Unit - II

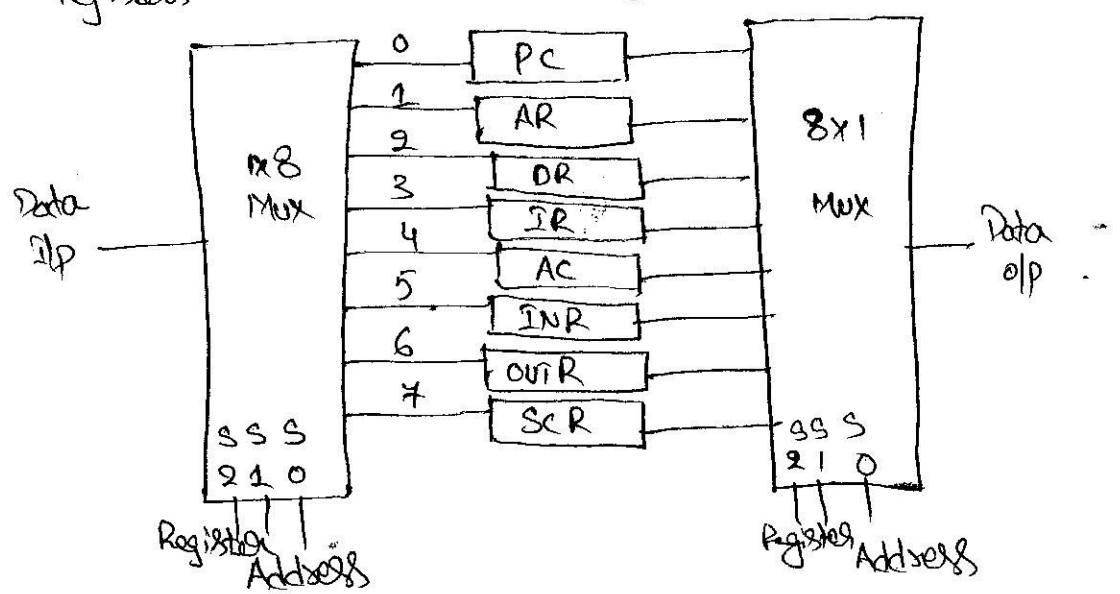
Register Transfer & Micro operations

*) **Register:-** is a collection of binary storage Elements
→ In theory Register is sequential logic which can be defined by state table
→ More often think of a register as storing a vector of binary values.
→ used to perform:- simple data storage ; data movement ; Processing
→ The operation on data in Registers are called Micro operations

Micro operations ← functions built in to Registers { shift, load, clear, increment, ... }

→ Registers are constructed using flip-flops and combinational circuits that enable one to
(i) Refresh volatile data ; (ii) Load (store) data
(iii) clear storages (change all bits to '0')
(iv) increment & decrement storage binarily
(v) complement storages
(vi) select individual storage bits

*) **Model of CPU Registers :-**
CPU registers are organized on an internal bus network
→ Accessible using a multiplexer
→ Registers are selected according to selecting inputs



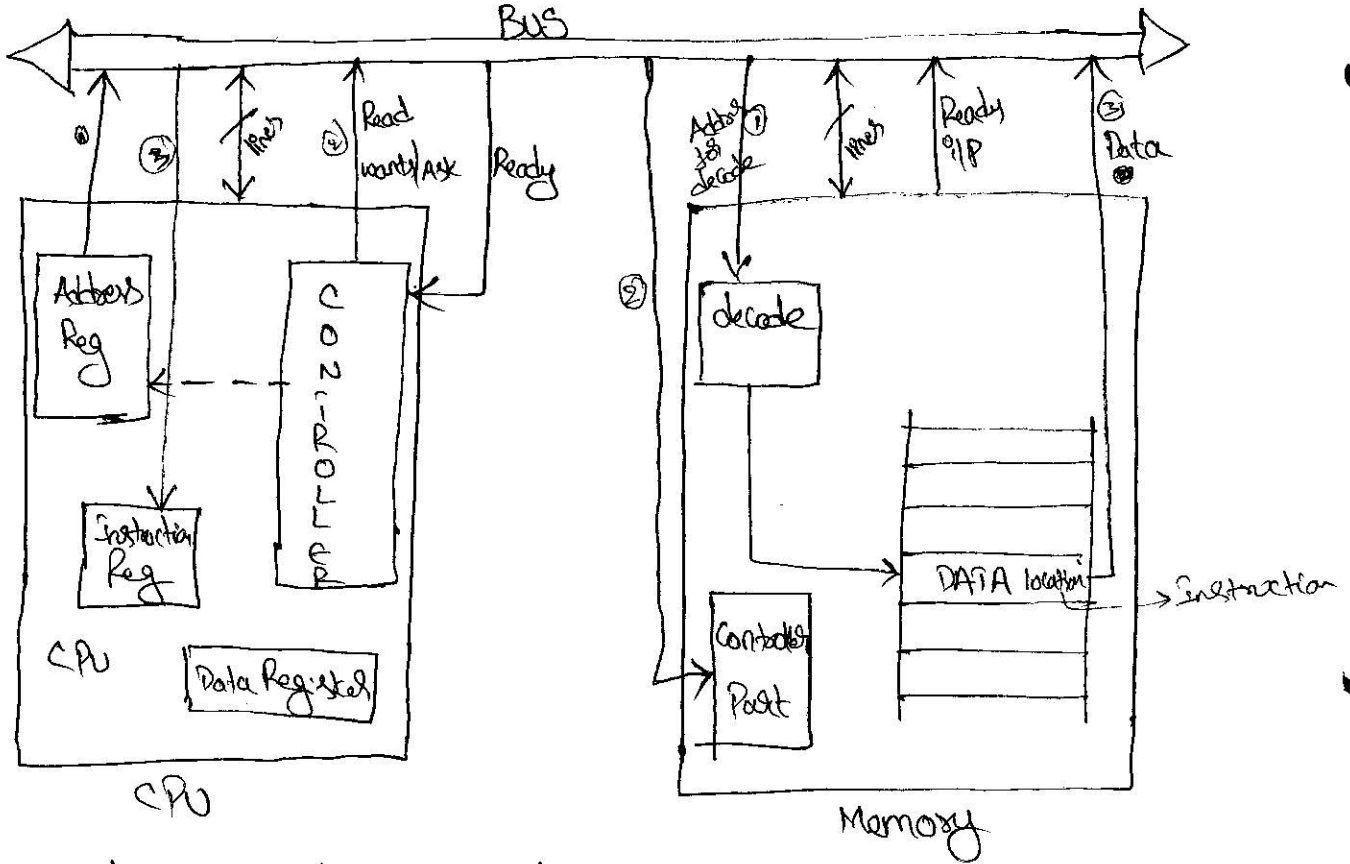
→ PC : Program Counter
→ IR : Instruction Register
→ AR : Address Register
→ DR : Data Register
→ AC : Accumulator
→ INR : Input buffer Register
→ OUTR : output Buffer Register
→ SCR : Sequence Counter Register

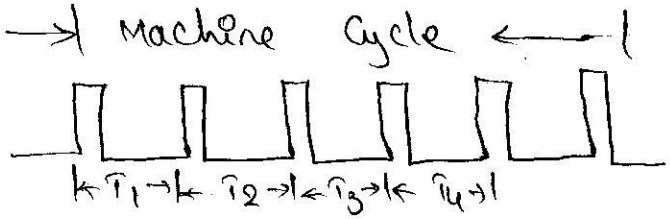
*) Register Activity :-

(i) Register - Transfer - Level (RTL)

Activity of CPU & Memory through Bus.

