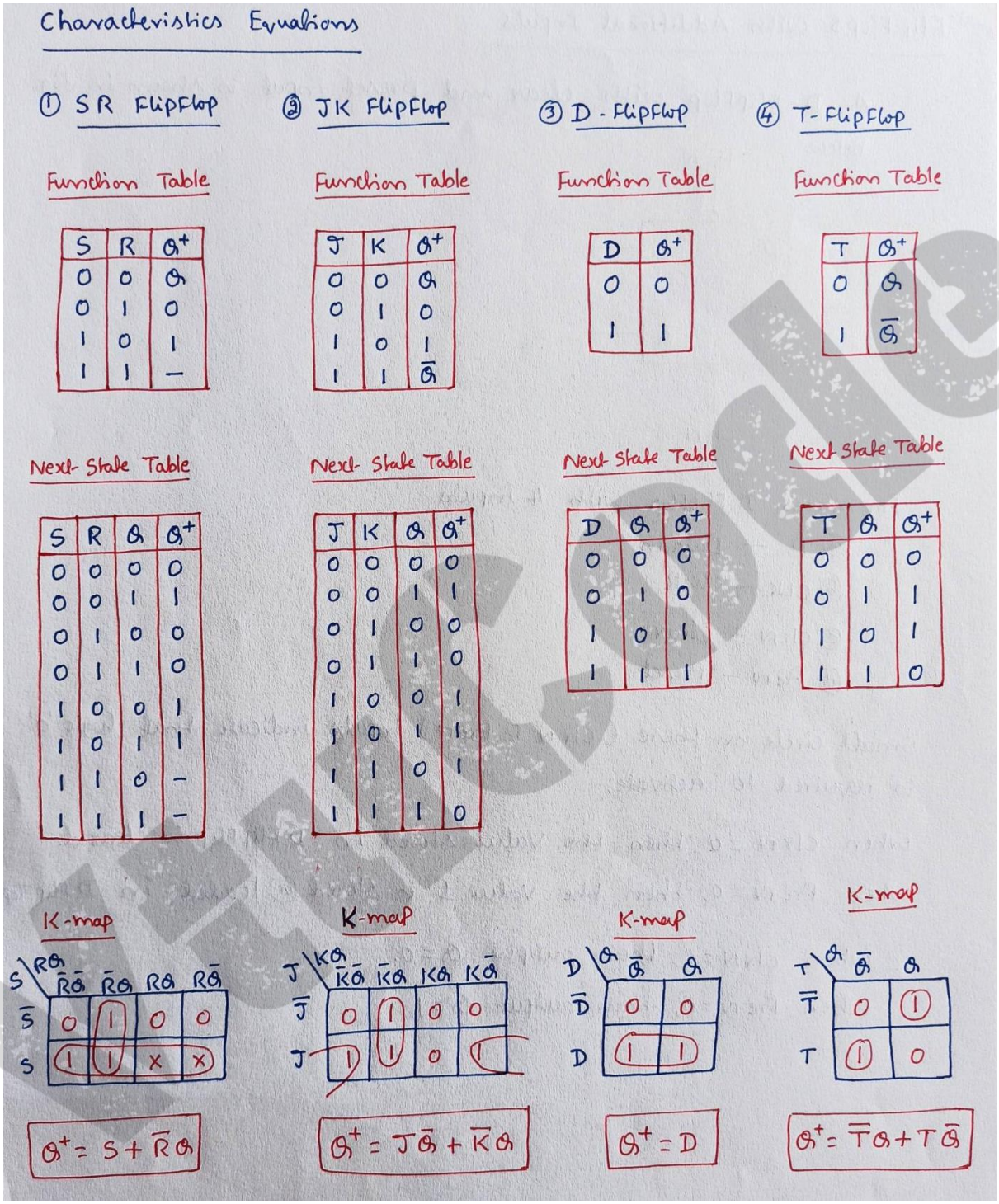
Implementation table:

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

Multiplexer Implementation:



20.Derive characteristic equation for JK, T, D and SR Flipflop



MODULE 3

1. **With** a neat diagram, **Explain** the basic operational concepts of a computer. Give operating steps.10

