q1)what is register transfer language?
sol)

- The Register Transfer Language is the symbolic representation of notations used to specify the sequence of micro-operations.
- In a computer system, data transfer takes place between processor registers and memory and between processor registers and input-output systems.

These data transfer can be represented by standard notations given below:

1.Notations R0, R1, R2..., and so on represent processor registers.

2.The addresses of memory locations are represented by names such as LOC, PLACE, MEM, etc.

3.Input-output registers are represented by names such as DATA IN, DATA OUT and so on.

4.The content of register or memory location is denoted by placing square brackets around the name of the register or memory location.

---

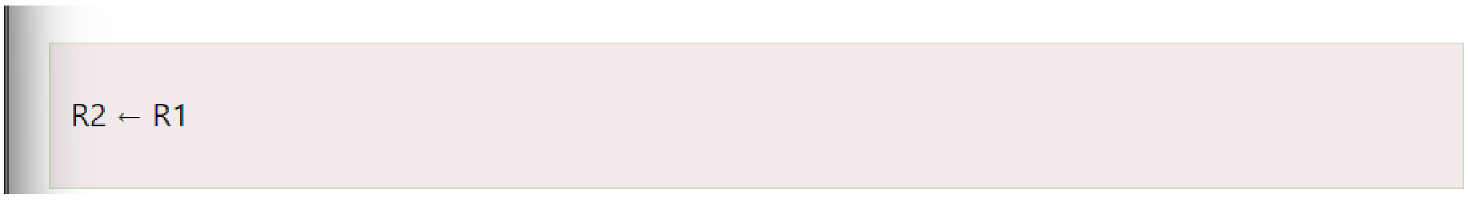
q2)what is register transfer?
sol)

- Information transfer from one register to another register is designated in symbolic form by means of a replacement operator
- Most of the standard notations used for specifying operations on various registers are stated below.

1. The memory address register is designated by MAR.
2. Program Counter PC holds the next instruction's address.
3. Instruction Register IR holds the instruction being executed.
4. R1 (Processor Register).

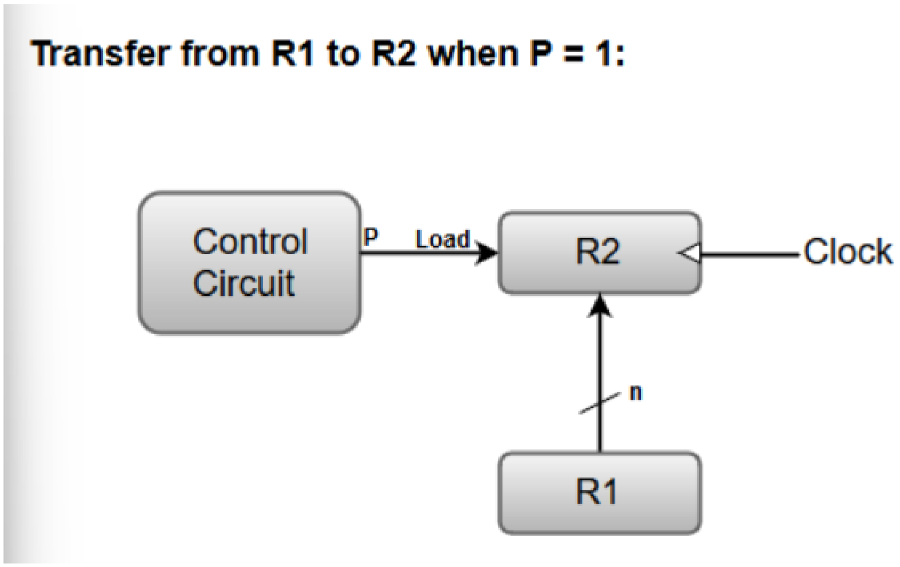
5. We can also indicate individual bits by placing them in parenthesis. For instance, PC (8-15), R2 (5), etc.

6. Data Transfer from one register to another register is represented in symbolic form by means of replacement operator.



```
R2 ← R1
```

- The following diagram shows the transfer of data from R1 to R2.



- Here, the letter 'n' indicates the number of bits for the register.
- The statement R2← R1 denotes a transfer of the content of register R1 into register R2.
- It designates a replacement of the content of R2 by the content of R1

The basic symbols of the register transfer notation are listed in below table:

Symbol	Description	Examples
Letters(and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow <--	Denotes transfer of information	R2 <-- R1
Comma ,	Separates two microoperations	R2 <-- R1, R1 <-- R2

....

q3)explain bus and memory transfer?

sol)

- A more efficient scheme for transferring information between registers in a multiple-register configuration is a Common Bus System.
- A common bus consists of a set of common lines, one for each bit of a register.

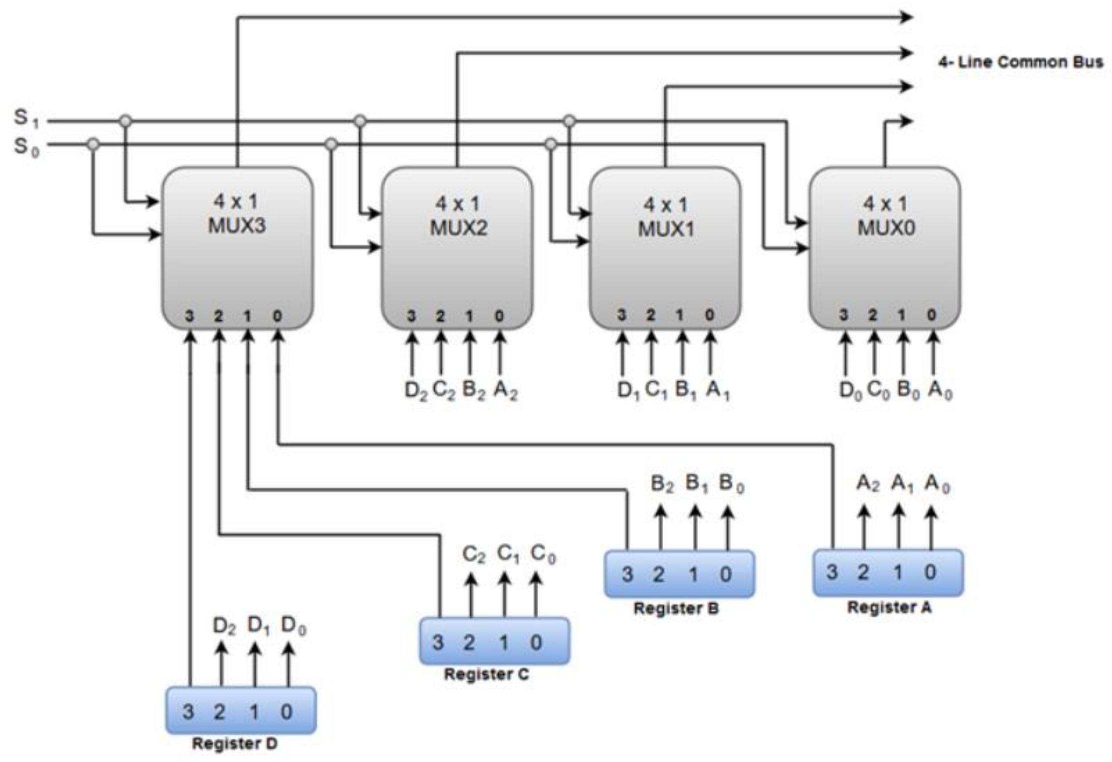
Different ways of constructing a Common Bus System:

1. Using Multiplexers
2. Using Three-state Buffers

1.Common bus system is with using multiplexers:

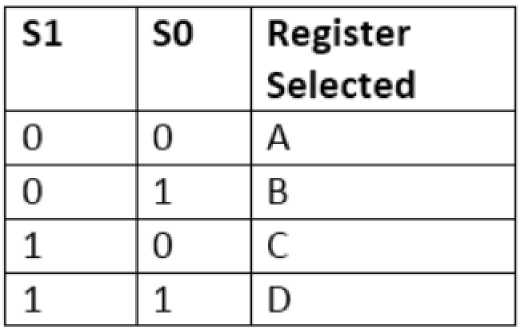
- The multiplexers select the source register whose binary information is then placed on the bus.
- The construction of a bus system for four registers is shown in below Figure.
-

Bus System for 4 Registers:



-
- The bus consists of four 4 x 1 multiplexers each having four data inputs and two selection inputs(S1 and S0)
- The two selection lines S1 and So are connected to the selection inputs of all four multiplexers.
- The selection lines choose the four bits of one register and transfer them into the four-line common bus.

- THE table shows the register that is selected by the bus for each of the four possible binary values of the Selection lines.



