Date : Page :

Read - This causes a transfer of info. to the DR from the memory word M selected by the address in AR.

$DR \leftarrow M(AR)$

Write - This causes a transfer of info. from R_1 into the memory word selected by the address in AR.

$M[AR] \leftarrow R_1$

$R_3 \leftarrow R_1 + R_2$

Arithmetic micro operation -

$D \leftarrow A + 1$

$R_3 \leftarrow R - 1$

Logic micro op^n. -

$F \leftarrow 0$ clear

$f \leftarrow A \wedge B$ AND

$f \leftarrow A \vee B$ OR

Shift micro op^n. -

(i) Logical shift

(ii) circular shift

(iii) Arithmetic shift

(i) Logical :-

$R_1 \leftarrow shl\ R_1$

$R_2 \leftarrow shr\ R_2$

(1)0010

After Left + shift

00100

Date : Page :

(ii) Circular :- (1)100100 — $R_1 \leftarrow cil\ R_1$, $R_2 \leftarrow cil\ R_2$

After left shift
1001001

(iii) Arithmetic :- 10011001 (-103) /2

ARS
11001100 (-52)

ALS (1)1011001 (-39)
10110010 (-78) x2

8 bit register → 

Left most bit indicates sign
1 → -ve
0 → +ve

* sign should remain same after ALS and ARS

These two micro op^n specify a 1 bit shift to the left of the content of R_1 register and a 1 bit shift to the right of the content of register R_2. The registers involved must be same on both sides of the arrow. The bit transferred to the end pos^n through the serial input is assumed to be 0 during a logical shift.

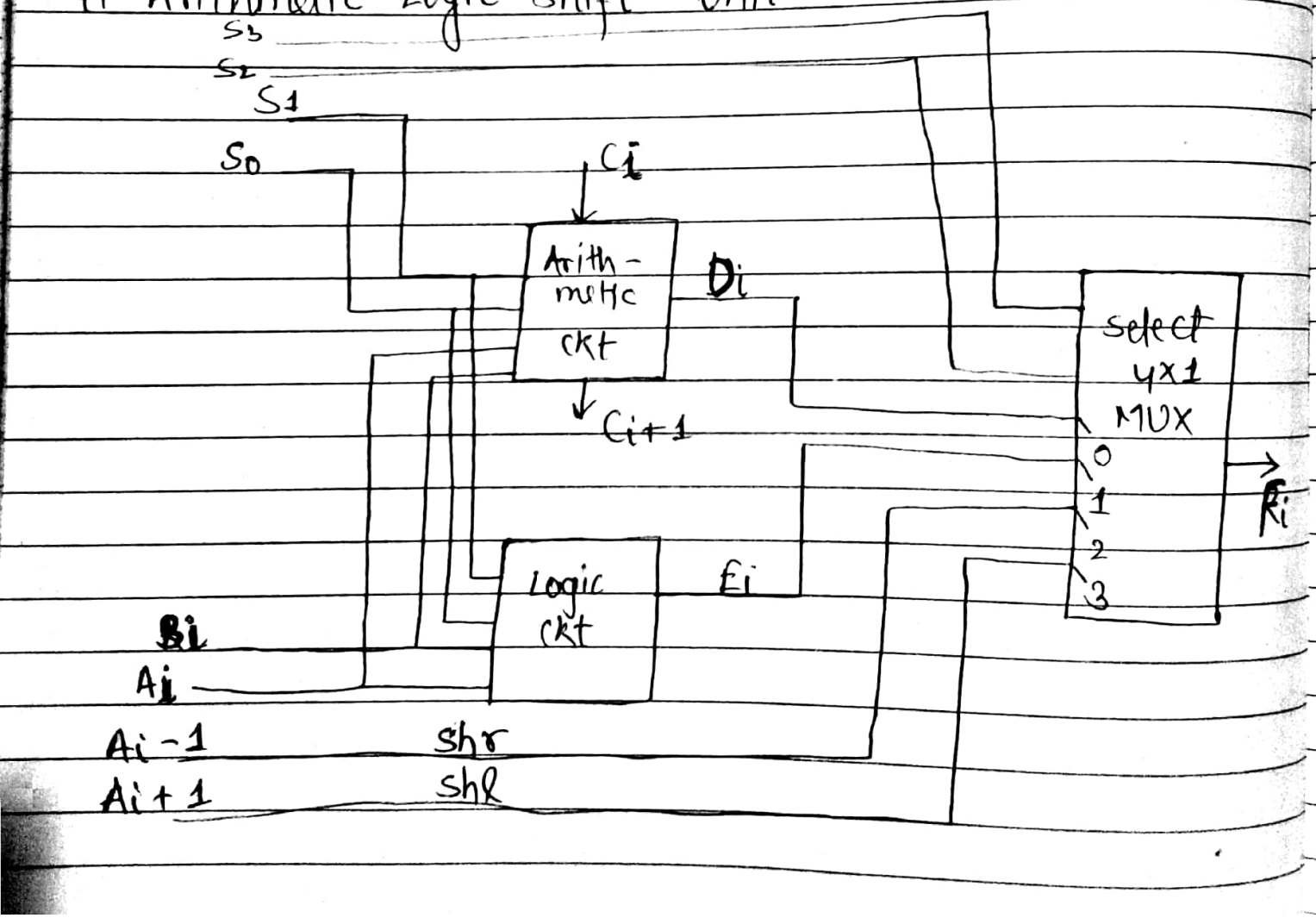
The Circular shift circulates the bits of the register around the two ends without the loss of info.

An arithmetic shift is a micro op^n that shifts a

Date : Page :

signed binary no. to the left or right. An arithmetic shift left multiplies a signed binary no. by 2. An arithmetic right shift divides the no. by 2. Arithmetic shift must be sign bit unchanged bcz the sign of the no. remains the same when it is multiplied or divided by 2. The left most bit in a register holds the sign bit and the remaining bits hold the no.

Arithmetic Logic shift Unit -



Date : Page :

S_3	S_2	S_1	S_0	C_{in}	op^n	
0	0	0	0	0	$f = A$	Arithmetic
0	0	0	0	1	$f = A+1$	
0	0	0	1	0	$f = A+B$	
0	0	0	1	1	$f = A+B+1$	
0	0	1	0	0	$f = A+\bar{B}$	
0	0	1	0	1	$f = A+\bar{B}+1$	
0	0	1	1	0	$f = A-1$	
0	0	1	1	1	$f = A$	
0	1	0	0	x	$f = A \wedge B$	Logic
0	1	0	1	x	$f = A \vee B$	
0	1	1	0	x	$f = A \oplus B$	
0	1	1	1	x	$f = \bar{A}$	
1	0	x	x	x	$f = shrA$	Shift
1	1	x	x	x	$f = shlA$	

op^n ~~x~~

~~f = A~~
~~f = A+1~~
~~f = A∧B~~
~~f = A+B+1~~
~~f = A∨B~~

(x) → don't care cond.
↓
No carry in case of logical and shift.

