Register Transfer and Micro operations

Register Transfer language :-

A digital system is an interconnection of digital hardware modules that accomplish a specific information processing task. Digital system design invariably uses a modular approach. The various modules are interconnected with common data and control paths to form a digital Computer system.

The digital modules are best defined by the registers they contain and the operations are performed on the data stored in them. The operations executed on data stored registers is called 'microoperations'. A micro operation is an elementary operation performed on the information stored in one or more operations. The result of the operation may be stored in the previous register or may be in another register.

The internal hardware organization of a digital Computer is best defined by specifying

1) The set of registers it Contains and their functions
2) The sequence of microoperations performed on the binary information stored in the registers
3) The control that initiates the sequence of microoperations

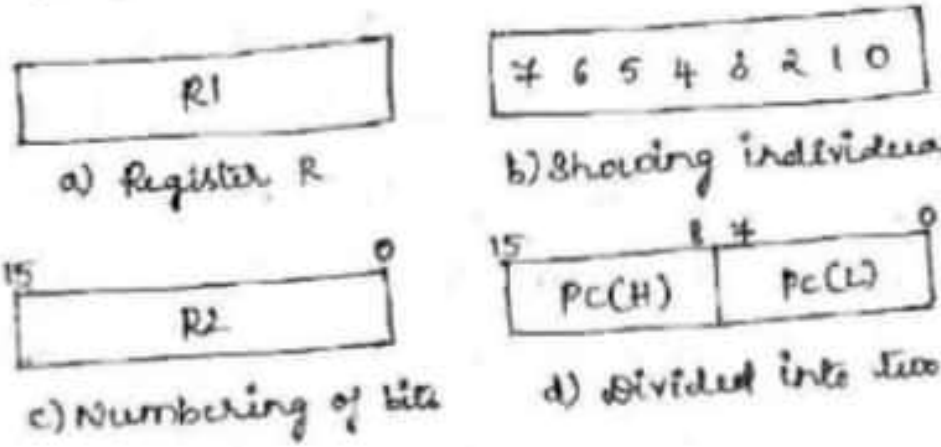
The symbolic notation used to describe microoperations transfers among registers is called a

Register transfer language. The term "register transfer" implies the availability of hardware logic circuits that can perform a stated microoperations and transfer the result of the operation to the same or another register. The word "language" is borrowed from programmers, who apply this term to programming language. A programming language is a procedure for writing symbols to specify a given computational process.

Register Transfer :-

Computer registers are designated by capital letters to denote the function of the register. For eg, the register that holds an address of the memory unit is called MAR. The individual flip-flops in an n-bit register are numbered from 0 through n-1, starting from '0' in the right most position and increasing the numbers towards the left

In fig. shows the representation of registers in the block diagram form.



a) Register R
b) Showing individual bits
c) Numbering of bits
d) Divided into two parts.

The fig.(a) represents the common way with the name of the register inside. The individual bits shown in the fig.(b). The numbering of bits in a 16-bit register are shown in fig.(c). i.e., in left of the box. The fig(d) shows the 16-bit register is partitioned into two parts. The symbol PC(0-7) or PC(L) refers to the low-order byte and PC(8-15) & PC(H) to the high-order byte.

"Information transfer from one register to another is designated in symbolic form by means of a replacement operation" is known as Register transfer.

The statement R2 ← R1, denotes a transfer of the content of register R1 into register R2. The content of the R1 can't be changed after transfer.

A statement that specifies a register transfer implies that circuits are available from the o/p's of the source register to the inputs of the destination register and that the destination register has a parallel load capacity. We want the transfer to occur only under a predetermined control condition. This can be shown by means of a if-then statement.

If (P=1) then (R2 ← R1)

Where 'P' is a control signal generated in the control section It is sometimes convenient to separate the control variables from the register transfer operations by specifying a "control function". A control function is a Boolean variable that is

equal to '1' or '0'.

The control function is as follows:

P : R2 ← R1.

The control condition is terminated with a colon. It symbolizes the requirement that the transfer operation be executed by the hardware only if P=1.

Every statement written in a register transfer notation implies a h/w construction for implementing the register. Fig. shows the block diagram that depicts the register from R1 to R2.

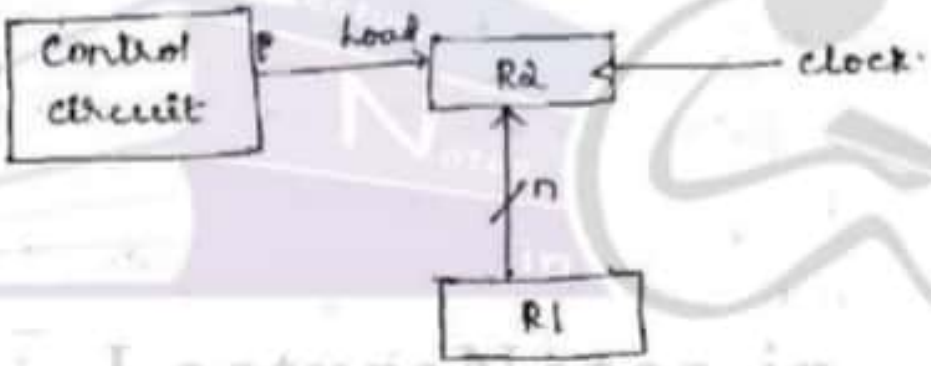
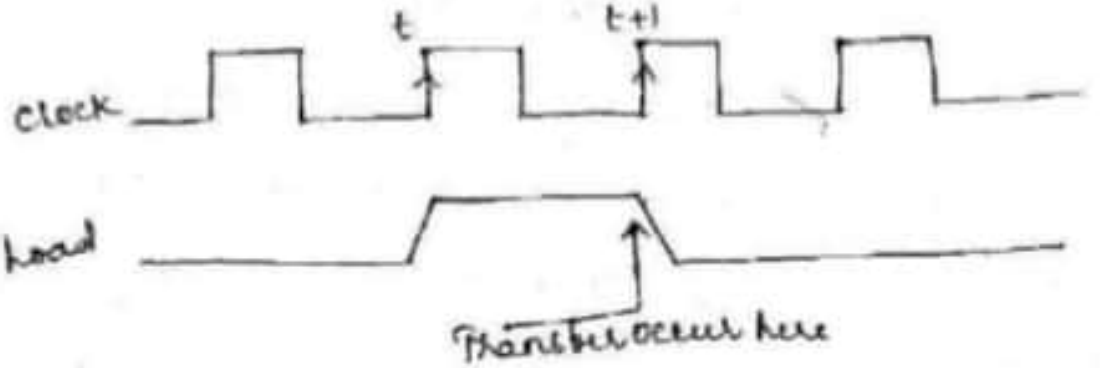


Fig: Transfer from R1 to R2 when P=1.