S1	S0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

In general a bus system has:

1. multiplex “k” Registers
2. each register of “n” bits

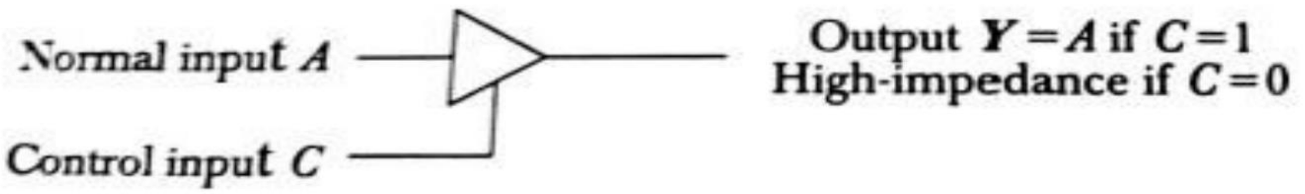
3. to produce “n-line bus”
4. no. of multiplexers required = n
5. size of each multiplexer = k x 1

2. Common bus system is with Three-state Buffers:

- A bus system can be constructed with three-state gates instead of multiplexers.
-
- A three-state gate is a digital circuit that exhibitsthree states.
- Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate.
- The third state is a high-impedance state.
- The high-impedance state behaves like an open circuit,which means that the output is disconnected and does not have logic significance
- Because of this feature, a large number of three-state gate outputs can be connected with wires to form a common bus line
- The graphic symbol of a three-state buffer gate is shown in Fig

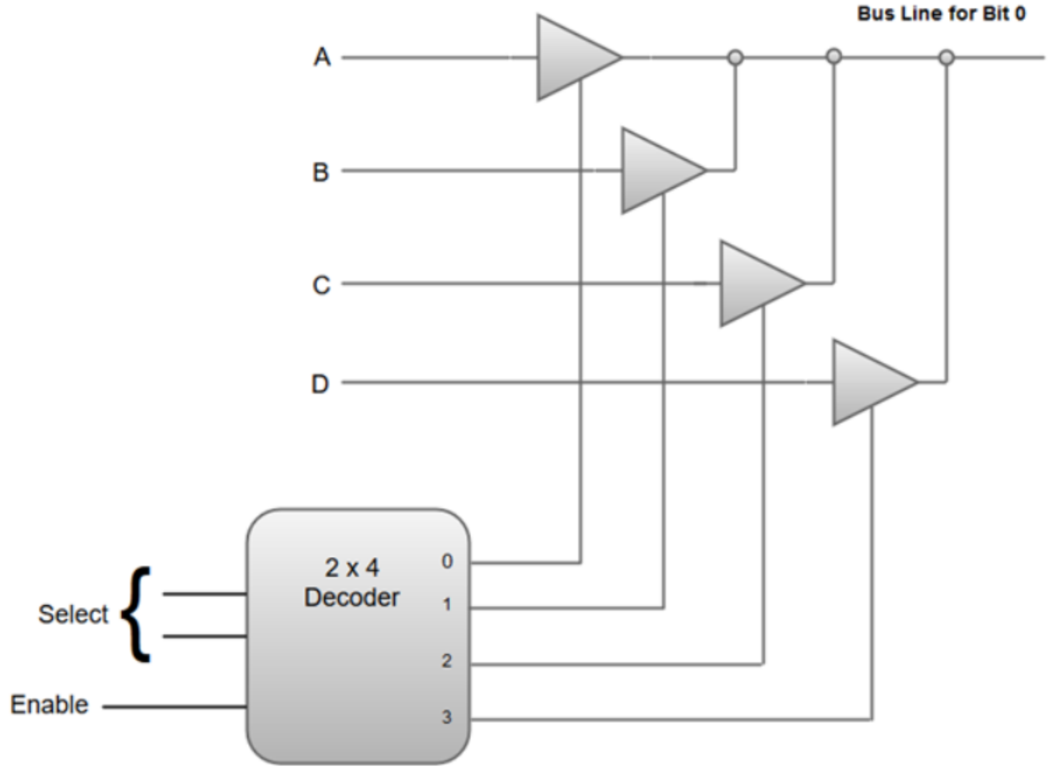
➢

Figure 4-4 Graphic symbols for three-state buffer.



- the difference is the normal buffer having both a normal input and a control input.
- The control input determines the output state.
- When the control input is 0, the output is disabled and the gate goes to a high-impedance state
- The construction of a bus system with three-state buffers is shown in Fig.

Bus line with three state buffer:



-
- The outputs of four buffers are connected together to form a single bus line.
- No more than one buffer may be in the active state at any given time.
- When the enable input of the decoder is 0, all of its four outputs are 0
- When the enable input is active, one of the three-state buffers will be active

memory transfer:

- The transfer of information from a memory unit to the user end is called a Read operation.
- The transfer of new information to be stored in the memory is called a Write operation.
- A memory word will be denoted by the letter M.
- It is necessary to specify the address of M when writing memory transfer operations.
-
- Consider a memory unit that receives the address from a register, called the address register, symbolized by AR.
- The data are transferred to another register, called the data register, symbolized by DR.

- The read operation can be stated as follows:

Read: DR<- M [AR]

where DR=data register

AR=address register

M=symbol of memory transfer

The write operation can be stated as follows:

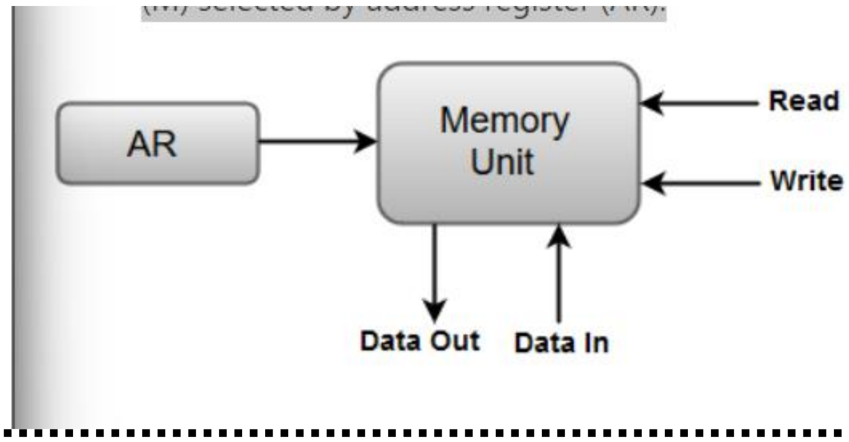
Write: M [AR] <- R1

where M=symbol of memory transfer

AR=address register

R1=register

- The Write statement causes a transfer of information from register R1 into the memory word (M) selected by address register (AR).



-

Q4)listout types of micro-operations?

sol)

- they are 4 micro-operations
  1. Register Transfer Micro-operations
  2. Arithmetic Micro-operations
  3. Logical Micro-operations

4. Shift Micro-operations

....

q5)explain Arithmetic Micro-operations?

sol)

- The basic Arithmetic Micro-operations are

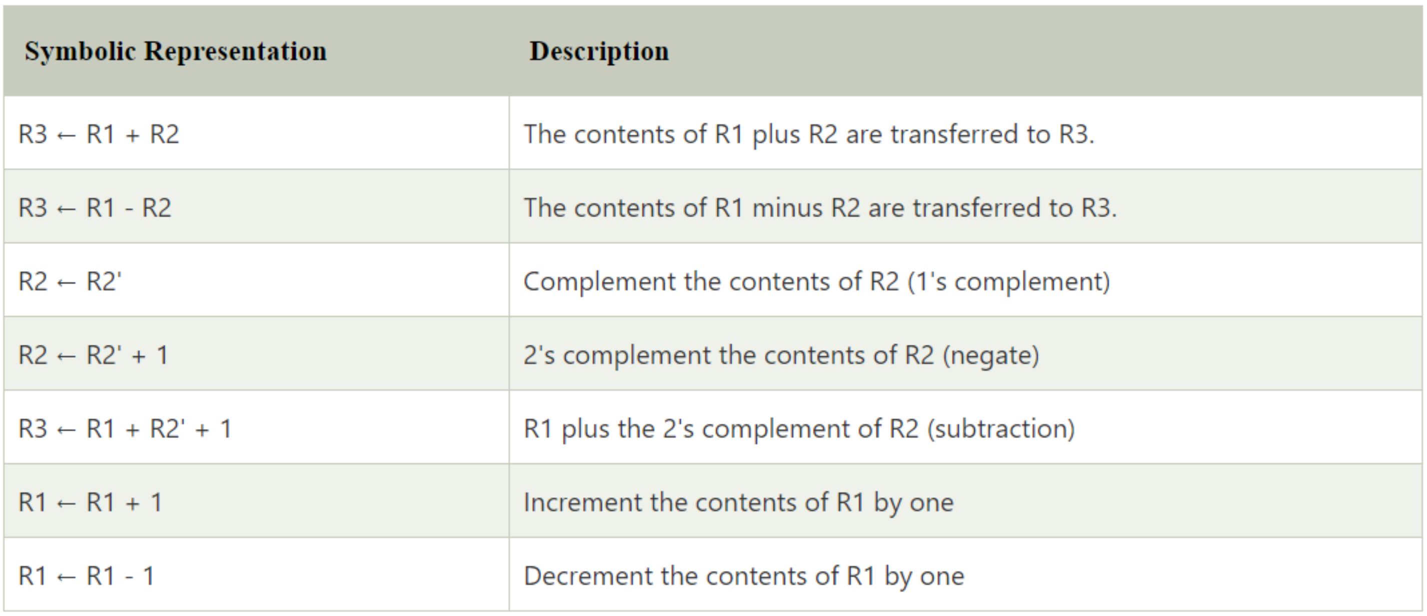
1.Addition
2.Subtraction
3.Increment
4.Decrement
5.Shift

Some additional Arithmetic Micro-operations are classified as:

1. Add with carry
2.Subtract with borrow

3.Transfer/Load, etc.