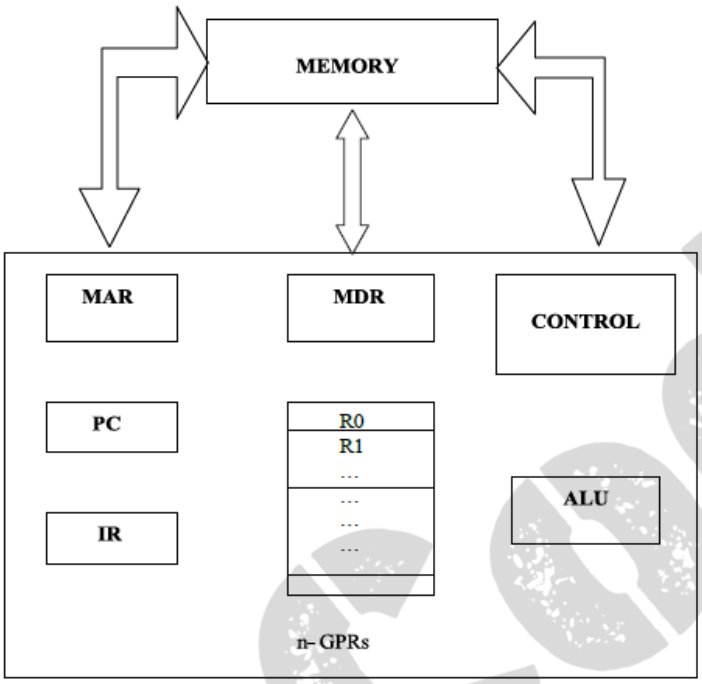


Fig b : Connections between the processor and the memory

The fig shows how memory & the processor can be connected. In addition to the ALU & the control circuitry, the processor contains a number of registers used for several different purposes.

The instruction register (IR):- Holds the instructions that is currently being executed. Its output is available for the control circuits which generates the timing signals that control the various processing elements in one execution of instruction.

The program counter PC:-

This is another specialized register that keeps track of execution of a program. It contains the memory address of the next instruction to be fetched and executed.

Besides IR and PC, there are n-general purpose registers R0 through R_{n-1}.

The other two registers which facilitate communication with memory are: -

1. **MAR – (Memory Address Register):-** It holds the address of the location to be accessed.
2. **MDR – (Memory Data Register):-** It contains the data to be written into or readout of the address location.

Operating steps are

1. Programs reside in the memory & usually get these through the I/P unit.
2. Execution of the program starts when the PC is set to point at the first instructionof the program.

3. Contents of PC are transferred to MAR and a Read Control Signal is sent to the memory.
4. After the time required to access the memory elapses, the address word is read out of the memory and loaded into the MDR.
5. Now contents of MDR are transferred to the IR & now the instruction is ready to be decoded and executed.
6. If the instruction involves an operation by the ALU, it is necessary to obtain the required operands.
7. An operand in the memory is fetched by sending its address to MAR & Initiatinga read cycle.
8. When the operand has been read from the memory to the MDR, it is transferred from MDR to the ALU.
9. After one or two such repeated cycles, the ALU can perform the desiredoperation.
10. If the result of this operation is to be stored in the memory, the result is sent to MDR.
11. Address of location where the result is stored is sent to MAR & a write cycle is initiated.
12. The contents of PC are incremented so that PC points to the next instruction thatis to be executed.

2. **Explain** Various **Addressing Modes** with examples.10

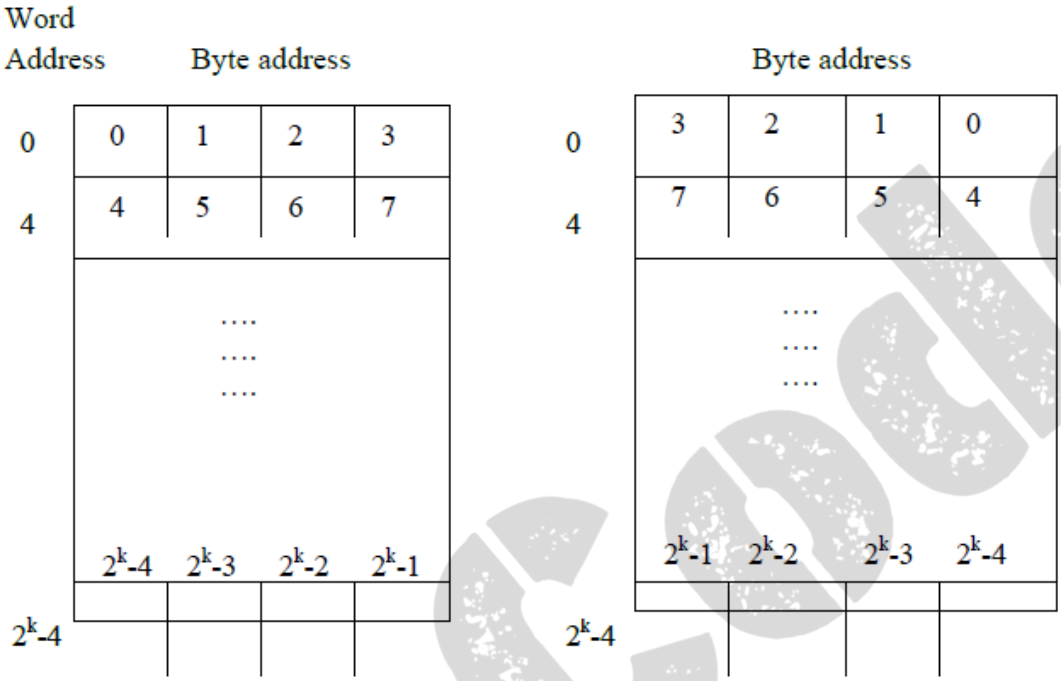
Name	Assembler syntax	Addressing function
Immediate	# Value	Operand = Value
Register	Ri	EA = Ri
Absolute (Direct)	LOC	EA = LOC
Indirect	(Ri)	EA = [Ri]
	(LOC)	EA = [LOC]
Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri, Rj)	EA = [Ri] + [Rj]
Base with index and offset	X (Ri, Rj)	EA = [Ri] + [Rj] + X
Relative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