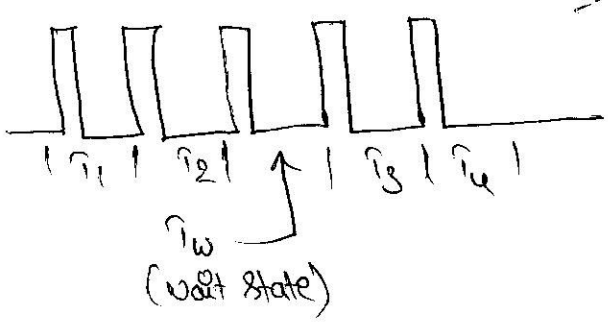
① CPU send Address to memory to fetch instruction data
② memory





Eg :-
CPU clock = 100 nsec (fast Process)
memory Access time = 200 nsec (slow respond in memory)

→ In this case wait (Ready i/p signal is Processed Tw (wait state cycle) → time signal

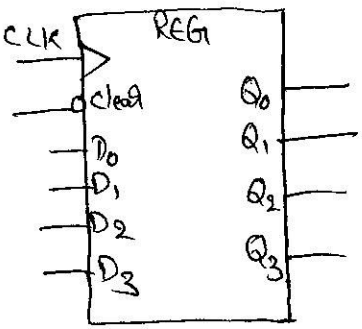


Activity of CPU Placing	on to Bus	at Time Cycles
Address	→ Address Bus	→ T_1 Cycle
		→ T_2 Cycle (fetch instruction)
Read Controller	→ Control Bus (Control Part)	→ T_3 Cycle
Data (D:Bus)	→ Data Bus (Data Part)	→ T_4 Cycle. } Execute
Data Bus (CPU)	→ Instruction Reg	

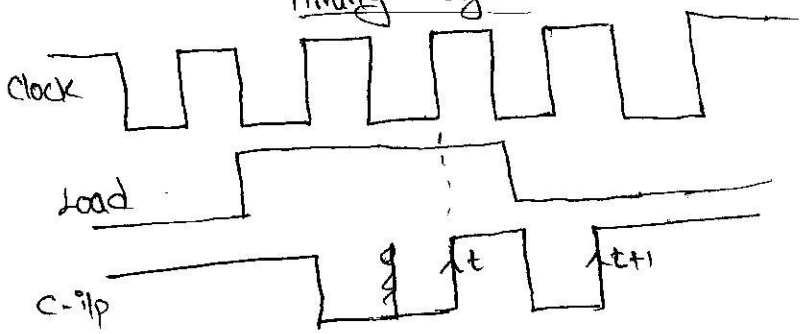
(*) Machine Cycle :-

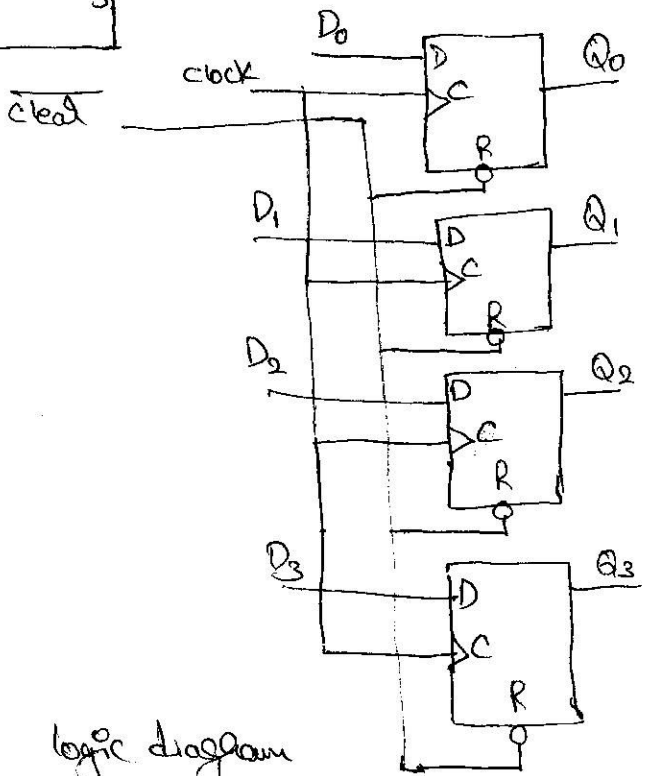
1. CPU addressing memory
2. CPU generates the control signal (Read/write)
3. Memory place the Data on the Bus
4. CPU Reads an instruction (data).

(*) Register and load Enable :-

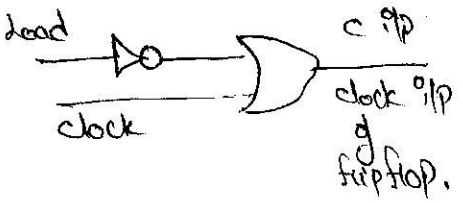


Timing diagram





logic diagram



*) Register Transfer Language :-

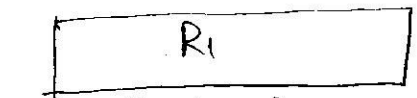
The symbolic notation used to describe microoperations transfers among registers is called Register transfer Language (RTL).

→ 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 same (or) other register.

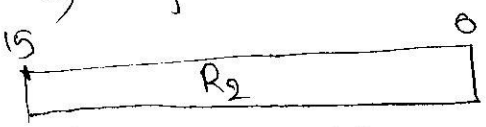
→ In Register transfer :- Registers are designated as ~~[illegible]~~ by Capital letters denote the function of register (R).

→ A register that hold register of memory unit is called MAR.

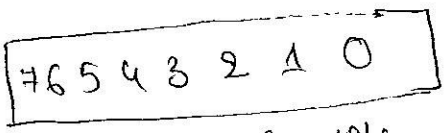
→ The individual flip-flop in an n-bit registered are numbered from 0 through n-1,



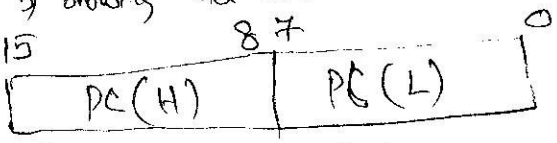
a) Register R



c) numbering of bits



b) showing individual bits



d) Divided into two Parts.

PC(L) (0-7) → Low Order byte

PC(H) (8-15) → Higher " "

Information transfer from one Reg to another Reg is designated symbolic

(Destination Reg) (Source Reg)

Eg: $R_2 \leftarrow R_1$

→ The Content of R_2 is overwritten by Content of R_1 Reg.

→ The Content of R_1 is not be changed after transfer.