The following table shows the symbolic representation of various Arithmetic Micro-operations.

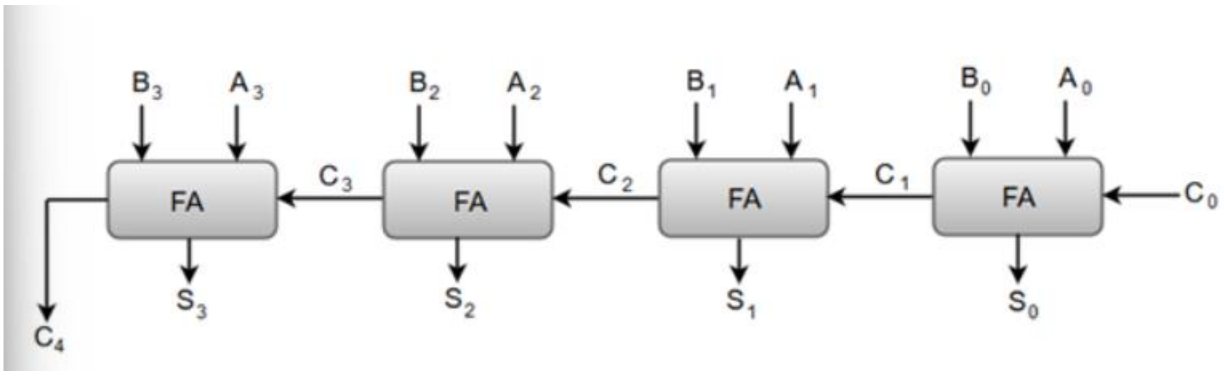


Symbolic Representation	Description
R3 ← R1 + R2	The contents of R1 plus R2 are transferred to R3.
R3 ← R1 - R2	The contents of R1 minus R2 are transferred to R3.
R2 ← R2'	Complement the contents of R2 (1's complement)
R2 ← R2' + 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2' + 1	R1 plus the 2's complement of R2 (subtraction)
R1 ← R1 + 1	Increment the contents of R1 by one
R1 ← R1 - 1	Decrement the contents of R1 by one

q6)explain binary adder?

sol)

- Digital circuit that formsthe arithmetic sum of 2 bits and the previous carry is called FULL ADDER.

- Digital circuit that generates the arithmetic sum of 2 binary numbers of any lengths is called BINARY ADDER
- the foloowing diagram shows the 4 bit binary adder
- 

- The 'A' and the 'B' are designated by subscript numbers from right to left, with subscript '0' denoting the low-order bit.
- The carry inputs starts from C0 to C3 connected in a chain through the full-adders
- C4 is the resultant output carry generated by the last full-adder circuit.

- The output carry from each full-adder(fa) is connected to the input carry of the next-high-order full-adder(FA).
- outputs (S0 to S3) generates the required arithmetic sum of 'A' and 'B' bits.

....

Q7) EXPLAIN Binary Adder-Subtractor?
SOL)

- The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.
- A 4-bit adder-subtractor circuit is shown in Fig

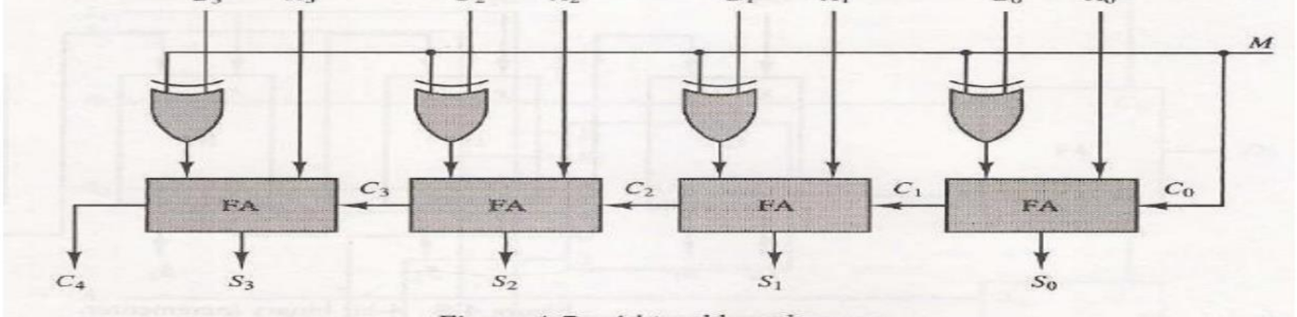


Figure 4-7 4-bit adder-subtractor.

- The mode input(M) controls the operation.
- When M = 0 the circuit is an adder and when M = 1 the circuit becomes a subtractor
- Each exclusive-OR gate receives input 'B' 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.
- The B inputs are all complemented and 1 is added through the input carry.
- The circuit performs the operation A plus the 2's complement of B.

....

q8) explain Binary Incrementer?
sol)

- The increment microoperation adds one to a number in a register.
- For example, if a 4-bit register has a binary value 0110, it will go to 0111 after it is incremented.
- this can be performed by using half-adders(HA)
- the diagram shows the 4-bit binary incrementer
-

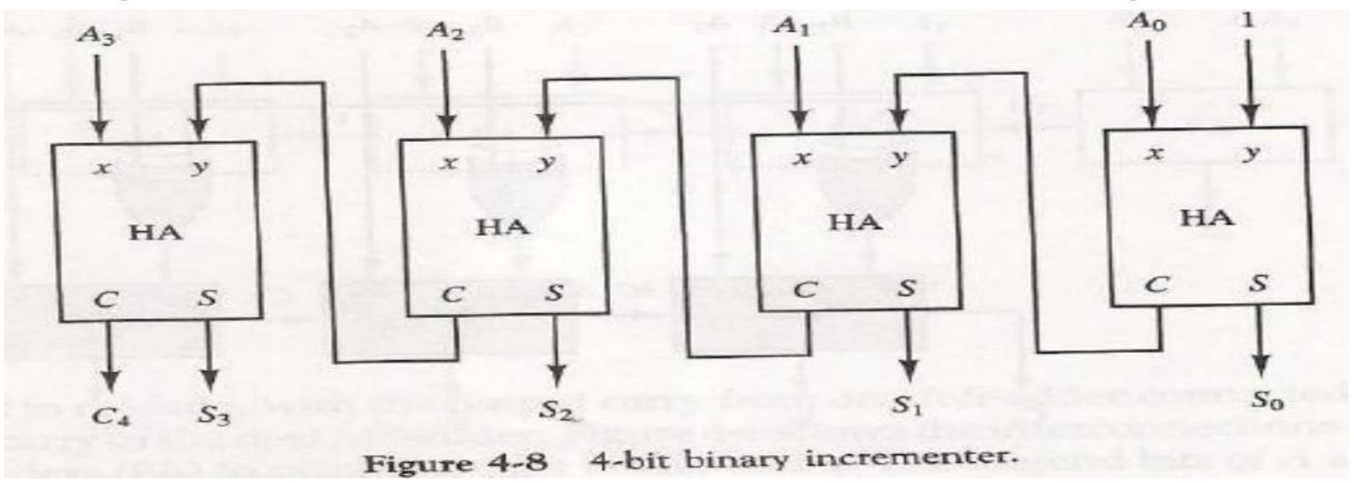


Figure 4-8 4-bit binary incrementer.

- The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder.
- The circuit receives the four bits from A0 through A3, adds one to it, and generates the incremented output in S0 through S3.
- The output carry C4 will be 1 only after incrementing binary 1111.