EA = effective address
Value = a signed number

In general, a program operates on data that reside in the computer's memory. These data can be organized in a variety of ways. If we want to keep track of students' names, we can write them in a list. Programmers use organizations called data structures to represent the data used in computations. These include lists, linked lists, arrays, queues, and so on.

Programs are normally written in a high-level language, which enables the programmer to use constants, local and global variables, pointers, and arrays. The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

3. **Describe Big Endian** and **Little Endian** methods of byte addressing with proper example5



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

There are two ways that byte addresses can be assigned across words, as shown in fig b. 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. In addition to specifying the address ordering of bytes within a word, it is also necessary to specify the labeling of bits within a byte or a word. The same ordering is also used for labeling bits within a byte, that is, b_7, b_6, …., b_0, from left to right.

4. **Write** a note on **Bus** structure5

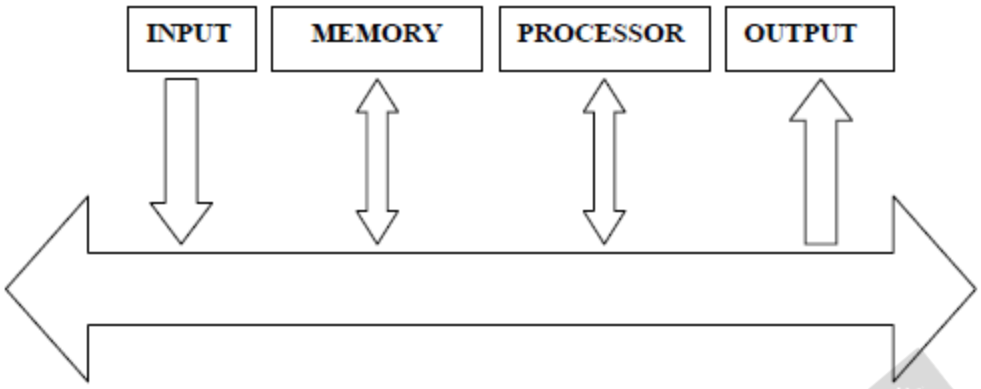


Fig c: Single bus structure

The simplest and most common way of interconnecting various parts of the computer. To achieve a reasonable speed of operation, a computer must be organized so that all its units can handle one full word of data at a given time.A group of lines thatserve as a connecting port for several devices is called a bus.
In addition to the lines that carry the data, the bus must have lines for address andcontrol purpose. Simplest way to interconnect is to use the single bus as shown.

Since the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time. Bus control lines are used to arbitrate multiple requests for use of one bus.
Single bus structure is

- Low cost
- Very flexible for attaching peripheral devices

Multiple bus structure certainly increases, the performance but also increases the cost significantly.

5. **Explain** How to measure **Performance** of a Computer ? 5

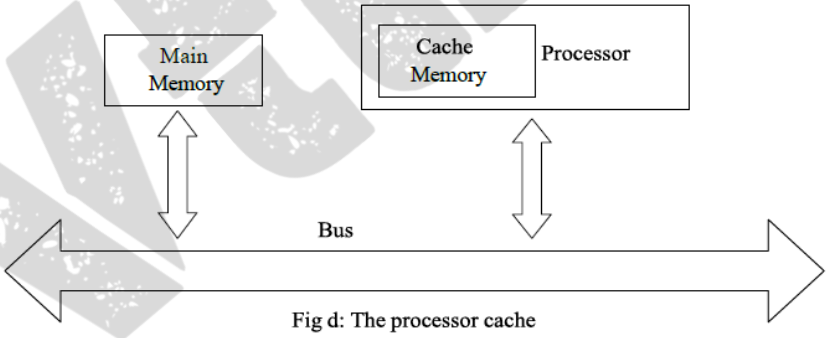


Fig d: The processor cache

At the start of execution, all program instructions and the required data are stored in the main memory. As the execution proceeds, instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache later if the same instruction or data item is needed a second time, it is read directly from the cache. The processor and relatively small cache memory can be fabricated on a single IC chip.
The internal speed of performing the basic steps of instruction processing on chip is very high and is considerably faster than the speed at which the instruction and data can be fetched from the main

memory. A program will be executed faster if the movement of instructions and data between the main memory and the processor is minimized, which is achieved by using the cache.
For example:- Suppose a number of instructions are executed repeatedly over a short period of time as happens in a program loop. If these instructions are available in the cache, they can be fetched quickly during the period of repeated use. The same applies to the data that are used repeatedly.