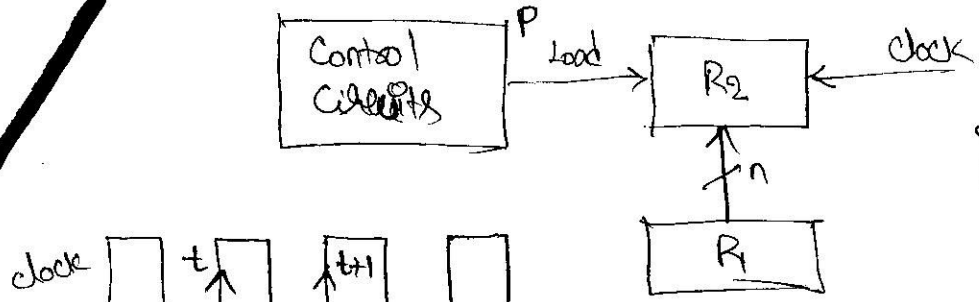
→ The transfer of Content of Registers two occurs only under a predetermined Control Condition.

By using "If-then" Statement

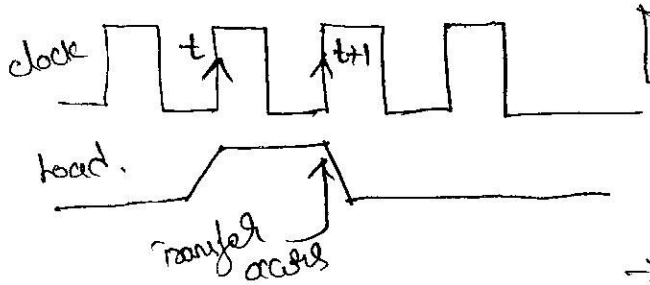
If $(P = 1)$ then $(R_2 \leftarrow R_1)$

where 'P' is a Control Signal

$P : R_2 \leftarrow R_1$



If $P = 1$ then $R_2 \leftarrow R_1$



$P \rightarrow$ is active in the control selection by rising edge of clock pulse at time 't' $\rightarrow$ next positive transition of clock at 't+1', finds the load i/p active & the data i/p of R_2 are then loaded into register in parallel.

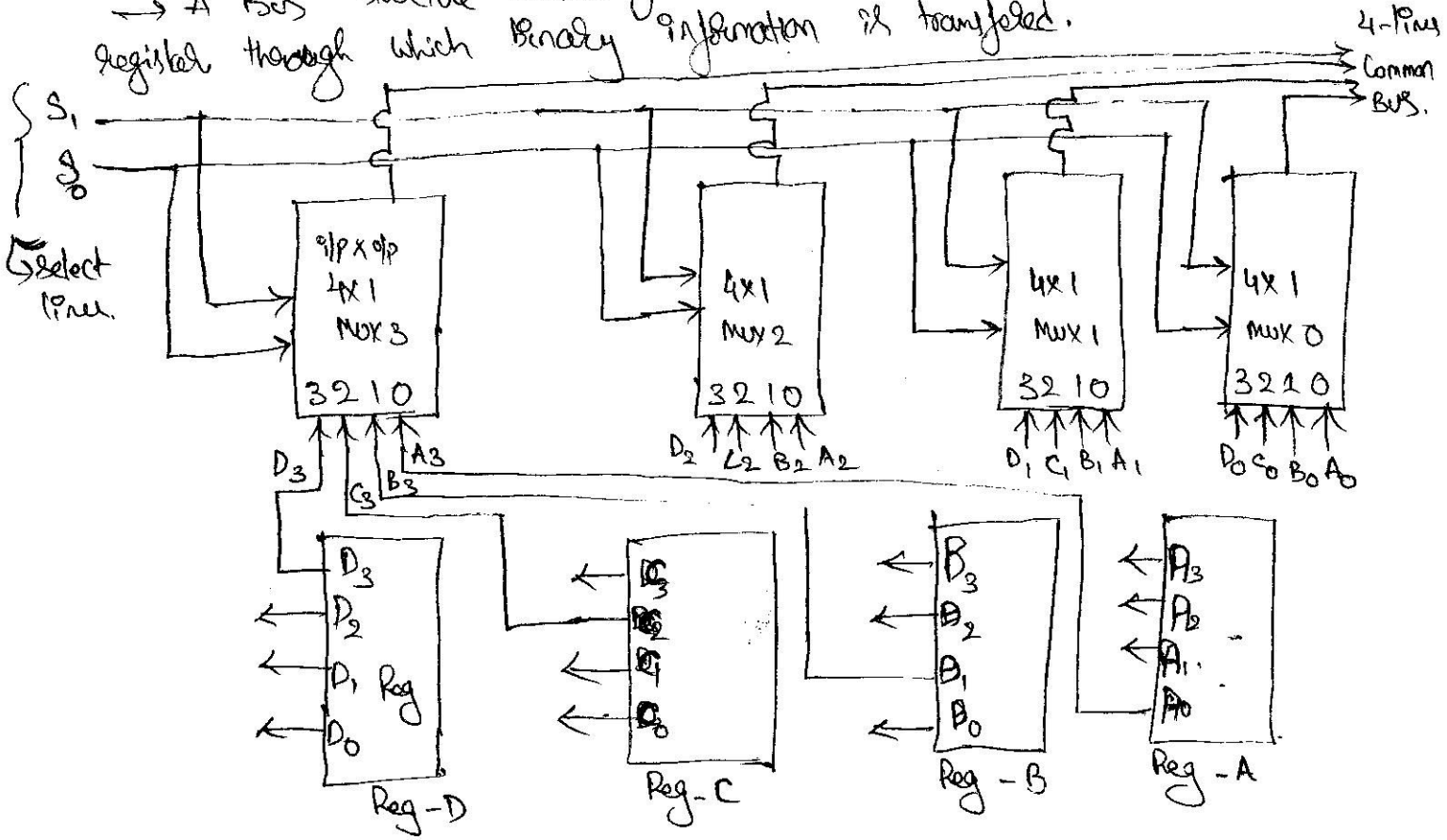
Eg.2 Statement :- $T : R_2 \leftarrow R_1 , R_1 \leftarrow R_2$

denotes an operation that exchanges the contents of two register during one common clock pulse, provided that $T = 1$ //.

2) Bus and Memory Transfer :-

→ The most significant scheme for transfering information in a register is a multiple register configuration is a Common Bus system.

↔ A Bus structure consists of common lines, one for each bit of a register through which binary information is transfered.



Functional Table:-

Select lines		Register Selected
S_0	S_1	
0	0	A
0	1	B
1	0	C
1	1	D.

→ When selection lines are S_0 & $S_1 = 00$ then the MUX will select the Reg A data as i/p and represents Reg-A through all MUX's and send through o/p as Common Bus.

→ If when the Bus is includes in statements, the registers transfer is denoted as

$BUS \leftarrow C$ (i.e Reg C is placed on Bus)

$R_1 \leftarrow BUS$ (Bus content is placed in Reg R_1).

∴ $\boxed{R_1 \leftarrow C}$.