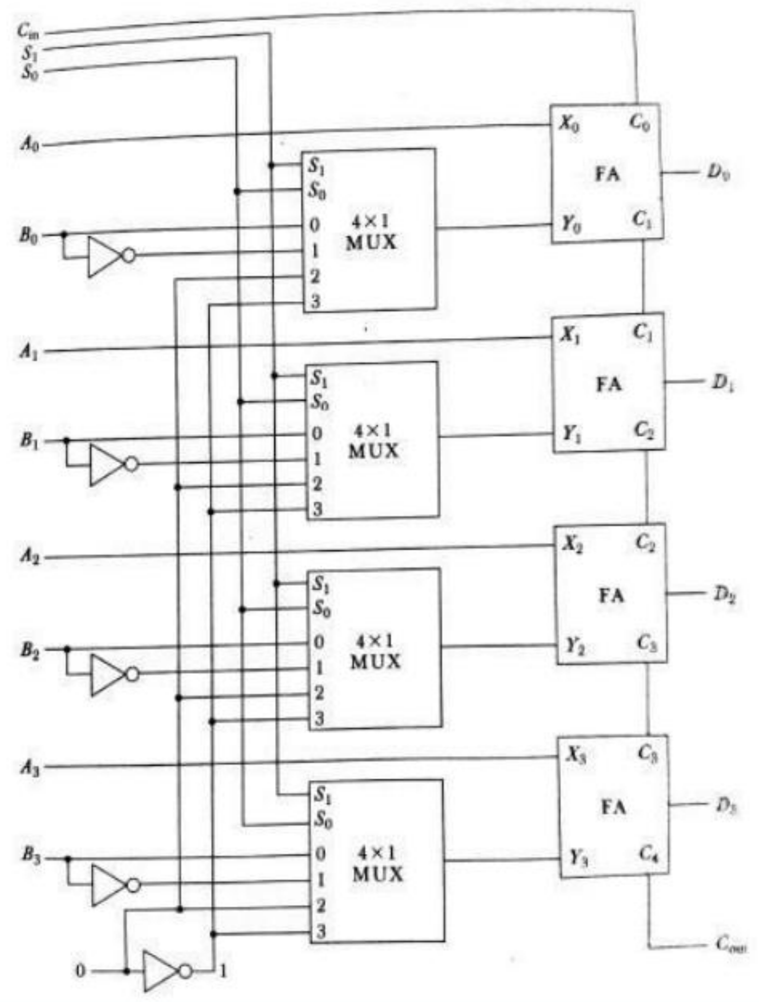
....

q9)explain Arithmetic Circuits?
sol)

- Arithmetic circuits can perform seven different arithmetic operations using a single composite circuit.
- It uses a full adder (FA) to perform these operations
- A multiplexer (MUX) is used to provide different inputs to the circuit in order to

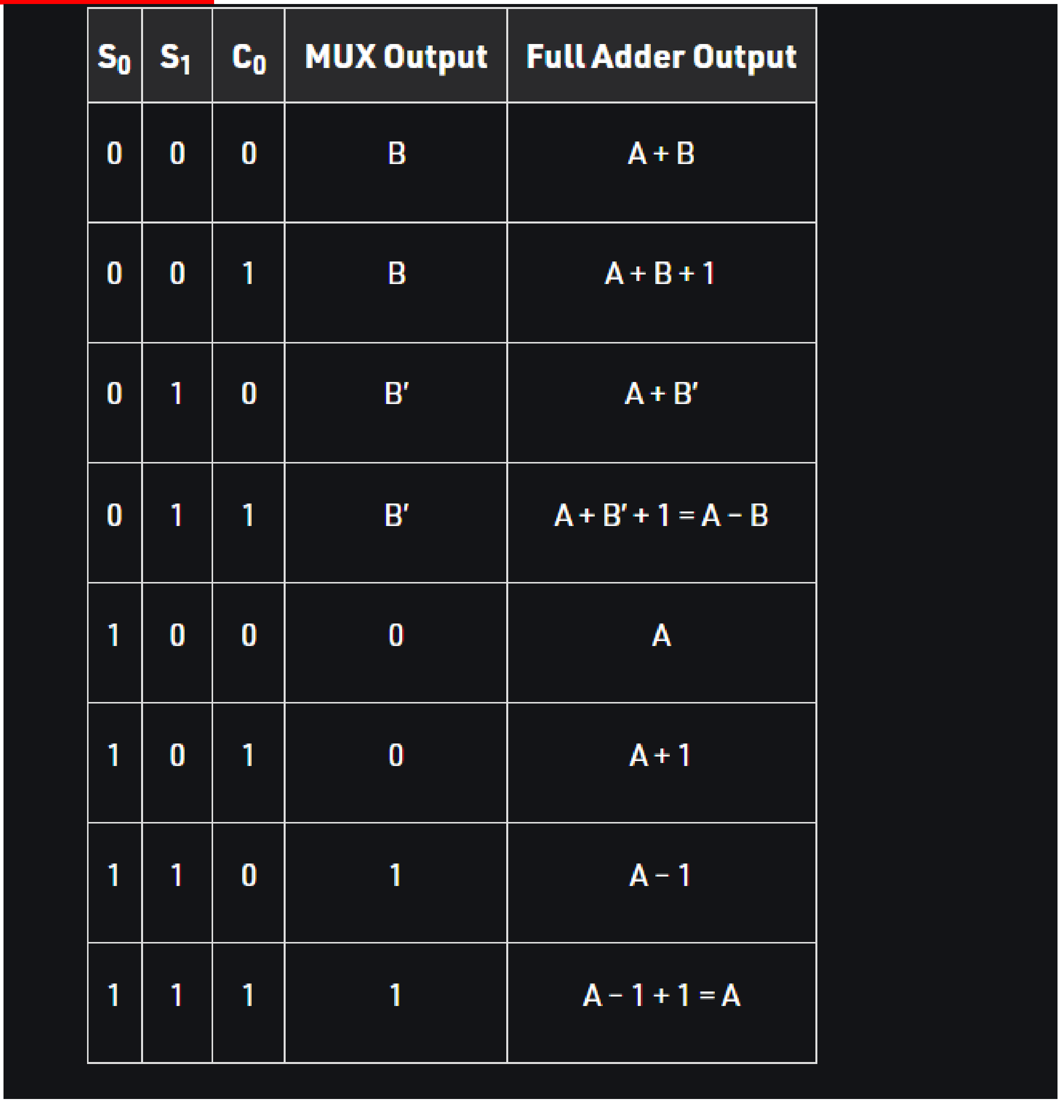
obtain different arithmetic operations as outputs.

- let Consider the following 4-bit Arithmetic circuit with inputs A and B
- It can perform seven different arithmetic operations by varying the inputs of the multiplexer and the carry (C0).



-

➢ Truth Table for the above Arithmetic Circuit :



S_0	S_1	C_0	MUX Output	Full Adder Output
0	0	0	B	$A + B$
0	0	1	B	$A + B + 1$
0	1	0	B'	$A + B'$
0	1	1	B'	$A + B' + 1 = A - B$
1	0	0	0	A
1	0	1	0	$A + 1$
1	1	0	1	$A - 1$
1	1	1	1	$A - 1 + 1 = A$

➢

....

q10)explain Logical Micro-operations?

sol)

- Logiacal micro operations specify binary operations for strings of bits stored in registers
- These operations consider each bit of the register separately and treat them as binary variables
- There are 16 different logic operations that can be performed with two binary variables.
- They can be determined from all possible truth tables obtained with two binary variables as shown in Table
-

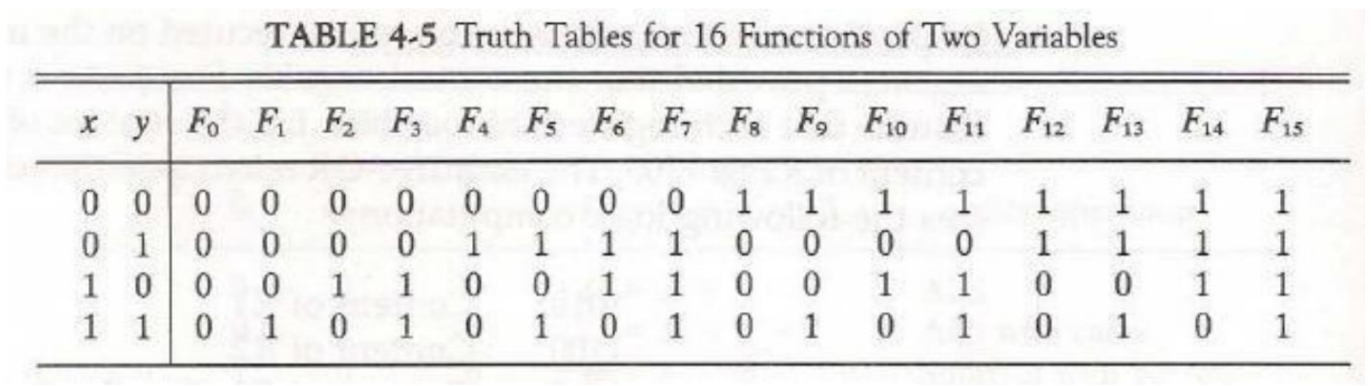


TABLE 4-5 Truth Tables for 16 Functions of Two Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- The 16 Boolean functions of two variables x and y are expressed in algebraic form