6. Compare Multiprocessors and Multicomputers.5

Multiprocessor & microprocessors:-

- Large computers that contain a number of processor units are called multiprocessor system.
- These systems either execute a number of different application tasks in parallel or execute subtasks of a single large task in parallel.
- All processors usually have access to all memory locations in such system & hence they are called shared memory multiprocessor systems.
- The high performance of these systems comes with much increased complexity and cost.
- In contrast to multiprocessor systems, it is also possible to use an interconnected group of complete computers to achieve high total computational power. These computers normally have access to their own memory units when the tasks they are executing need to communicate data they do so by exchanging messages over a communication network. This properly distinguishes them from shared memory multiprocessors, leading to name message-passing multi computer.

MODULE 4

1. **Explain** DMA transfer with bus arbitration.

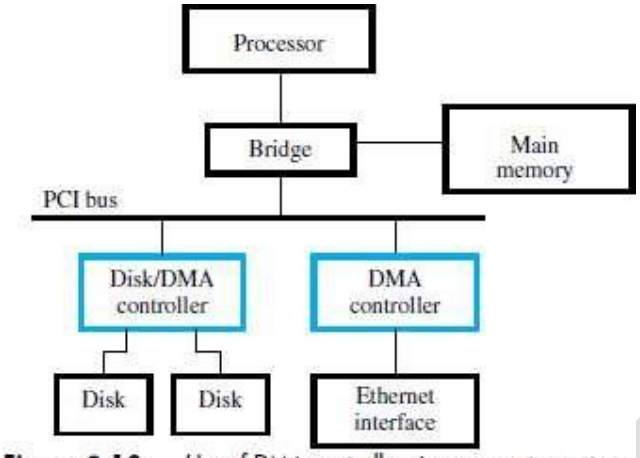


Figure 8.13 Use of DMA controllers in a computer system.

Figure 8.12 Typical registers in a DMA controller.

a. The transfer of a block of data directly b/w an external device & main-memory w/o continuous involvement by processor is called DMA.
b. DMA controller

→ is a control circuit that performs DMA transfers (Figure 8.13).
→ is a part of the I/O device interface.
→ performs the functions that would normally be carried out by processor.

While a DMA transfer is taking place, the processor can be used to execute anotherprogram

c. DMA interface has three registers (Figure8.12):
   i. First register is used for storing starting-address.
   ii. Second register is used for storing word-count.
   iii. Third register contains status- &control-flags.
d. The R/W bit determines direction of transfer.

If R/W=1, controller performs a read-operation (i.e. it transfers data from

memory to I/O), Otherwise, controller performs a write-operation (i.e. it transfers data from I/O to memory).

e. If Done=1, the controller
→ has completed transferring a block of data and
→ is ready to receive another command. (IE →Interrupt Enable).
f. If IE=1, controller raises an interrupt after it has completed transferring a block of data.
g. If IRQ=1, controller requests an interrupt.
h. Requests by DMA devices for using the bus are always given higher priority than processor requests.
i. There are 2 ways in which the DMA operation can be carried out:
    i. Processor originates most memory-access cycles.
        1. DMA controller is said to "steal" memory cycles from processor.
        2. Hence, this technique is usually called **Cycle Stealing**.
    ii. DMA controller is given exclusive access to main-memory to transfer a block of data without any interruption. This is known as **Block Mode** (or burst mode).

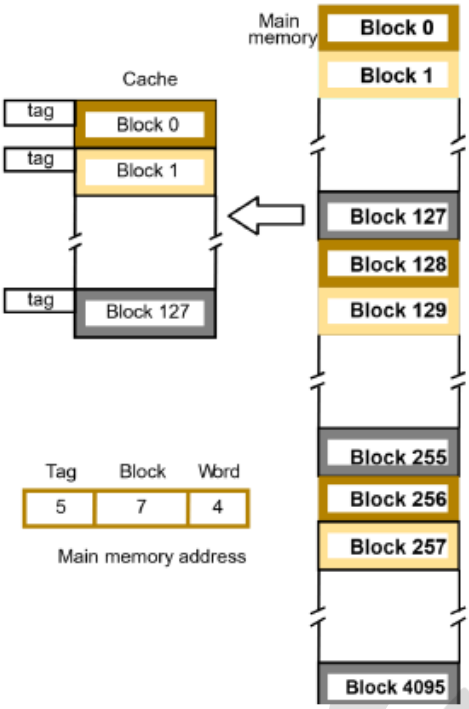
2. What is cache memory? **Explain** different mapping function with diagram.

Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is. Cache memory is based on the property of computer programs known as “locality of reference”.