3) Three-State Bus buffers:-

→ A Bus can be constructed with three-gates instead of multiplexers

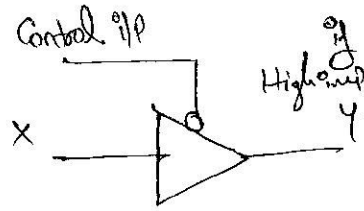
→ It is also known as 3-State drivers. These are designed to have greater o/p current & voltage capability than ordinary circuit.

→ The 3-State drivel is a digital circuit exhibits 3-states

① Logic 1 ② Logic 0 ③ High impedance state (or) floating

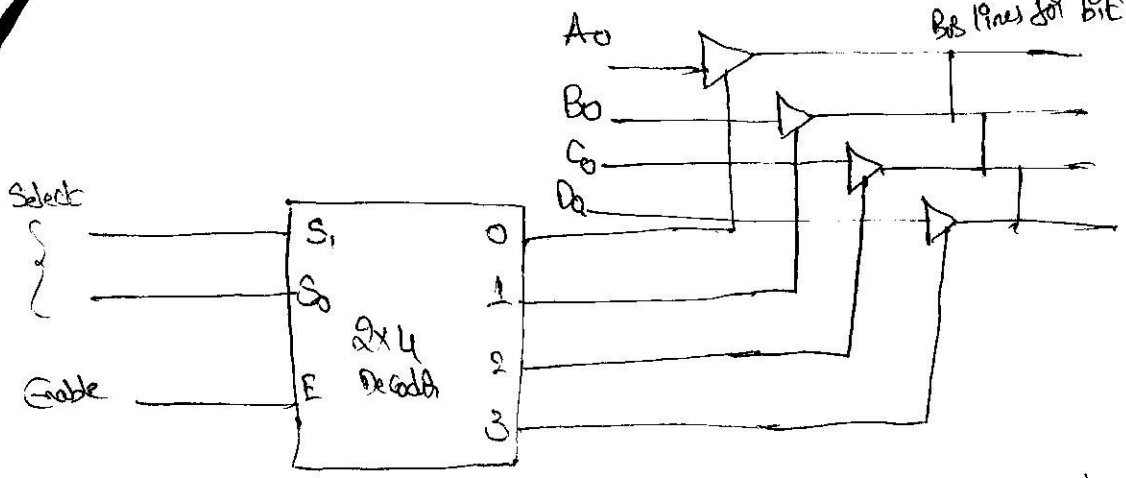
→ The high impedance state is o/p appears as disconnected like an open state

Enable	x	y
0	x	Z (Hi-impedance)
1	1	1
1	0	0



Enable	x	y
0	0	0
0	1	1
1	X	Z (Hi-Z)

High impedance :- A point in a circuit allows respectively small amount of current per unit applied voltage. [High impedance circuits are low current, high voltage].



Construction of Common Bus using three state buffers.

4) Memory Transfers :-

→ Transfer of information from a memory word to outside environment is called a Read operation

→ the transfer of new information to be stored in to memory is called a write operation.

→ It is necessary to specify the address of M when writing memory transfer operation.

i.e the address is enclosed with in brackets of 'M'

→ the memory unit that receives the address from a register called the address register [AR].

→ the data are transfered to another register (Data register [DR])

Read operation Read : DR ← M[AR]

write operation write : M[AR] ← R1 // transfer of R1 into memory word //

5) Arithematic micro operations :- (4 - Categories used)

(1) Register transfer m.o transfer binary information from one Reg to Reg.

(2) Arithematic m.o . Perform arithematic operations on numeric data stored in Register.

(3) Logic operation Perform bit manipulation operation on non-numeric data stored in Registers.

(4) Shift operation Perform shift operation on data stored in Register.

The Micro Operations:-

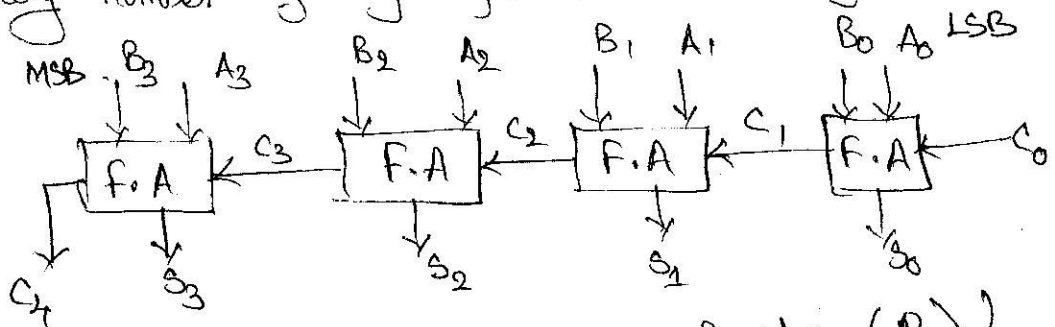
Symbolic designation	Description
$R_3 \leftarrow R_1 + R_2$	Contents R_1 Plus R_2 transfered to R_3
$R_3 \leftarrow R_1 - R_2$	Contents R_1 minus R_2 " " R_3
$R_2 \leftarrow \overline{R_2}$	Complements the Contents of R_2 (1's Complement)
$R_2 \leftarrow \overline{R_2} + 1$	2's Complement of R_2 Contents (negate)
$R_3 \leftarrow R_1 + \overline{R_2} + 1$	R_1 Plus 2's Complement of R_2 (subtraction)
$R_1 \leftarrow R_1 + 1$	Increment the Content of R_1 by one
$R_1 \leftarrow R_1 - 1$	Decrement " " R_1 by one.

(1) Binary Adder:-

To implement add micro operation with hardware, we need the Registers to hold the data and the digital component that perform the arithmatic operation.

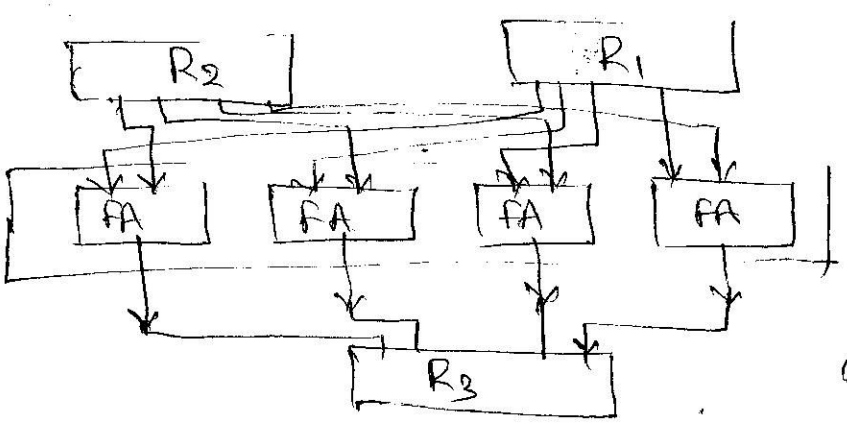
The digital circuit that perform the arithmatic sum of two bits and previous carry is called a full adder.