The 'n' o/p's of register R1 are connected to the 'n' i/p's of register R2.

It is assumed that the control variable(P) is synchronized with the same clock as the one applied to the register



In the timing diagram, P is activated in the control section by the rising edge of a clock pulse at time t. The next positive transition of the clock at time t+1 finds the load i/p active & the data i/p's of R2 are then loaded into the register in parallel.

The basic symbols of the register transfer notation are listed:

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	MAR, R2
parentheses ( )	Denote a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Comma (,)	Separates two micro-operations	R2 ← R1, R1 ← R2

The statement T: R2 ← R1, R1 ← R2 denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T = 1. The simultaneous operation is possible with registers that have edge-triggered flip-flops.

Bus and Memory Transfers :-

The most efficient scheme for transferring information b/w registers in a multiple-register configuration is a common bus system. A bus structure consists of a set of common lines, one for each bit of a register thro' which

binary information is transferred at one time.

One way of constructing a common bus system is with multiplexers. The multiplexers select the source reg whose binary information is then placed on the bus. The construction of a bus system for four registers as shown in the fig.

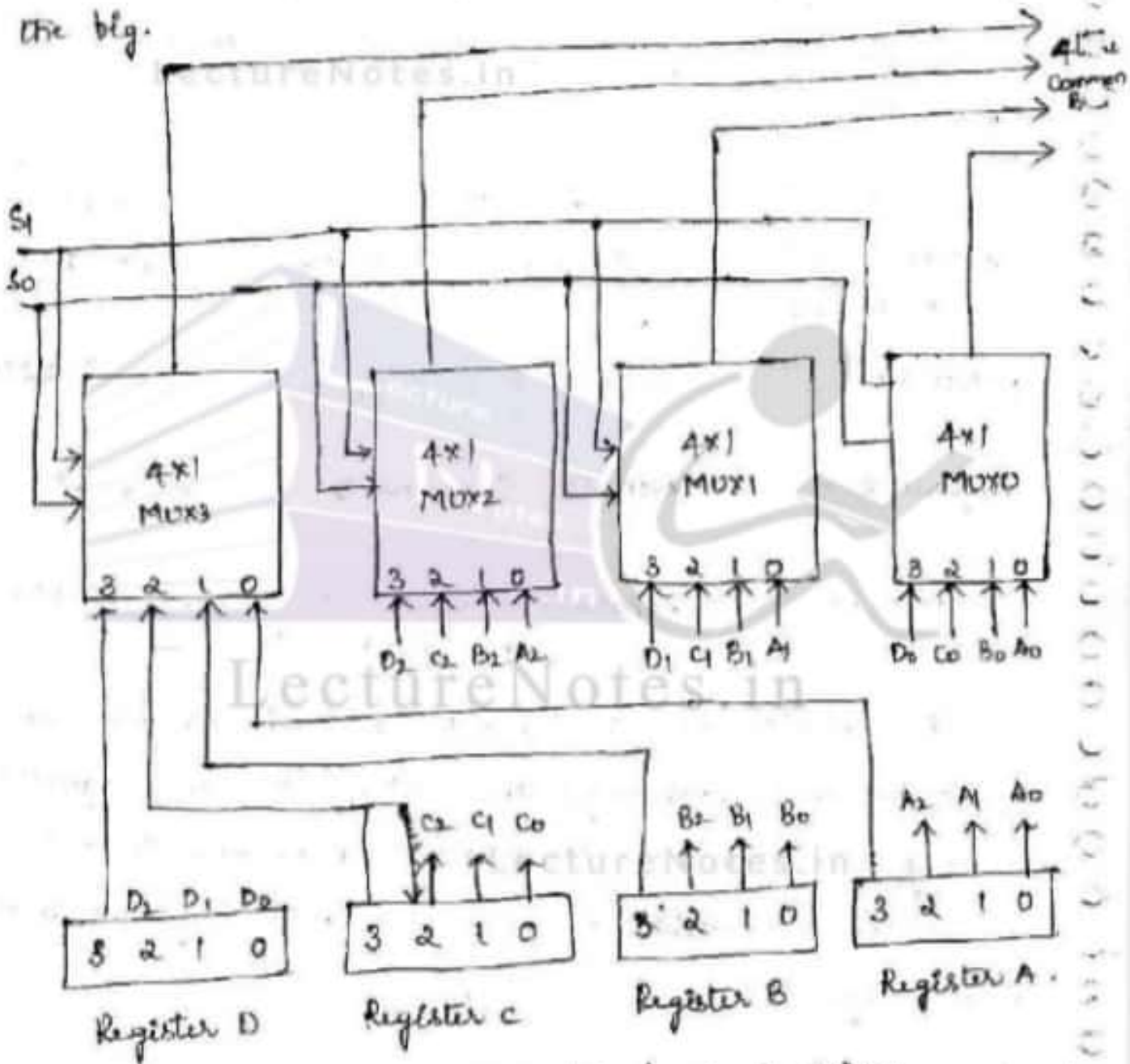


Fig: Bus system for four registers

Each register has four bits numbered 0 through 3. Not to complicated the diagram with 16 lines crossing each other we use labels to show the connections from the o/p's of the registers

to the input of the registers.

For eg, o/p 1 of register A is connected to the input 0 of MUX1 because this I/p is labelled as 'A1'. This MUX0 multiplexes the lower 0 bits of the registers and soon.

The two selection lines S_1 and S_0 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. When $S_1S_0 = 00$, the '0' data inputs of all four multiplexers are selected and applied to the o/ps that form the bus. This causes the bus lines to receive the content of register A since the o/p's of this register are connected to the '0' data inputs of the multiplexers. Similarly register B is selected if $S_1S_0 = 01$ and soon.

The below table shows the register is selected by the bus for each of the four possible binary values of the selection lines.

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

In general, a bus system will multiplex 'k' registers of 'n' bits each to produce an n-line common bus. The number of multiplexers needed to construct the bus is equal to n, the

number of bits in each register. The size of each multiplexer must be k×1 since it multiplexes 'k' data lines.

The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to the inputs of all destination registers and activating the load control of the particular destination register.

When the bus is includes in the statement, the register transfer is symbolized as follows:

BUS ← C, R1 ← BUS.

The content of register 'C' is placed on the bus and the content of the bus is loaded into register R1 by activating its load control i/p.

R1 ← C.