Instead of having individual registeres performing the micro-op^n directly, computer system employ a no. of registers connected to a ALU. To perform a micro-op^n the contained of specified registers are placed in the inputs of the common ALU. The ALU performs an op^n & the register of the op^n is then transferred to a destination register.

Date : Page :

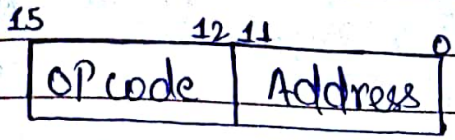
Inputs Ai and Bi are applied to both the arithmetic & logic units. A particular micro op^n is then selected with inputs S_1 and S_0. A multiplexer at the output chooses b/w an arithmetic output Di and a logical output in Ei. The data in the multiplexer are selected with inputs S_3 and S_2. The other two data inputs to the MUX receive inputs A_{i-1} for the right shift op^n and A_{i+1} for the left shift op^n. The output carries C_{i+1} of a given arithmetic stage must be connected to the input carry C_i of the next stage in sequence. The input carry to the first stage is the Cin which provides a selection variable for the arithmetic op^n. The first 8 are arithmetic op^n and are selected with $S_2 = 0$ and $S_3 = 0$. The next 4 are the logic op^n for which $S_2 = 1$ & $S_3 = 0$. The input carry has no effect during logic op^n and is marked with X. The last 2 op^n are shift op^n for which $S_3S_2 = 10$ & $S_3S_2 = 11$.

Date : Page :

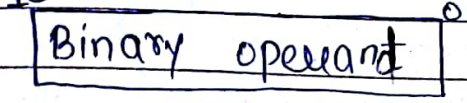
UNIT - II

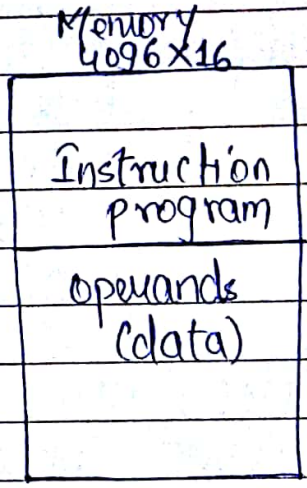
Instruction Code

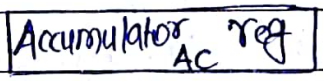
Stored program Organization,



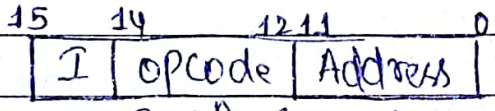
Instruction format



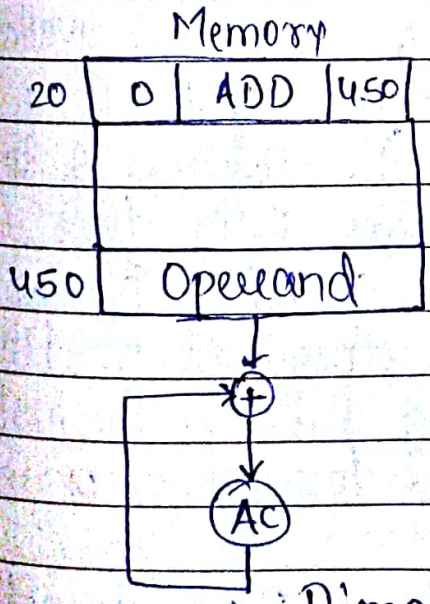




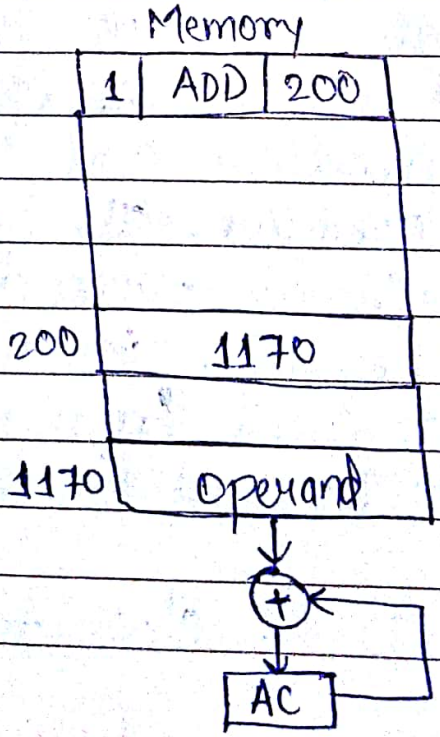
Indirect address



Inst^n format



Direct add.



Indirect add.

Date : Page :

An inst^n code is a group of bits that instruct the computer to perform a specific op^n. It is divided in 2 parts one is address and another is opcode. The opcode code of an inst^n is a group of bits that define op^n such as ADD, SUB, MUL, DIV. The no. of bits req. for the opcode of an inst^n depends on the total no. of operations available in the system. The opcode must consist of atleast n bits for a given 2^n distinct op^n.

In spo,

The first part specify the op^n to be performed and the second specifies an address.

The memory add. tells the control where to find an operand in memory. This operand is data from memory and uses as the data to be operated on together with the data stored in the processor reg (AC).

for a memory unit with 4096 words, we need 12 bits to specify an address if we store each instruction code in one 16 bit memory word, we have available 4 bits for the operation code to specify upto 1 out of 16 operations and 12 bits to specify the address of an operand. The operation is performed with the memory operand and the content of AC. If an operation in an instruction code does not need an operand from memory the rest of bits in the inst^n can be used for other purposes. for ex:- operation such as clear AC, complement AC. They do not need

Date : Page :

an operand from memory.

▷ Indirect address –
When the II part of an inst^{n} code specify an operand the inst^{n} is set to have an immediate operand.
When II part specify the address of the operand, the inst^{n} is set to have a direct address.
When the bits in the II part of inst^{n} contains the add. of a memory word in which the address of the operand is found in direct add.
One bit of a inst^{n} code can be used to distinguish b/w a direct and indirect address.

