COA

Intro

A digital computer is a digital system that performs various computational tasks
Digits → binary system → 0 & 1's.

→ binary digit – bit
→ Info is rep in group of bits
→ using various coding schemes groups of bits can be made to represent not only binary no's but also discrete symbols such as decimal digits or letters of alphabets

Computer System
Functional Entities
Hardware Software

Hardware:- The hardware of the computer system consists of all the electronic components & electromechanical devices that comprise the physical entity of the device.

__/__/____

Software: Computer s/w consist of instructions & data that a computer manipulates to perform various data processing tasks.

✡ Basic Terms related to Digital Computer

Program: A sequence of Instruction for the computer

Database: The data that is manipulated by the program constitutes the database

Sys s/w: Collection of programs
purpose - effective use of PC
required to:- co-ordinate all activities of CS.
Eg:- OS

App s/w: Written by user
purpose - solving problems.
It allows user to do things like creating text docs, playing games, use to surf on

__/__/____

✱. BLOCK DIAGRAM OF DC. — components

1) CPU.
It contains
a) ALU for manipulating data.
b) Registers for storing data.
c) CU for directing other components to perform certain actions such as fetching & execution of instructions.

2). MM.
* contains storage
* for Inst & data.
* It's called RAM.
* bcoz CPU can access any memory location at random & can retrieve info in fixed interval of time

3) SM.
* non volatile memory.
* Used :- permanent storage of data & prog.
* Eg :- Hard disk, floppy disk, Magnetic tape

4) I/p & O/p processor
It contains electronic circuits for communicating & controlling the transfer of info b/w PC & outside world.

5) I/p devices
It sends info to the comp sys for processing
Eg:- Keyboard, mouse, Mic, Webcam.

6) O/p devices
It displays the result of processing
Eg:- Monitor, projector, Speaker.

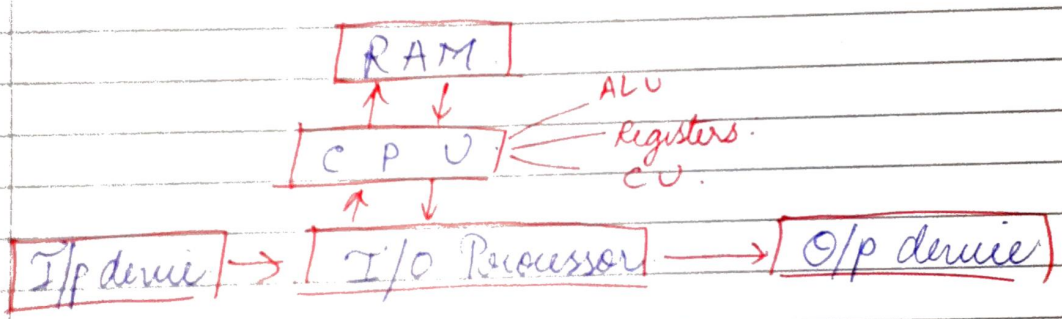


Fig. Block diag of DC

Computer Organisation
CO is concerned with the way the HARDWARE COMP Operate & the way they are connected together to form the CS.

Computer Design
CD is concerned with the HARDWARE DESIGN
It is " " " determination
of What hardware should be used &
how the parts should be connected.

__/__/____

Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system

Computer Architecture

CA is concerned with STRUCTURE AND BEHAVIOUR of the computer. It includes,

- Instruction formats
- instruction sets
- techniques for addressing memory.

The Architectural design of CS is concerned with the specifications of various funtional models such as processors & memories and structuring them together into a CS.

Two basic types of CA are.

a) Von Neumann Architecture

* The vast majority of computers in use today operate according to VNA.
* VNA is based on stored-program computer concept, where data & instructions are stored in same memory
* This architecture has only one bus for both data transfers & Inst fetches.

__/__/____

* ∴ data transfers & instruction fetches should be Scheduled as they can't perform at the same time.
* The components of VNA are.
  a) RAM.
  b) ALU.
  c) CU.
  d) Man-made Interface.
* Eg:- desktop PC

Harvard Architecture.

* It uses physically seperate memory for storing data & Inst.
* In a computer with HA, the CPU can read both D & I from memory at the same time using two diff busses (I bus, D bus) leading to double the memory bandwidth
* Eg:- Microcontroller based CS.
  DSP based CS

__/__/____

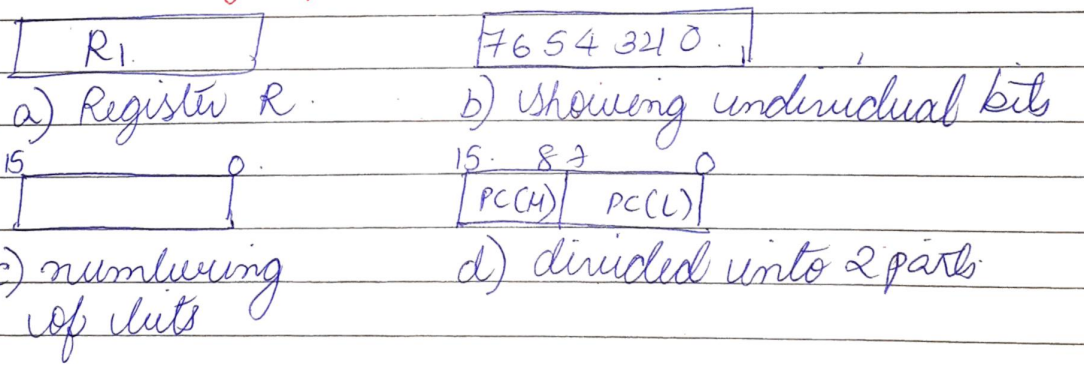
Registers :-

* Collection of flip flops where each flip-flop can store 1 bit of info
* If we take 3-bit register it stores 3 bits of info.

Micro-operation

* MO is an operation which is performed on the contents of registers
* MO is a basic operation
* MO are classified into 3 types AMO, LMO, SMO.

Block Diag of Registers



a) Register R
b) Showing individual bits
c) numbering of bits
d) divided into 2 parts

Register Transfer

Transfering the contents from one register to another register

$R2 \leftarrow R1$

↓

replacement operator

replaces the contents of R1 & by R2

$R_2 \leftarrow R_1$.

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

$P: R_2 \leftarrow P_1$

control func.

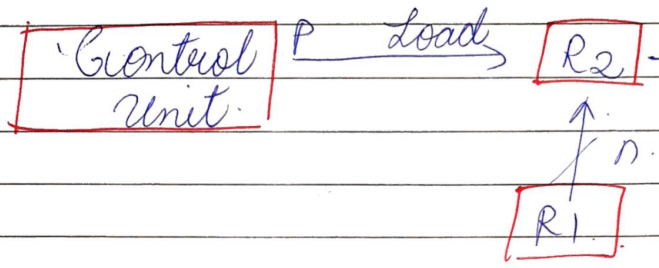


fig: Block diag depicting the transfer from R_1 to R_2

* When the clock pulse is applied then n bits of register R_1 will be transfered to R_2.

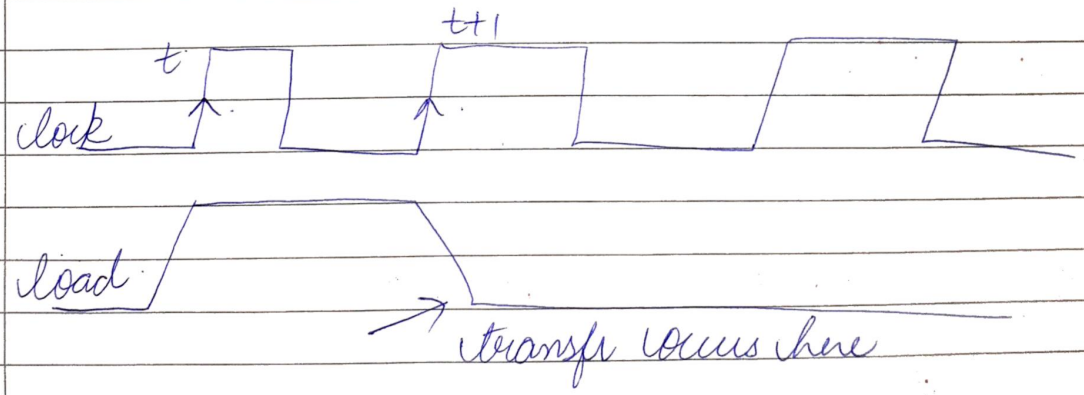


fig :- Timing Diagram

* For transferring the ~~clock pulse~~ content we require minm one clock pulse. to transf n bits from R_1 to R_2.

* When the clock pulse is over, the entire n bits will be transfered from R_1 to R_2
* at time 't' we have initiated the transfer of R_1
* at time 't+1' the transfer occurs completely

✱ Basic Symbols for RT.

1) letters & no's :- used to denote registers
   Eg :- R_1, R_3, PC, IR, MBR.

2) paranthesis :- Denotes parts of registers
   Eg :- $R_1(0-8)$, $PC(M)$.

3) Arrows $\leftarrow$:- Transfer contents of one Reg to the
   Eg :- $R_2 \leftarrow R_1$ (replacement operator)

4) Comma , :- Denotes sequence of MO.
   Eg :- $R_1 \leftarrow R_2$, $R_2 \leftarrow R_4$

✱ Register Transfer Language RTL

* RTL is a symbolic notation which Specifies RTMO.

✡. Bus & Memory Transfers.

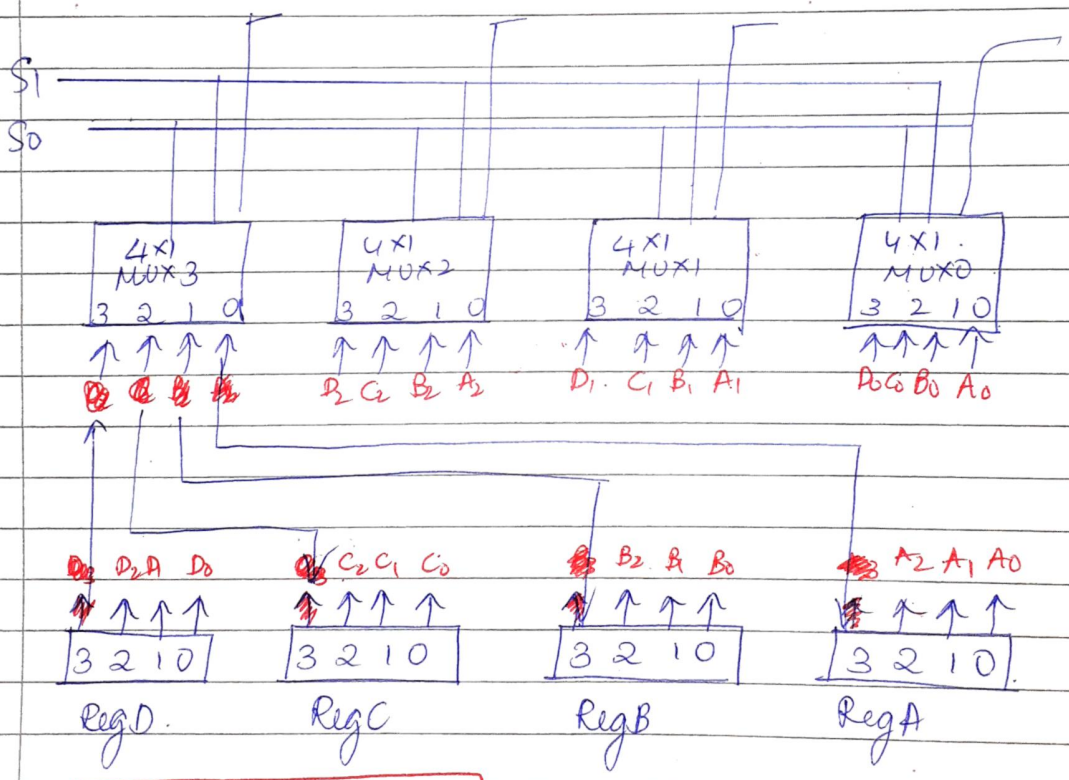
* Bus is mainly useful inorder to transfer info from one register to another register in less expensive manner.
* We can implement common bus system in 2 approaches.
  a) using multiplexers
  b) using decoders in 3 state buffers.

Implementing bus using multiplexers

* We have 4 registers, Reg A, B, C, D.
* Where the size of each register is 4 bits. (0 to 3).
* Then we have 4 multiplexers.
* The no. of multiplexers depend upon no of registers.
* The size of each multiplexer also depends upon size of register (4bits)
* Advantage of multiplexers is that it accepts 'n' selection lines
* ∴ n selections lines are to be applied to the multiplexers.
  $n \rightarrow 2^n$ o/p's $\rightarrow$ 1. $(2 \rightarrow 4)$.
* So we get 2 selection lines S_1 & S_0
* S_1 & S_0 must be applied to all multiplexers.
* 4x1 denotes 4 i/p & 1 o/p.

__/__/____

* Each multiplexer receives i/p from registers
* The '0' o/p of all registers must be applied as the i/p to MUX '0'.
* MUX '1' receives i/p from o/p 1 of all the registers



S_1	S_0	Bus
0	0	A
0	1	B
1	0	C
1	1	D

* Once the info is available in the bus, then the o/p of the bus has to be connected with the i/p to the destination register.
* So in this way we can transfer the contents of one reg to the other with the help of bus.

Memory Transfer

2 Operations :-

a) Read :- Transferring the data from memory to any register
(MRR)
IR ← m [AR]

(RWM)
b) Write :- Transferring the data from any register to memory.
m[AR] ← IR

3 State Buffer / Tri State Buffer

* Buffer is mainly used to store bits
* 3 State buffer means it produces 3 states
  1st State – 1
  2nd State – 0
  3rd State – High Impedence (no o/p) (open circuit)

graphical rep

If control Input is 1, then o/p
$Y = A$
If control Input is 0, then o/p
High Impedence

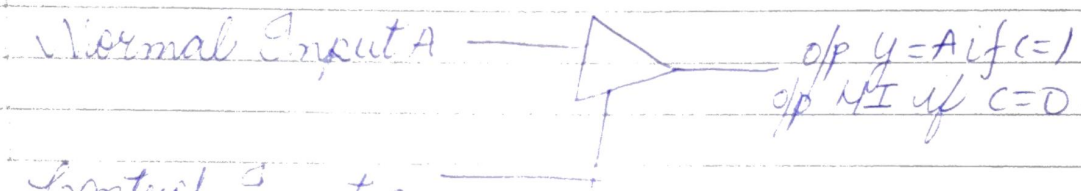


fig: graphical symbols for 3-state buffer

Bus lines with 3 state buffer

* We take a 2×4 decoder
* On decoder we have to apply enable i/p either 1 or 0
* S_1 & S_0 are the two selection i/p's
* If there are n inputs, it produces 2^n outputs

$n \rightarrow 2^n$

$2 \rightarrow 2^2 = 4$ o/p

* 3 state buffer accepts a normal i/p as well as control Input

* for this 3 State buffer, it is taking i/p from output 0 of decoder
* Let us take 4 registers.