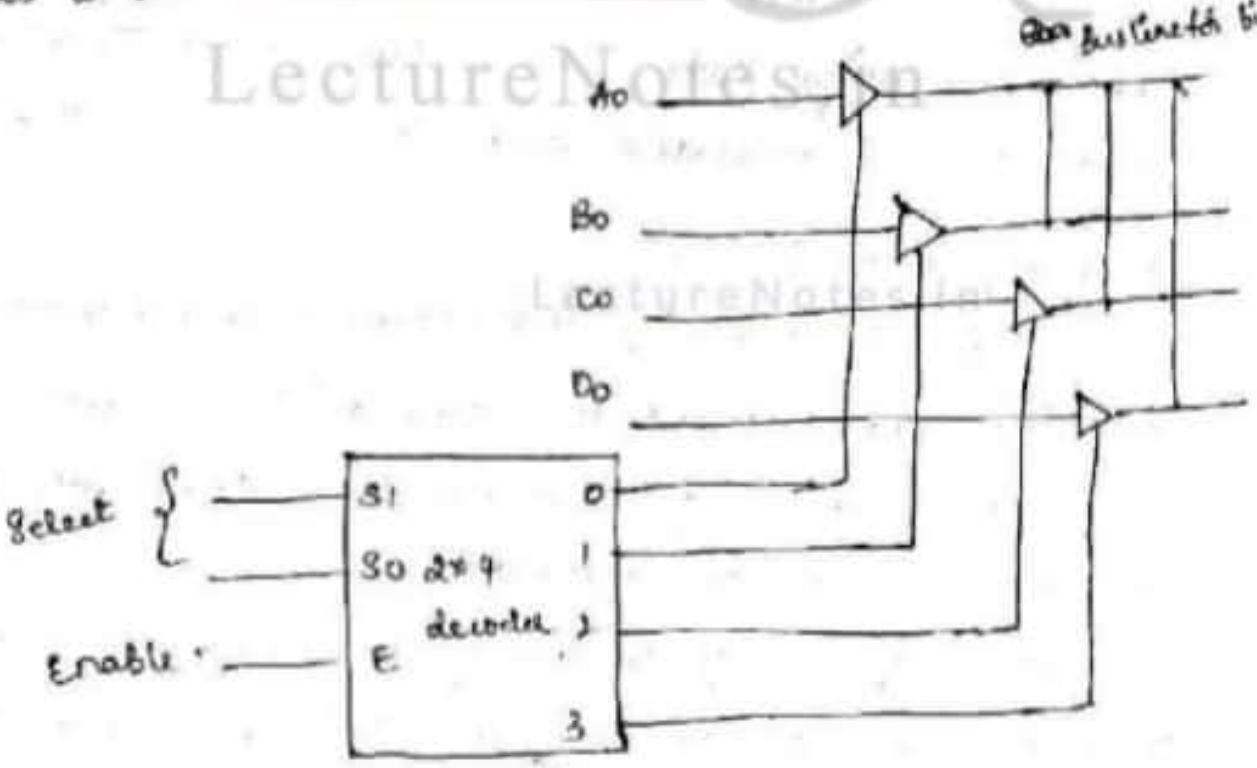
Three - State Bus Buffers :-

A bus system can be constructed with three - gate instead of multiplexers.

A three - state gate is a digital circuit that exhibits three states. Two of the states are signals equivalent to logic '1' and '0'. The third state is a high - impedance state. The high - impedance state behave like a open circuit. Three-state gates may perform any conventional logic such as AND or NAND. The one most commonly used in the design of a bus system is the buffer gate.

The graphic symbol of a three-state buffer gate is shown in fig.

It is distinguished from a normal buffer by having both a normal i/p and a control input. The control input determines the o/p state. When the control i/p is equal to 1, the o/p is enabled and the gate behaves like any conventional buffer, with the o/p equal to the normal i/p. When the control i/p is equal to 0, the o/p is disabled and the gate goes to a high-impedance state. The high-impedance state of a three state gate provides a special feature not available in other gates. The three state gate o/p's can be connected with wires to form a common bus line without any loading effects.



The construction of a bus system with three-state buffers as shown in fig.

The o/p's of four three-state buffers are connected to form a single bus line. The control i/p's to the buffers determine which of the four normal i/p's will communicate with the bus line. No more than one buffer may be active state at any time. remaining are maintained in a high-impedance state.

One way to ensure that no more than one control i/p is active at any given time is to decoder as shown in fig. When the enable i/p is of the decoder is '0', all four i/p's are '0' and the bus line is in a high-impedance state. When the enable i/p is '1', one of the three-state buffer will be active, depending on the binary value it selects the i/p's to the decoder.

To construct a common bus for four registers of 'n' bits each using three-state buffers, we need 'n' circuits with four buffers and each common o/p will produce.

Memory Transfer :-

The transfer of info. from a memory word to the outside environment is called a 'read' operation

The transfer of new info. to be stored into the memory is called a 'write' operation.

It is necessary to specify the address of M when writing memory transfer operation. The address is enclosed with in the square brackets of M.

The memory unit that receives the address from a register called the address register (AR). The data are transferred to another register (Data register (DR).

The read operation as follows

Read : DR ← M[AR]

The write operation as follows

Write : M[AR] ← R1.

This causes a transfer of info. from R1 into memory word.

Arithmetic Microoperations :-

The micro operations most often used in digital computers are classified into 4 categories:

1) Register transfer microoperations transfer binary information from one register to another
2) Arithmetic microoperations perform arithmetic operations on numeric data stored in registers
3) Logic microoperations perform bit manipulation operations on non-numeric data stored in registers
4) Shift microoperations perform shift operation on data stored in registers.

The basic arithmetic micro operations are addition, subtraction, increment, decrement and shift.

The arithmetic micro operation defined by the statement

R3 ← R1 + R2 .) Specifies an add micro operation.

It states that the content of reg. R1 are added to the contents

of Register R2 and the sum transferred to register R3.

The other microoperations are listed below:

Symbolic designation	Description
$R3 \leftarrow R1+R2$	Contents of R1 plus R2 transferred to R3.
$R3 \leftarrow R1-R2$	" minus " "
$R2 \leftarrow \overline{R2}$	Complement the contents of R2 (1's complement)
$R2 \leftarrow \overline{R2}+1$	2's complement of R2 contents (negate)
$R3 \leftarrow R1+\overline{R2}+1$	R1 plus 2's complement of R2 (subtraction)
$R1 \leftarrow R1+1$	Increment the contents of R1 by one.
$R1 \leftarrow R1-1$	Decrement " " " " .