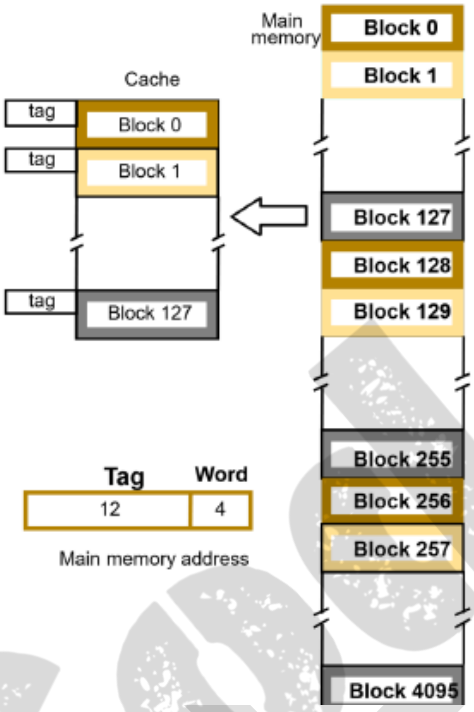
Three mapping functions can be used.

1. Direct mapping
2. Associative mapping
3. Set-associative mapping.

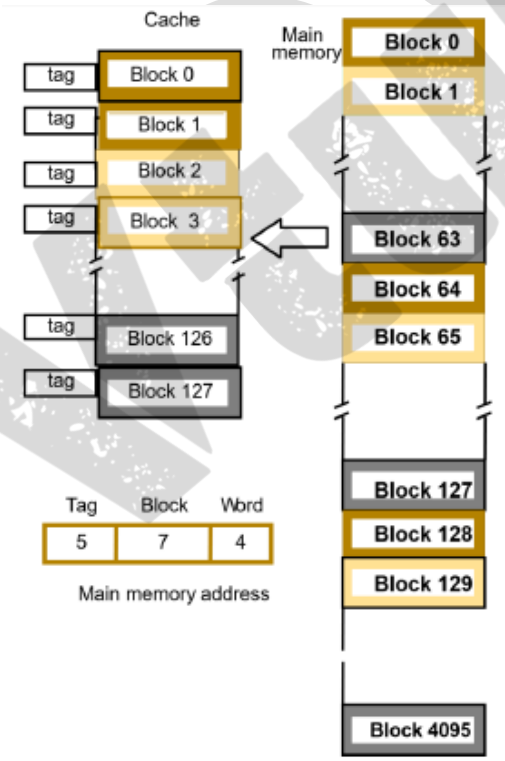
Direct mapping



Associative mapping



Set-associative mapping



3. **Explain** centralized bus arbitration

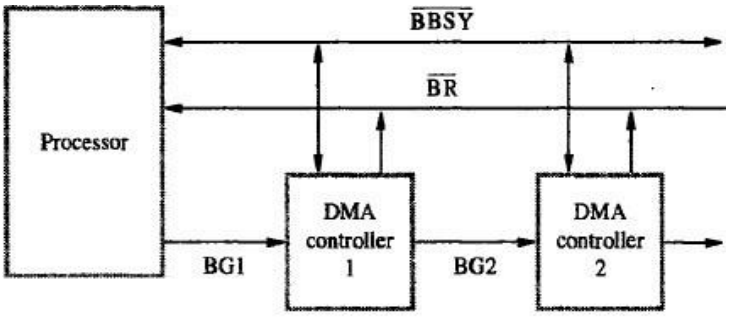


Figure 4.20 A simple arrangement for bus arbitration using a daisy chain.

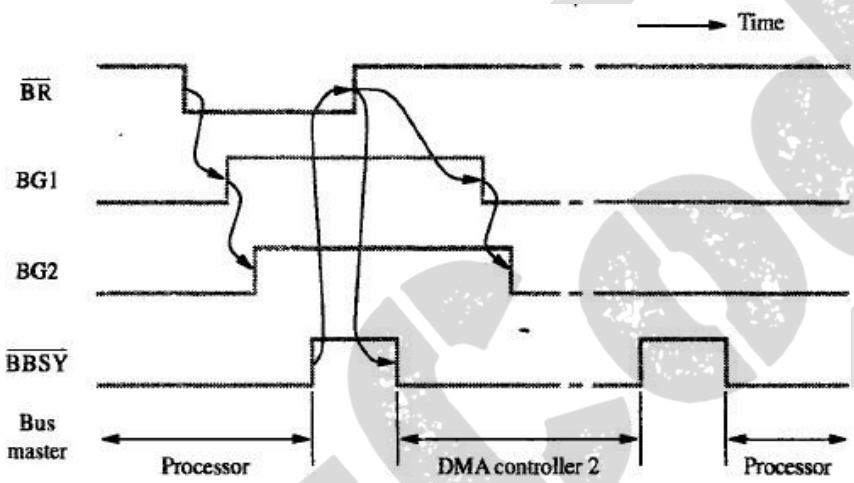


Figure 4.21 Sequence of signals during transfer of bus mastership for the devices in Figure 4.20.