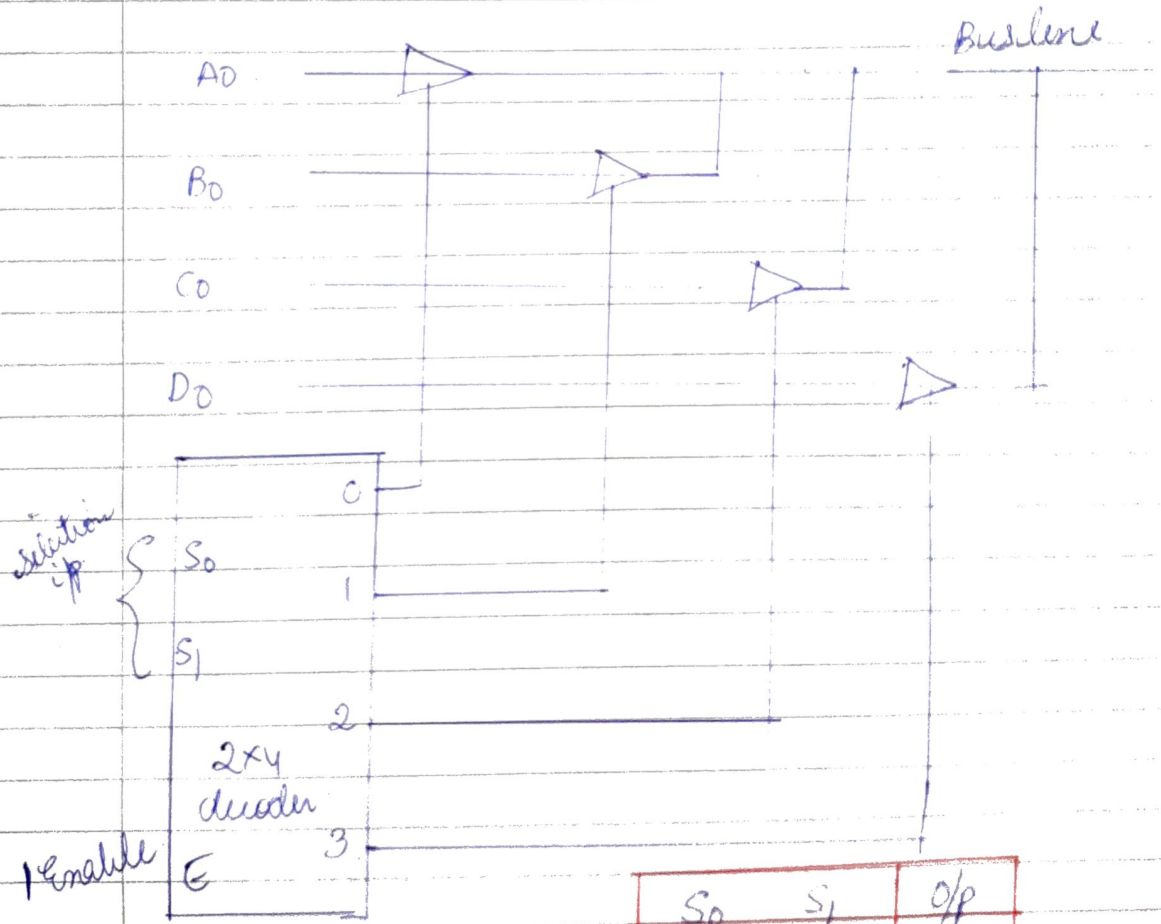
A ⟶ A_0, A_1, A_2, A_3

B ⟶ B_0, B_1, B_2, B_3

C ⟶ C_0, C_1, C_2, C_3

D ⟶ D_0, D_1, D_2, D_3

* Now, it's take only 1st bit from each register i.e A_0, B_0, C_0, D_0



S_0	S_1	O/p
0	0	0
0	1	1
1	0	2
1	1	3

Arithmetic MO.

There are 7 AMO.

1) $R_3 \leftarrow R_1 + R_2$ +
2) $R_3 \leftarrow R_1 - R_2$ −
3) $R_2 \leftarrow \overline{R_2}$ (1's comp)
4) $R_2 \leftarrow \overline{R_2} + 1$ (2's comp)
5) $R_3 \leftarrow R_1 + \overline{R_2} + 1$ (−)
6) $R_1 \leftarrow R_1 + 1$ ++
7) $R_1 \leftarrow R_1 - 1$ −−

4 bit binary adder

* Binary adder is useful inorder to add 2 binary no's whose size here is 4 bits.
* Let the 1st binary no be A_0, A_1, A_2, A_3 where A_3 is most sig bit & A_0 is least sig bit
* Likewise, 2nd Binary no be B_0, B_1, B_2, B_3.
* If we want to add 2 bits, we use half adder & 3-bit - full adder
* Binary adders is a collection of full adders which are connecting in cascading
* The o/p of 1 FA will be applied as I/p to other FA

* If the carry value is 1 then its -ve, & if the carry value is 0. then its +ve.
* Digital circuit that forms the arithmetic sum of 2 bits & the previous carry is called. FULL ADDER
* Digital circuit that generates the arithmetic sum of 2 binary no's of any length is called BINARY Adder

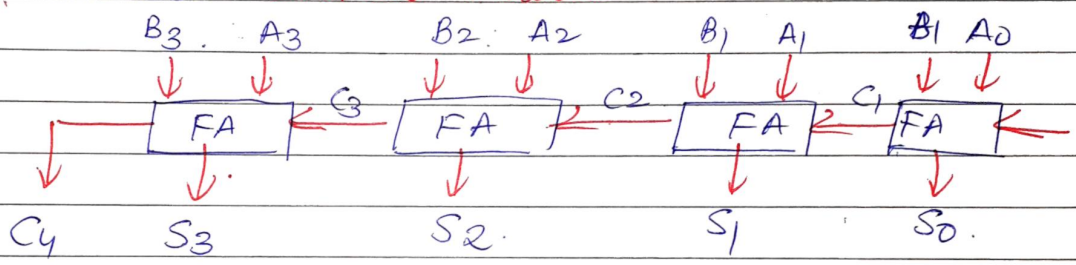
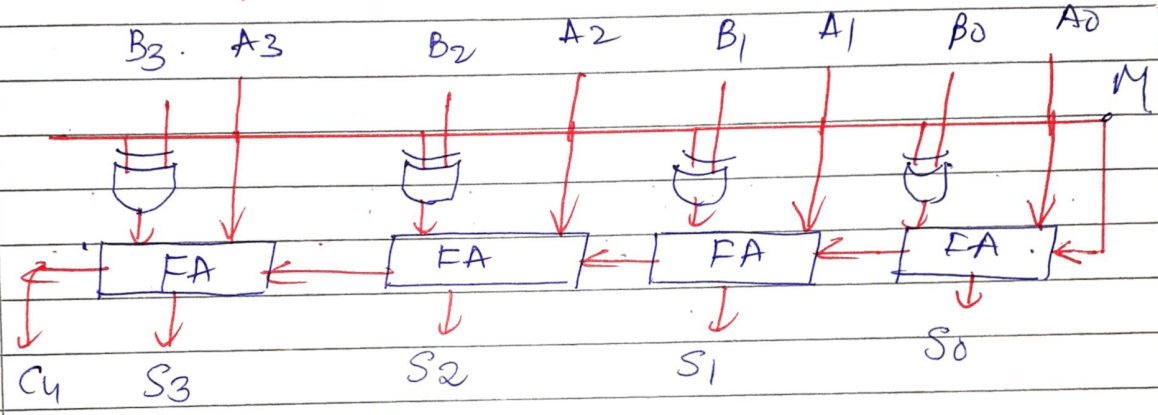


fig: Interconnections of 4 FA's to provide 4 bit BA.

* The carries are connected in a chain through the full-adders.
* The i/p carry to BA is C_0 & o/p carry is C_4.
* The S o/p's of FA generate the required amt of bits (sum).
* An 'n' bit adder of binary requires n-full adders

4 BIT BINARY ADDER — Subtractor

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

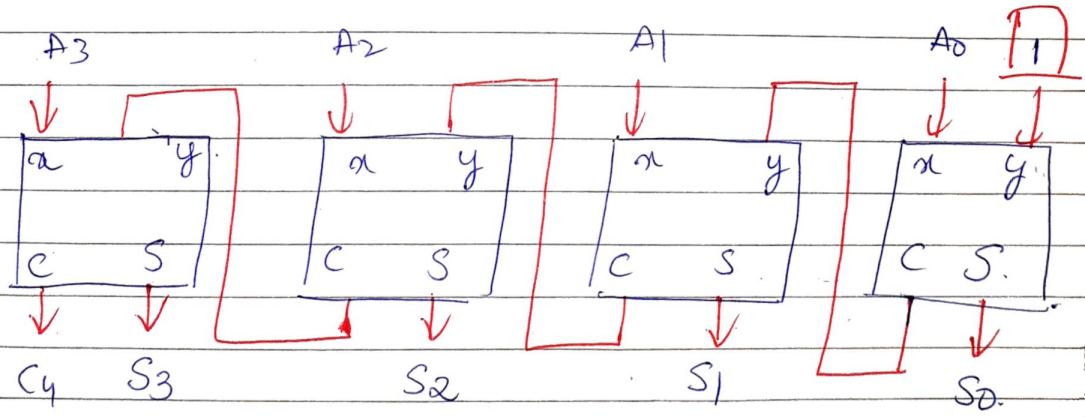


* Thus the FA accepts 3 inputs,
  1st — A & 2nd B.
* exclusive or requires 2 inputs
  1st — M & 2nd B
  M — Mode bit,
* if M value is 1 — circuit is subtractor
  0 — circuit is adder.
* if M value is 1 — carry is 1.
  0 — carry is 0
* if M value is 1 — we have B XOR 1
  $= B^1$ and $C_0 = 1$.
  0 — we have B XOR 0
  $= B$ and $C_0 = 0$.

* The B inputs are all complemented and a 1 is added through the Ip carry.
* The circuit performs the operations A plus the 2's comp of B
  $(A + \overline{B} + 1)$

4 BIT BINARY INCREMENTER

* It increments the contents of the register by 1.
* We have to take 4 half adders
* A_0, A_1 A_2 are the i/p to x while 1 is the i/p to y
* S_0 is the o/p
* Carry will be passed as i/p to the next HA
* In this way we can add 1 to the corresponding binary adder



* n bit binary increments — n HA
* The least significant bit must have 1 input connected to logic -1

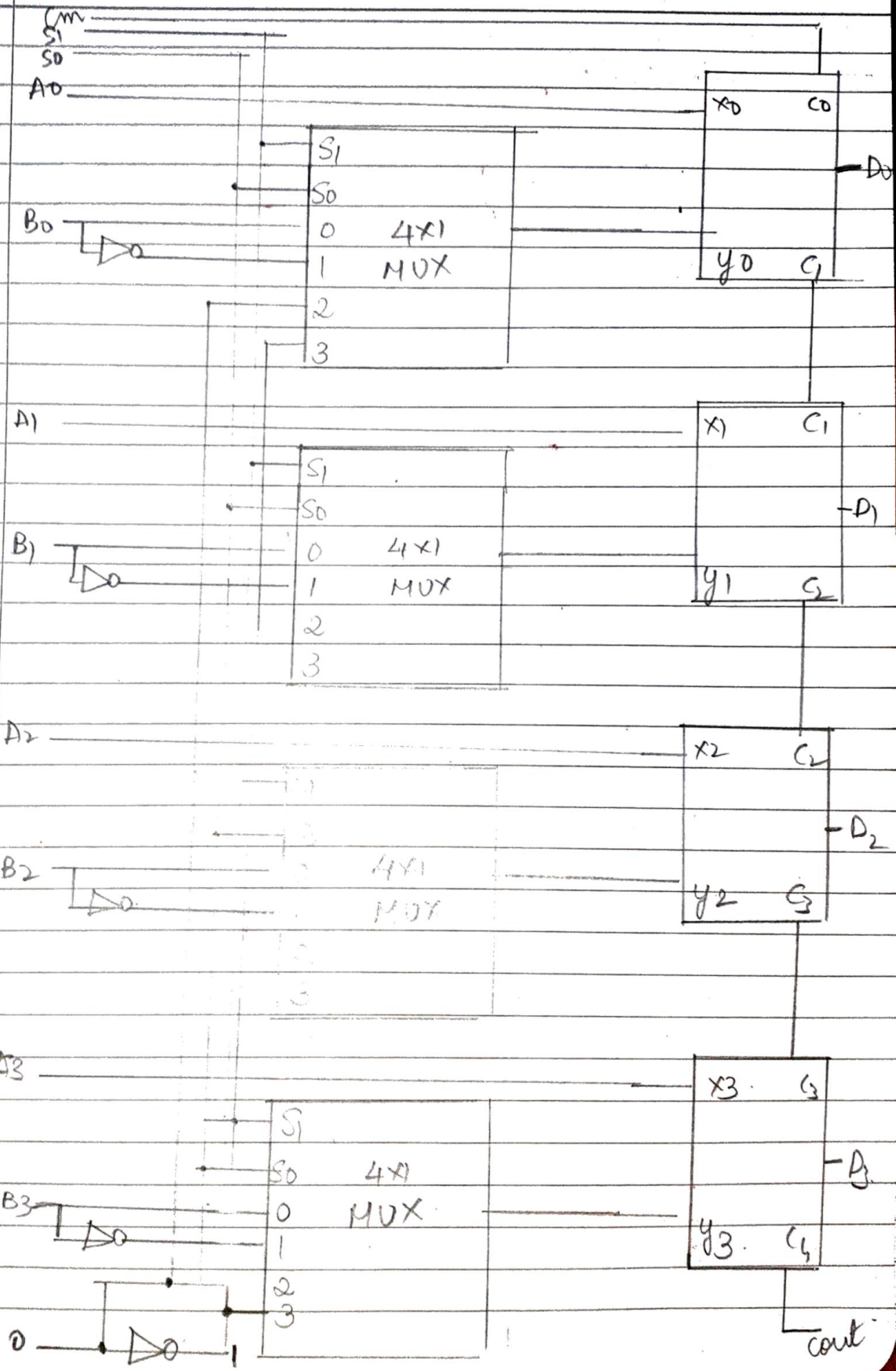
4 BIT ARITHMETIC CIRCUIT

* With the help of this we can perform all 7 AMO.
* 4 FA's are taken.
* FA advantage :- performs addition operation on 3 bits (x, y, c_0)
* The input to the 'x' comes from 1st binary no A_0; x_1 receives info from A_1, x_2
  " " " A_2, x_3.
  " " " A_3
* The 2nd i/p to the FA comes from MUX
* The o/p of MUX is applied as i/p to the y_0....
* The FA accepts carry as the i/p Cin. is the carry value which is passed to C_0. Now C_0 contains the corresponding carry value.
* So this FA performs addition operation on x_0, y_0, c_0, the resultant sum value is D_0 where the carry carry value is C_1
* Carry C_1 is propogated as i/p to the next FA
* The last carry contains Cout

* We have taken 4 multiplexers
* By changing selection i/p we can perform all 7 AMO
* The size of MUX is 4×1 where it accepts 4 inputs & produces 1 o/p 0,1,2,3
* ∴ We must have 2 selection i/p i.e S_1 & S_0
* The selection i/p's are applied to all the mux
* Each multiplexer '0' i/p receives data from Register B
* Each multiplexer of i/p '1' receives data from complement of B
* Each multiplexer of i/p '2' receives data from '0' i/p (logic gate)
* Each multiplexer of i/p '3' receives data from '1' i/p (∵ complement = 1)

(or)

S_1	S_0	Cin	I/p (y)	O/p (D=A+y+Cin)
0	0	0	B	D = A+B
0	0	1	$\overline{B}$	D = A+B+1
0	1	0	$\overline{B}$	D = A+$\overline{B}$
0	1	1	$\overline{B}$	D = A+$\overline{B}$+1
1	0	0	0	D = A
1	0	1	0	D = A+1
1	1	0	1	D = A−1
1	1	1	1	D = A


Cm
S1
S0
A0
B0
S1
S0
0
1
2
3
4X1
MUX
X0
C0
D0
y0
C1
A1
B1
X1
C1
D1
y1
C2
A2
B2
X2
C2
D2
y2
C3
A3
B3
X3
C3
D3
y3
C4
0
1
cout

Addition

* When $S_1S_0 = 00$ the val of B is applied to Y i/p of the adder
  - ✓ If $C_{in} = 0$, o/p $D = A + B$.
  - ✓ If $C_{in} = 1$, o/p $D = A + B + 1$.
* Both cases perform the add micro operation with or without adding the carry.

Subtraction

* When $S_1S_0 = 01$, the comp of B is applied to Y i/p of the adder.
  - ✓ If $C_{in} = 1$, then $D = A + \bar{B} + 1$. This produces A plus 2's comp of B, which is = $A - B$.
  - ✓ If $C_{in} = 0$, then $D = A + B$. This is = to a subtract with borrow, $A - B - 1$.

Increment

* When $S_1S_0 = 10$, the i/p's from B are neglected & 0's are inserted into Y i/p's
* Case 1: direct transfer from i/p A to o/p D.
* case 2: val of A is incremented by 1

Decrement

* When $S_1S_0 = 11$, 1's are inserted into Y i/p's.

* This is because 1a no with all 1's is equal to 2's comp.

LOGIC MICRO OPERATIONS.