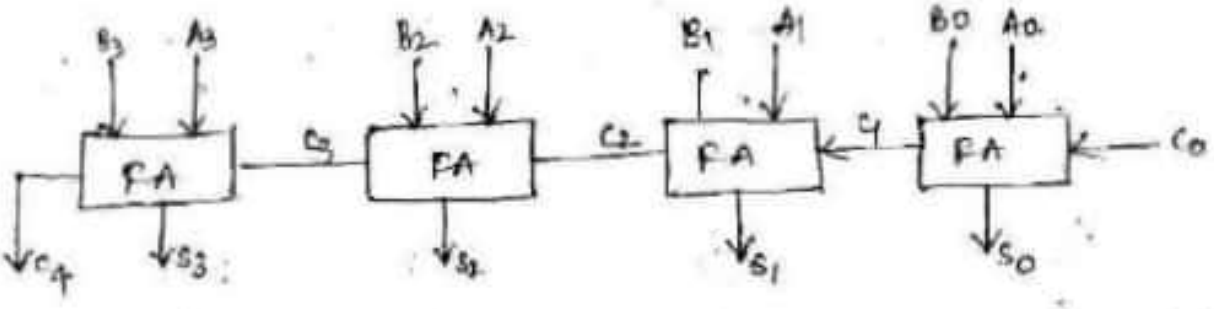
Binary Adder :-

To implement the add microoperation with hw, we need the registers to hold the data and the digital component that performs the arithmetic addition.

The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a full adder. The digital circuit that generates the arithmetic sum of two binary numbers of any length is called binary adder.

The binary adder is constructed with full adders connected in cascade to each other.

The fig. shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder.



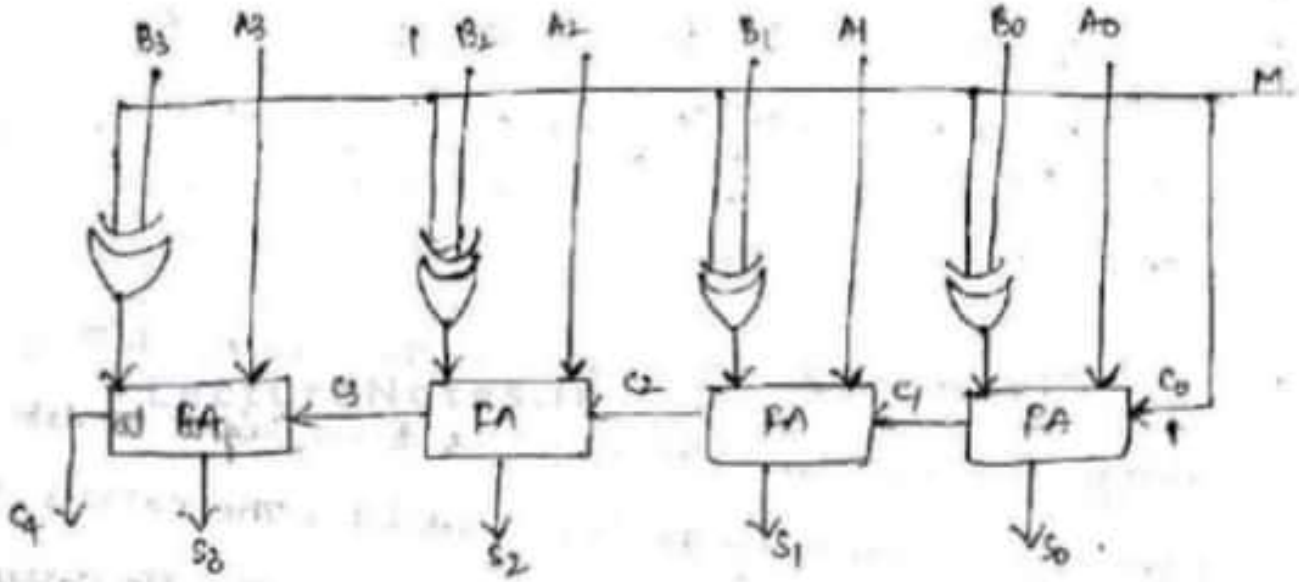
The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript '0' denoting the low-order bit. The carries are connected in a chain thro' full adders. The i/p carry to the binary adder is C_0 and the o/p carry is C_4. The 's' o/ps of the full-adders generate the required sum bits.

An n-bit binary adder requires n full-adders. The 'n' data bits for the 'A' i/p's come from one register (R1) and 'n' data bits for the 'B' i/p's " " " (R2), the sum can be transferred to a third register or to one of the source registers by replacing its previous content.

Binary Adder - Subtractor :

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

The Mode input M controls the entire operations. When M=0, the circuit is an adder and when M=1, the circuit becomes a subtractor.



Each Ex-OR gate receives input M and one of the i/p's of B. When M=0, we have $B \oplus 0 = B$. The full adder receive the value of B, the i/p carry is '0' and the circuit performs A plus B.

When M=1, we have $B \oplus 1 = B'$, $C_0 = 1$. The 'B' i/p's are all complemented and a '1' is added through the i/p carry.

The circuit performs the operation A plus 2's complement of B. For unsigned numbers, this gives A-B if $A \geq B$ or the 2's complement of (B-A) if $A < B$. For signed numbers, the result is A-B.