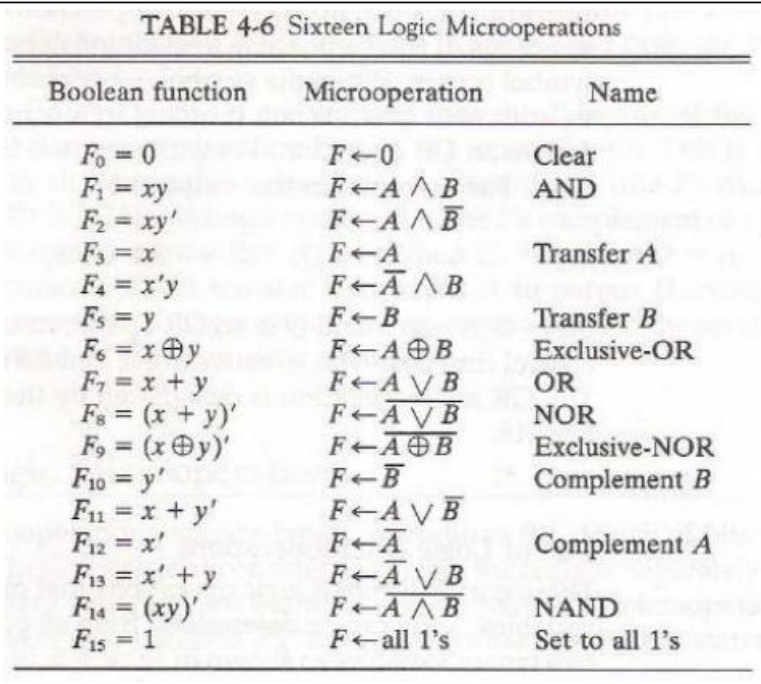


TABLE 4-6 Sixteen Logic Microoperations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

-

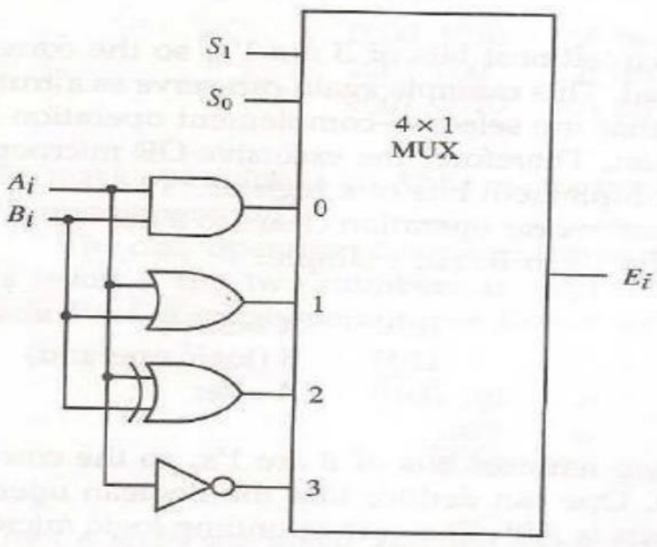
Hardware Implementation:

1. The hardware implementation of logic micro operations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function.

2. there are 16 logic micro operations, most computers use only four--AND, OR, XOR

3. below diagram shows that circuit that generates the four basic logic micro operations.

Figure 4-10 One stage of logic circuit.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

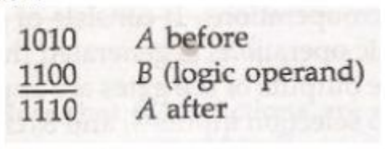
(b) Function table

4. It consists of four gates and one multiplexer

5. The two selection inputs S1 and S0 choose one of the data inputs of the multiplexer and direct its value to the output.

Some Applications:

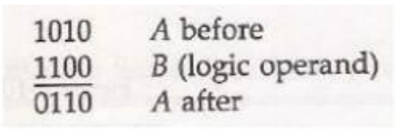
1. Selective set:

1. The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B.
2. It does not affect bit positions that have 0's in B.
3. The followingnumerical example clarifies this operation:
4. 

1010	*A* before
1100	*B* (logic operand)
1110	*A* after

2.Selective complement:

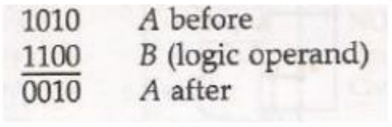
1. The selective-complement operation complements bits in A where there are corresponding 1's in B
2. It does not affect bit positions that have 0's in B
3. For example:



1010	*A* before
1100	*B* (logic operand)
0110	*A* after

3. Selective clear:

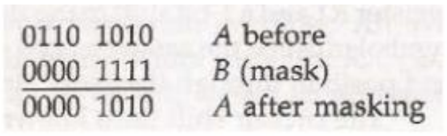
1. The selective-clear operation clears to 0 the bits in A only where there are corresponding l's in B.
2. For example:
3.



1010	*A* before
1100	*B* (logic operand)
0010	*A* after

4.Mask:

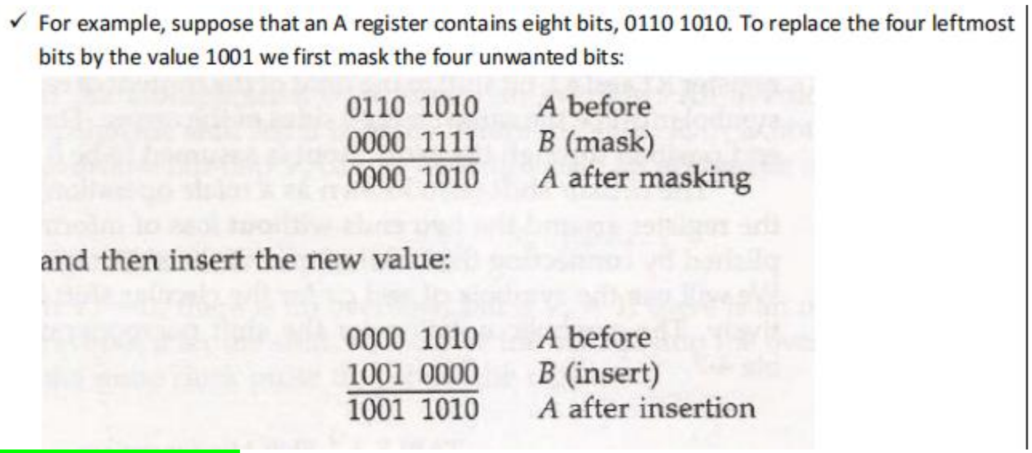
1. The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding O's in B

2. 

```
0110 1010    A before
0000 1111    B (mask)
---------
0000 1010    A after masking
```

5.Insert:

1. the insert operation inserts a new value into a group of bits.



✓ For example, suppose that an A register contains eight bits, 0110 1010. To replace the four leftmost bits by the value 1001 we first mask the four unwanted bits:

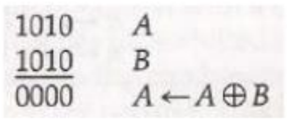
```
0110 1010    A before
0000 1111    B (mask)
---------
0000 1010    A after masking
```

and then insert the new value:

```
0000 1010    A before
1001 0000    B (insert)
---------
1001 1010    A after insertion
```

6.Clear:

1. The clear operation compares the words in A and B and produces an all 0's result if the two numbers are equal.



$$\begin{array}{ll} 1010 & A \\ \underline{1010} & B \\ 0000 & A \leftarrow A \oplus B \end{array}$$

....

q11)explain shift micro operation?
sol)

➢ Shift microoperations are used for serial transfer of data.

➢

➢ The contents of a register can be shifted to the left or the right.

➢

- During a shift-left operation the serial input transfers a bit into the rightmost position.
-
- During a shift-right operation the serial input transfers a bit into the leftmost position.

There are three types of shifts:

1.logical
2.circular
3.arithmetic.