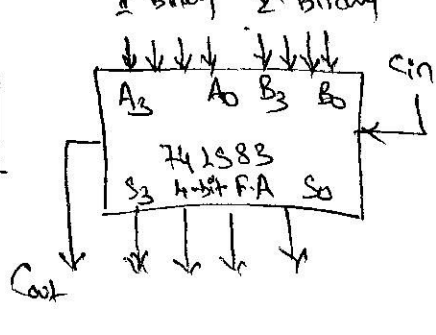
→ The digital circuit that generate the arithmatic sum of two binary number of any length is called binary adder. (4-bit Parallel Adder too.)



(Parallel Adder)

A i/p Comes from one Register (R_1)
B i/p " " " Register (R_2)
} + transfered to Register - R_3 by replacing its previous Contents.





full Adder :- FA 1 (2-bit full Adder)

$$Sum = A_0 \oplus B_0 \oplus C_0$$

$$= \bar{A}_0\bar{B}_0C_0 + A_0\bar{B}_0\bar{C}_0 + \bar{A}_0B_0\bar{C}_0 + A_0B_0C_0$$

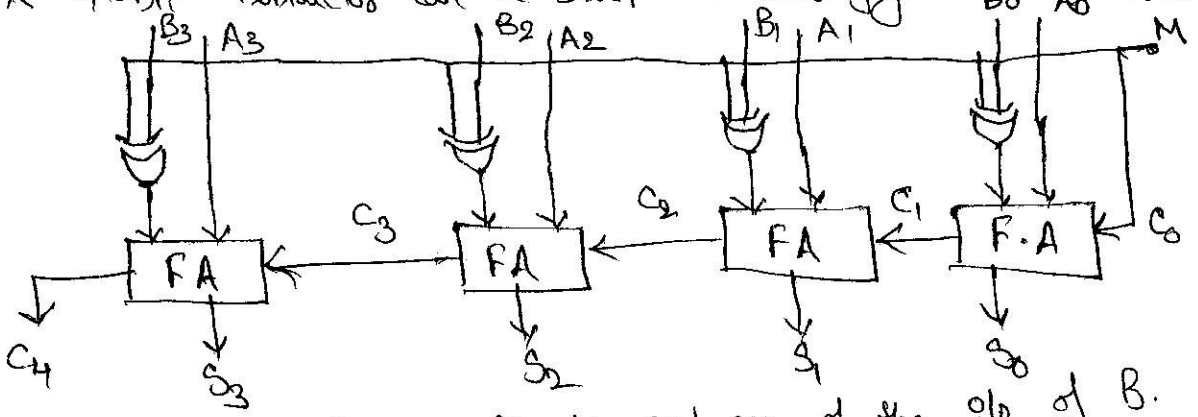
$$C_1 = A_0B_0 + A_0C_0 + BC_0 : 0 \text{ (1)}.$$

I/p A.				I/p B				Addition				
A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	C	S_3	S_2	S_1	S_0
1	0	0	0	0	0	1	0	0	1	0	1	0
1	0	0	0	1	0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0	0	1	0	1	0
0	0	0	1	0	1	1	1	0	1	0	0	0
1	0	1	0	1	0	1	1	1	0	0	1	0
1	1	1	0	1	1	1	1	1	1	0	1	0
1	0	1	0	1	1	0	1	1	0	1	1	1

Binary Adder - Subtractor :-

The addition & subtraction operation can be combined into one common circuit by including exclusive-OR gate with each full adder.

→ A 4-bit subtractor can be shown in below fig.



Each Ex-OR gate receives i/p M and one of the i/p of B.
When M=0, we have $B \oplus 0 = B$. then F.A receives the value of B, the i/p Cary is '0' and operation performed by circuit is A plus B ⇒ A+B.

i.e when M=0 each i/p of Ex-OR is $B \oplus 0 = B$
operating of Cary = 0 ; then it perform **A+B**

when M=1 each i/p of Ex-OR is $B \oplus 1 = B'$ & $C_0 = 1$
i.e B is complemented and '1' is added through Cary i/p. and circuit perform 2's complement of 'B'.

→ for unsigned number this gives (A-B) if (A ≥ B (or) the 2'complement of (B-A) · if (A < B)

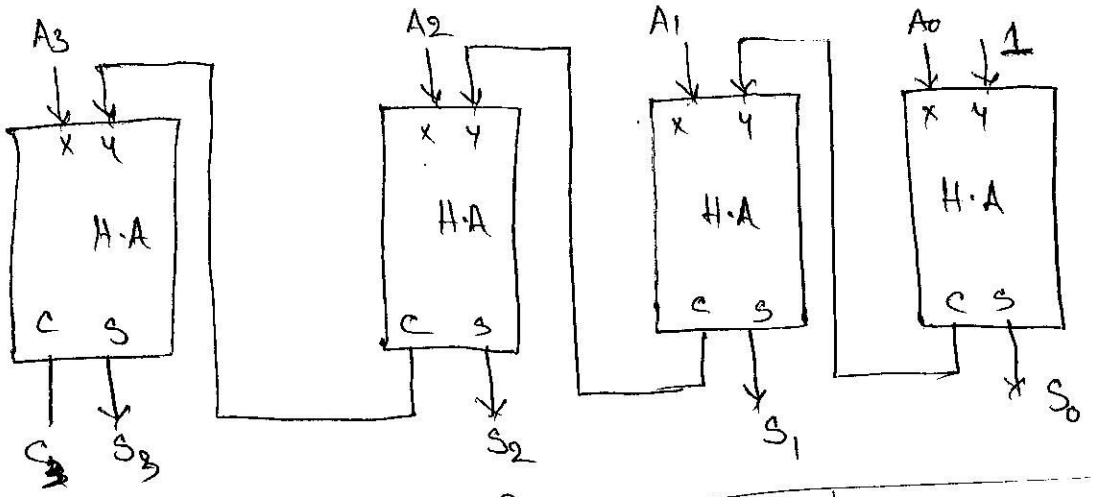
→ for signed number result is 'A-B'.

Binary Increment :-

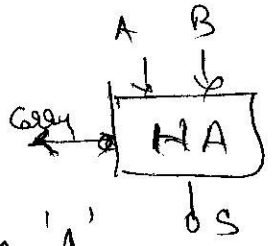
The increment micro operation adds one to a number in Register

Eg:- 0110 + 1 = 0111

The least significant half Adder in circuit is connected to logic-'1' and another i/p is connected to LSB of the number to be incremented.



w.r.t H.A ⇒



A	B	Sum = A⊕B = $\bar{A}B + B\bar{A}$	Carry = A·B
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

If one i/p is logic '1'

Eg: If Reg A = 1000 then,

1000
1
1001 ← incremented

Arithematic Circuit :-

The basic Component of an Arithematic circuit is Parallel Adder by Controlling the data i/ps to adder it is possible to obtain different types of Arithematic operations,

→ It has full Adder circuits that consists 4-bits adder and 4-Multiplexer for chossing different operations

→ There are two 4-bits i/p A & B and 4-bit o/p 'D'.

→ The 4-bits i/p from A go directly to X-i/p of the Binary Adder.

→ Each of 4-i/p of B Connected to the data i/ps of multiplexers.

→ The multiplexed data i/p also received the complement of B.