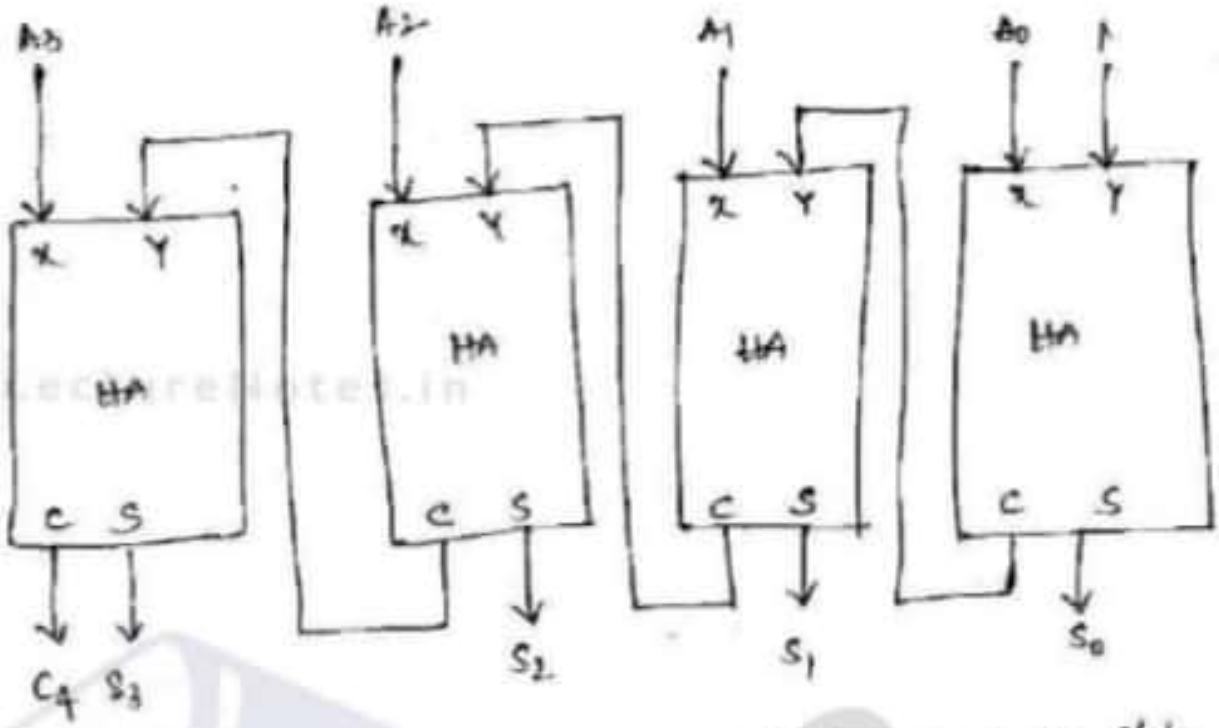
Binary Incrementer :-

The 'increment micro operation adds one to a number in a register.

Eg: 0110 + 1 = 0111

The diagram of a 4-bit combinational circuit incrementer is shown in fig. one of the i/p's to the least significant half-adder (HA) is connected to logic-1 and the other i/p is connected

to the LSB of the number to be incremented.



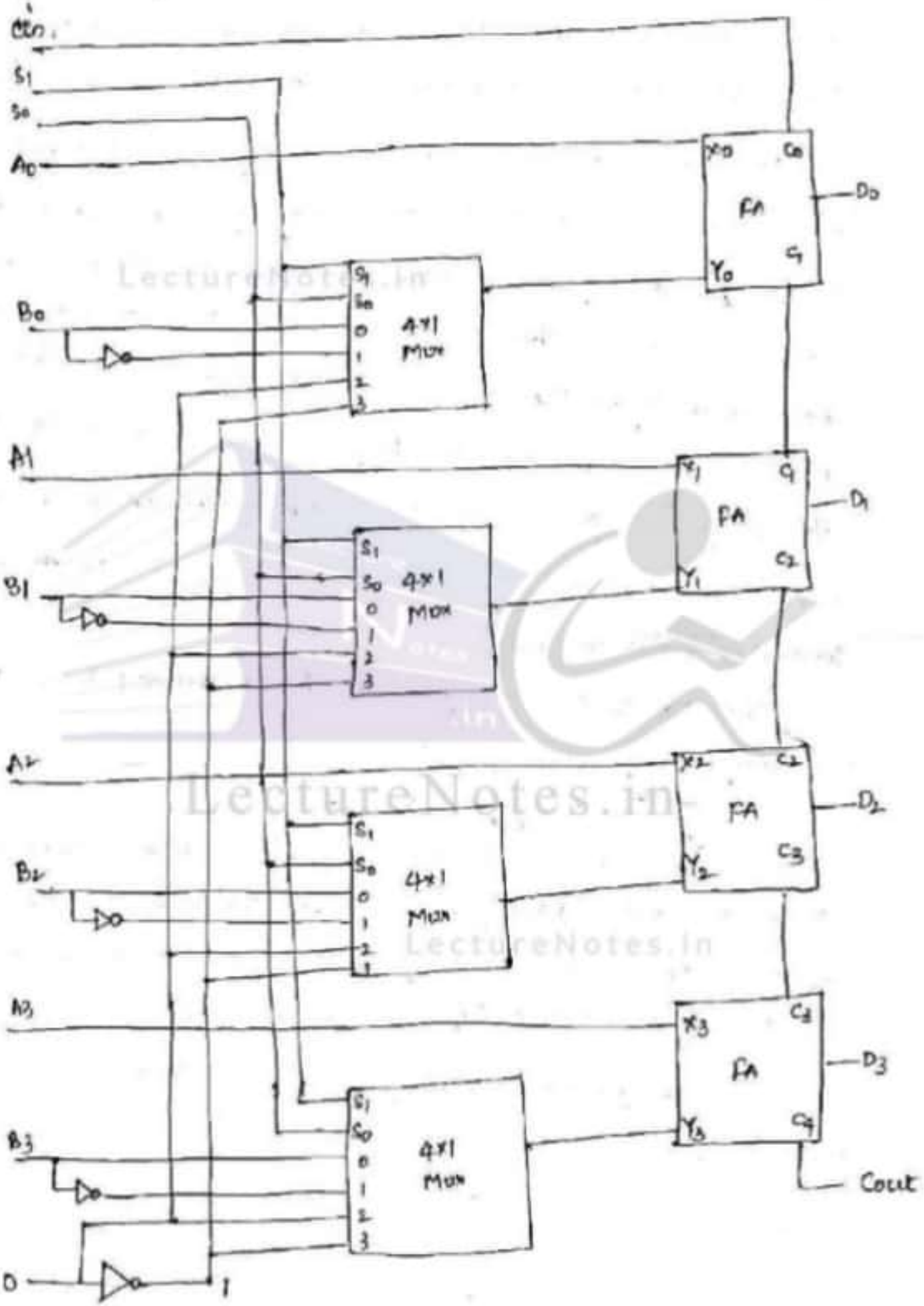
The o/p carry of one HA is connected to one of the I/p's of the next-higher-order half adder. The circuit receives the four bits from A_0 thro' A_3 adds one to it and generates the incremented output in S_0 through S_3. The o/p carry C_4 will be 1 only after incrementing binary 1111. This also causes o/p's S_0 thro' S_3 to go to '0'.

The circuit diagram can be extended to an n-bit binary incrementer by extending the diagram to include 'n' half-adders.

Arithmetic circuit :-

The basic component of an arithmetic circuit is the parallel adder. By controlling the data I/p's to the adder, it is possible to obtain different types of arithmetic operations.

The diagram of a 4-bit arithmetic circuit is shown in fig.



It has the bell-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations. There are two 4-bit inputs A & B and a 4-bit output D. The four inputs from A go directly to the X inputs of the binary adder. Each of the four i/p's of B are connected to the data i/p's of the multiplexers. The multiplexers data i/p's also receive the complement of B. The other two data i/p's are connected to logic-0 and logic-1. The four multiplexers are connected by two selection inputs S_1 & S_0. The i/p carry C_{in} goes to the carry i/p of the FA in the least significant position. The other carry i/p of the FA are connected from one stage to the next.