The symbolic notation for the shift microoperations is shown in Table

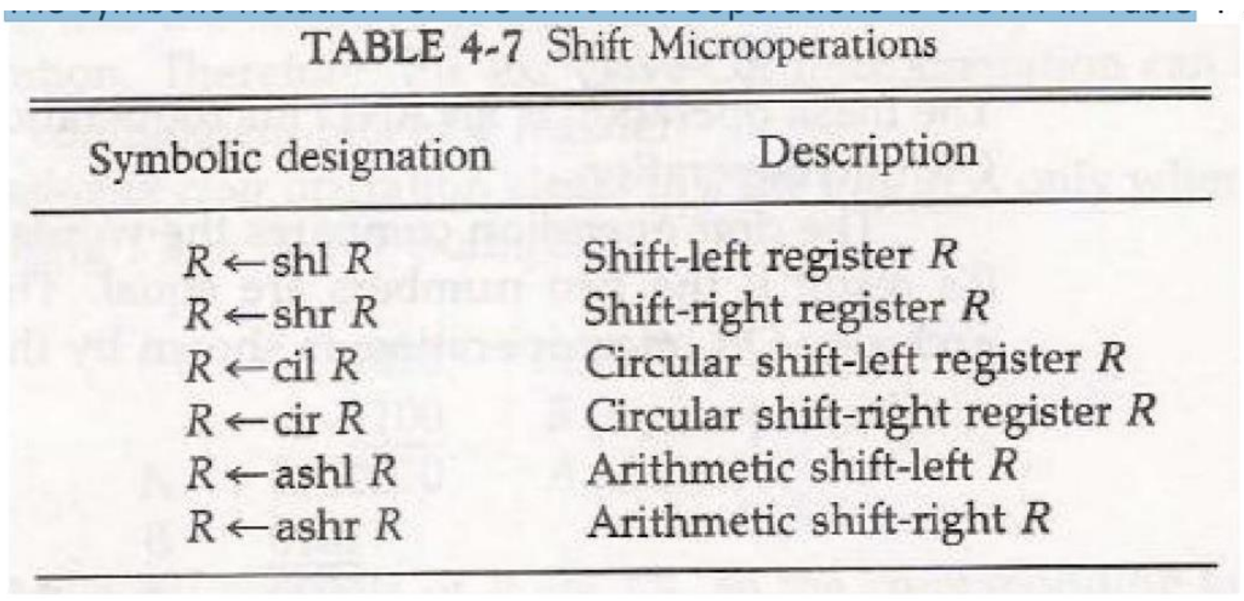
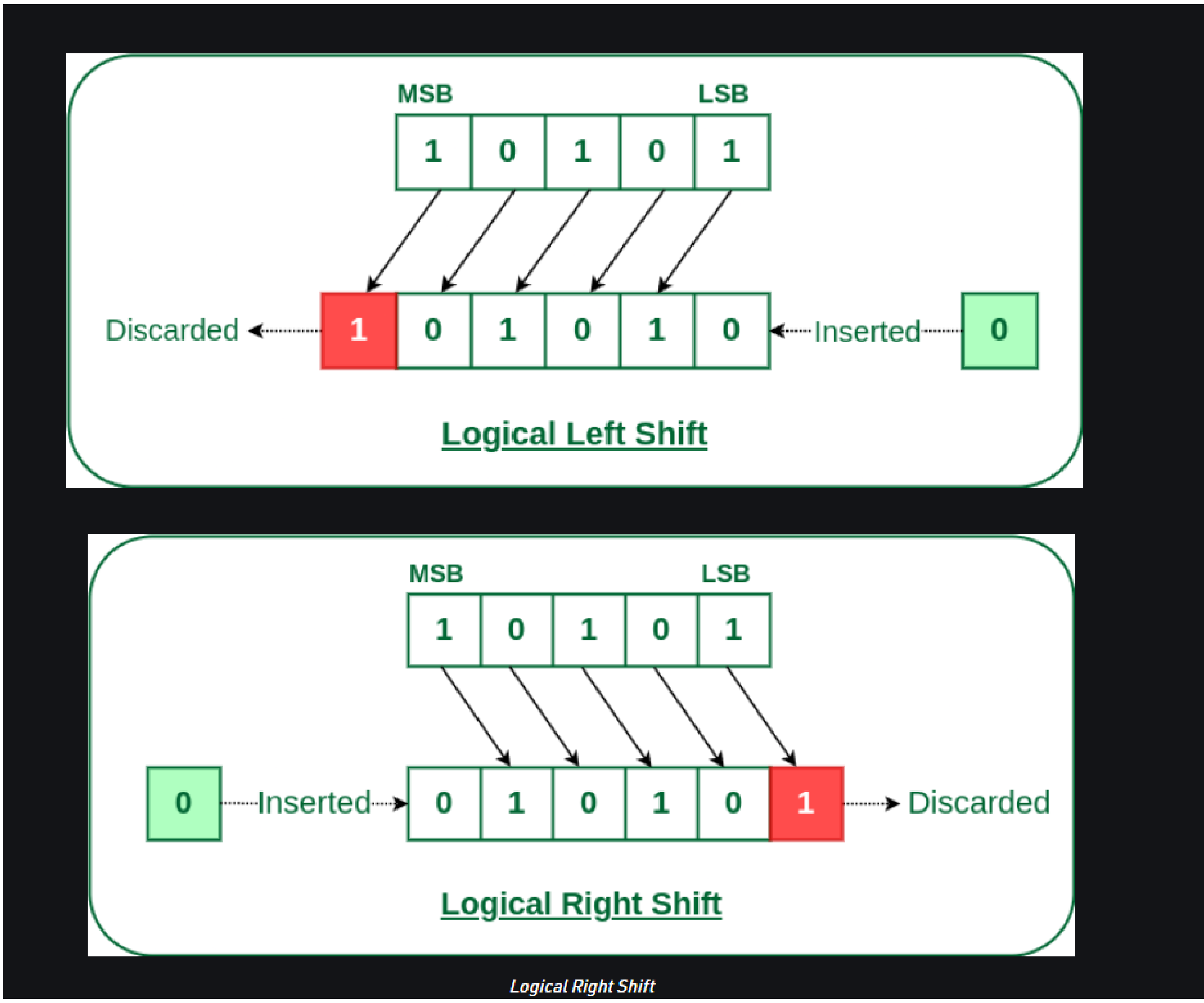


TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

1. Logical Shift:

1. A logical shift is one that transfers 0 through the serial input.
2. The symbols 'shl' and 'shr' for logical shift-left and shift-right microoperations.
3. The microoperations that specify a 1-bit shift to the left of the content of register R and a 1-bit shift to the right of the content of register R

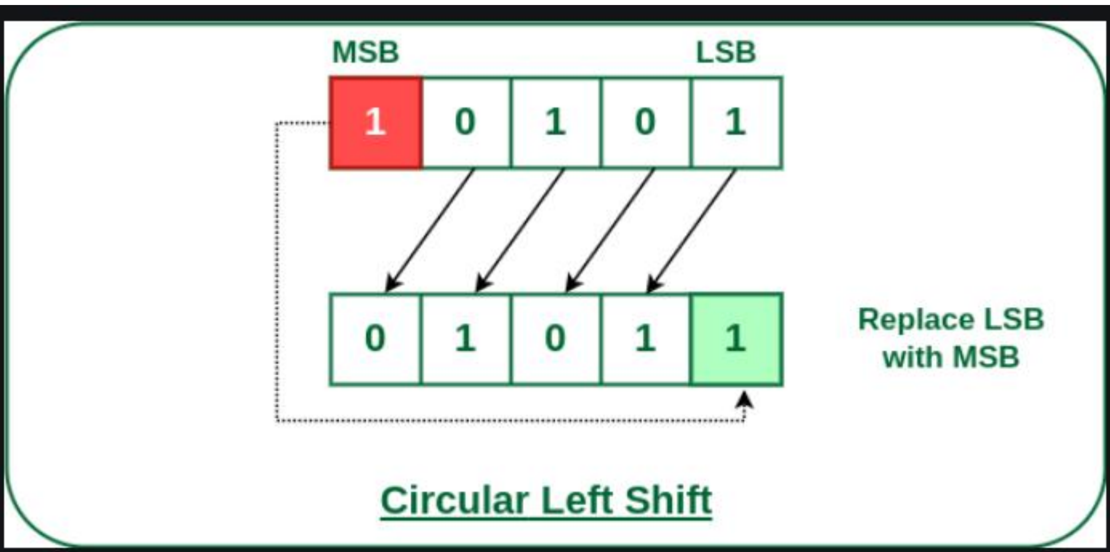


2.Circular Shift:

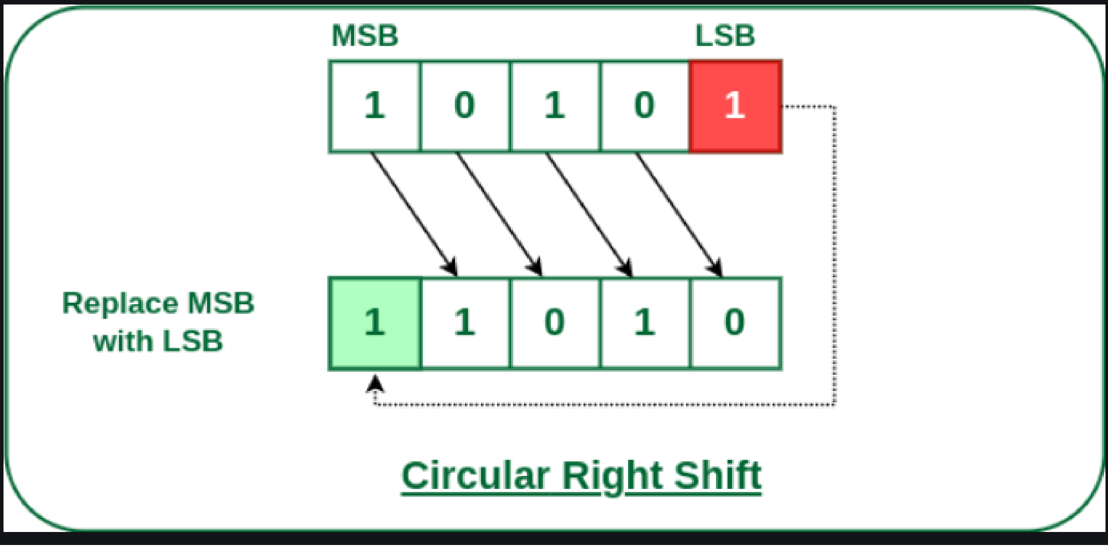
1. The circular shift (also known as a rotate operation) circulates the bits of

the register around the two ends without loss of information.