a. A single bus-arbiter performs the required arbitration (Figure:4.20).
b. Normally, processor is the bus-master.
c. Processor may grant bus-mastership to one of the DMA controllers.
d. A DMA controller indicates that it needs to become bus-master by activating BRline.
e. The signal on the BR line is the logical OR of bus-requests from all devices connected to it.
f. Then, processor activates BG1 signal indicating to DMA controllers to use bus whenit becomes free.
g. BG1 signal is connected to all DMA controllers using a daisy-chain arrangement.
h. If DMA controller-1 is requesting the bus,

Then, DMA controller-1 blocks propagation of grant-signal to other devices. Otherwise, DMA controller-1 passes the grant downstream by asserting BG2.

i. Current bus-master indicates to all devices that it is using bus by activating BBSYline.
j. The bus-arbiter is used to coordinate the activities of all devices

requesting memory transfers.

k. Arbiter ensures that only 1 request is granted at any given time according to a priority scheme. (BR →Bus-Request, BG →Bus-Grant, BBSY →Bus Busy).

l. The timing diagram shows the sequence of events for the devices connected to theprocessor.

m. DMAcontroller-2

→ requests and acquires bus-mastership and
→ later releases the bus. (Figure: 4.21).

n. After DMA controller-2 releases the bus, the processor resources bus-mastership.

4. **Explain** synchronous bus transfer during input operation

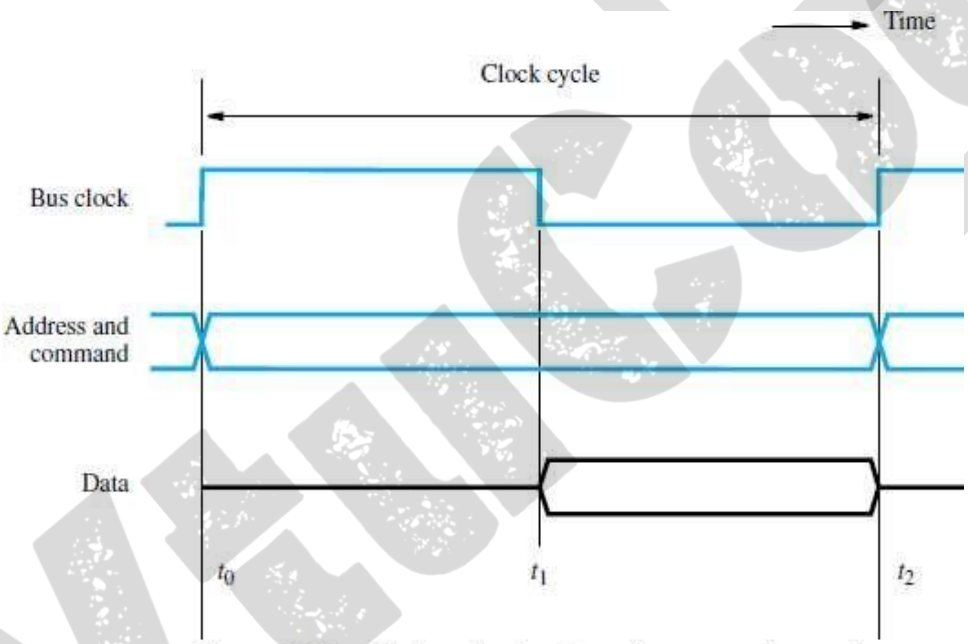


Figure 7.3 Timing of an input transfer on a synchronous bus.

a. All devices derive timing-information from a common clock-line.
b. Equally spaced pulses on this line define equal time intervals.
c. During a "bus cycle", one data-transfer can take place.

A sequence of events during a read-operation

d. At time t0, the master(processor)

→ places the device-address on address-lines &
→ sends an appropriate command on control-lines (Figure 7.3).

e. The command will

→ indicate an input operation &
→ specify the length of the operand to be read.

f. Information travels over bus at a speed determined by physical &

electrical characteristics.

g. Clock pulse width(t_1-t_0) must be longer than max. propagation-delay b/w devices connected to bus.
h. The clock pulse width should be long to allow the devices to decode the address & control signals.
i. The slaves take no action or place any data on the bus before t_1.
j. Information on bus is unreliable during the period t_0 to t_1 because signals are changing state.
k. Slave places requested input-data on data-lines at time t_1.
l. At end of clock cycle (at time t_2), master strobes (captures) data on data-lines into its input-buffer
m. For data to be loaded correctly into a storage device,

data must be available at input of that device for a period greater than setup-time of device.

MODULE 5

1. **Illustrate** the sequence of operation required to execute the instruction **ADD (R3), R1** on single bus processor.

   Consider the instruction *Add (R3),R1* which adds the contents of a memory-location pointed by R3 to register R1. Executing this instruction requires the following actions:

   1) Fetch the instruction.
   2) Fetch the first operand.
   3) Perform the addition &
   4) Load the result into R1.