The o/p of the binary adder is calculated from the following arithmetic sum:

$$D = A + Y + C_{in}$$

where 'A' is the 4-bit binary number at the X i/p's and Y is the 4-bit binary number at the Y i/p's of the binary adder. C_{in} is the input carry, which can be equal to '0 & '1'. Symbol '+' is the arithmetic addition.

We can generate the eight possible arithmetic micro operations.

Select			Input	o/p	Micro operation
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry.
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow.
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A.

When $S_1S_0 = 00$, the value of B is applied to the Y i/p's of the adder. If $C_{in} = 0$, the o/p $D = A + B$. If $C_{in} = 1$, the o/p $D = A + B + 1$. Both cases perform the add micro operation with or with out carry.

When $S_1S_0 = 01$, the complement of B is applied to Y i/p's of the adder. If $C_{in} = 1$, then $D = A + \overline{B} + 1$. This produces the 2's complement of B, which is equivalent to a subtraction of A-B. When $C_{in} = 0$, then $D = A + \overline{B}$ i.e., $A - B - 1$.

When $S_1S_0 = 10$, the i/p's from B are neglected, and all zeros are inserted into the Y i/p's. The o/p becomes $D = A + 0 + C_{in}$. This gives $D = A$ when $C_{in} = 0$ and $D = A + 1$ when $C_{in} = 1$. In the first case, a direct transfer from i/p A to o/p D. In the second case, the value of A is incremented by 1.

When $S_0 = 1$, all 1's are inserted into the Y i/p's of the adder to produce the decrement operation $D = A - 1$, when $C_{in} = 0$.

In the function table, the o/p $D = A$ may appear twice, so we have only seven distinct arithmetic micro operations.

Logic micro operations :-

Logic micro operations specify binary operations for strings of bits stored in registers.

Special symbols :-

The some of the special symbols are :

OR is denoted by $\vee$ and AND is denoted by $\wedge$.

In the operations, we never use 'OR' statement.

Eg: $R4 \leftarrow R5 \vee R6$.

List of logic micro operations :-

There are 16 different logic operations that can be performed with two binary 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 each of the 16 columns F_0 through F_{15} represents a truth table of one possible Boolean functions for the two variables x and y.

The 16 Boolean functions of two variables x and y are expressed in algebraic form in the first column as shown below:

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

The 16 logic micro operations are derived from these functions by replacing variable x by the binary content

binary content of register A and variable y the binary content of register B.

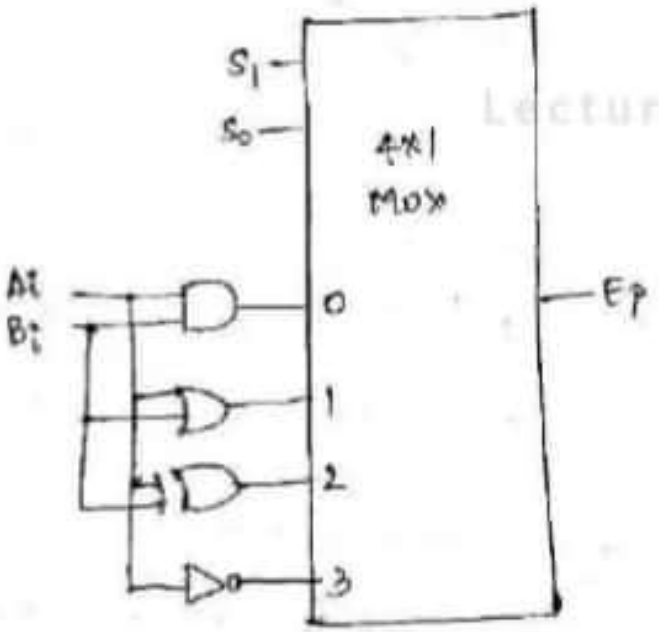
The logic microoperations listed in the second column represent a relationship b/w the binary contents of two registers A and B. Each bit of the register is treated as a binary variable and the micro operation is performed on the string of bits stored in the registers.

Hardware implementation :-

The h/w implementation of logic micro operation requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function.

Most computers use only four - AND, OR, XOR and Complement.

The below fig shows the one stage of a circuit that generates the four basic logic micro operations



Function table

S_1	S_0	O/p	operation
0	0	E=A∧B	AND
0	1	E=A∨B	OR
1	0	E=A⊕B	XOR
1	1	E=$\bar{A}$	Complement

It consists of four gates & a multiplexer. Each of the four logic operations is generated thru a gate that performs the required logic. The o/p's of the gates are applied to the data i/p's of the multiplexer. The two selection i/p's S_1 & S_0 choose one of the data i/p's of the multiplexer and direct its value to the o/p. The diagram shows one typical stage with subscript i. For a logic circuit with n bits, the diagram must be repeated 'n' times.

The function table shows all the logic microoperations obtained for the combination of the selection variables.

Some Applications :-

Logic micro operations are very useful for manipulating individual bits of a portion of a word stored in a register. They can be used to change the bit values, delete a group of bits, & insert new bit values into a register.

Some of the types of operations are going to perform.

Selective-Set :-

The selective-set operations sets to 1 the bits in Register A where there are corresponding 1's in register B. It does not affect bit positions that have 0's in B.

Eg :

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

The two left most bits of B are 1's, So the corresponding bits of A are set to 1. One of these two bits are set already as and the other has been changed from 0 to 1. The two bits of A with corresponding 0's in B remain unchanged.

From the truth table we note that the bits of A after the operation are obtained from the logic-OR operation of bits in B and previous value of A. Therefore, the OR micro operation can be used to selectively set bits of a register.

Selective - Complement :-

The selective complement operation complements bits in A, where there are corresponding 1's in B.

Eg:

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

Again the two left most bits of B are 1's, so the corresponding bits of A are complemented. This operation is just like an exclusive-OR operation.

Selective - clear :-

The selective-clear operation clears to 0 the bits in A only where there are corresponding bits in B are 1's.

for eg,

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

Again the two left most bits of B are 1's, so the correspon-ding bits of A are cleared to 0.

From (c) we can deduce from the Boolean operation performed on the individual bits is AB'. The corresponding logic micro operation is

$$A \leftarrow A \wedge B'.$$

The mask operation is similar to selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in B. The mask operation is an 'AND' operation

```
1010    A before
1100    B (logic operand)
------
1000    A after masking
------
```

The two right most bits of A are cleared because the corresponding bits of B are 0's. The two left most bits are left unchanged. The mask operation is more convenient to use than the selective clear and this mask operation uses 'AND' micro operations.