2. We will use the symbols 'cil' and 'cir' for the circular shift left and right, respectively

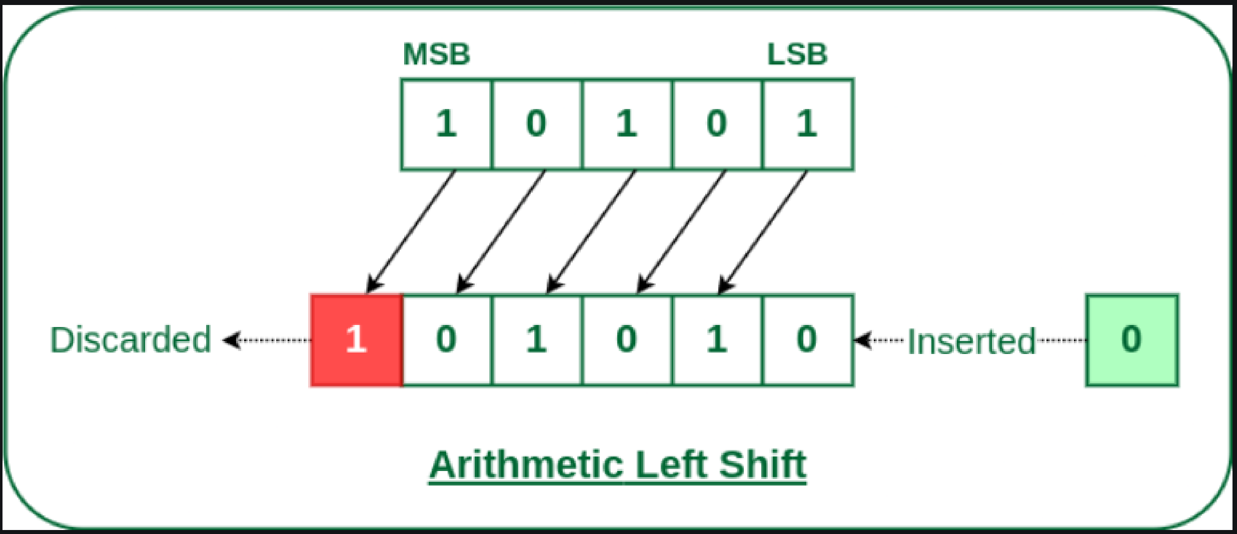


Circular Left Shift

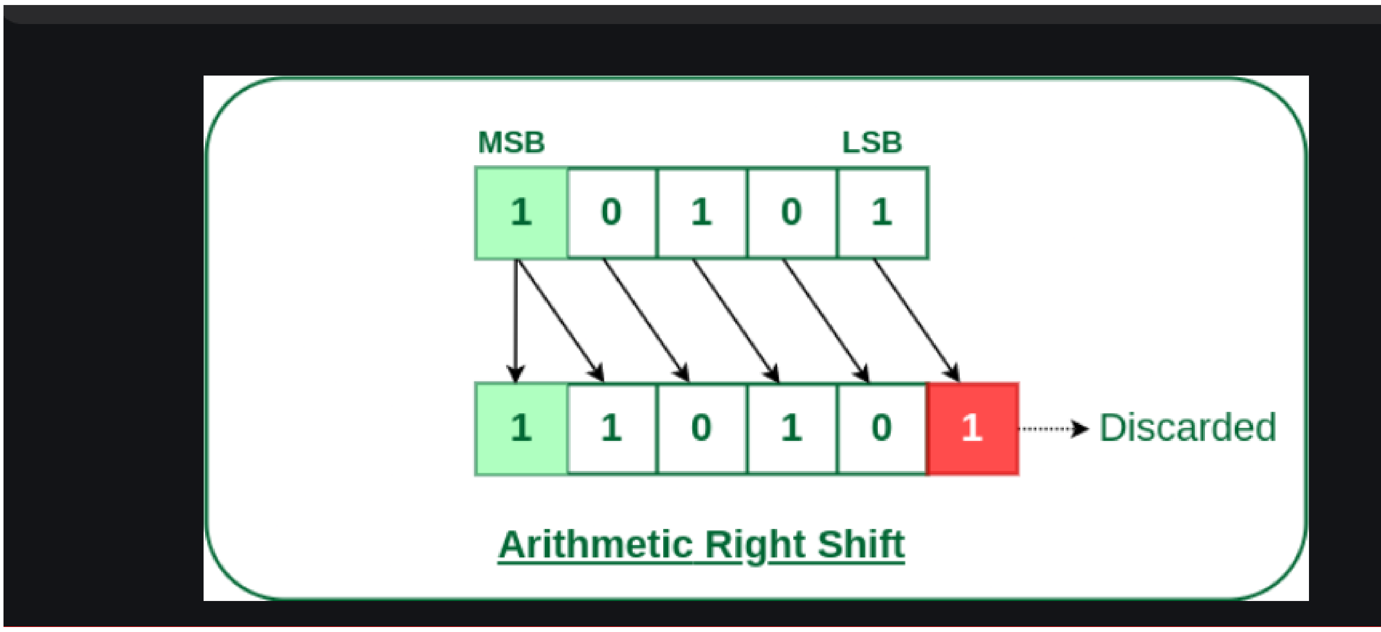


3.Arithmetic Shift:

1. arithmetic shift is a microoperation that shifts a signed binary number to the left or right
2. An arithmetic shift-left multiplies a signed binary number by 2.
3. An arithmetic shift-right divides the number by 2.
4. Arithmetic shifts must leave the sign bit unchanged



Arithmetic Left Shift

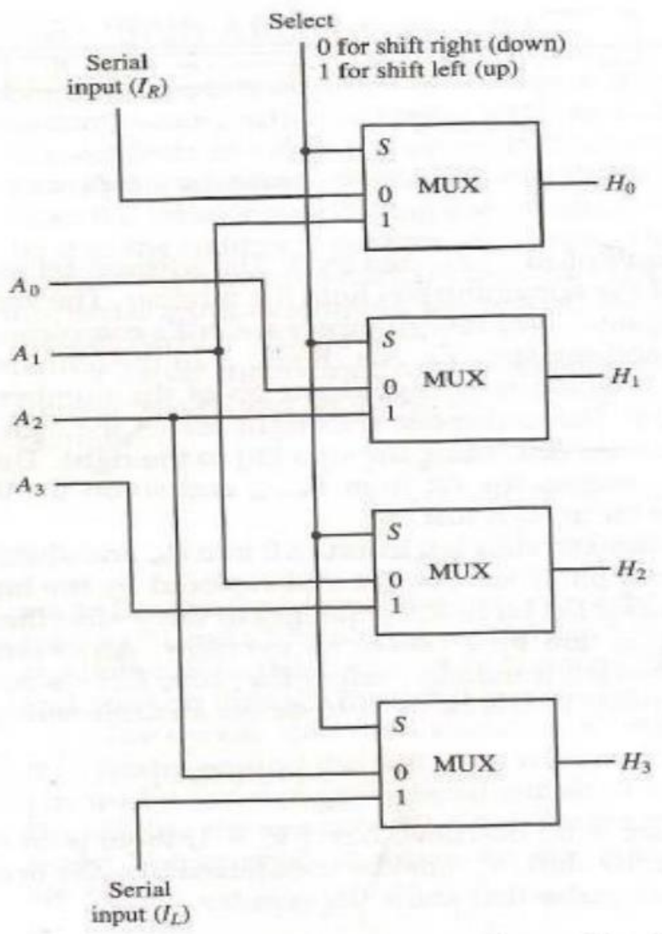


Hardware Implementation:

1. A combinational circuit shifter can be constructed with multiplexers
2. The 4-bit shifter has four data inputs, A0 through A3, and four data outputs, H0 through H3
3. There are two serial inputs, one for shift left (IL) and the other for shift right (IR).