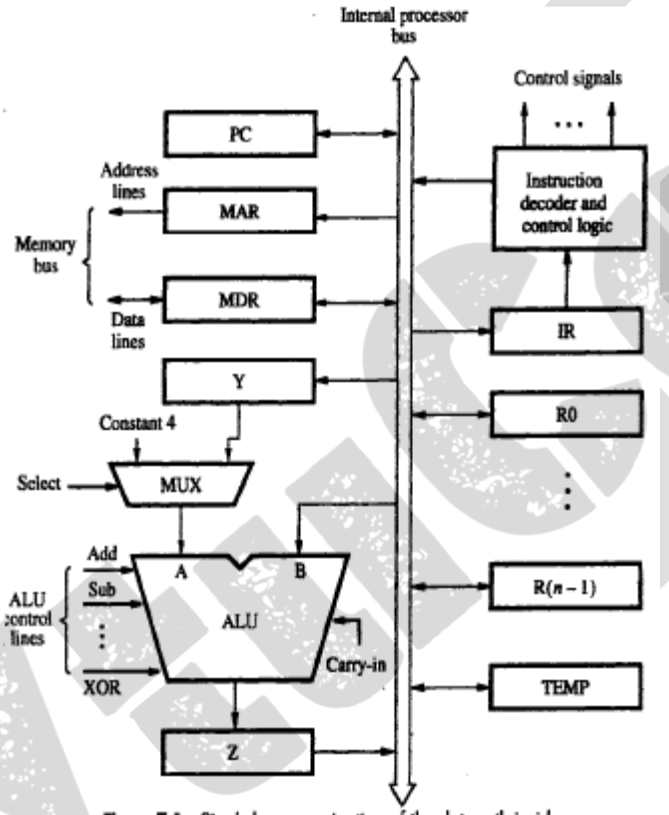


Figure 7.1 Single-bus organization of the datapath inside a processor.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

Figure 7.6 Control sequence for execution of the instruction Add (R3),R1

2. With a neat diagram, **Explain** multiple bus organization of computer and functional concepts

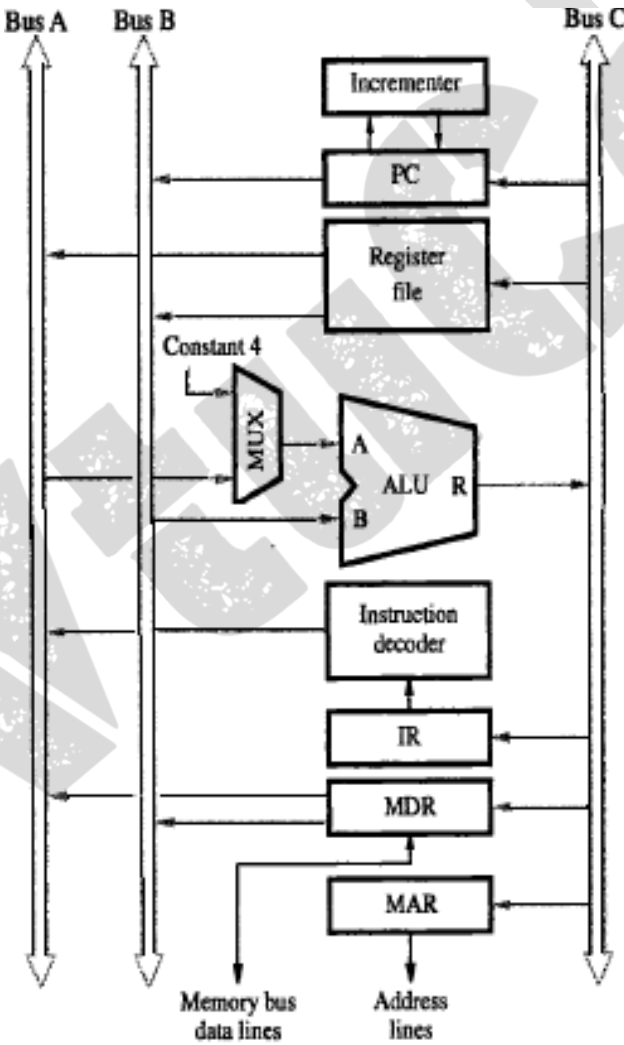


Figure 7.8 Three-bus organization of the datapath.

Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

Figure 7.9 Control sequence for the instruction Add R4,R5,R6

Disadvantage of Single-bus organization: Only one data-word can be transferred over the bus in a clock cycle. This increases the steps required to complete the execution of the instruction

Solution: To reduce the number of steps, most processors provide multiple internal-paths. Multiple paths enable several transfers to take place in parallel.

• As shown in fig 7.8, three buses can be used to connect registers and the ALU of the processor.

• All general-purpose registers are grouped into a single block called the **Register File**.

• Register-file has 3 ports:

1) Two output-ports allow the contents of 2 different registers to be simultaneously placed on buses A & B.

2) Third input-port allows data on bus C to be loaded into a third register during the same clock-cycle.

• Buses A and B are used to transfer source-operands to A & B inputs of ALU.

• The result is transferred to destination over bus C.

• **Incrementer Unit** is used to increment PC by 4.