Registers –

1) Data reg. → It holds memory operand.
2) Address reg. → It holds address for memory.
3) Accumulator reg. → Processor reg.
It holds the operand.
4) Inst^{n} reg. → It holds the inst^{n} code.
5) Program counter → It holds the add. of next inst^{n} to be executed.
6) Temporary reg. → It holds temporary data.
7) INPR → It holds input character.
8) OUTR → It holds output character.

Date : Page :

No. of bits	functions of Register
DR - 16	Holds memory operand
AR - 12	Holds address for memory
AC - 16	Processor Register
IR - 16	Holds inst^{n} code
PC - 12	Holds address of next inst^{n}
TR - 16	Holds temporary data
INPR - 8	Holds input characters
OUTR - 8	Holds output characters

// Diagram

The no. of wires will be excessive if connections are made b/w the outputs of each register and the input of other register. A more efficient scheme for transferring info. in a system with many registers is to use a common bus. The output of 7 registers and memory are connected to the common bus. The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2, S_1, S_0. for example, if $S_2S_1S_0 = 011$ means Data reg. is selected to place data on the bus. The particular reg. whose load (LD) input is enabled receives the data from the bus during next clock pulse.

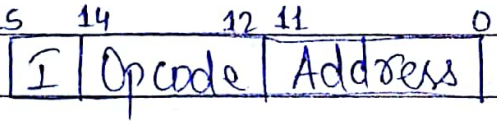
The memory place its 16 bit output onto the bus when the read input is activated and $S_2S_1S_0 = 111$ The memory receives the content of the bus when its write input is activated.

Date : Page :

INPR is connected to provide info. to the bus but OUTR can only receive info. from the bus. This is b/s INPR receives a character from an input device which is then transferred to AC and delivers it to an output device. There is no transfer from outer to any of the other reg. The 16 lines of the common bus receive info. from six reg. and the memory unit. The bus lines are connected to the inputs of 6 registers and the memory unit. The input data and output data of the memory are connected to the common bus but the memory add. is connected to AR.

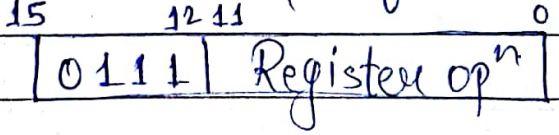
Therefore, AR must always be used to specify a memory address. By using a single reg. for the address we eliminate the need for an add. bus that would have been needed otherwise.

Computer Instⁿ –



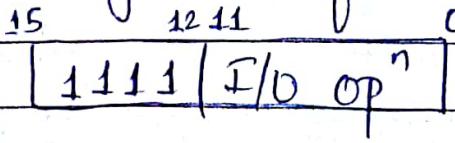
opcode = 000 – 110

→ Memory reference instⁿ



opcode = 111

→ Register reference instⁿ



Opcode = 1111

→ I/O instⁿ

Date : Page :

➢ Commands – (1) Memory reference inst^{n}
AND, ADD, LDA, STA, ...
(2) Register reference inst^{n}
CLA, CLE, CMA, ...
(3) I/O inst^{n}
INP, OUT, ION, IOFF, ...

The basic computer has 3 inst^{n} code formats, each format has 16 bits. The opcode part of the inst^{n} contains 3 bits and the meaning of the remaining 13 bits depends on the operation code encountered.

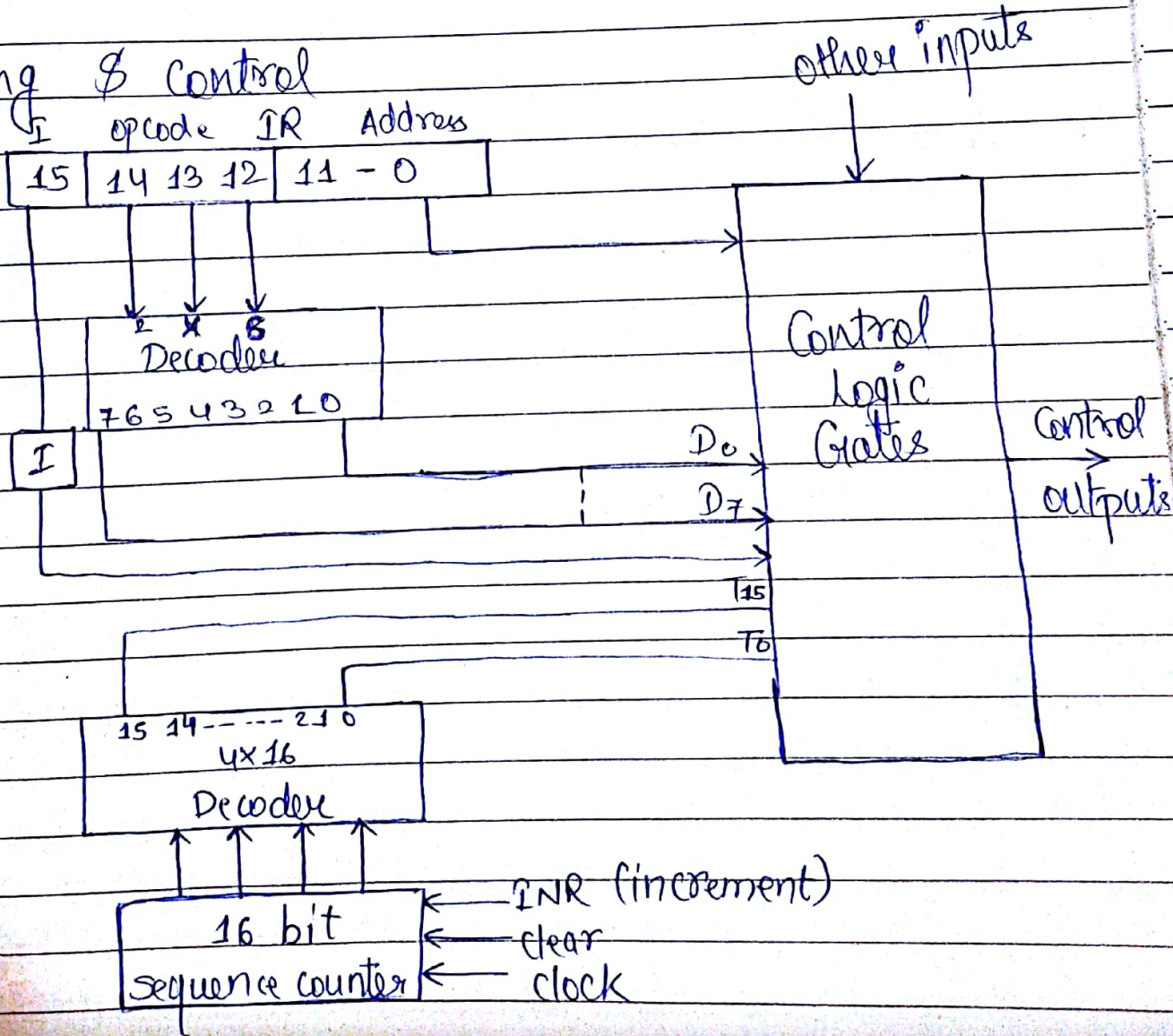
A memory reference inst^{n} uses 12 bits to specify an address and 1 bit to specify the addressing mode I. I = 0 for direct address and I = 1 for indirect address. The register reference inst^{n} are recognized by the opcode 111 with 0 in the leftmost bit of the inst^{n}.