Truth functions for 16 func of 2 variables

8421

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

$\bar{x}y + x\bar{y}$

* F_1 to F_{15} are the decimal no's.
* If there are 2 binary variables we can have 16 logical functions.

Boolean func	Micro operath	Meanig
$F_0 = 0$	$F \leftarrow 0$	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
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	E-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR

Boolean func	Micro operation	Mean
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Ex-NOR
$F_{10} = y'$	$F \leftarrow \bar{B}$	Comp B
$F_{11} = x + y'$	$F \leftarrow A \vee \bar{B}$	—
$F_{12} = x'$	$F \leftarrow \bar{A}$	Comp 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 all 1s.

and $\wedge$, or $\vee$.

Hardware Implementation of Logical & Micro-Operations.

* Let us take 4×1 MUX, 2 selection lines.
* Then we apply 2 bits Ai & Bi

A	B	MO
0	0	$E \leftarrow A \wedge B$
0	1	$E \leftarrow A \vee B$
1	0	$E \leftarrow A \oplus B$
1	1	$E \leftarrow \overline{A}$

* The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the req logic func.
* Although there are 16 logic MO, most PC's use only 4 -- AND, OR, XOR, Complement from which all others can be derived.

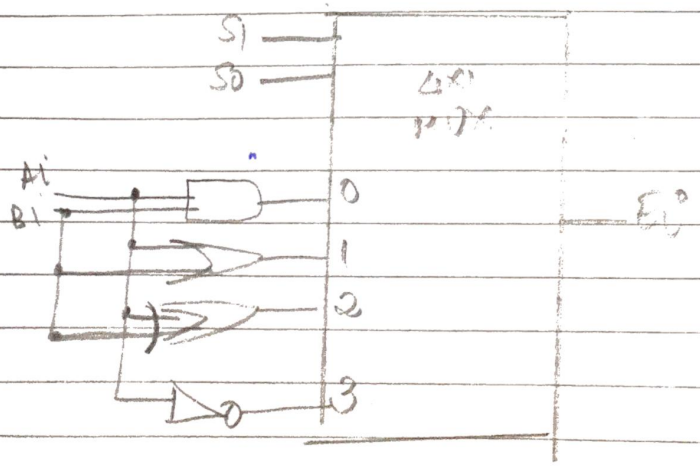


fig: One stage circuit that generates 4 basic logic MO.

* It consists of 4 gates & a multiplexer
* Each of 4 logic operations is generated through a gate that performs required logic.
* The outputs of the gates are applied to the data inputs of the multiplexer.
* The 2 selection i/p S1 & S0 choose one of the data i/p's of MUX & direct its value to o/p

Applications of Logic MO.

1) Selective Set
2) Selective Complement
3) Selective Clear
4) mask
5) Insert
6) Clear

1) Selective set

1010 & 12.	1010	A before
	1100	B (logic operand)
	1110	A after

Set the register A

* If the bit in B register is 1, then the same bit in A will become 1
* If the bit in B register is 0, then there is no need to change
* 'OR' microoperation is used to selective set

2) Selective Complement

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

* If the Bit in register B is '0' then no need to change register A bit
* If the bit in register B is '1' then the bit in A will be complemented
* Exclusive-OR microoperation is used in order to complement the selective bits in the register

3) Selective Clear

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

* B = 0, no change
* B bit = 1, clear the corresponding A bit to '0'
* $A \cap \overline{B}$ mO is used in order to clear the selective bits in register

4). Mask.

1010	A (before)
1100	B (logic operand)
1000	A (after)

B = 0, mask A with 0.
B = 1, no change
0 priority >>>>
AND. MO.

5). Insert. (Mask + OR)
AND & OR MO.

1010 1100
0011 1100
(change)

1010	1100	(mask.
0000	1111	
0000	1100	
0011	0000	(OR)
0011	1100	

6) Clear.

1010
1100
0110

Bits same = 0.
Bits diff = 1
Ex-OR.

Shift Micro Operations

* SMO are useful inorder to shift the contents either from left to right or right to left
* During shift left, the serial i/p transfer the bit towards right side & vice versa
* Serial i/p gives a value 0 or 1.
* There are 3 types of SMO
  a) logical Shift
  b) Circular Shift (or) Rotate Shift
  c) Arithmetic Shift

1) logical Shift. (shl & shr)

logical Shift left.	logical Shift Right
$R_1 \leftarrow shl\ R_1$	$R_1 \leftarrow shr\ R_1$
Left most bit will be discarded	Right most bit will be discarded

2) Circula Shift (CIL or CIR).

$R_1 \leftarrow cil\ R_1$	$R_1 \leftarrow cir\ R_1$.
Transfers left Most bit to right side	Transfer Right most bit to

3) Arithmetic Shift (SHAL SHAR)

$R_1 \leftarrow$ ashl R_1	$R_1 \leftarrow$ ashr R_2
If left most bits are same no overflow	We should not change the left most bit
mul by 2	division by 2

Shift Micro operations

* SMO are used for serial transfer of data.
* The contents of the register can be shifted to the left or the right
* During a shift-left operation the serial input transfers a bit into right-most position
* Shift-right operation ——— left most position

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

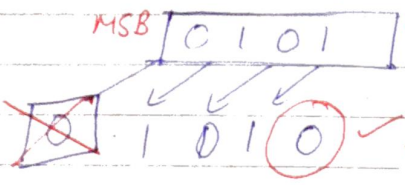
Shift micro-operations

- Logical
  - shl
  - shr
- Circular
  - cil
  - cir
- Arithmetic
  - ashl
  - ashr

Logical Shift

* A logical shift is one which transfers '0' through serial i/p.
* The symbols shl & shr for logical shift-left & shift-right micro-operations
* The MC that specify ca
* The bit transferred to the end position through the serial i/p is assumed 0 during a logical shift

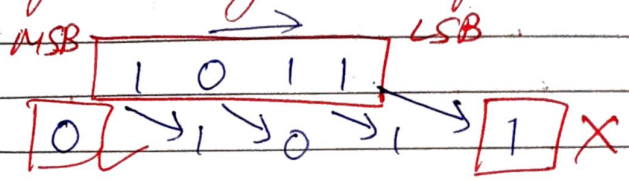
Ex: logical left shift



∴ 0101 after logical left shift gives 1010

LSB is filled with 0

Ex of logical right shift.



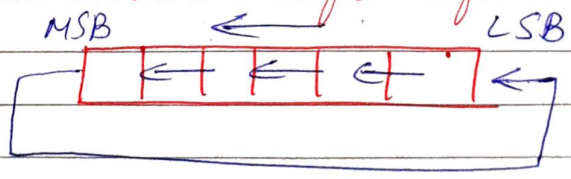
∴ 1011 → 0101.

MSB is filled with zero.

Circular Shift

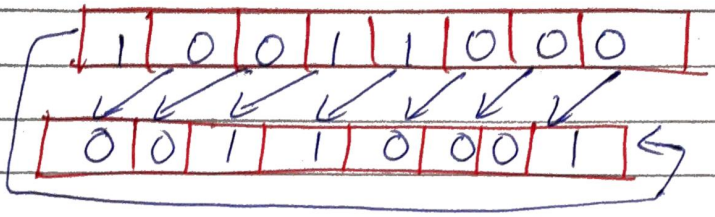
* The circular shift (also knows as rotate operation) circulates the bits of register around the two ends without loss of info
* This is accomplished by connecting the serial o/p of shift register to its serial i/p.
* We will use the symbols cil & cir for circular shift left & right respectively

Ex of circular shift left.

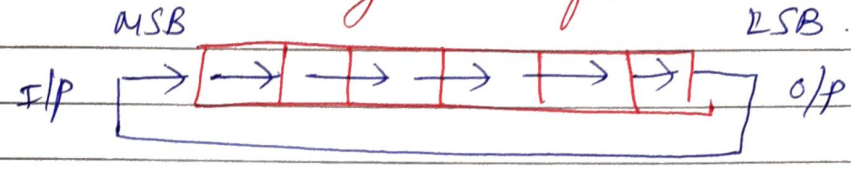


Instead of LSB filled with 0, we r fill LSB with MSB.

Ex.

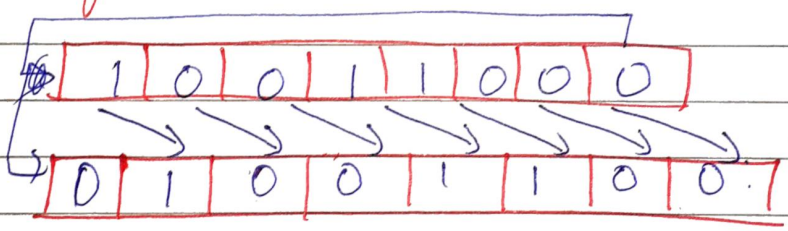


Circular right shift



Instead of filling MSB with 0, we fill MSB with LSB.

Ex of Cir.

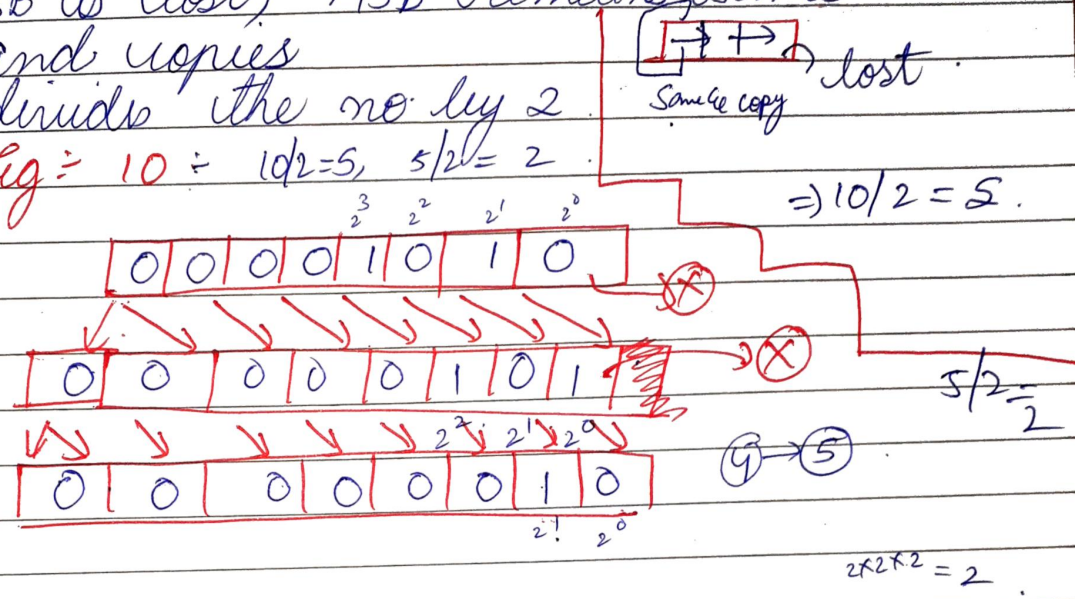


Arithmetic Shift

* An arithmetic shift is a micro-operation that shifts a signed binary no. to left or right.
* ashl multiplies a signed binary no. by 2.
* ashr divides the no. by 2.
* Arithmetic shift must leave the sign bit unchanged because the sign of no. remains the same when div by 2.

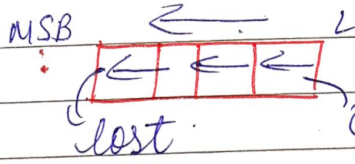
Arithmetic right shift

* bits move towards the RHS the LSB is lost, MSB remains same and copies
* divides the no by 2
* Eg :- 10 ÷ 10/2=5, 5/2 = 2



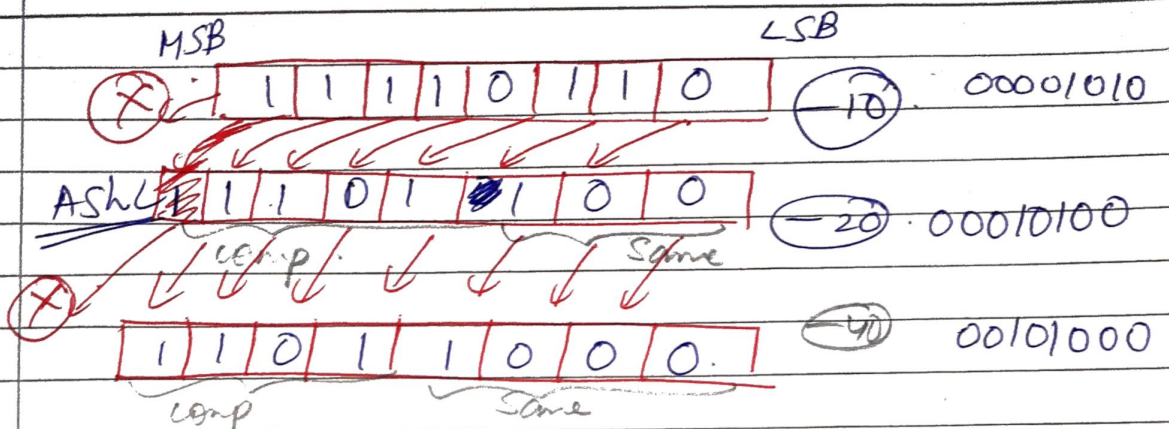
Arithmetic left Shift

* multiplies by 2
* bits move towards LHS, the MSB is lost, LSB is filled with 0



* Eg. When we see 1 in in MSB it is -ve no.
  So from LSB only 1st one should be there and all should be complimented

$2^3\ 2^2\ 2^1\ 2^0$ calculate 1's.



Overflow condition is there if MSB is 0 and can be solved by XOR

Shifting Operations — HARDWARE IMPLEMENTATION

* A combinational unit shifter can be constructed with multiplexers
* The 4 bit shifter has 4 data inputs A_0, A_1, A_2, A_3
* It also has 4 data outputs H_0, H_1, H_2, H_3
* There are 2 serial i/p, 1 for shift left & other for shift right
* When the selection i/p $S=0$, then i/p data are shifted right
* When $S=1$, the i/p data is shifted left
* Functional Table shows which i/p goes to each o/p after the shift
* Shifter with 'n' data i/p's & o/p's require 'n' multiplexers