3. **Explain** pipeline performance

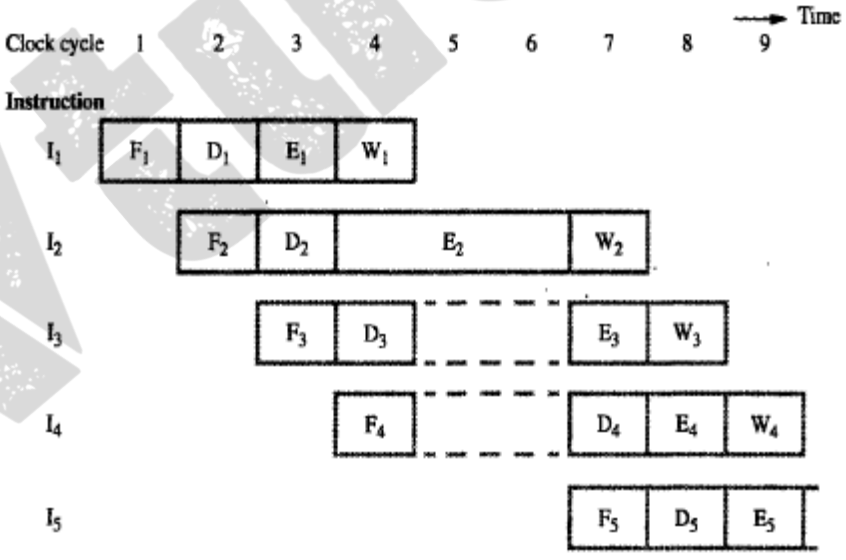


Figure 8.3 Effect of an execution operation taking more than one clock cycle.

Figure 8.3 shows an example in which the operation specified in instruction I_2 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, stage 1 are blocked from accepting new instructions because the information in B1 cannot be overwritten. Thus, steps D_4 and F_5 must be postponed as shown.

4. **Explain** basic idea of pipelining and 4 stage pipelining

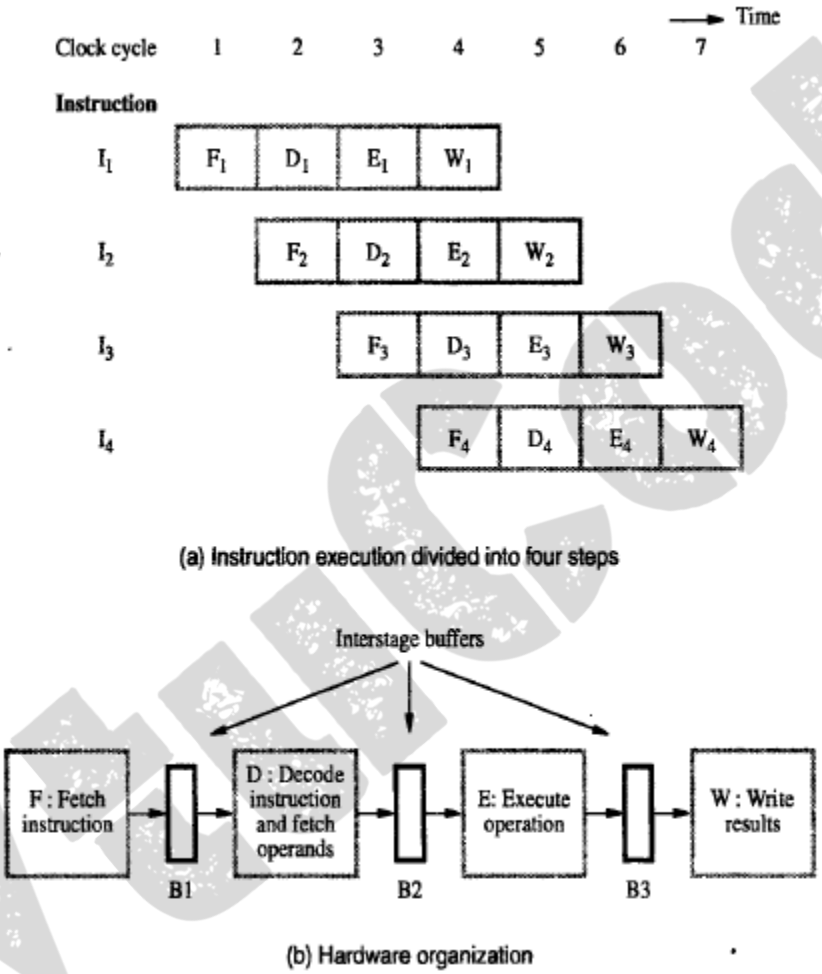


Figure 8.2 A 4-stage pipeline.

This consideration is particularly important for the instruction fetch step, which is assigned one clock period in Figure 8.2*a*. The clock cycle has to be equal to or greater than the time needed to complete a fetch operation. However, the access time of the main memory may be as much as ten times greater than the time needed to perform basic pipeline stage operations inside the processor, such as adding two numbers. Thus, if each instruction fetch required access to the main memory, pipelining would be of little value.

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