A register reference inst^{n} specifies an op^{n} on the AC reg. and operand from memory is not needed. Therefore the other 12 bits are used to specify the op^{n} to be executed. An I/O inst^{n} does not need a ref. to memory and is recognized by opcode 111 with 1 in the leftmost bit. The type of inst^{n} is recognized by the computer control from the 4 bits in positions 12-15 of the inst^{n}.

A computer should have a set of instructions so that the user can construct machine language program to

evaluate any f^n that is known to be computable. The set of $inst^n$ are said to be complete, if the computer includes a sufficient no. of instructions in each of the following categories :-

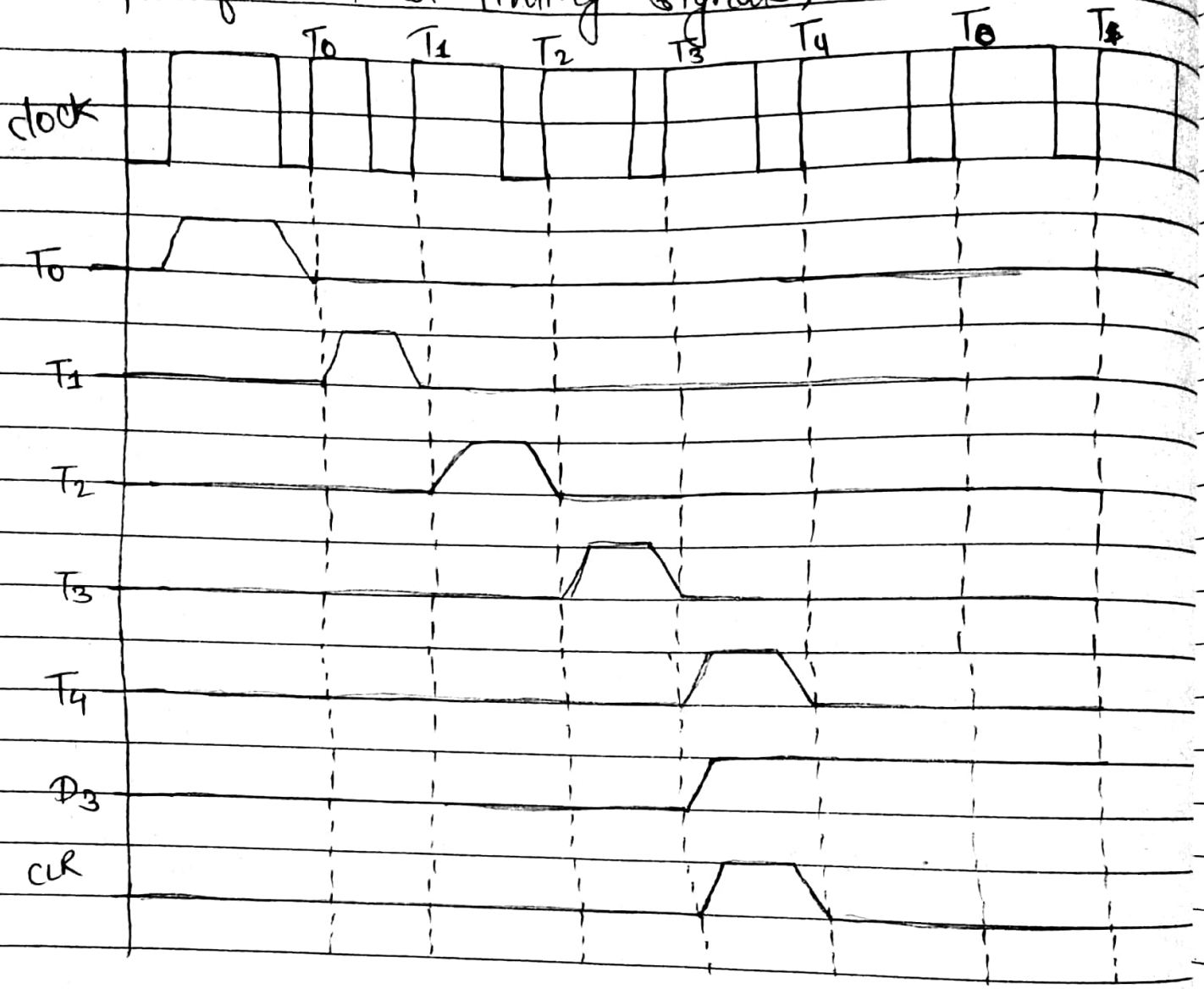
(a) Arithmetic, Logical and Shift $inst^n$
(b) $Inst^n$ for moving info. to and from memory and processor registers.
(c) Program control $inst^n$ together with $inst^n$ that check status conditions.
(d) I/O $inst^n$



$D_3T_4 : SC \leftarrow 0$

Date : Page :

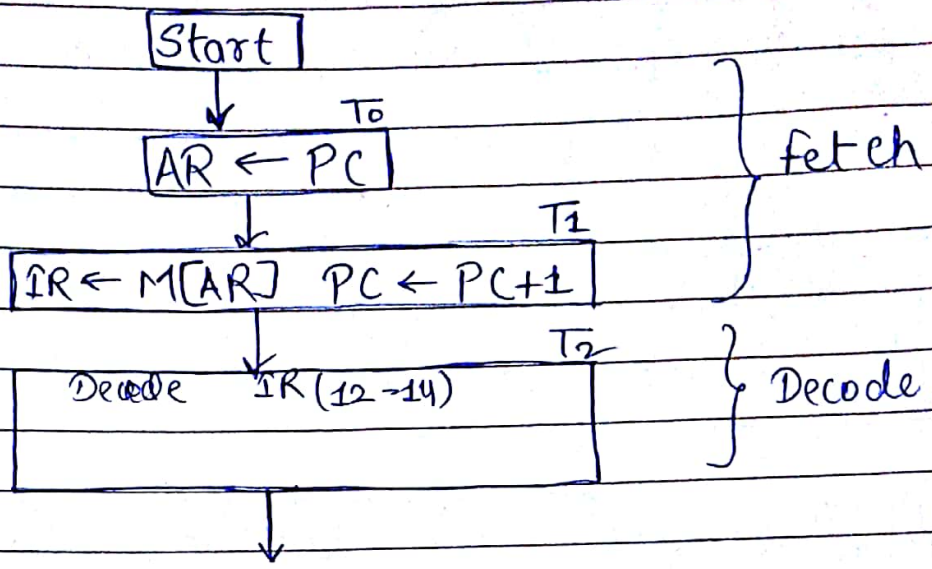
Example of control timing signals,



Instruction Cycle

Date : Page :