Ex. $A_3\ A_2\ A_1\ A_0$

$A = 1\ 0\ 1\ 0$

$Shl = A_2\ A_1\ A_0\ I_L$

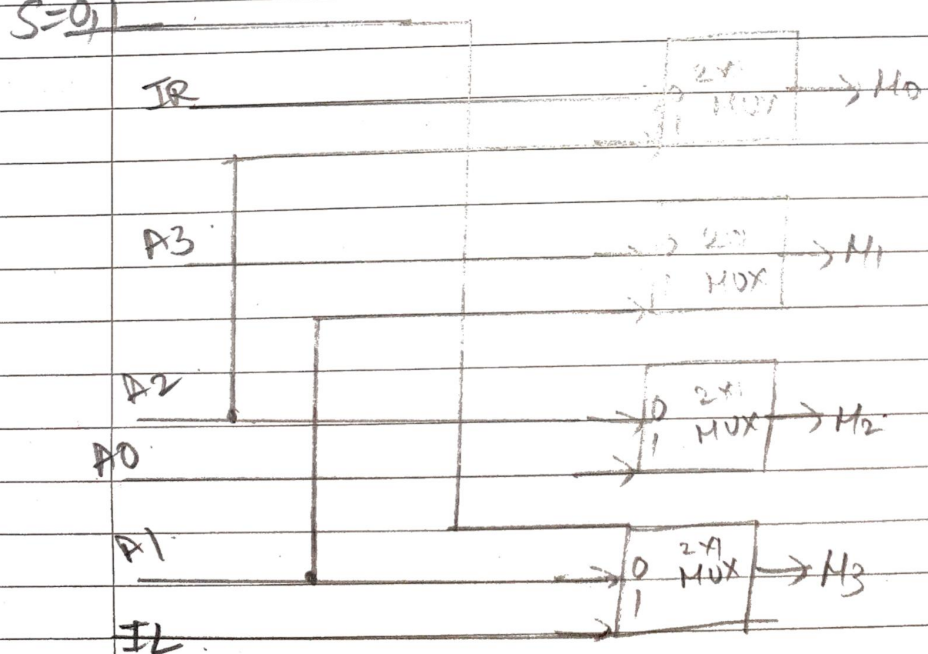
$0\ 1\ 0\ 0$

$Shr = I_R\ A_3\ A_2\ A_1$

$0\ 1\ 0\ 1$

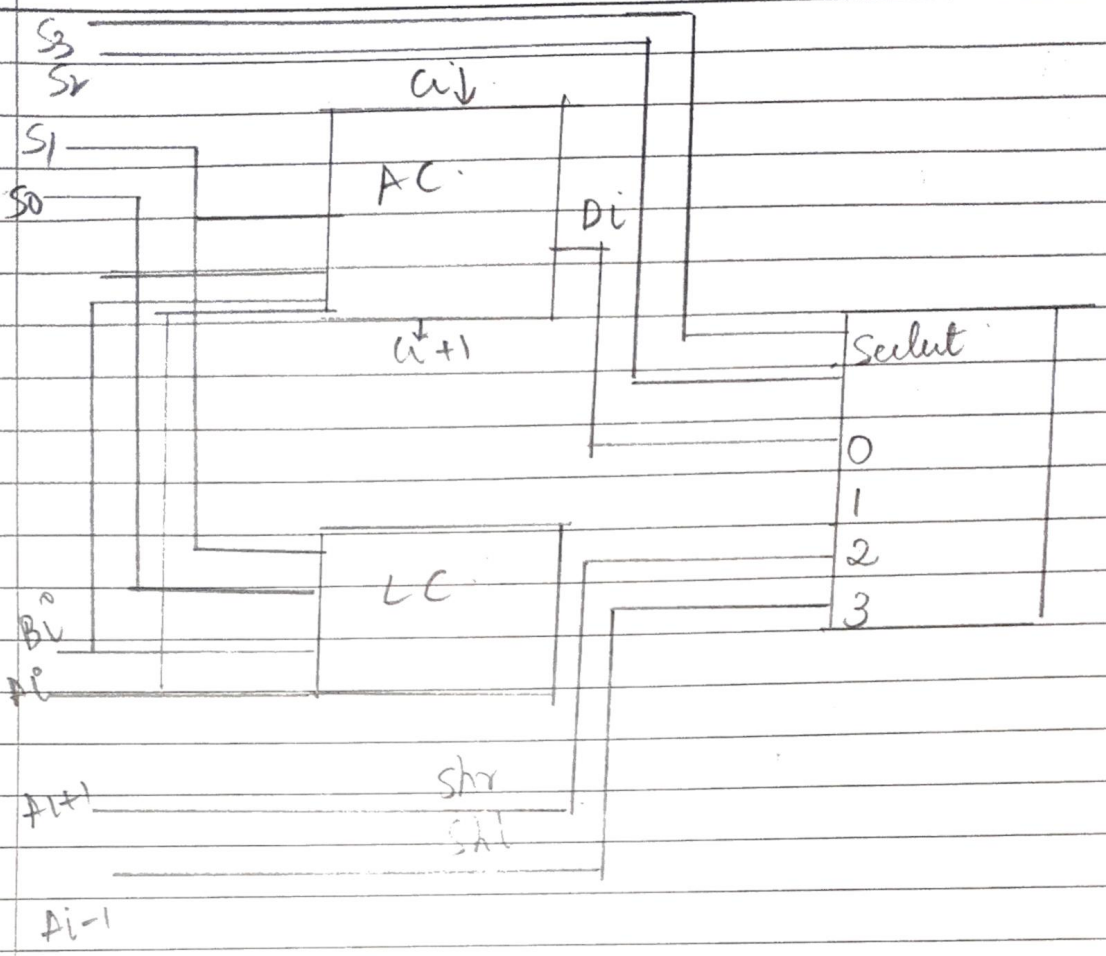
Function Table

Select	o/p			
S.	M_0	M_1	M_2	M_3
Shr → 0	I_R	A_3	A_2	A_1
Shl → 1	A_2	A_1	A_0	I_L



Arithmetic logic Shift Unit

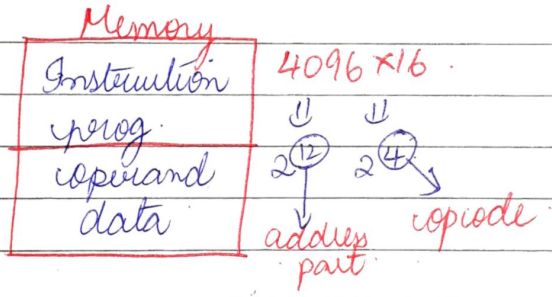
* Instead of having individual registers performing the micro-operations directly, computer systems employ a no of storage registers connected to common operational unit — ALU.
* ALU is a combinational circuit. So that the entire register transfer operation from the source register through ALU and into the destination Register, can be performed during one clock pulse period.
* Particular MO is selected with inputs S_1 & S_0, A 4x1 MUX at the output
* The circuit has one stage of Arithmetic circuit, logic circuit, 4x1 MUX
* In the circuit, at a time only 1 unit can activate
* A_i & B_i are given as i/p either for Arithmetic circuit (or) logic circuit
* AC also uses C_i carry as i/p. But for LC there is no carry
* S_1 & S_0 are the i/p given to both LC & AC
* S_2 & S_3 are connected to the MUX
* The MUX is going to SELECT which circuit is in operation.
* i/p for shr is A_{i-1} & shl is A_{i+1}

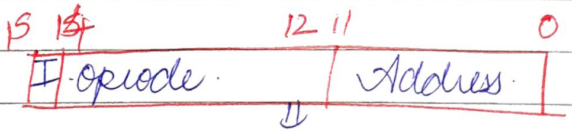


S_3	S_2	S_1	S_0	Cin	Opern	Func.
0	0	0	1	0	F=A	Transfer A
0	0	0	1	1	F=A+1	Increment A
0	0	0		0	F=A+B	Addition
0	0	0	1	1	F=A+B+1	Add with Carry
0	0	1	0	0	F=A+B'	Sub with borr
0	0	1	0	1	F=A+B'+1	Subtraction
0	0	1	1	0	F=A-1	decrement A
0	0	1	1	1	F=A	transfer A
0	1	0	0	X	F=A∧B	AND
0	1	0	1	X	F=A∨B	OR
0	1	1	0	X	F=A⊕B	XOR
0	1	1	1	X	F=A'	Comp A
1	0	X	X	X	F=shrA	shr with F
1	1	X	X		F=shlA	shr with F

Instruction Codes

* Instruction codes is a group of bits (0101...) that tells to the processor to perform some specific task
* Every PC has its own Instruction code format
* The IC can be org into 2 parts
  a) op code b) address
* In the memory, there are IC
* The size of the memory for 16 bit Insn is 4096 × 16.





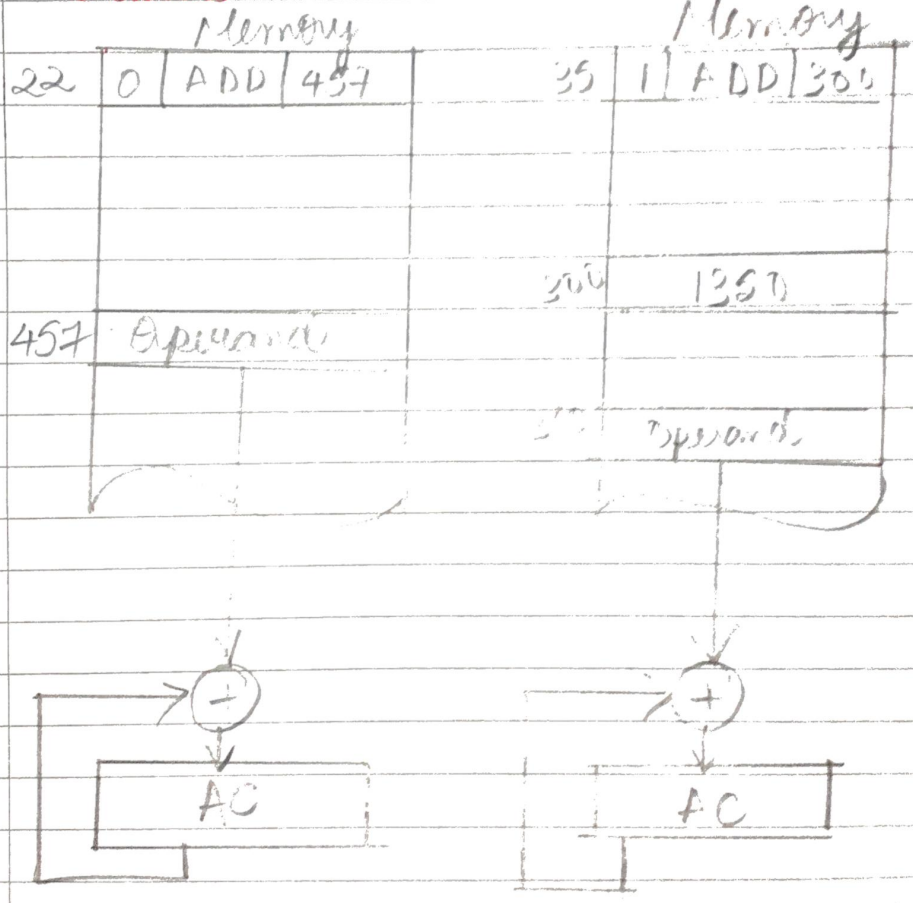
Instruction format

3 Types of Instructions

a) Memory Reference Instn.
b) Regstr Reference Inst n.
c) I/O Reference Inst n.

* The opcode & address would be of
  a) Immediate addrs
  b) Direct address
  c) Indirect address

Demonstration of direct & Indirect Address



b) Direct addrs

Address field contains operand

__/__/____

b) direct address

* 22 is address of MM.
* We k.t, inst is divided into 3 parts. Indirect bit - 0
  opcode - ADD
  address - 457.
* If the indirect bit is 0, then the address which specifies the operand is known as effective address. (The address field itself is)
* So the control goes to 457 Memory location.
* In 457 memory location, operand will be available
* Let the operand be 10 & think we are performing addition operation
* The (+) is the adder circuit we are using
* The adder perform addition operation on the contents of the accumulator as well as the operand
* After performing the operation, the result will be transfered to the accumulator
* data of operator = 10,
  data of accumulator = 20,
  adder adds 10+20 = 30.
* ∴ 30 will be transfered to the accumulator

c) Indirect address

* Let us assume Inst add is at 35th addr
* Address field contains adress of the operand.

* Mode bit - 1
* Opcode - ADD
* Address = 300

* If i=1, its called indirect address.
* address field contains effective address
* If have to go to 300 address in order to know address of the operand.
* 1350 becomes effective address
* First the control goes to 300, then we see that 1350, then the control again goes to 1350,
* The 1350 now contains the operand.
* The adder perform addition operation on the acc & operand & transfer the result into the acc

Computer Registers

* Registers are mainly useful for storing the data.
* If the data resides in register, CPU can access the data in a faster manner. (Registers are faster compared to MM)
* 6 registers

a) AR.
* Address Register
* AR stores address of the operand.

b) PC
* Program Counter
* PC always contains address of the

next instruction to be executed

c) DR
* Data Register
* DR stores operand.

d) AC
* Accumulator
* General purpose register

e) IR.
* Instruction register
* It contains the inst to be loaded from the main memory
* IR stores inst^n.

f) TR
* Temporary register.
* To perform temp calculations

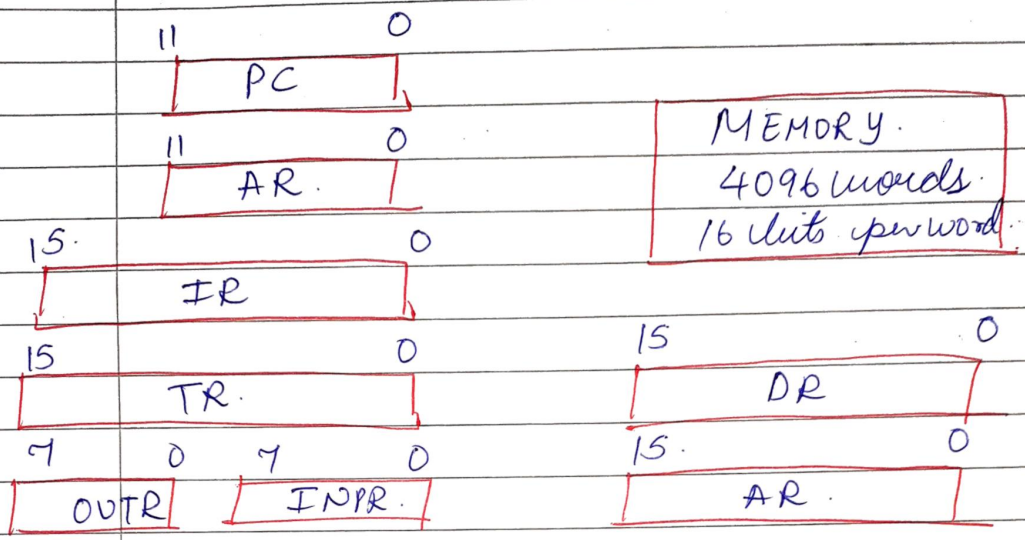
g) INPR
* Input register
* contains i/p data which is received from i/p device.

h) OUTR
* Output register
* contains o/p data which is received from o/p device

*a) The AR & PC mainly stores the address
b) Out of 16 bits, 12 bits specify addrs
c) ∴ size of AR & PC is 12 bits.

*a) DR, AC, IR, TR stores inst^n.
b) The size of these 4 are 16 bits

* INPR & OUTR.
  a) Size – 8 bits.



* Common Bus System.

① * By using CBS we can transfer the data from register to another register, R to R, R to M, M to R in a less expensive manner.
② * S_2, S_1, S_0 are selection lines i/p which are to be applied on a bus.
③ * Of the are 3 i/p total $2^3 = 8$ i/p can be used.
④ * The size of memory unit is 4096×16
⑤ * The memory places the info on bus 7. when S_2, S_1, S_0 are 111
⑥ * We can perform 2 operations on Memory unit = write & read.

(7) * Read means to transfer mem to reg, So place the content of the memory on the bus.

(8) * Whenever the write (or) Input is enabled then the bus info will be placed on the mem. as Write means transfering the data from register to memory

(9) * AR specifies address of the operand which is present in MM.

(10) AR, PC, DR, AC, TR * ~~AR~~ has 3 control i/p.
→ LD – load
INR – Increment.
CLR – clear

(14) * AR places the contents on the bus '1' when $S_2 S_1 S_0$ are 001

(15) * PC places the contents on the bus '2' when $S_2 S_1 S_0$ are 010

*- PC ~~also contains~~ 3 i/ps. LD, INR, CLR.

(11) * When load control ~~i~~p is enabled then the particular register receives info from the bus

(12) * If INR control ~~i~~p is enabled, then its content will be incremented by 1.

(13) * If CLR control i/p is enabled then all the content will be 0.