→ The other two data i/p are connected to logic '0' & '1'

→ The 4-Multiplexer are Connected by selection i/p -2 S0 & S1

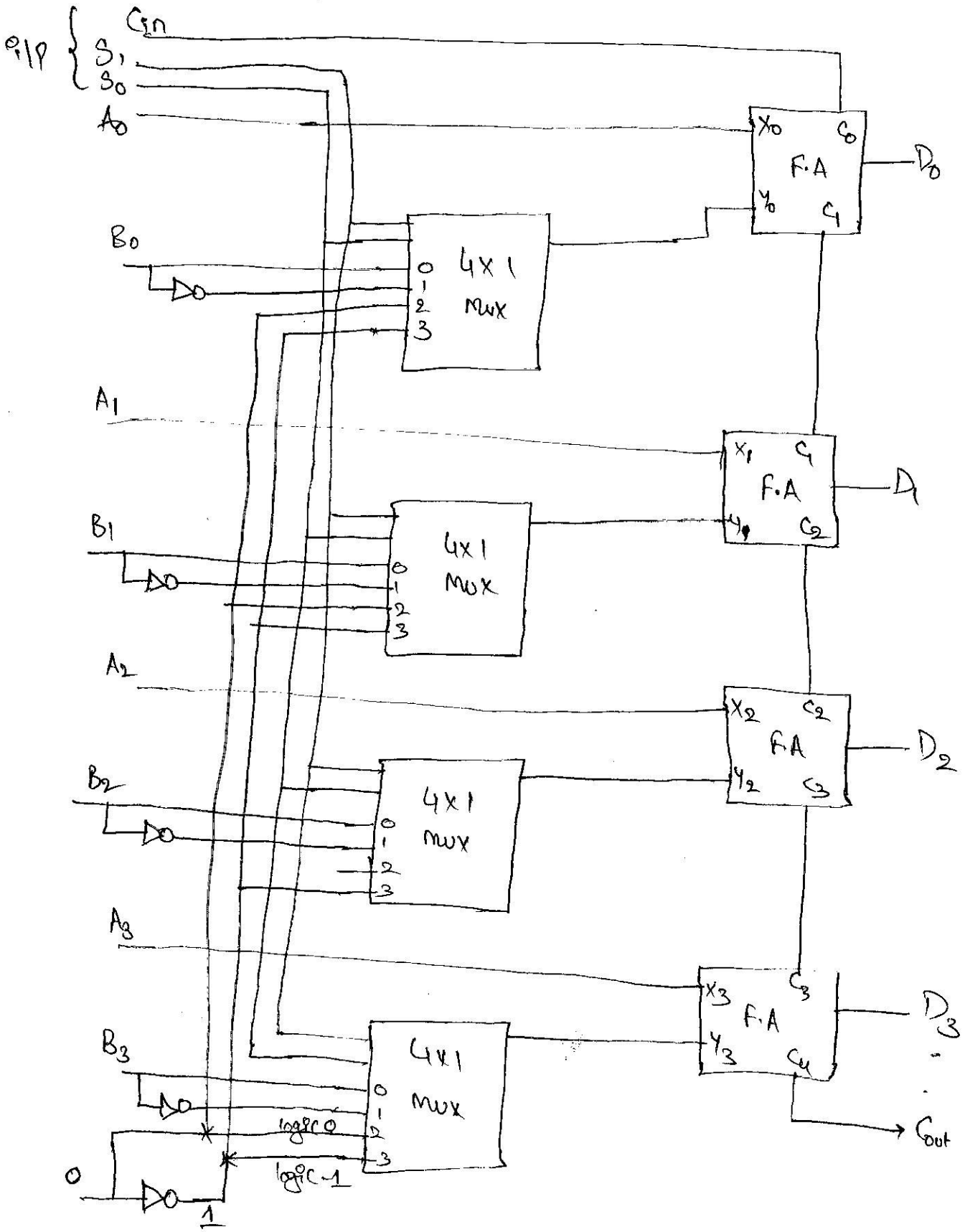
→ The i/p Carry is fed to Carry i/p of F.A in least significant position.

→ The other Carry i/p of F.A are connected from one stage to next

→ the o/p of binary adder is calculated from arithmetic sum

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

→ where A is the 4-bit binary number at X-i/p and Y-i/p is 4-bit binary number at Y-i/p of binary Adder.; C_{in} is i/p carry which can be Equal to '0' (or) '1'



Select			I/p	o/p	MiCro operation
S_0	S_1	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	$\bar{B}$	$D = A + \bar{B}$	Subtract with borrow
0	1	1	$\bar{B}$	$D = A + \bar{B} + 1$	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

Eg:- $A = \overset{A_3\ A_2\ A_1\ A_0}{1101} \Rightarrow$ ~~A2~~ = A, ~~A~~

$B = \underset{B_3 B_2 B_1 B_0}{1001}$

If select = 011 then, $D = A + \bar{B} + 1$

$A = 1101$; $\bar{B} = 0110$

$+ \quad 1 = 1$

0111

$\therefore D = A + \bar{B} + 1$

$= 1101 = 13$

$10111 \quad -9$

$\overset{1}{\textcircled{1}}0100 = 4$

$(-)\overset{13}{9}$
4

---

*) Logic MiCro operations :-

Specify operations for strems of bits stored in registers.

Special Symbols :- OR - denoted as 'V'

AND - " " '$\wedge$'

Eg:- $R_4 \leftarrow R_5 \vee R_6$ // R_5 OR R_6.

(i) List of logic MiCro operations:-

There are 16 - different logic operations that can be performed with two binary variables.

2 - variables x & y

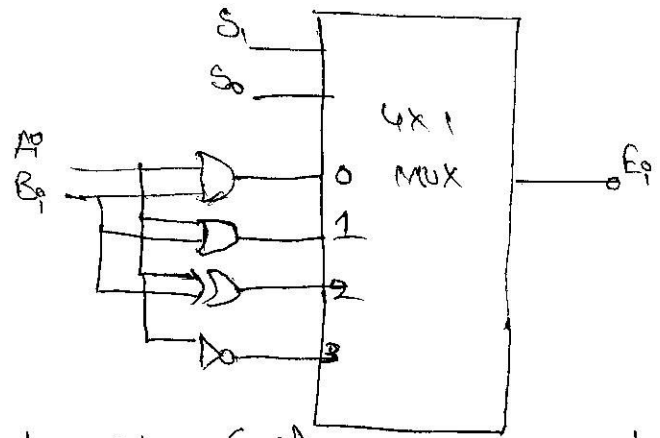
16 - Colourmns ($f_0 \rightarrow f_{15}$).