fetch
Decode
Decision
Execute

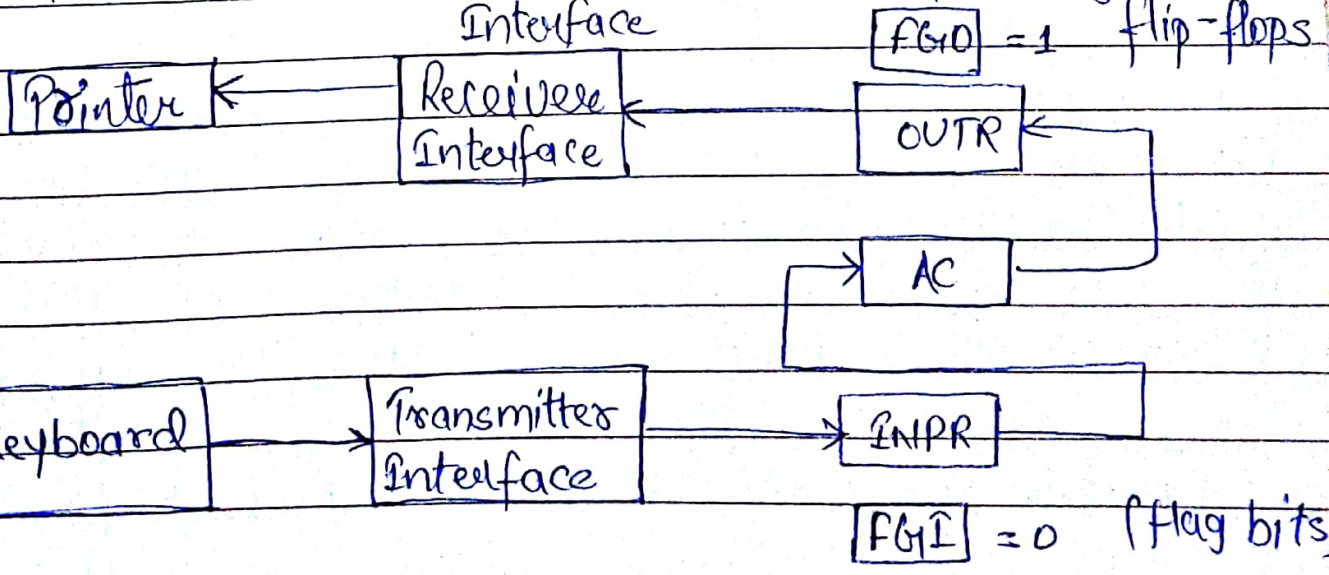


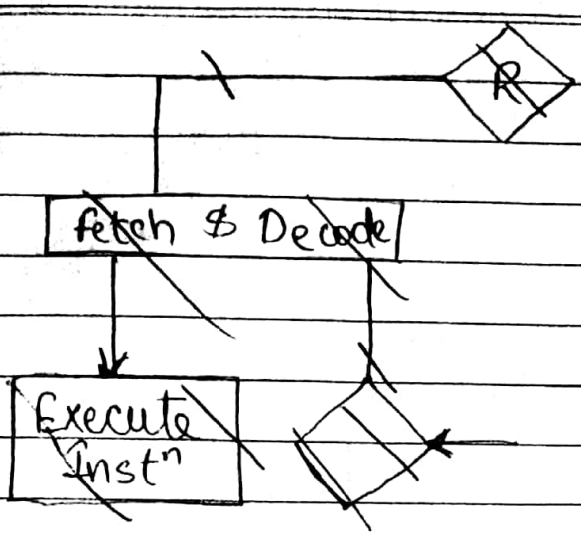
(I) Input - Output

I/o terminal

Serial comm^n Interface

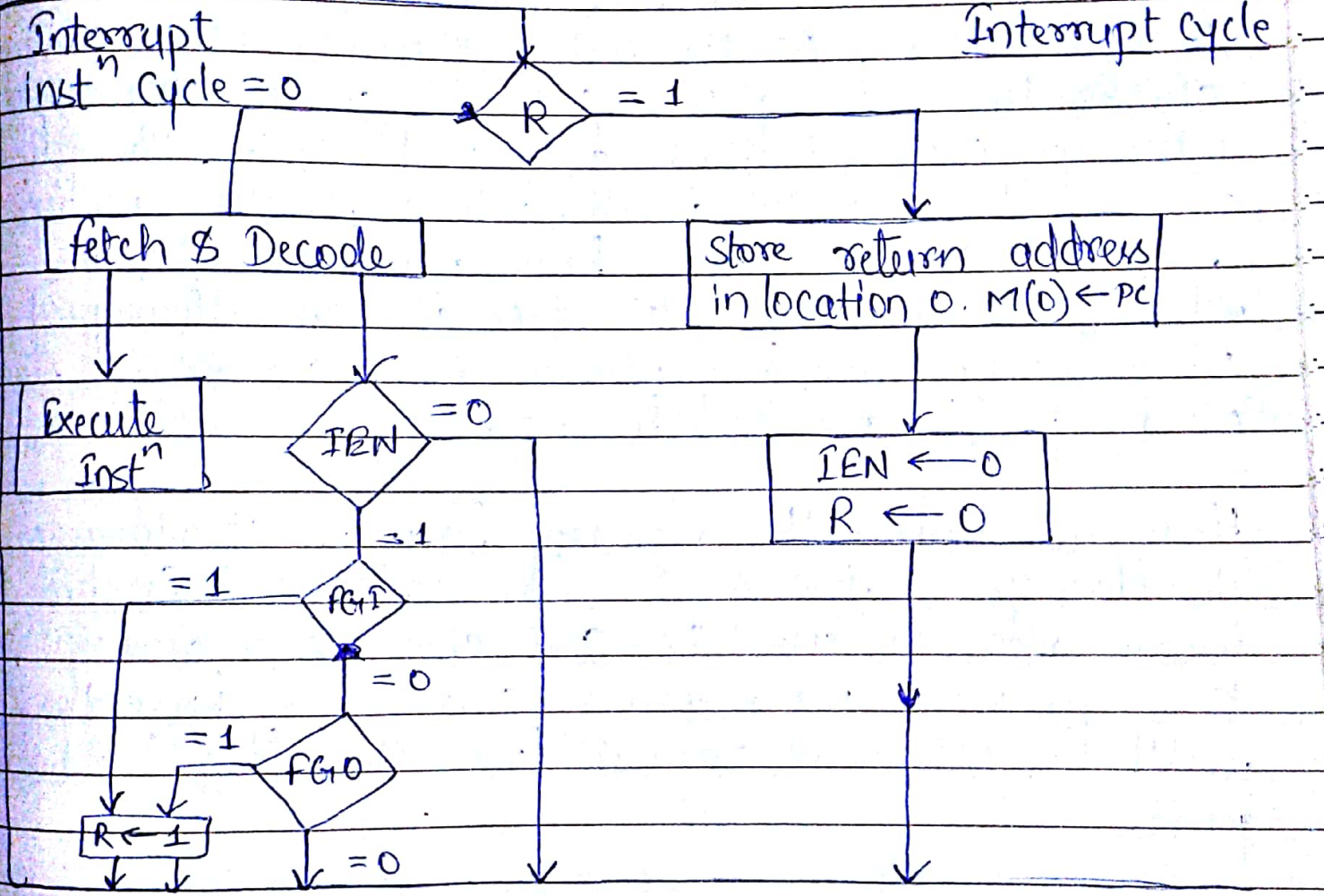
Computer registers & flip-flops





(I) The serial info. from the Keyboard is shifted into the input register INPR. The info. for the printer is stored in the outer reg. These two registers communicate with a comm^n interface serially and with the AC in parallel. The input reg. INPR consist of 8 bits and hold an input info. The one bit input flag FGI is a control flip-flop. The flag bit is set to 1 when ~~diff~~ new info. is available in the input device and is cleared to 0 when the info. is accepted by the computer. As long as the flag is set, the info. in INPR cannot be changed by striking another key. The computer checks the flag bit, if it is 1, the info. from INPR is transferred in parallel into AC and FGI flag is clear to 0. Once the flag is cleared new info. can be shifted into INPR. The outer reg. works similarly but the dir^n of info. flow is reversed. Initially the output flag FGO is set to 1. The Computer checks the flag bit, if it is 1, the info. from AC is transferred into OUTR and FGO is cleared to 0.

The output device accept the info., prints the corresponding character and when the operation is completed, it sets FGO to 1. The computer does not load a new character into outer when FGO is 0 b/s this cond. indicates that the output device is in process of printing the character.