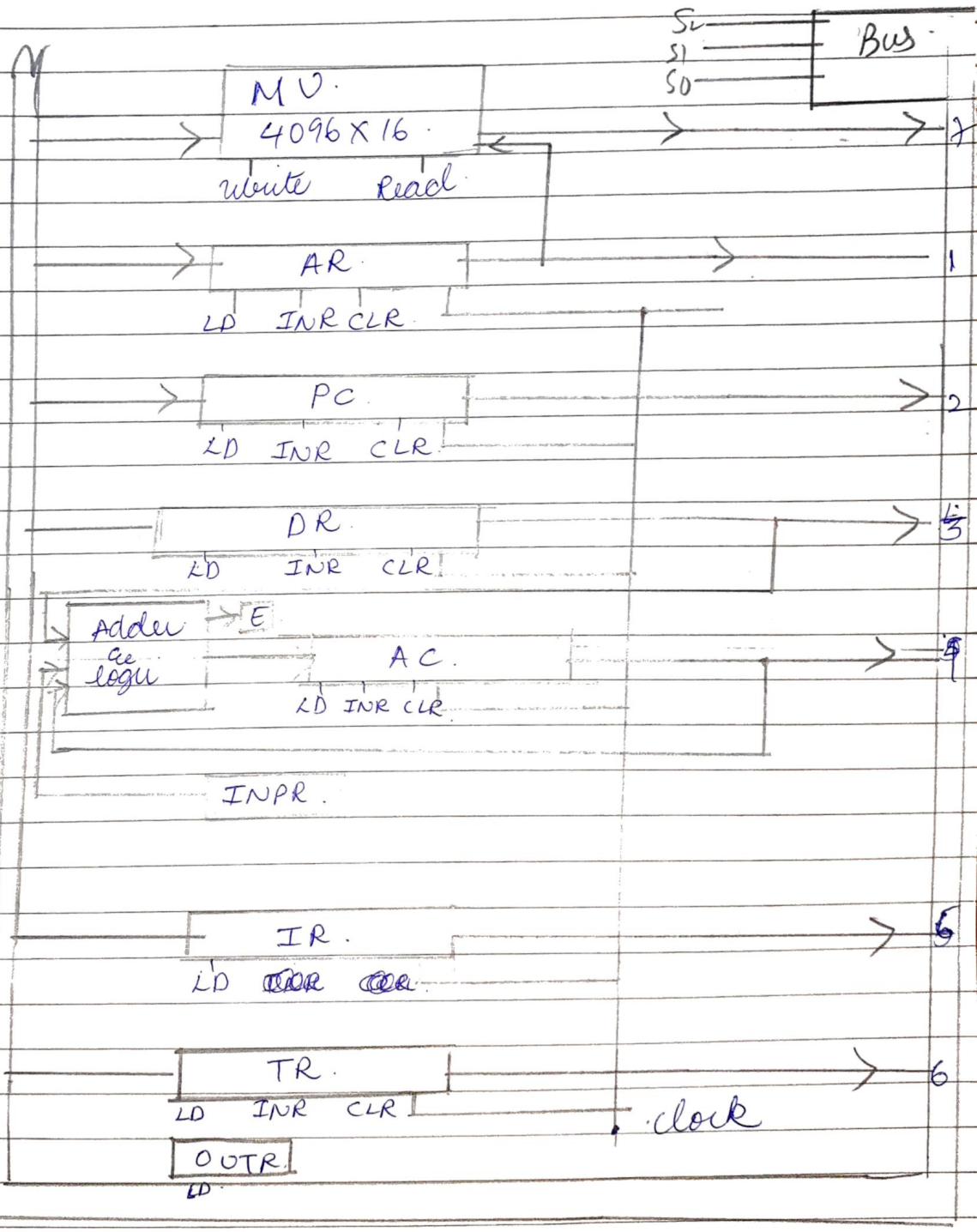
(16) * IR & OTR has only 1 control i/p i.e LD.

(17) * ~~INPR~~ doesn't provide any info to the bus are INPR & OUTR

(18) * Adder & logic gets data from DR & AC

19) * E block stores carry.
3 — 011, DR.

20) * Clock pulse is applied on all Reg

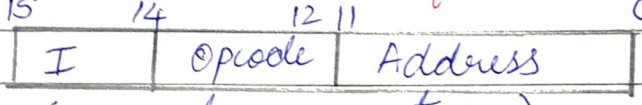


✱. Computer Instructions

1) memory-reference Inst.
2) Register-Reference Inst.
3) I/O-Instr

1) MRI.

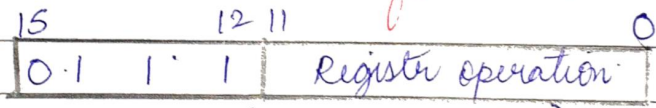
* MRI is useful to perform ~~assist~~ operations on operands located in MM.
* M-R instruction format



(opcode: 000 to 110).

2). RRI

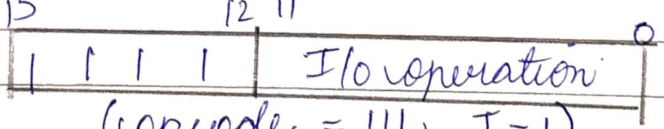
* RRI is useful to perform operations on the registers (no operands)
* R-R Instn format



(opcode: 111, I=0).

3) I/O I.

* IOI is useful to perform operations on I/O registers



(opcode = 111, I=1)

1) MRI.

Inst	I=0	I=1
~~AAAA~~		
AND	0XXX	8XXX
ADD	1XXX	9XXX
LDA	2XXX	AXXX
STA	3XXX	BXXX
BON	4XXX	CXXX
BSA	5XXX	DXXX
INC	6XXX	EXXX

2) RRI.

Inst	
CLA	7800
CMA	7400
CLE	7200
CME	7100
CIR	7080
CIL	7040
INE	7020
SPA	7010
SNA	7008
SZA	7004
SZE	7002
HLT	7001

3) IOI.

INP	F800
OUT	F400
SKI	F200
SKO	F100
ION	F080
IOF	F040

__/__/____

TIMING AND CONTROL / HARDWIRED CU

* CU is present in the CPU.
* Mainly useful to generate Timing signals & control signals
* CU mainly co-ordinates b/w CPU, MM, I/O devices
* We can design CU in 2 ways
  a) hardwired CU
  b) micro programmed CU.

* a) HCU
  * The CU will be designed with the help of hardware components like logic gates, flip flops, decoder, registers
  * It is extremely fast — ADV
  * modifications are tough — DIS.

* An Instruction will be stored in IR in MM.
* The size of IR = 16 bits; 1st 12 bits specifies address (0-11); next 3 bits denotes opcode (12-14); next bit specifies indirect bit (15)
* The size of opcode is 12, 13, 14 i.e 3 bits, so we can perform 2^3 operations = (8).
* In order to perform those operations we have to decode those 3 bits with the help of decoder.

* The 3 bits of the opcode will be passed as i/p to decoder [3×8] 2^n
* Decoder takes 3 i/ps produces 8 o/p's
* All these o/p's should be passed to control logic gates
* The 12 bits in the address will also be passed as i/p to CLG
* Indirect bit will be transfered to flip flop E and that E bit will be transfered to CLG.
* Then we have a sequence counter whose size is 4 bits.
* So, we have 24 combinations i.e 16 timing signals
* The 4 bit seq counter will be decoded with the help of decoder.
* The decoder accepts 4 i/p from the sequence counter, all all the 16 o/p will be passed as i/p to CLG.
* Now, the first signal is T_0, so when AND operation is performed then Timing signal T_0 will be active
* The sequence counter accepts 3 i/p. 1) clock pulse CLK, 2) Increment INC 3) Clear CLR.
* So if we want to perform any operation on the seq counter, then we have to enable both CLK & INC then only we get next timing signals

* Whenever CLR control i/p is enabled the prgm will start from the first timing signal T_0
* D_0, D_1, D_2 are decoders outputs, T_0, T_1, T_2 are timing signals
* Every operation is performed by synchronizing D_0 with T_0
  - $D_0 T_0$ – AND
  - $D_1 T_1$ – ADD

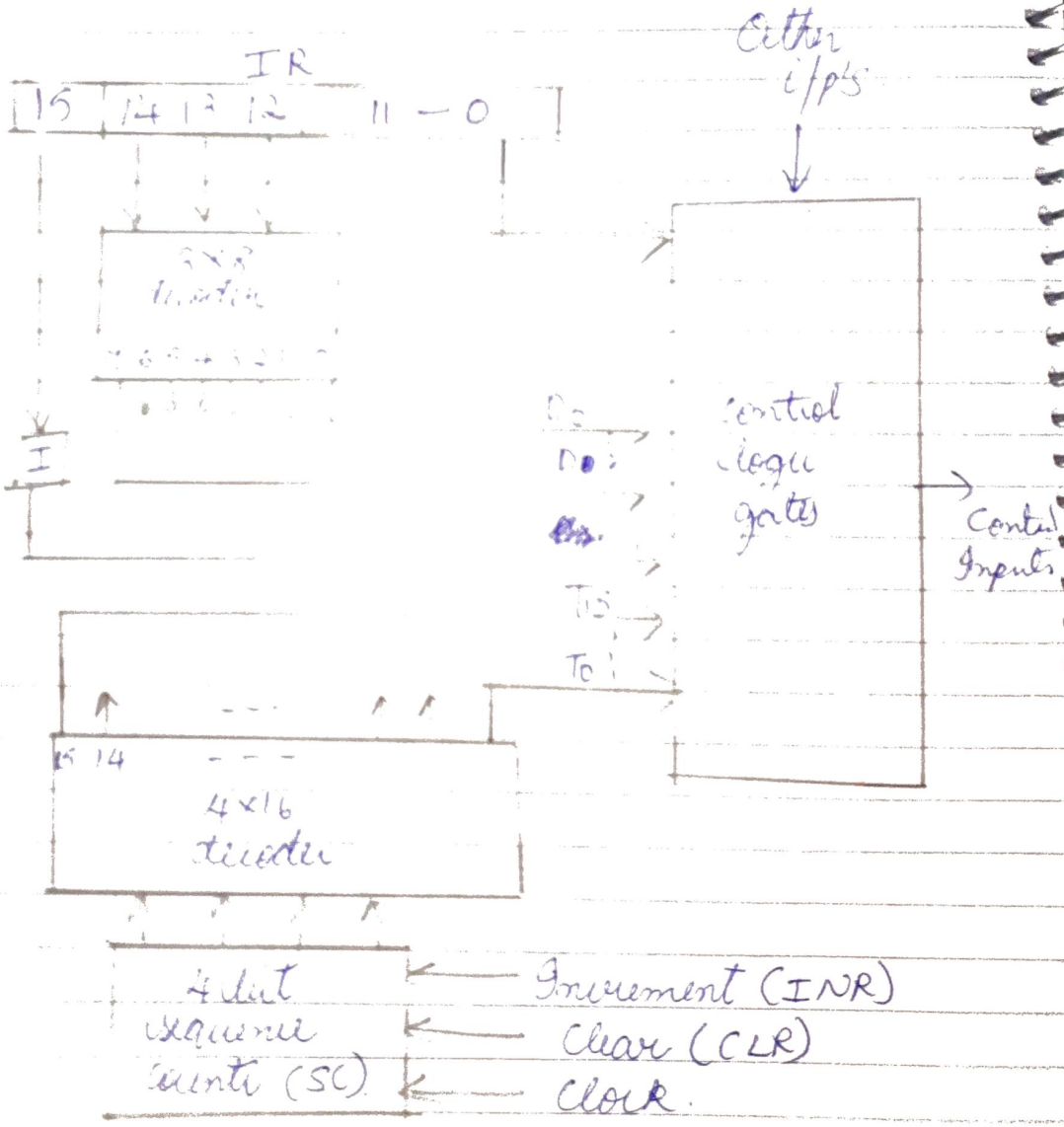
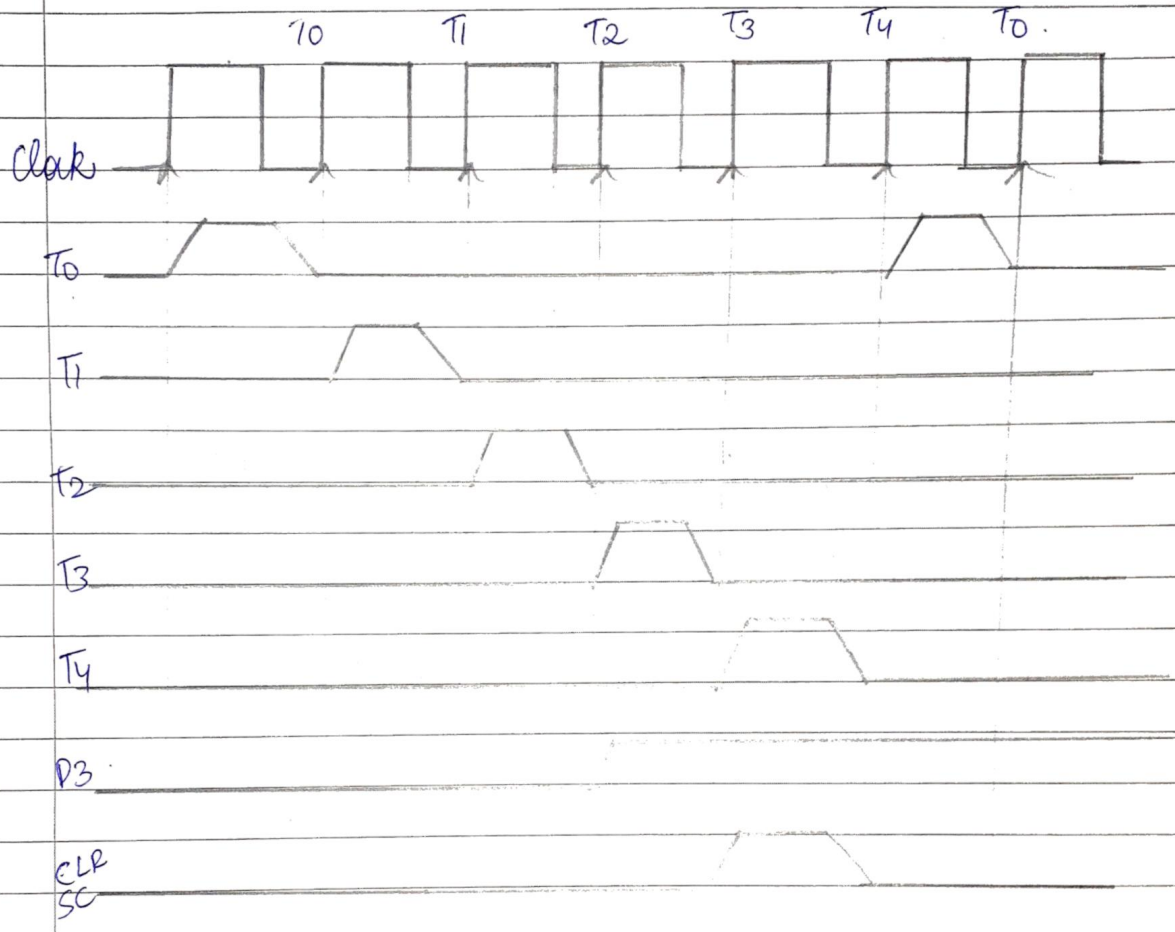


fig:- CU of Basic computer

Control Timing Signals

T_0, T_1, T_2, T_3, T_4 | $SC \leftarrow 0$ T_4. | $D_3T_4 : SC \leftarrow 0$

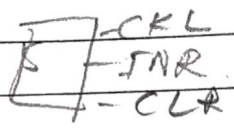
$SC \uparrow$. | P_3 is active | ~~D_3T_4~~



* The sequence counter SC corresponds to the +ve transition of clock.
* Initially CLR i/p of SC is active
* The 1st positive transition of clock clears SC to 0, which in turn activates To out of decoder
* SC is ↑ with every of +ve clock

transition, unless CLR i/p is active.

* This produces sequence of timing signals T_0, T_1, T_2, T_3, T_4.
* The last 3 waveforms show how SC is cleared when $D_3 T_4 = 1$.
* O/p D_3 from the operation decoder becomes active at the end of timing signal T_2.
* When TS T_4 becomes active, the o/p of AND gate that implements control func $D_3 T_4$ becomes active.
* The signal is applied to CLR i/p of SC
* On the next +ve clock transition, The counter is cleared to 0.
* This causes the TS T_0 to become active instead of TS that would have been active if SC were incremented instead of cleared.



Memory Reference Inst.

1) AND - ~~D0~~ T4
D0 T4 : DR ← m[AR].
D0 T5 : AC ← AC ∧ DR, SC ← 0.

2) ADD
D1 T4 : DR ← m[AR].
D1 T5 : AC ← AC + DR, E ← Cout, SC ← 0

3) LDA (load mem word to ac).
D2 T4 : DR ← m[AR].
D2 T5 : AC ← DR, SC ← 0.

4) STA (store ac into mem).
D3 T4 : m[AR] ← AC, SC ← 0.

5) BUN (branch unconditionally)
D4 T4 : PC ← AR, SC ← 0

6) BSA (branch & save address)
D5 T4 : m[AR] ← PC, AR ← AR+1
D5 T5 : PC ← AR, SC ← 0.

7) ISZ (Increment & skip zero)
D6 T4 : DR ← m[AR].
D6 T5 : DR ← DR+1.
D6 T6 : m[AR] ← DR, if (DR = 0)
then PC ← PC+1,
SC ← 0.

Register Reference Inst.

1) CLA (clear accumulator)
CLA : rB_{11} : AC ← 0.

2) CLE (clear extended flip flop)
CLE : rB_{10} : E ← 0.

3) CMA (complement accumulator)
CMA : rB_9 : $\overline{AC}$ ← AC.