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
Boolean function		0	xy	$x\bar{y}$	x	$\bar{x}y$	y	$x \oplus y$	$x + y$	$\overline{x + y}$	$\overline{x \oplus y}$	$\bar{y}$	$x + \bar{y}$	$\bar{x}$	$\bar{x} + y$	$\overline{xy}$	1
			"	"	"	"	"	"	"	"	"	"	"	"	"	"	"
			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}

	Boolean function	Microoperation ($x = A$, $y = B$)	Name
1)	$F_0 = 0$	$F \leftarrow 0$	clear.
2)	$f_1 = xy$	$F \leftarrow A \wedge B$	AND
3)	$f_2 = x\bar{y}$	$F \leftarrow A \wedge \bar{B}$	Transfer-A
4)	$f_3 = x$	$F \leftarrow A$	
5)	$f_4 = \bar{x}y$	$F \leftarrow \bar{A} \wedge B$	Transfer-B
6)	$f_5 = y$	$F \leftarrow B$	
7)	$f_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
8)	$f_7 = x + y$	$F \leftarrow (A \vee B)$	OR.
9)	$f_8 = \overline{(x + y)}$	$F \leftarrow \overline{(A \vee B)}$	NOR.
10)	$f_9 = \overline{(x \oplus y)}$	$F \leftarrow \overline{A \oplus B}$	exclusive NOR.
11)	$f_{10} = \bar{y}$	$F \leftarrow \bar{B}$	Complement B
12)	$f_{11} = x + \bar{y}$	$F \leftarrow (A \vee \bar{B})$	
13)	$f_{12} = \bar{x}$	$F \leftarrow \bar{A}$	complement-A
14)	$f_{13} = \bar{x} + y$	$F \leftarrow (\bar{A} \vee B)$	
15)	$f_{14} = \overline{(xy)}$	$F \leftarrow \overline{(A \wedge B)}$ → NAND	
16)	$f_{15} = 1$	$F \leftarrow$ all 1's	Set all 1's.

Hardware Implementation :-

To design logic micro operation in hardware, requires that logic gates



function table :-

S_1	S_0	o/p	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \bar{A}$	Complement

(i) Selective Set :- (OR)

If $S_1 S_0 = 01$ then select '1' in B's

Performs 'OR' operation.

Eg :-
```
  1010   A (before)
+ 1100   B (logic operand)
------
  1110   After
```
↳ OR - operation.

Any ← X X X X (A)
(B) 1 0 1 0
1 X 1 X
↳ A (After)

(ii) Selective - Complement.

If selective Complement bits in A, when corresponding to bits with 'B' as 1's

Eg :-
```
  1010   A before
  1100   B (logic operand)
------
  0110   (A - After)
```
↳ Ex - OR operation.

(iii) Selective clear :-

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

```
  1010  → A (before)
  1100  → B (logic operand)
------
  0010  → A (After)
```

logic micro operation ($A \leftarrow A \wedge \bar{B}$)

(iv) The Mask operation :- Similar to selective clear operation.

Except that the bits of A are cleared only where there are corresponding 0's in B.

→ It is an AND operation.

```
  1010   A (before)
  1100   B (logic operand)
------
  1000 → A - After
```
↳ AND operation.

(v) Insert operation :-

Inserts a new value into a group of bits. This is done by first masking the bits and then perform OR - operation.

If A-Register has (8-bits) = 0110 1010 to replace four left most bits by value 1001 we first mask the four unwanted bits.

AND operation
```
  0110 1010  → A before
  0000 1111  → B (mask)
------
  0000 1010  → A after masking
```
changed ↓ ↳ not changed

and then insert new value

OR operation
```
  0000 1010   A - before
  1001 0000   B (insert)
------
  1001 1010   A - After insertion
```
↳ left most bits are changed.

(vi) The Clear operation :-

It Compares the words A & B and Produces all 0's result if two numbers are equal.

→ If 2-Registers data is equal then clears and sets to 0's.

→ It is an Ex-OR operation.

```
1010    A
1010    B
----
0000    A ← A⊕B //.
```

*) Shift Micro operations :-

are used for serial transfer of data. The contents of registers are shifted to the left (or) Right

3-types of shifts

1) Logical 2) circular 3) Arithematic.

1) Logical shift :- is one that transfers '0' through the serial i/p.

denoted as shl & shr

(logical shift left) / (logical shift Right)

Specify

Eg:- $R_1 \leftarrow shl\ R_1\ (R \rightarrow 2)$ → 1 bit shift left of content of R_1 Register

$R_2 \leftarrow shr\ R_2\ (2 \rightarrow R_0)$ → 1 bit shift to right content of R_2 Reg

2) Circular shift :- (Rotate operation) the bits of register around the two ends without loss of information.

denoted as cir & cil

circular right / circular left

Symbolic designation :-

$R \leftarrow shl\ R$

$R \leftarrow shr\ R$

$R \leftarrow cil\ R$

$R \leftarrow cir\ R$

$R \leftarrow ashl\ R$

$R \leftarrow ashr\ R$

Description

shift-left register-R

shift-Right " "

circular shift left R

circular shift right R

Arithematic shift left R

" " R.

...ft Operations :-

...ithematic Shift :-

It a micro operation that shift a signed binary number to the left (or) the right.

→ An arithematic shift - left multiplies a signed binary number by 2.

→ An arithematic shift - right divided the number by 2.

→ ashl → Arithematic shift left means multiply by 2.
ashr → " " right means divide by 2.

Eg:- Unsigned number:- (Right shift)

	Position
1 0 0 0 →	0
1 0 0 0 →	1
1 0 0 0 →	2
1 0 0 0 →	3
1 0 0 0 →	4

micro operation. shift Right

	position.
1 0 0 0 →	(8)
0 1 0 0 →	(4)
0 0 1 0 →	(2)
0 0 0 1 →	(1)
0 0 0 0 →	(0)

divide by '2'

Unsigned:- (left shift)

	Position
0 0 0 1 ←	0
0 0 0 1 ←	1
0 0 0 1 ←	2
0 0 0 1 ←	3
0 0 0 1 ←	(4)

micro operation left shift

	value
0 0 0 1 ←	(1)
0 0 1 0 ←	(2)
0 1 0 0 ←	(4)
1 0 0 0 ←	(8)
0 0 0 0 ←	? → there after multiply of 8 with 2 = 16. so it needs 5-bits

multiplies of '2'.

Eg:- Signed operation:- (2's Complement Representation)

Right Shift

	value
1 0 0 0	= -8

MSB = 1 (-ve)

	Position
1 0 0 0 →	0
1 0 0 0 →	1
1 0 0 0 →	2
1 0 0 0 →	3
1 0 0 0 →	4.

shift operation.

micro operation. shift right.

	value
1 0 0 0 →	(-8)
1 1 0 0 →	(-4)
1 1 1 0 →	(-2)
1 1 1 1 →	(-1).

divided by '2'.

Left Shift.

1 0 0 0 →	0
1 0 0 0 →	1

left micro operation.