(IEN → Interrupt enable flag)

When R = 0, the computer goes through the inst^n cycle. During the execution phase of the inst^n cycle, IEN is checked by the control if it is 0, it indicates

Date : Page :

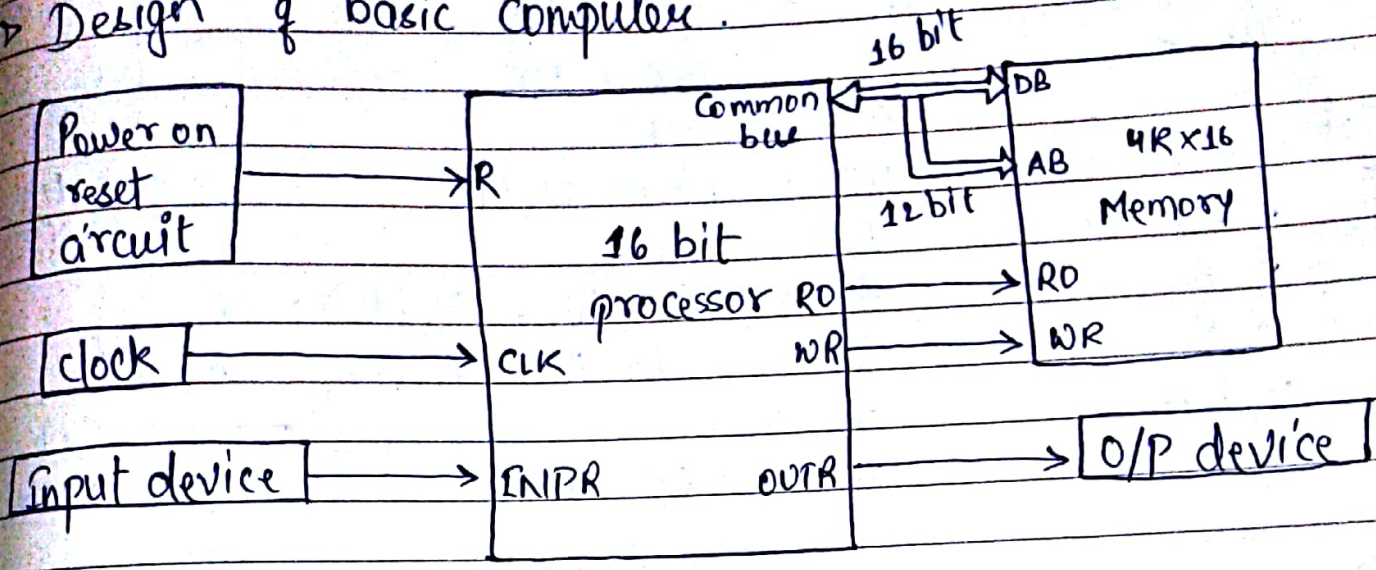
that the programmer does not want to use the interrupt so control continue with the next inst^{n} cycle. If IEN is 1, control checks the flagbit, if both flags are zero, it indicates that neither the input nor the output registers are ready for the transfer of info. In this case, control continue with the next inst^{n} cycle. If either flag is set to 1, while IEN=1, flipflop R is set to 1. At the end of execution phase, control checks the value of R, if it is 1 it goes through an interrupt cycle, instead of an inst^{n} cycle. The return add. available in PC is stored in a specific location where it can be found later when the program returns to the inst^{n} at which it was interrupted. This location may be processor reg., a memory stack or a specific memory location.

Here, we choose the memory location at address 0, as the place for storing the return address. Control then insert address 1 into PC and clears IEN and R so that no more interruption can occur from the flag until the interrupt request from the flag has been passed.