4) CME (complement E).
CME : rB_8 : $E \leftarrow \overline{E}$

5) CIR (circulate right)
CIR : rB_7 : AC ← shr AC, E ← AC(0).
AC(15) ← E

6) CIL (circular left)
CIL : rB_6 AC ← shl AC, E ← AC(15),
AC(0) ← E

7) INC (Inc Acc)
INC : rB_5 AC ← AC+1

8) SPA (skip next Inst if ac is +ve)
SPA : rB_4 if (AC(15)=0) then
PC ← PC+1

9) SNA (skip next Inst if ac is -ve)
SNA : rB_3 if (AC(15)=1) then
PC ← PC+1

10) SZA (Skip next Inst if ac is zero)
SZA : rB_2 : if (AC=0) then PC ← PC+1

11) SZE (Skip next Inst if E is 0)
SZE : rB_1 : if (E=0) then PC ← PC+1

12) HLT (Halt)
HLT : rB_0 : S ← 0,
↳ start/stop flip flop.

Input/Output Instrn

1) INP: Input reg
INP : AC(0-7) ← INPR, FGI ← 0.

2) OUT: Output reg
OUT : OUTR ← AC(0-7), FGI ← 1.

3) SKI : Skip on Input Flag
SKI : if (FGI=1) then PC ← PC+1

4) SKO: Skip on Output Flag.
SKO : if (FGO=1) then PC ← PC+1

5) ION: Interupt Enable On
ION: IEN ← 1

6) IOFF : Interupt enable off
IOFF : IEN ← 0

Instruction Cycle.

1) Fetch 2) Decode 3) Read effective address
4) execute effective address

1) Fetch :

* PC contains address of next Instruction to be executed.
* The address will be loaded to AR. during timing signal T_0
* If we apply one more clock pulse to seq counter we get T_1
* During T_1, we have to read instruction from the memory and the instr to be loaded into IR
* Then we have to inc the PC so that even the next Inst will be executed

T_0 : AR ← PC.
T_1 : IR ← M[AR], PC ← PC+1

2) Decoder.

* During decoding phase, the 1st 12 bits from the IR (0-11) will be transfered to AR at T_5 T_2.
* The 3 opcode bits are decoded with the help of decoder then we get 2^3 operations (D_0 to D_7)
* The 15 bit of IR is transfrd to I

where I stands for Indirect bit
if I=0, its unDirect address.
I=1, its direct address.

T_2 : AR ← IR(0-11),
D7D6 ··· D_t ← IR(12-14),
I ← IR(15)

Register Transfer statements for fetch phase.

T_0: AR ← PC.
T_1: IR ← m[AR], PC ← PC+1.

Interupt cycle

* Interupt is a special signal which needs to be executed immediatily
* Whenever CPU receives interupt signal, the CPU stops execution of the currently running program.
* The CPU now executes interupt related program.
* If R = 1 then we have interupt signal, R = 0 no interupt signal then CPU executes instruction cycle (fetching, decoding, executing)
* IEN is a flip flop, If IEN = 1, then there is a interupt signal, IEN = 0 then no Interupt signal.
* FGI value is 1 then there is i/p ~~flag~~ interupt
  FGI value is 0 then there is ~~o/p flag~~ no i/p interupt
  ↳ assigns 1.
* If FGO value is 0 then there is o/p interupt so it assigns R ← 1.
  If FGO value is 0 then there is no o/p interupt
* After all this, the control goes to R.
* If R = 1, it means interupt signal.
* We store return address in location 0.
  M[0] ← PC
* Program counter location will be shifted to 1, branch to location 1.
* We have successfully executed

interrupt related program
* Then IEN will be cleared to 0.
  R will also be cleared to 0
  as there is no interrupt
* Now again the control goes to R

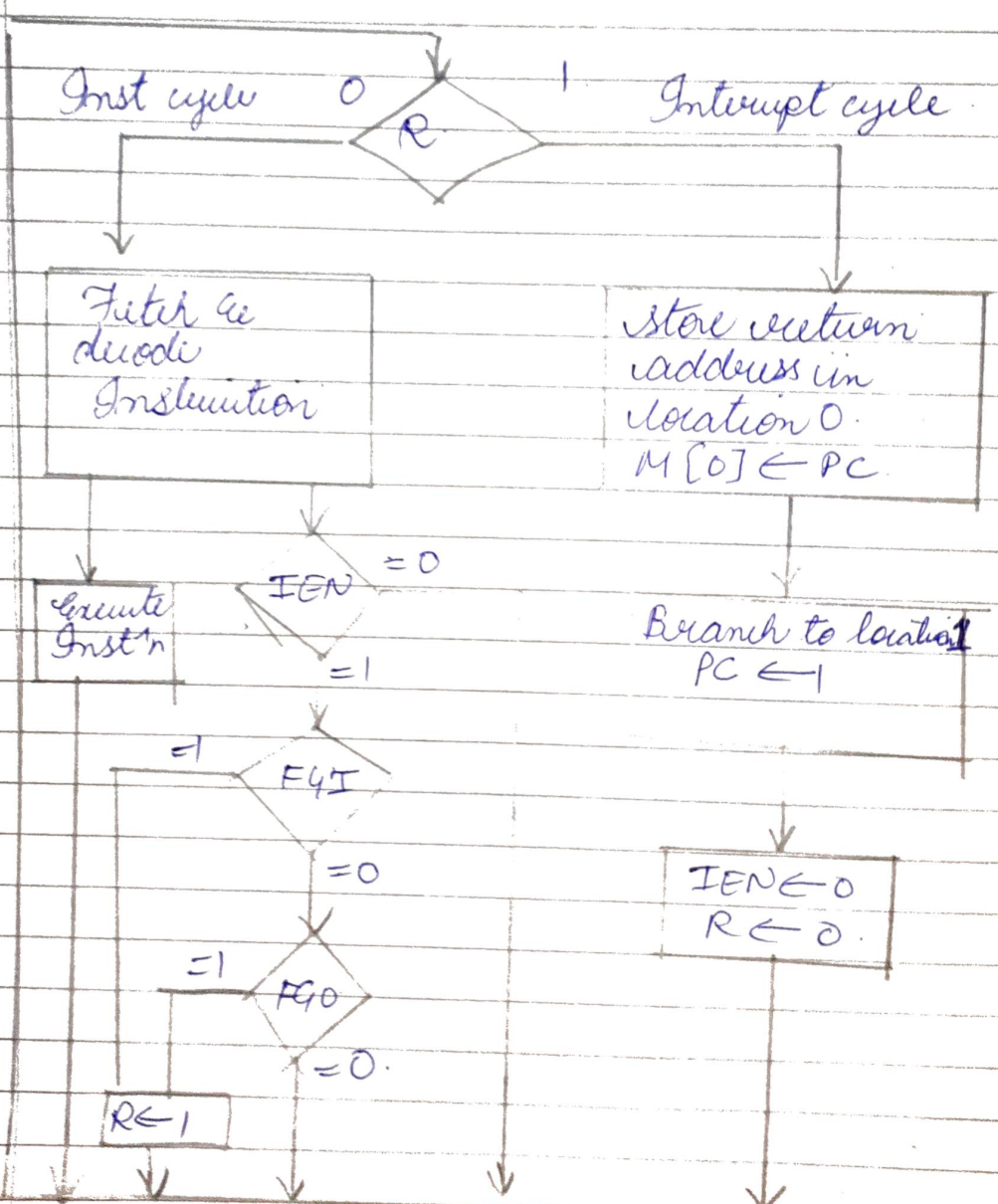


Fig :- flowchart of Interrupt cycle

__/__/__

Demonstration of Interupt cycle.

a) Before interupt

Memory

0		
1	0 BUN 1120	
255	Main	
PC = 256	Memory	
1120	I/O program	
	1 BUN 0	

b) After Interupt

Memory

0	256	
PC=1	0 BUN 1120	
255	Main	
256	Memory	
1120	I/O program	
	1 BUN 0	

* At 255, The CPU recieves interupt signal.
* So, the CPU stops the execution of currently running prog
* And it stores the next Inst address PC = 256 in an address location
  m[0] ← PC
* So we have to store m[0] in 256 location.
* So after interupt 256 is stored in 0th location.
* Now PC will be at location 1
* As BUN is there (branch unconditionally) without checking any condition it goes to 1120, then without checking condn it goes to 0

__/__/____

UNIT-2.

Control Unit

* Refer Timing & control / Hardwired CU.

Difference b/w HCU & MPU.

Hardwired Control	Micro programmed Control
* Technology is circuit based.	* Technology is S/W based.
* Implemented through flip-flops, gates, decoders	* Micro inst^n generate signals to control the execution of inst^n.
* Fixed Inst^n format	* Variable inst^n format.
* Inst^n & Reg based	* Inst^n & not reg based.
* ROM is not used.	* ROM is used.
* It is used in RISC	* Used in CISC.
* Faster decoding	* Slower decoding.
* diff to modify.	* Easily modified
* chip area is less	* Chip area is large

Important Terms

Control word

- The control variables at any time are represented by 1's & 0's
- It can be programmed to perform various operations

__/__/____

Micro programmed Control Unit
A CU whose binary control variables are stored in the memory.

Micro Instr^n's.
- Each word in CU contains Microinst^n's
- It specifies one or more operations for the systems.

Micro program.
- Sequence of micro inst^n's
- It control the functions of CPU.
- Alterations are not needed once the CU is in operation
- CU can be Read-only memory (ROM).
- The contents of words in ROM is fixed. We can't be altered since there is no writing capacity in ROM.

Dynamic Micro programmy.
- Microprog is loaded initially from a Auxiliary memory (Sec mem). Such as magnetic disk.
- It employs a writable control mem which allows the user to change the micro program, though it is mostly used for reading

Types of Memory in MPC
1) Main memory-
   - MM is used for storing programs

MP - microprograms
MI - micro instruction
MO - micro-operations

CU - control unit
CM - control memory
MM - main memory
EA - effective add

Branch Micro Inst
__/__/____

* The content of MM can be altered.
* Program in MM contains machine Instⁿ & data.

2) Control Unit Memory

* Memory that is part of CU.
* CU holds MP that can't be altered by user.
* The MP consists of MI to execute MO.
* M/C instⁿ initiates a series of MI in CM
* MI generates MO to fetch instⁿ from MM to evaluate effective address

* Any computer contains 3 components CPU, I/O devices, memory
* CPU contains 3 components - ALU, registers, CU
* CU is mainly useful for generating control signals
* CU initiates MO

Micro programmed Control operations

* The control signals are implemented with the help of MCO.
* MP is stored in CU.
* The CM is contained as ROM. (no alterations)
* Control address Register contains address of the MO i.e it reads from CM.
* Control data register contains the instⁿ that is to be executed.
* The next addr generator (sequencer) generates the next Instⁿ addr

* We know that CDR contains some MI that is to be executed, in addition to MI it contains some additional bits in order to generate next instⁿ
* CDR — pipeline register
  ↳ executing multiple tasks simultaneously in less amt of time

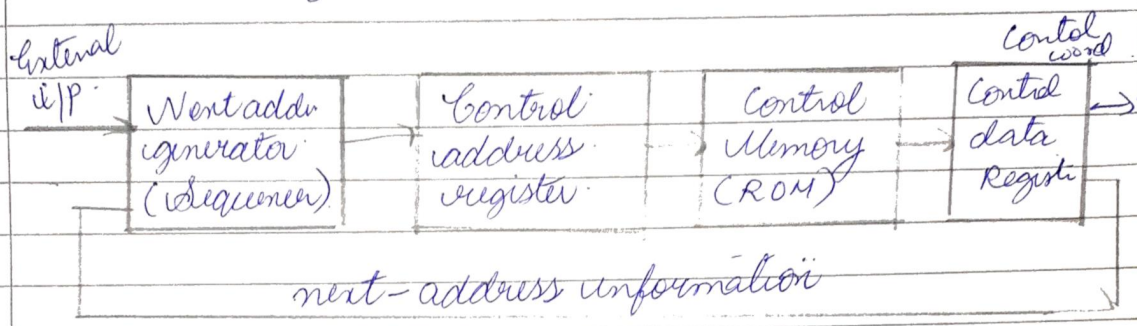


fig :- general config of MPCU.

Addressing Sequencing (QR).
Micro program Sequencer

* MI are stored in CU.
* An initial address is stored in CAR.
* fetch routine may be sequenced by incrementing CAR.
* At end of fetch routine instⁿ is in IR.
* The EA routine in CM can be reached through BMI which is conditioned on status mode bits of instr
* MO steps to be generated in processor depend on opcode of instr
* Subroutine will req can ext reg for

__/__/____

Storing return address.

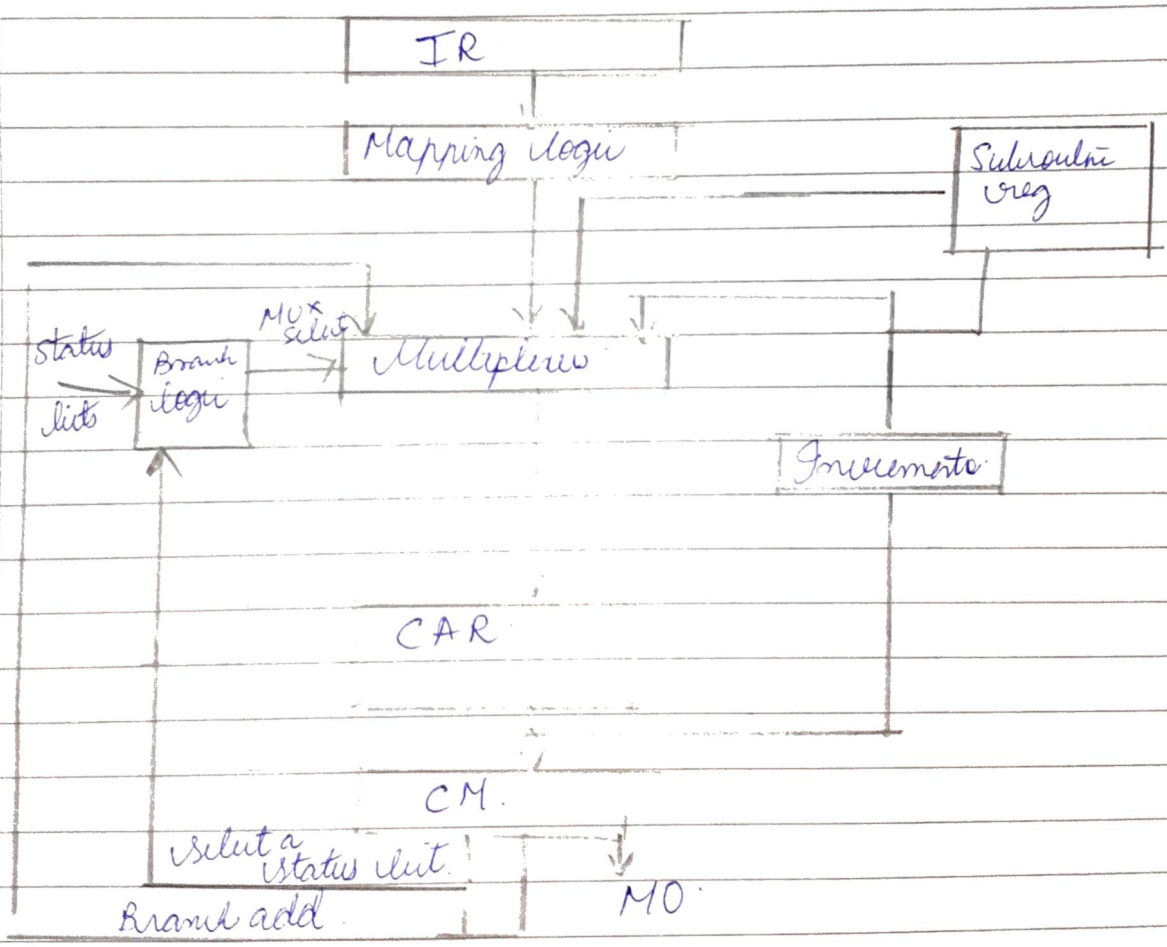
* Return address can't be stored in ROM because it has no writing capability
* The duty of MP sequencer is to generate next MI address.
* There are 4 approaches.