1 0 0 0 →	(-8)
1 1 0 0 0 →	(-16)

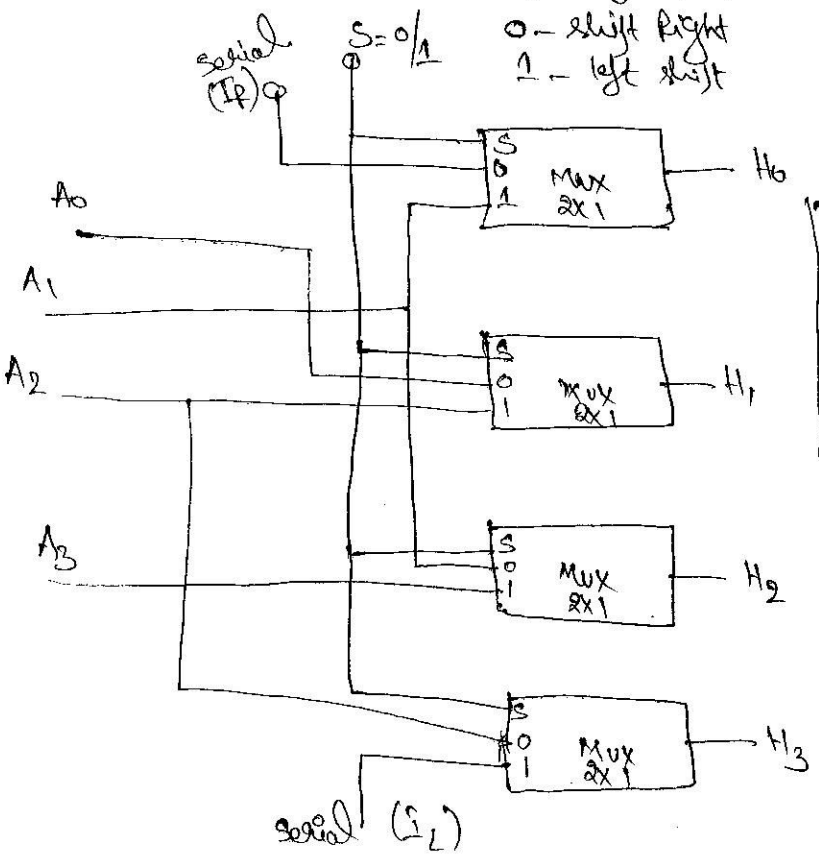
} multiple by '2'

Hardware implementation:-

Information can be transfered to the register in parallel and then shifted to the right (or) left.

The combination circuit is constructed using multiplexers.

→ The 4-bit shifter has 4-data i/p ($A_0 \rightarrow A_3$) and 4-data o/p ($H_0 - H_3$)
→ has 2-serial i/p → 1-shift left (I_L) & other 1-shift right (I_R)
→ When selection S = 0 data shift Right / down.
S = 1 data shift left / up.



functional 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

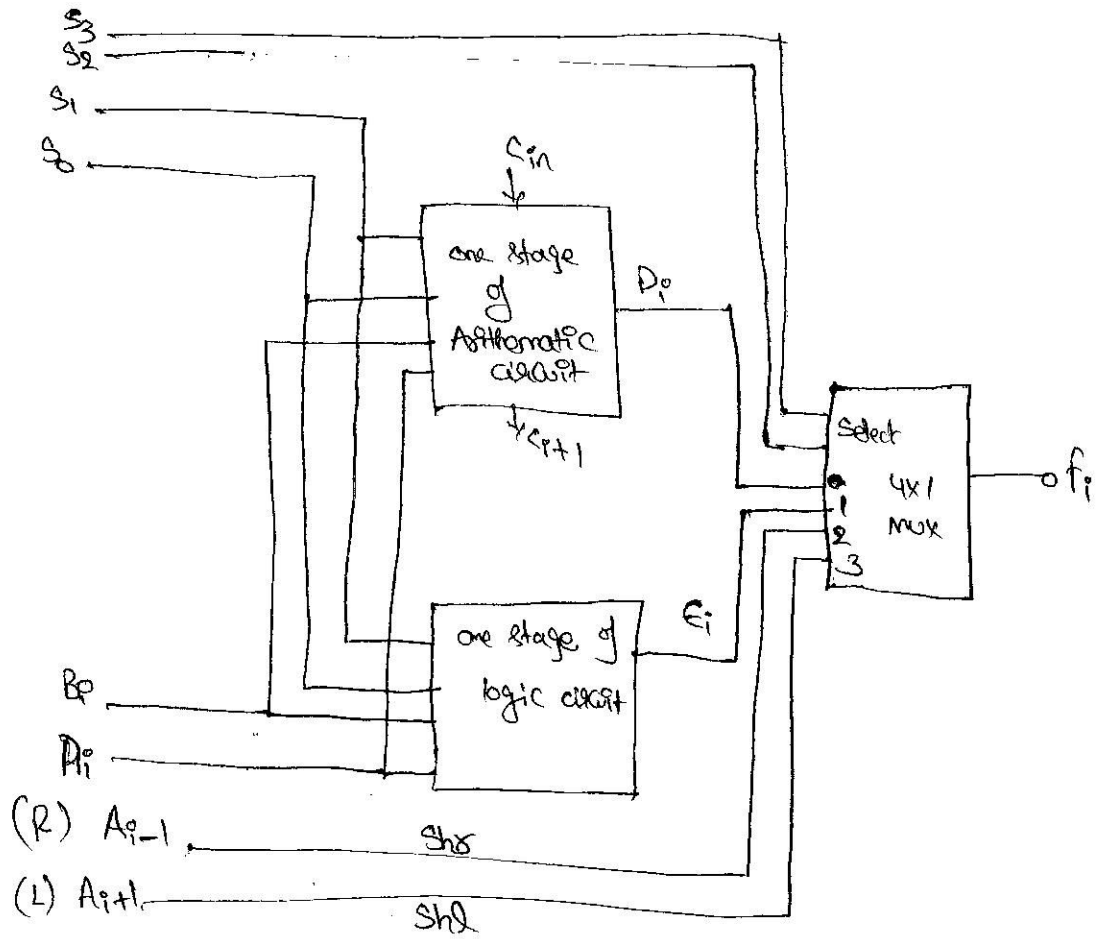
(ii) Arithematic logic shift unit :-

→ A no. of storage registers connected to a common operational unit called an arithematic logic unit (ALU).

→ ALU performs operations and result of operation is transfered to destination registers.

→ The Arithematic, logic, shift circuit can be combined into one ALU with common selection variable.

→ In this type configuration a clock pulse is needed for loading the data into the registers and another pulse is needed to initiate the shift.



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

✳ Instruction Codes:-

→ A Computer instruction is binary code specifies sequence of micro operations.

→ I.C together with data are stored in memory.

→ Computer reads each instruction from memory and place in Control Reg (CR).

→ IC is a group of bits that instruct Computer to perform a specific operation.

→ this is divided in-to parts.

① Basic Part of operation is operation part.

→ The operation code of instruction code is a group of bits that define such operations Add, Subtract, multiply, shift, & complement.

→ the operation code must consists of atleast n-bits for a given 2^n (or) less distinct operations.

(*) Consider a Computer 64-distinct operations, one of them begin an Add operation.

The operation code consists of 6-bits with bit Configuration 110010 assigned to ADD operation.

When this operation code is decoded in the Control unit the Computer issues Control signals to read an operand from memory and add. the operand to processor register.

→ in Computer binary code that tells the Computer to perform a specific operation

→ the Control unit receives the instruction from memory and interprets the operation code bits.

→ For every operation code the Control issues a sequence of micro operations needed for the hardware implementation of specified operation. for this reason an operation code is sometimes called macro operation because specifies a set of micro operations.

→ An instruction Code must therefore specify not only the operation but also the registers or memory words where the operands are to be found, as well as where result is to be stored.