Date : Page :

(4K = 4096)

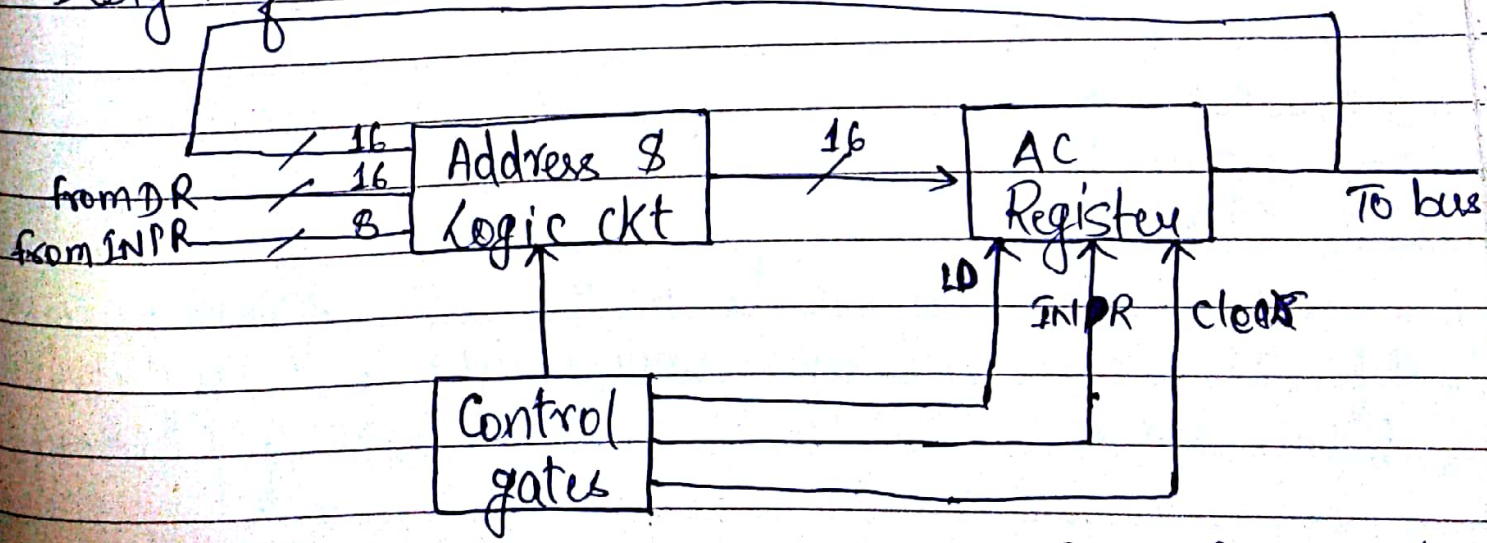
Design of basic computer.



(a) A memory unit with 4096 words of 16 bit each.
(b) 9 registers → PC, AR, INPR, OUTR, DR, AC, IR, MAR, TR
(c) 7 flipflops → I, S, E, R, IEN, FGI, FGO.
(d) 2 decoders → A 3 x 8 operation decoder and a 4 x 16 timing decoder.

(e) A 16 bit common bus, control logic gates, adder & logic ckt connected to the input of AC.

Design of Accumulator (AC) Logic.



INR → increment

Date : Page :

D_0T_5 : $AC \leftarrow AC \wedge DR$
D_1T_5 : $AC \leftarrow AC + DR$
D_2T_5 : $AC \leftarrow DR$
PB_{11} : $AC(0-7) \leftarrow INPR$
$r B_9$: $AC \leftarrow \overline{AC}$
$r B_7$: $AC \leftarrow shr\ AC,\ AC(15) \leftarrow E$
$r B_6$: $AC \leftarrow shl\ AC,\ AC(0) \leftarrow E$
$r B_{11}$: $AC \leftarrow 0$
$r B_5$: $AC \leftarrow AC + 1$

Load (LD)
CLR
INR

» Instruction format -

| Opcode | Mode | Address |

↳ Single Accumulator organization
↳ General register org.
↳ Stack org.

(1) ADD X
(2) ADD R_1, R_2, R_3
$R_1 \leftarrow R_2 + R_3$
ADD R_1, R_2
$R_1 \leftarrow R_1 + R_2$
MOV R_1, R_2
ADD R_1, X
(3) PUSH X

R_1, R_2, R_3 → registers
X → address
R_1 → Destination

The no. of address fields in the inst" format of a computer depends on the internal organization of its register.
Most computers fall into one of 3 types of CPU org".

Date : Page :

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

Solve. ① Three address inst^n

~~ADD X~~ ADD	R_1, A, B	$R_1 \leftarrow M[A] + M[B]$
~~ADD A, B~~ ADD	R_2, C, D	$R_2 \leftarrow M[C] + M[D]$
~~A $\leftarrow$ A * B~~ MUL	X, R_1, R_2	$M[X] \leftarrow R_1 * R_2$

② Two address inst^n

~~ADD A, B~~ ADD A, B	~~A $\leftarrow$ A + B~~ A $\leftarrow$ A + B
~~ADD C, D~~ ADD C, D	~~C $\leftarrow$ C + D~~ C $\leftarrow$ C + D
~~MUL~~ MUL	

② One address inst^n

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	$M[X] \leftarrow AC$

③ Two adddress inst^n

MOV	R_1, A	$R_1 \leftarrow M[A]$
ADD	R_1, B	$R_1 \leftarrow R_1 + M[B]$
MOV	R_2, C	$R_2 \leftarrow M[C]$
ADD	R_2, D	$R_2 \leftarrow R_2 + M[D]$
MUL	R_1, R_2	$R_1 \leftarrow R_1 * R_2$
MOV	X, R_1	$M[X] \leftarrow R_1$

Date : Page :

④ zero address inst^n (Stack org.)

PUSH A	TOS ← A
PUSH B	TOS ← B
ADD	TOS ← A+B
PUSH C	TOS ← C
PUSH D	TOS ← D
ADD	TOS ← C+D
MUL	TOS ← (C+D)* (A+B)
POP X	M[X] ← TOS

~~Imp~~

Addressing modes –

opcode	mode	address

The addressing mode specifies a rule for interpreting the address field of the inst^n before the operand is actually referred.

➢ Adv. of addressing modes →

(i) To reduce the no. of bits in the address field of the inst^n.

(ii) To provide programming versatility to the user by providing such facilities as pointers to memory, indexing of data and progeam relocation.

(a) Implied mode ⇒ In this mode the operands are specified implicitly in the def^n of the inst^n.

Ex:- Complement of AC → $\overline{AC}$

(b) Immediate mode ⇒ The operand is specified in the inst^n itself.

Date : Page :

(c) Register mode ⇒ Operands are in the registers that resides within the CPU.