The 'insert' operation inserts a new value into a group of bits. This is done by first masking the bits and then perform OR with the required value.

For eg, suppose, A register A contains eight bits, 0110 1010. To replace the four left most 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.

The mask operation is an 'AND' micro operation and the insert operation is an 'OR' micro operation.

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

For eg.,

1010	A
1010	B
0000	A ← A ⊕ B.

When A & B are equal, the two corresponding bits are either both 0 or both 1. In the case, the EX-OR operation produces '0'. The all 0's result is then checked to determine if the two numbers are equal.

Shift microoperations:-

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 i/p transfers a bit into right most position. During a shift-right operation the serial i/p transfers a bit into the left most position.

There are three types of shifts

1) logical 2) circular 3) Arithmetic

A logical shift is one that transfers 0 thro' the serial i/p. The symbols shl & shr for logical shift-left and shift-right microoperations

Ex: R1 ← shl R1. (R→L)

R2 ← shr R2. (L→R)

are two microoperations that specify a 1-bit shift to the left of the content of register R1 and a 1-bit shift to the right of the content of register R2. The bit transferred to the end position thro' the serial i/p is assumed to be 0 during a logical shift.

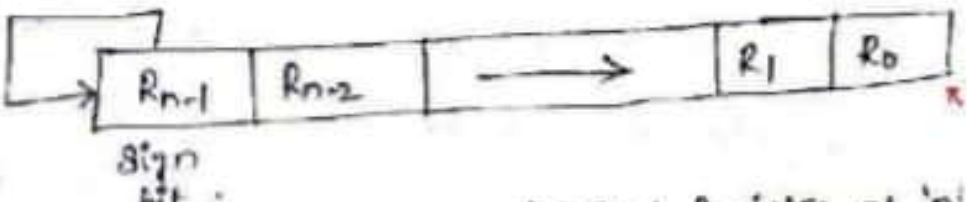
The "circular shift" (rotate operation) circulates the bits of the register around the two ends with out loss of information. We use the symbols cil & cir for the circular shift left & right respectively.

The symbolic notation for the shift microoperations is shown in below Table.

Symbolic designation	Description
R ← shl R	Shift-left register R
R ← shr R	Shift-Right " "
R ← cil R	circular shift left " "
R ← cir R	" " right "
R ← ashl R	Arithmetic shift left R
R ← ashr R	" " " right R.

An arithmetic shift is a microoperation that shifts a signed binary number to the left or the right. An arithmetic shift-left multiplies a signed binary number by 2. An arithmetic shift-right divides the number by 2.

An arithmetic shift must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2. The left most bit in a register holds the sign bit, and the remaining bits holds the number. The sign bit is 0 for +ve numbers and '1' for -ve numbers. -ve numbers are 2's complement form.



The above fig. shows a typical Register of 'n' bits. Bit R_{n-1} in the left most position holds the sign. R_{n-2} is the right most significant bit and R_0 is the least significant bit.

An arithmetic shift-right leaves the sign bit unchanged and shifts the number (including the sign bit) to the right. The bit R_0 is lost.

The arithmetic shift-left inserts 0 to R_0 and shifts all other bits to the left. The initial bit of R_{n-1} is lost.

An overflow flip-flop V_S can be used to detect an arithmetic shift-left overflow.

$$V_S = R_{n-1} \oplus R_{n-2}$$

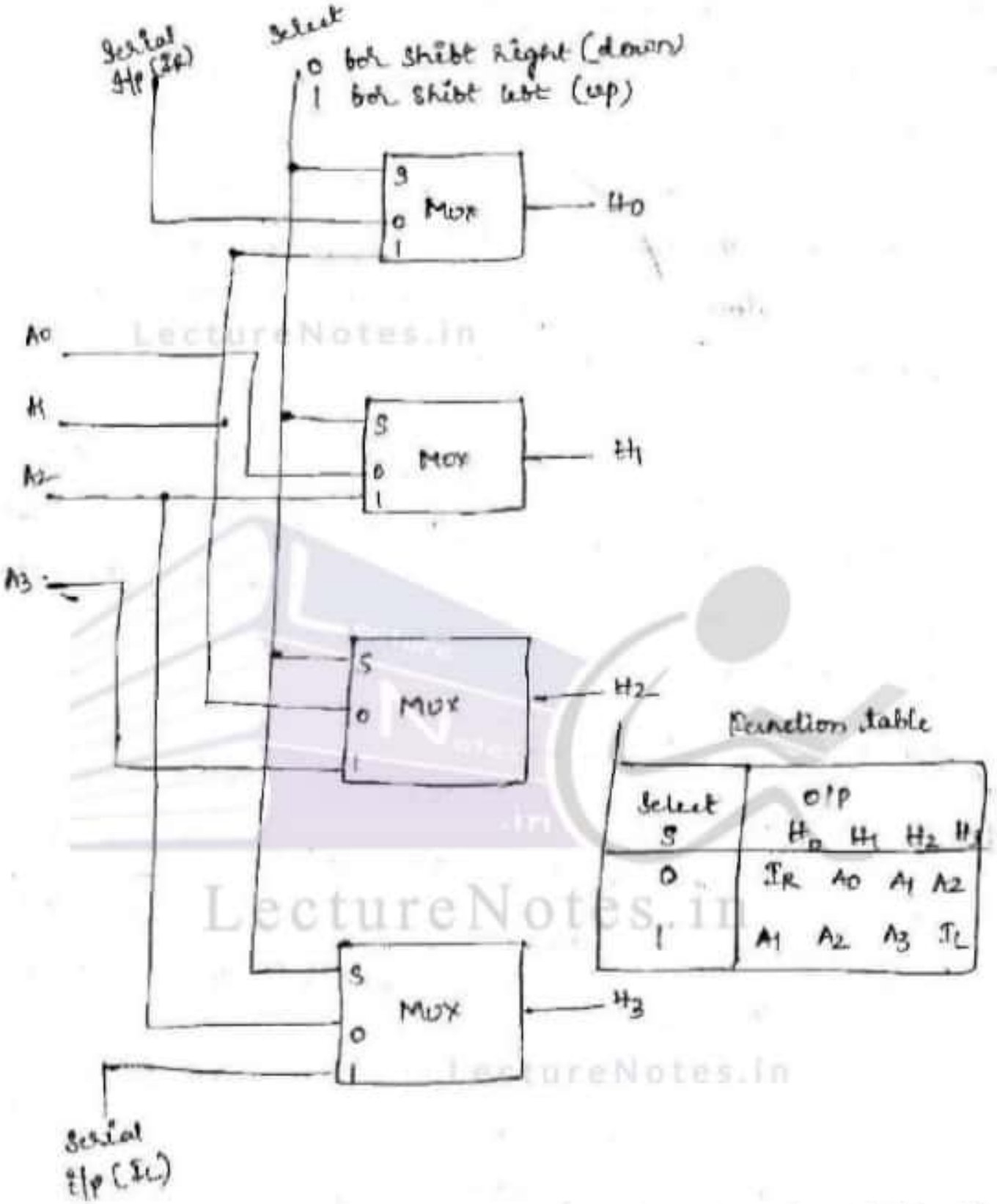
If $V_S = 0$, there is no overflow, but if $V_S = 1$ there is an overflow and a sign reversal after the shift. V_S must be transferred into the overflow flip-flop with the same clock pulse that shifts the register.