1) Incrementing CAR by 1
   * CAR contains address of MI stored in CM.
   * So we use an Incrementer circuit.
   * The Incrementer circuit increments CAR by 1.
   * Once incrementation is over, then the MUX transfers the unit to CAR
2) Subroutine Register.
   * Subroutine is a func which we use in C, C++
   * Subroutine contains collection of units
   * Mainly we use subroutines for repetition purpose.
   * The next statement will be stored in SR. (After the func in Main prog executes).
   * The MUX transfers the ~~inst~~ next to the CAR. then the inst will be executed.
3) Unconditional or conditional Branch
   * Unconditional - without checking the condition, the respond add → CAR.
   * Conditional - check condition → so that we have status bit
   * Both will be stored in CAR with the help of MUX
   * Branch logic checks whether the status is ~~wrong~~ right or ~~right~~ wrong

__/__/____

4) mapping Instruction-

* We use IR.
* On the opcode, mapping is done.
* add one zero on left & 2 zeros on right which is the mapping logic.
* The MUX transfers Mapping logic to CAR.



Micro program Example

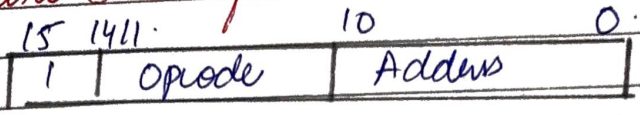
* We have MM & CM.

MM
* MM – stores data & inst
* MM – 4 reg. (DR, AC, AR, PC).
* MM size – 2048 × 16. (2^{11}) 11 bits req to specify address, 4 bits for opcode
* AR, PC size – 11 bits
* DR, AC size – 16 bits

CM
* CM – stores MP.
* CM – 2 reg. (SBR, CAR)
* CM size – 128 × 20 (2^7)
* SBR, CAR size – 7 bits

* Here we are using MUX to transfer the info among Reg.
* DR receives info from AR, memory, AC
* AR receives info from DR, PC.
* PC receive info from AR.
* Memory receive info from AR.
* ALU performs operation on contents of DR & AC & the res is being transfer to AC
* CAR receives info from AR

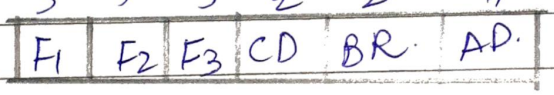
Machine Inst format.



Sample machine Inst.

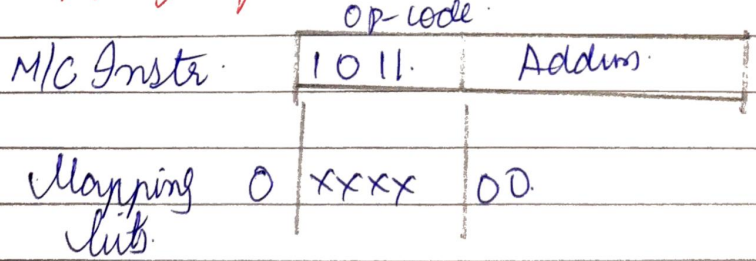
Symbol	Op-code	Desc
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC -ve < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

Micro Inst^n format.

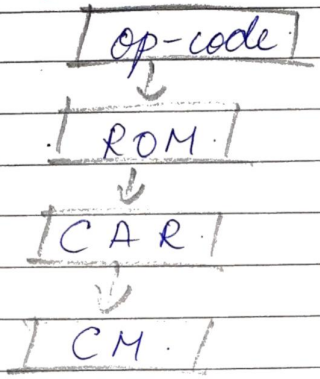


F1, F2, F3 – MO fields
CD – condition for branching
BR – Branch field
AD – Address field.

Mapping of Inst to MicroRoutine



Mapping func Imply ROM.



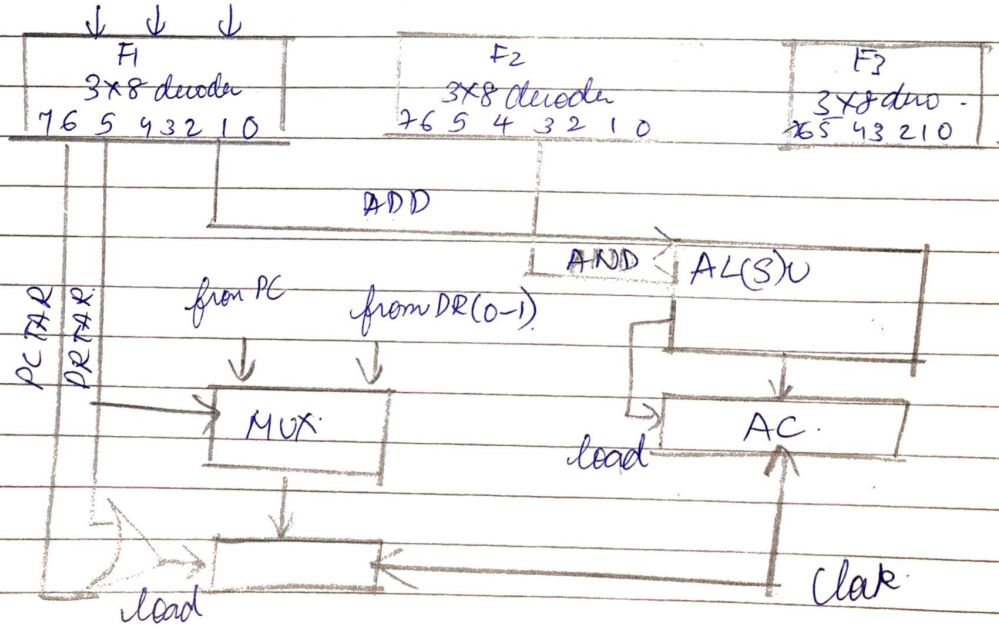
Symbolic MI.

1) Label: may be empty / symbolic address :
2) micro-ops : 1, 2, 3 (symbols used),
3) CD : one of (U, I, S, Z) where U: Uncond Branch.
I : Indirect Bit.
S : Sign of AC.
Z : Zero val of AC.
4) BR : one of {JMP, CALL, RET, MAP}
5) AD : one of { Symbolic address, next, empty }

4. Design of CU.

* CU design - 2 ways - HWC, & MPC
* MP is a collection of MI, each MI - 3-MO fields
* They are F_1, F_2, F_3, in addition to that we have Cond field, Branch field, Address field
* Size of F_1, F_2, F_3 is 3 bits.
* It is decoded with 3x8 decoder.

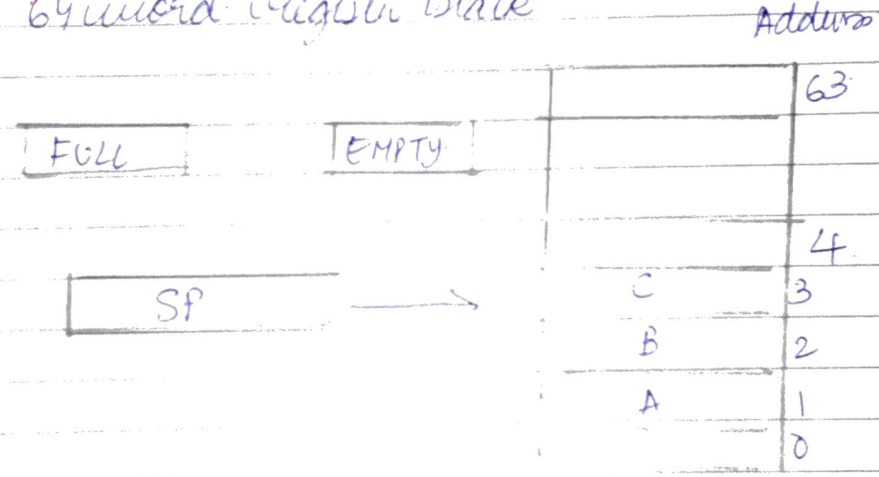
* A decoder accepts n i/p & produces 2^n o/p's
* ∴ 8 o/p's.
* Out of 8 operations, one doesn't perform any operation. So we are left with 7 operations. (In F_1 & F_2)
* In F_3 we can perform only 6 operations
* So in total – 20 operations
* Out of 20 – we demo 5 operations.
* o/p of 3 in F_2 — connected to – ALU.
* ALU performs AND operation on AC & DR. & is transfed to AC (when load enabled)
* O/P 1 of F_1 — connected to ALU.
  ALU performs ADD on AC & DR
  & is transf to AC. (when load is enabled)
* O/p 5 of F1 (DRTR) i.e transfer the contents of AR to DR.
* O/p 6 of F_1 (PCTAR) i.e transfer the content of AR to PC.
* load & clock pulse is applied to AR.



* Stack Organisation -

* Stack works on principle of LIFO
* PUSH - inserting element
* POP - deleting "
* We can implement Stack in 2 ways
  a) Register Stack
  b) Memory Stack

a) Register Stack
* 64 word register stack



* The Stack pointer SP contains a binary no whose val is equal to the address of the word that is on top of Stack.
* 3 items are placed in the stack A, B, C.
* Item C is on top of stack so that content of SP is now 3
* To remove the top item, the stack is popped by reading memory word at address 3. & decrementing content of SP

* Item B is now on the top of stack since SP holds address of 2.
* To insert a new item, the stack is pushed by incrementing SP & writing a word in the next-higher location of stack.
* 64 bit stack — 6 bits — $2^6 = 64$.
* SP — 6 bits — cant exceed >63 (111111)
* 63 ↑ by 1, res = 0 since 111111 + 1 = 1000000.
* 1 bit reg FULL — set to 1 when stack is full.
* 1 bit reg EMPTY — set to 1 when stack is EMPTY.
* DR hold binary data.

PUSH.

* If stack is not full, a new item is inserted with a push operation.
* push operation consists the foll ops.

SP ← SP+1	Inc stack pointer
M[SP] ← DR.	Write item on top of stack.
If (SP=0) then (FULL←1)	Check if stack is full.
EMPTY ← 0.	Mark the stack not empty.

* SP is ↑ so that it points to the addr of next higher word.
* Memory write operation inserts the word from DR into top of stack
* SP holds the address of top of stack.
* M[SP] denotes memory word
* The 1st item stored in a stack is at address 1, last item — 0

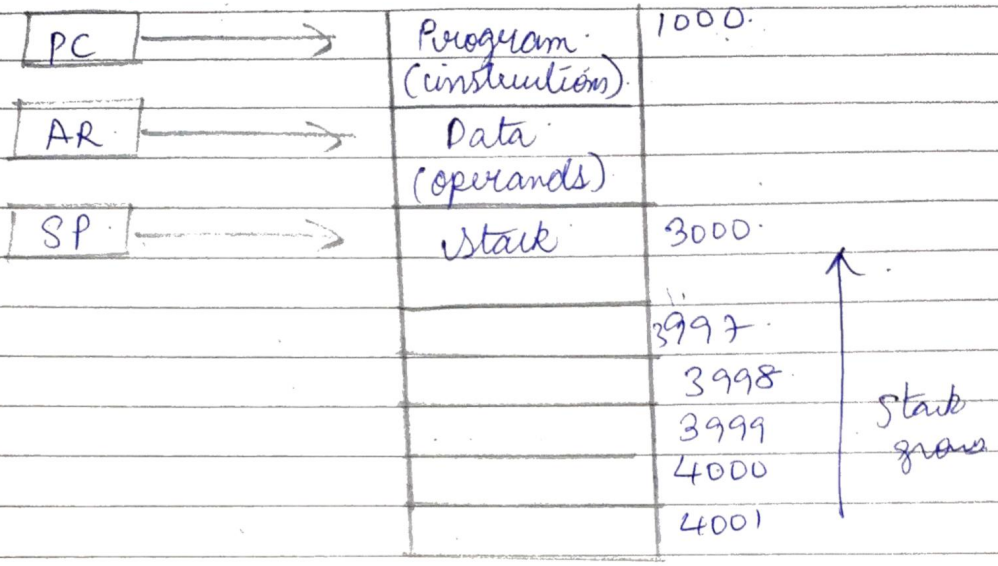
* If SP reaches 0, the stack is full of items. So FULL is set to 1.
* Once an item is stored in location 0 there is no more empty register in the stack
* If an item is written in stack, then the stack can't be empty. So EMPTY is cleared to 0.

POP

* A new item is deleted from the stack if stack is not empty.
*
```
DR ← M[SP]        | Read item on top of stack.
SP ← SP - 1.      | ↓SP.
IF (SP=0) then (EMPTY←1) | Check if stack is empty.
FULL ← 0.         | Mark the stack not Full
```

* The top item is read from the stack into DR.
* SP is then ↓
* If val = 0, Stack is empty.
* So EMPTY is set to 1.
* This condition is reached if the item read was in location 1.
* Once this item is read out, SP is ↓ & reaches the value 0, which is init val of SP
* SP changes to 111111 = 63
* In this config, the word in address 0 receives the last item in stack.
* Erroneous operation will ~~receive~~ result if stack is pushed when FULL = 1 or popped when EMPTY = 1

Memory Stack



* Computer memory is partitioned into 3 segments: program, data, stack
* PC points at the addrs of next instⁿ in the prog which is used during fetch phase to read an inst
* The AR points to array of data
* SP points at top of stack
* 3 reg are connected to Common Bus.
* Init val of SP – 4001, stack grows with ↓ values
* 1st item – @4000
  2nd item – @ 3999.
  last item – @ 3000
* Asme – stack communicates with DR
* Stack grows ↓ by ↑ Mem Addess.
* Advantage – CPU can refer to it without having to specify an address

__/__/____

PUSH:-

* A new item is inserted with push operation.

SP ← SP − 1
M[SP] ← DR

* SP is ↓ so that it points to address of next word.
* A memory write operation inserts the words from DR to top of stack.

POP:-

* A new item is deleted with POP operation.

DR ← M[SP]
SP ← SP + 1

* Top item is read from stack from stack into DR.
* Then SP is ↑.

★. INSTRUCTION FORMATS.

1) Address field
2) opcode
3) mode.

1) Address field.
   a) Single Accumulator Org.
   b) General Purpose Reg
   c) Stack Org

Single Acc Org.

* No need to specify Acc explicitly.
* Eg CLA & CMA are implicit

GPR.

* Inst contains multiple addresses (reg / operand)

Eg: ADD R1, R2, R3.

$R_1 \leftarrow R_1 + R_2 + R_3$

(source & dest.)

Stack

* push, pop.
* Operations — ADD, SUB, MUL, DIV

Based on the no. of Inst, addresss can be classified into 3 types.

$X = (A+B) * (C+D)$

three-address-Instn.

ADD R1, A, B.	$R_1 \leftarrow m[A] + m[B]$
ADD R2, C, D.	$R_2 \leftarrow m[C] + m[D]$
MUL X, R1, R2.	$m[X] \leftarrow R_1 * R_2$

two-address-Inst.

mov R1, A.	$R_1 \leftarrow m[A]$
ADD R1, B.	$R_1 \leftarrow R_1 + m[B]$
mov R2, C.	$R_2 \leftarrow m[C]$

ADD R_2, D	$R_2 \leftarrow R_2 + m(D)$
MUL R_1, R_2	$R_1 \leftarrow R_1 \times R_2$
MOV X, R_1	$m[X] \leftarrow R_1$

One address Inst (LOAD & STORE)

LOAD A	$AC \leftarrow m[A]$
ADD B	$AC \leftarrow AC + m[B]$
STORE T	$m[T] \leftarrow AC$
LOAD C	$AC \leftarrow m[C]$
ADD D	$AC \leftarrow AC + m[D]$
MUL T	$AC \leftarrow AC * m[T]$
STORE X	$M[X] \leftarrow AC$

Zero address Inst (STACK)

PUSH A	$TOS \leftarrow A$
PUSH B	$TOS \leftarrow B$
ADD	$TOS \leftarrow (A+B)$
PUSH C	$TOS \leftarrow C$
PUSH D	$TOS \leftarrow D$
ADD	$TOS \leftarrow (C+D)$
MUL	$TOS \leftarrow (C+D) * (A+B)$
POP	$m[X] \leftarrow TOS$

Risc (LOAD & STORE)

LOAD R_1, A	$R_1 \leftarrow m[A]$
LOAD R_2, B	$R_2 \leftarrow m[B]$
LOAD R_3, C	$R_3 \leftarrow m[C]$
LOAD R_4, D	$R_4 \leftarrow m[D]$
ADD R_1, R_2	$R_1 \leftarrow R_1 + R_2$
ADD R_3, R_4	$R_3 \leftarrow R_3 + R_4$
MUL R_1, R_3	$R_1 \leftarrow R_1 * R_3$
STORE X, R_1	$m[X] \leftarrow R_1$

__/__/____

Address Modes

* Used to determine effective address or operand.