4. When the selection input S=0 the input data are shifted right

5. When S = 1, the input data are shifted left



Function table

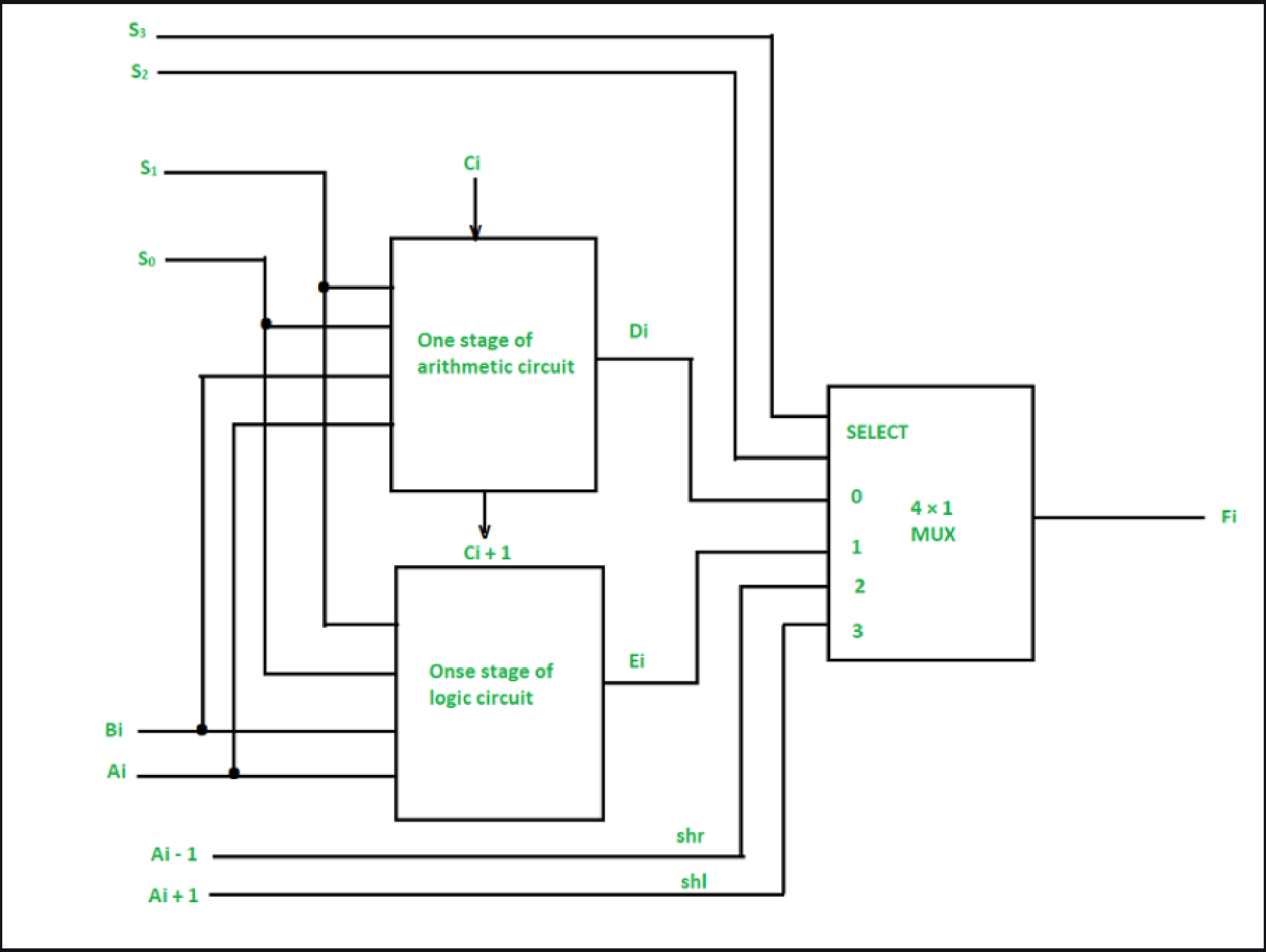
Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

Figure 4-12 4-bit combinational circuit shifter.

q12)explain Arithmetic Logic Shift Unit(ALSU)?
sol)

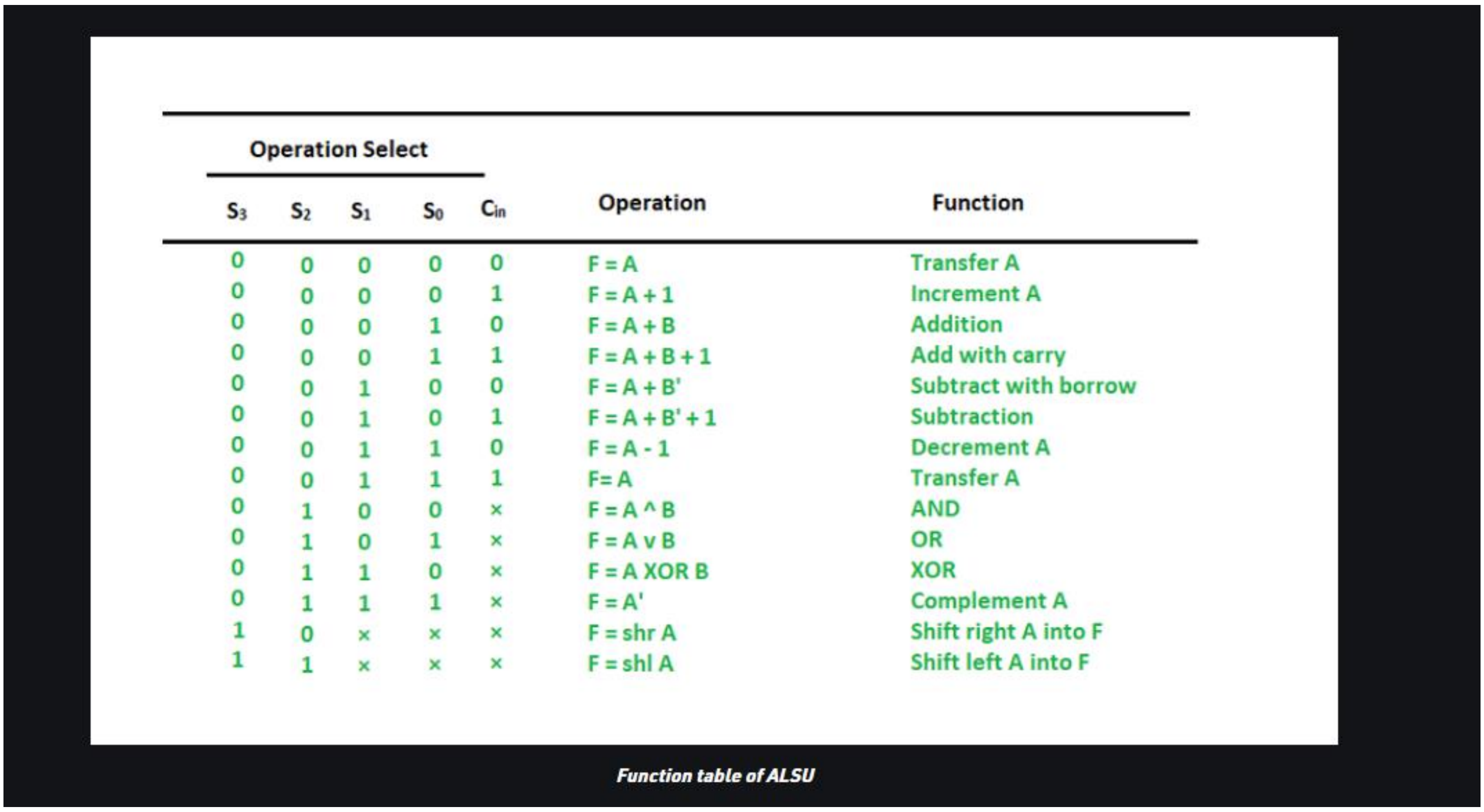
- Arithmetic Logic Shift Unit (ALSU) is a member of the Arithmetic Logic Unit (ALU) in a computer system.
- It is a digital circuit that performs logical, arithmetic, and shift operations.
- The Arithmetic Logic Unit performs an operation that leads as a result and gets transferred to a destination register.
- Sometimes, the shift micro operations are performed in a separate unit, but sometimes it is made as a part of full Arithmetic Logic Unit (ALU)

➢



One stage of ALSU

function table of Arithmetic Logic Shift Unit



Operation Select					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Increment A
0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	F = A + B + 1	Add with carry
0	0	1	0	0	F = A + B'	Subtract with borrow
0	0	1	0	1	F = A + B' + 1	Subtraction
0	0	1	1	0	F = A - 1	Decrement A
0	0	1	1	1	F= A	Transfer A
0	1	0	0	×	F = A ^ B	AND
0	1	0	1	×	F = A v B	OR
0	1	1	0	×	F = A XOR B	XOR
0	1	1	1	×	F = A'	Complement A
1	0	×	×	×	F = shr A	Shift right A into F
1	1	×	×	×	F = shl A	Shift left A into F

Function table of ALSU