Hardware implementation :-

Information can be transferred to the register in parallel and then shifted to the right or left. In this type of configuration, a clock pulse is needed for loading the data into the register and another pulse is needed to initiate the shift.

A combinational circuit shifter can be constructed with multiplexers as shown in fig.

The 4-bit shifter has four data i/p's A_0 through A_3, and four data o/p's H_0 through H_3. There are also serial i/p's, one for shift left (IL) and the other for shift right (IR). When selection i/p $S = 0$, the i/p data are shifted right (down in the diagram)



Function table

Select S	O/p 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

When Sel, the i/p data are shifted left (up in the diagram)

A shifter with n data i/p's and o/p's requires 'n' multiplexers

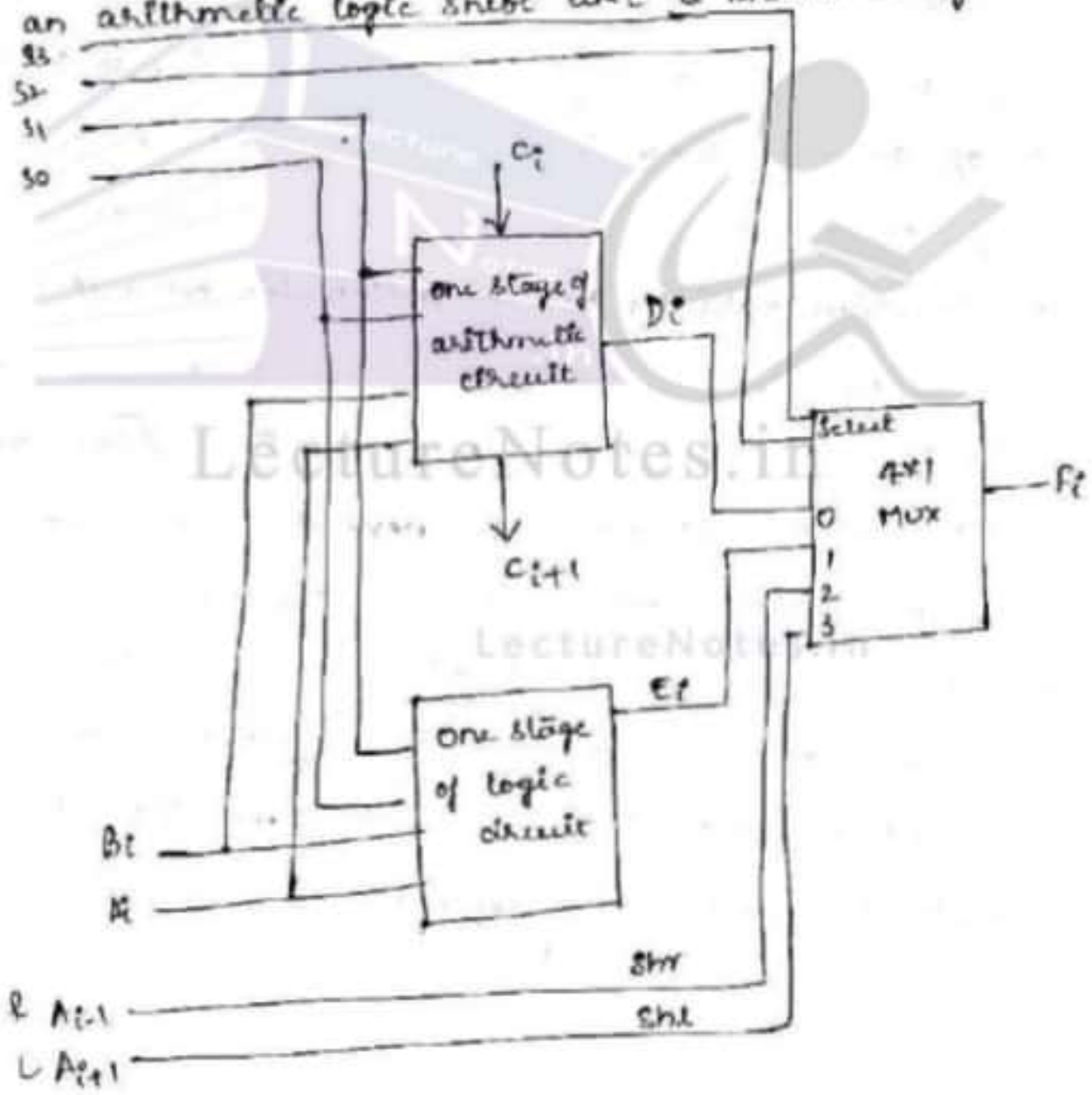
The two serial i/p's can be controlled by another multiplexer to provide the three possible types of shift.

Arithmetic logic Shift unit :-

A number of storage registers connected to a common operational unit called an arithmetic logic unit (ALU).

The ALU performs an operation and the result of the operation is then transferred to a destination register.

The arithmetic, logic, shift circuits can be combined into one ALU with common selection variables. One stage of an arithmetic logic shift unit is shown in fig.



operation select

S_3	S_2	S_1	S_0	C_{in}	operation	Function
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	$F = A+B+1$	Addition with carry
0	0	1	0	0	$F = A+\bar{B}$	Subtract with borrow
0	0	1	0	1	$F = A+\bar{B}+1$	Subtract
0	0	1	1	0	$F = A-1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	X	$F = A \wedge B$	AND
0	1	0	1	X	$F = A \vee B$	OR
0	1	1	0	X	$F = A \oplus B$	XOR
0	1	1	1	X	$F = \bar{A}$	Complement A
1	0	X	X	X	$F = shr\ A$	Shift right A into F
1	1	X	X	X	$F = shl\ A$	Shift left A into F.