Types of Addressing Modes

1) Implied AM.
2) Immediate AM.
3) Register AM.
4) Register Indirect AM.
5) Auto Increment AM.
6) Auto Decrement AM.
7) Direct AM.
8) Indirect AM.
9) Relative AM
10) Base Register AM.
11) Indexed AM.

1). IAM :-
* Operand is implicitly available in the definition of Instr.
* no need to specify the operand explicitly.
* Eg :- CLA, CMA.

2). Im AM :-
* Instr :- should not contain any Address
* In place of address we will be having operand.

3) RAM.
* The corresponding register contains the content of the AC.

4) RI,AM :-
* Register contains effective address

5) AI AM
* Post increment approach, similar to RI, AM

6) ADAM
* Pre decrement approach

7) D AM
* The inst contains address, which becomes effective address

8) I AM.
* The inst contains an addres, we have to go to that address, to determine the EA.

9) Rel AM.
* The content of PC is added to address field which to get EA.

10) B.R :- The content of BR is added to AF to get EA
I.R - " " of IR " " " " " "
↳ values — both 100.

	200	Load to AC | Mode
	201	Address = 500
↷	202	Next Inst
PC		- - -
		- - -
PC = 200		- - -
	399	450
R_1 = 400	400	700
XR = 100	500	800
AC	600	900
↳	702	325
	800	300

AM	E.A	Content of AC
IN AM	201	500
R AM	—	400
R Idl AM	400	700
Auto Inc AM	400	700
Auto Dec AM	399	450
Dir AM	500	800
Ind AM	800	300
Rel	702	325
B.R	600	900
I.R	600	900

__/__/__

Data Transfer Inst.

Diff types of Data Transfer Inst.

Name	Mnemonic	
Load	LD	transfr mem word to AC
Store	ST.	stores AC into Mem X
Move	MOV.	transfr 1Reg to other
Exchange	XCH.	exchng conten = 2 Mem / 2 Reg
Input	IN	Ip info is provided to processor reg
Output	OUT.	Op info is received from O/P dev
push	PUSH	push to stack from Proces Reg
pop	POP.	pop from stack to Proces Reg

8 Addressing modes for LOAD Inst.

Mode	Assembly Conv	Reg Transf
Dir Address	LD ADR.	AC ← M [ADR]
I A.	LD @ADR.	AC ← M[M[ADR]]
Rel A.		AC ← M [PC + ADR]
Imm A.	LD #NBR.	AC ← NBR.
Indexed A.	LD ADR(X)	AC ← M [ADR + XR]
Reg "	LD R1.	AC ← R1.
Reg Ind A.	LD (R1)	AC ← M [R1]
Auto Inc A.	LD (R1)+.	AC ← M [R1], R1 ← R1 + 1.

__/__/____

UNIT-3.

Floating point Rep & IEEE.

* MUL & DIV on FP no's.
* Similar – Scientific notation.
* A floating point no. consists of signed (+ve / -ve) digits string given in the given bias
* 3 parts.
  a) mantisa b) base c) Exponent
  Mantisa × Base^exp.
* Decimal rep.

1.234×10^{4} → exp.
mantisa, Base

* All teh – Number system.
* Scientific notation — No sys in FP.
* Eg of no. sys — a) speed of light
  b) charge of e⁻
  c) age of universe.
* They have rep in form of $\pm M \times B^{E}$
* FP nos used to rep — very small
  — very large
* ∴ we foll FP instead of fixed point

Floating point Add & sub Algo.

1) Check for zeroes.
2) alignment of mantissa
3) add or sub mantisa
4) normalise the res.

Eg:- $0.\underbrace{23456}_{M.} \times \underbrace{10^{5}}_{B.}$ (MSB: 2, E: 5)

* The FP is in Normalised form.
* The Most Significant Bit of Mantisa is a non zero value.

Eg:-

$AC = 0.123 \times 10^3$

$BR = 0.345 \times 10^3$

Addition	Subtraction
0.123×10^3	0.123×10^3
$+ \quad 0.345 \times 10^3$	$- \quad 0.345 \times 10^3$
0.468×10^3	0.222×10^3

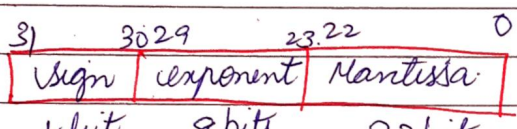
* if exp are equal, alignment of Mantisa is over
* if suppose the res is 1.468×10^3, 1 is overflow.
* We have to perform Shift right operation
* 0.468×10^4
* if suppose the res of sub is 0.022×10^3 0 is underflow.
* We have to perform shift left opn.
* Normalisation → overflow → Shift right opn + Inc exponent
  → underflow → Shift left opn + dec exponent.
  0.110×10^2

IEEE 754 Floating pt rep

* IEEE – Inst of electrical & electronics org.
* 754 – specification no.
* Rep by.

① Single Precision format.

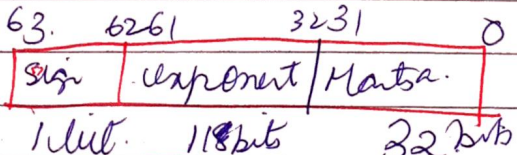
* SPF is a format prop by IEE for rep of Floating pt no's.
* occupies – 32 bits



* Used in simple prog like games.
* 8 bits – exp.
* 23 bits – Mantissa.
* Range of Single Precision – 126 to + 127

② Double Precision format.

* DPF is format prop by IEE for rep of FPN
* occupies – 64 bits.



* Used in complex prog – scientific.
* 11 bits – exp.
* 32 – mantsa.
* Rang of Double Pre — 1022 to + 1023

__/__/____

Unit 4.

✱. Asynchronous Data Transfer

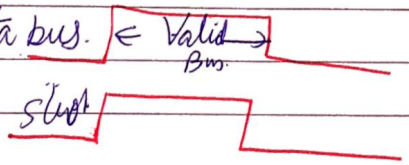
Explain Strobe & handshaking control methods of Asynchronous data.

A). Strobe Control Method of AD.

* Uses – Single control signal for each transfer
* That control signal is called Strobe.
* Strobe Control method – 2 types.
  a) Source Initiated Strobe
  b) Destination Initiated Strobe.

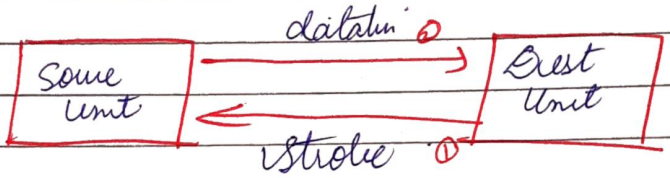
1). Source Initiated Strobe.

* data bus carries binary info from S to D.
* data bus carries 0's + Strobe signal.
* Source Initiated Strobe means Source Informs the destination, the valid data available in data bus. ie Initiating Strobe
* Timing Signals.

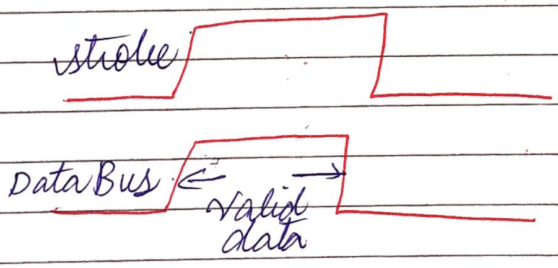


* Source unit 1st places the data on the data bus.
* Strobe is saying that a valid data is there in the data bus. so accept valid data that info is sent by strobe signal.

2) Destination Initiated Strobe



* Destination says to strobe signal please keep valid data in data bus
* The source unit keeps the valid data on the bus.

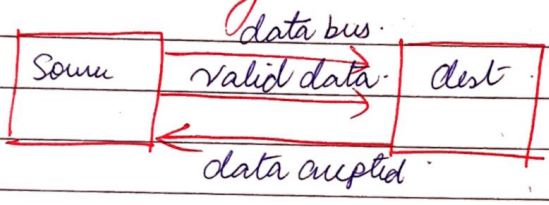


B) Handshaking control Method

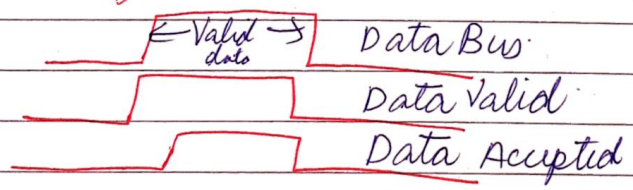
* We go for handshaking method even though we have strobe control
* disadv of Strobe :- The is no knowing whether the data placed on bus is receiving data or not. acknoledgment is not there in Strobe Control

* To overcome this problem we use MSM.
* MSM — 2 types
  a) MS source to destination
  b) MS dest to source

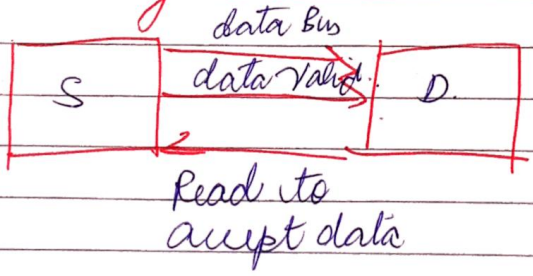
1). Hand Shaking Source to Destination



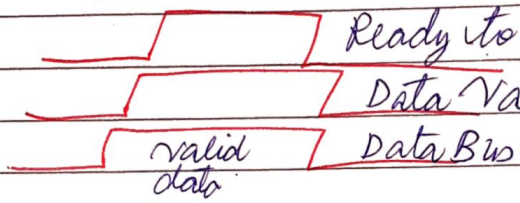
* The data bus carry binary info along with valid data from S to D
* data valid is received by D
* D sending S "data is accepted" then source will understant what data I sent is received.
* This is acknowledgement
* Timing Signals



2) Hand Shaking Destination to Source.



* 1st, the D sends signal to S "Ready to accept data" means whatever data I previously sent that is finish so I'm ready to accept another data.
* When S sees it, It places data on the data bus along with valid data
* Timing signals

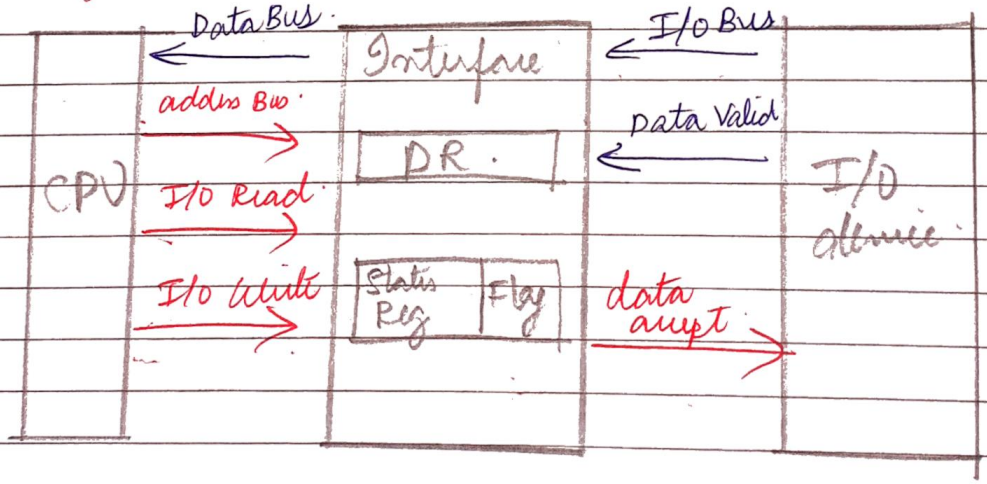


Modes of data transfer

There are 3 modes of data transfer

1) Programmed I/O
2) Interrupt driven I/O
3) DMA (Direct Memory Access)

(1) Programmed I/O



* programmed I/O is used to transfer data b/w I/O devices & memory
* I/O devices — I/P & O/P → keyboard monitor
* programmed I/O is implemented by prog.
* b/w CPU & I/O devices we have interface.
* I/O device places data on data bus. after placing data IO device will say that bit is valid data
* Data Valid says that some data is present in I/O bus
* I/O bus data was placed in DR.
* The flag bit is 0 or 1.
* The flag bit is 1, means some info is there, is accepted & sent to I/O device.
* The info placed in DR will be send to CPU with the help of data bus
* CPU can access with I/O device by specifying its address, every I/O device contain some address.
* I/O Read & I/O write are control lines mainly. we perform 2 opn Read & Write.
* In this way, data is transfed to CP.

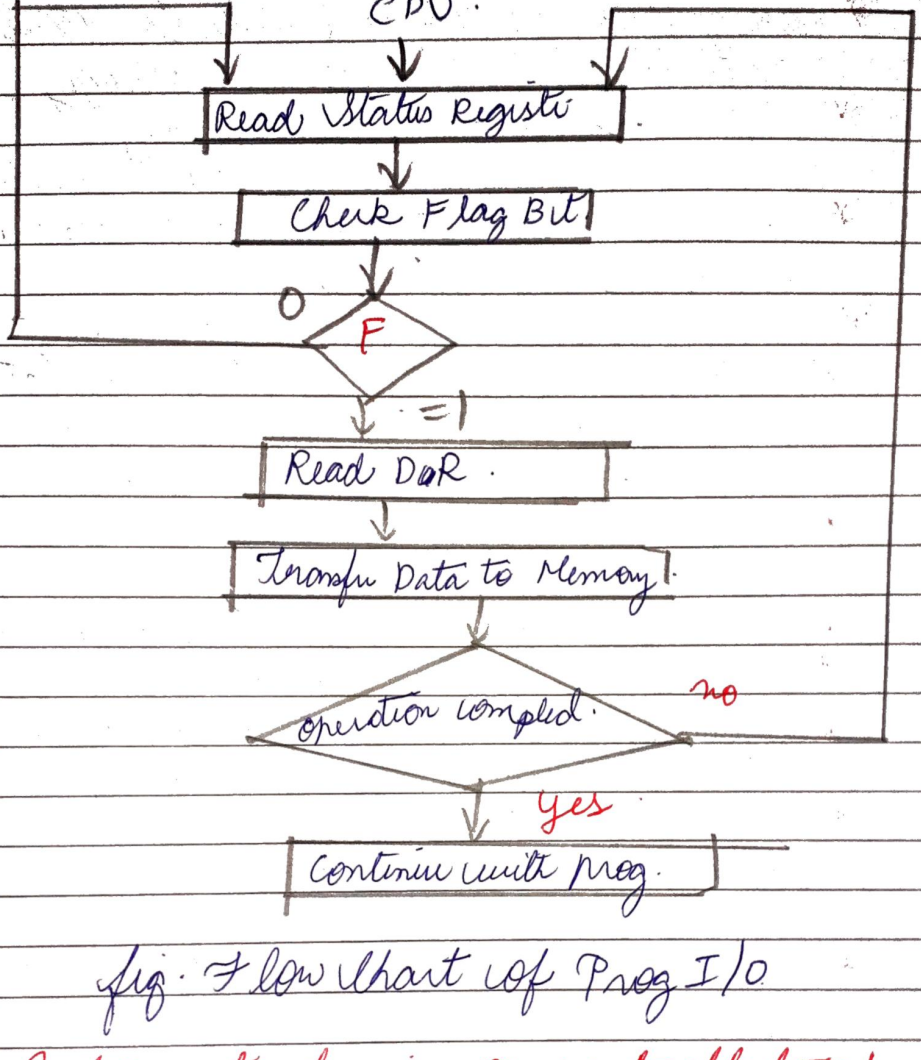


fig. Flow chart of Prog I/O

2) Interuppt driven or controlled I/O.

* disad of PI/O :- CPU taks long time
* Overcom - ID CI/O.
* IDC I/O - need not wait.
* I/O module ready then it interupts CPU.
* If CPU is doing some work and Interupt occurs from I/O module. It stops the execution & takes Interupt request.