(d) Register Indirect ⇒ Inst^n Register specifies a register in the CPU whose contents give the address of the operand in the memory.

(e) Auto increment or Auto decrement ⇒
It is similar to reg. indirect mode except that the register is incremented or decremented after or before its value is used.

(f) Direct modes ⇒ Operand resides in the memory and its address is given directly by the add. field of the inst^n.

(g) Indirect add ⇒ The address field of the inst^n gives the address where the effective add. is stored in memory. A few addressing modes require that the add. field of the inst^n be added to the content of a specific reg. in the CPU.

Effective add. = add. part of the inst^n + content of the CPU reg.

(h) Relative add. ⇒ PC + add. part of the inst^n
(i) Indexed add. ⇒ Index reg. + add. part of inst^n reg
(j) Base register ⇒ add. part + reg.

Date : Page :

Ex:- 2 word inst^{n} at address 200, 201 is a load to AC, address will contain 500

PC = 200

R = 400

XR = 100 → Indexed reg.

AC

Memory

200	Load to Ac mode
201	Add = 500
202	Next Inst^{n}
399	450
400	700
500	800
600	900
702	325
800	300

	Add mode	Effective Add.	Content of AC
1)	Direct	500	800
2)	Immediate	201	500
3)	Indirect	800	300
4)	Relative	702	325
5)	Indexed	600	900
6)	Register	—	400
7)	Register Indirect	400	700
8)	Auto increment	400	700
9)	Auto decrement	399	450

Date : Page :

Data Transfer and manipulation.

Data transfer		Data manipulation	
Load	LD	(1) Arithmetic Inst^n	
STORE	ST	Increment	INC
MOVE	MOV	Decrement	DEC
EXCHANGE	XCH	Add	ADD
Input	In	Subtract	SUB
Output	OUT	Multiply	MUL
Push	PUSH	Divide	DIV
Pop	POP	Add with carry	ADD -C
		Subtract with borrow	SUB -B

(2) Logical & bit manipulation

Clear	CLR
Complement	COM
And	AND
Or	OR
Exclusive-OR	XOR
Clear carry	CLR C
Set carry	SET C
Complementary	COM C
Enable interrupt	EI

(3) Shift Inst^n

Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHRL

Date : Page :

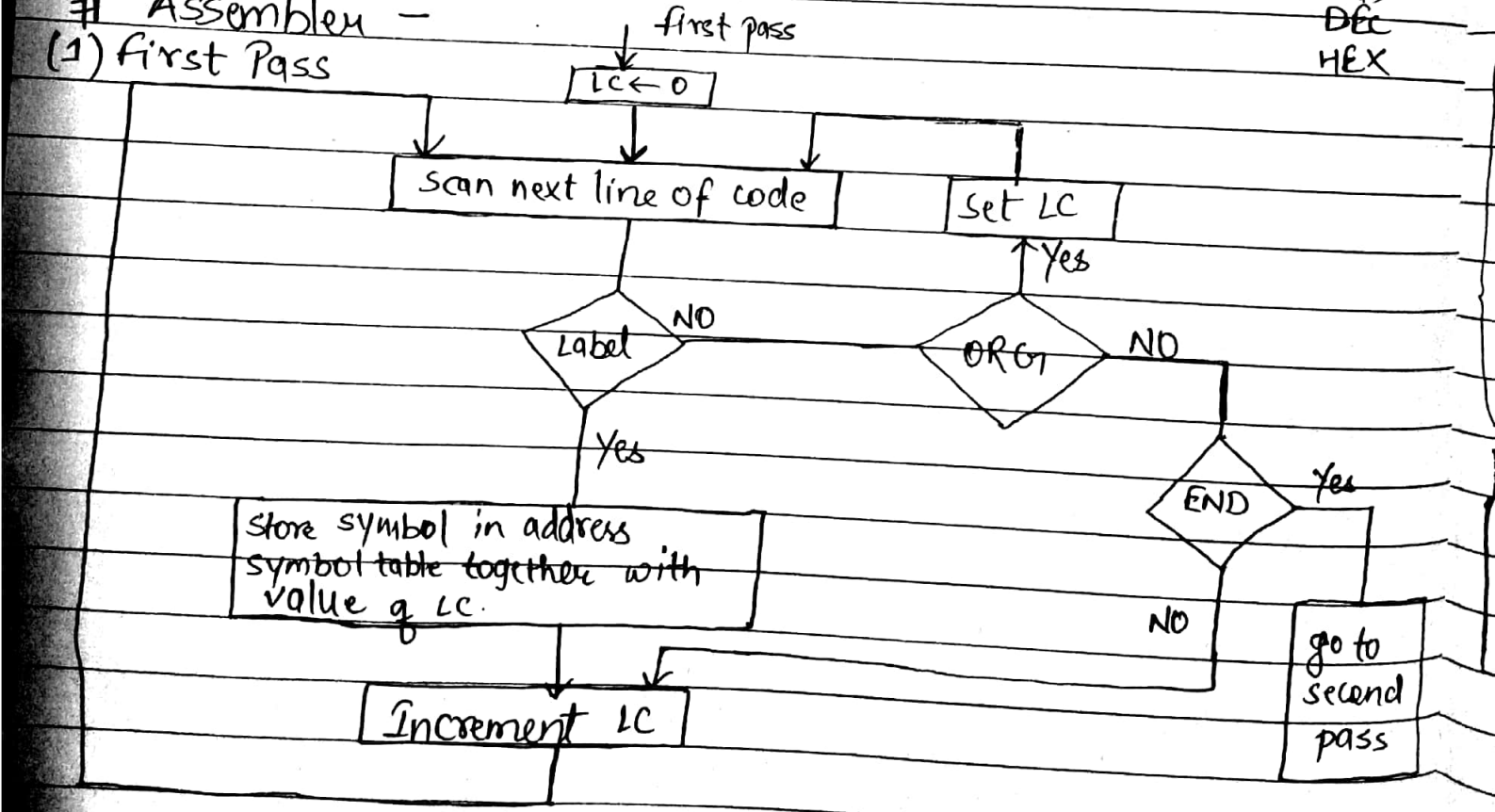
Rotate right ROR
Rotate left ROL
Rotate right through Carry ROR C
Rotate left through carry ROL C

- Data Transfer inst^n move data from one place in computer to another without changing the data content. The most common transfers are b/w memory and processor registers, b/w processor registers and input/output and b/w the processor registers.
- Data manipulation inst^n are those that perform arithmetic, logic and shift op^n.

ORG
END
DEC
HEX

Assembler –

(1) First Pass

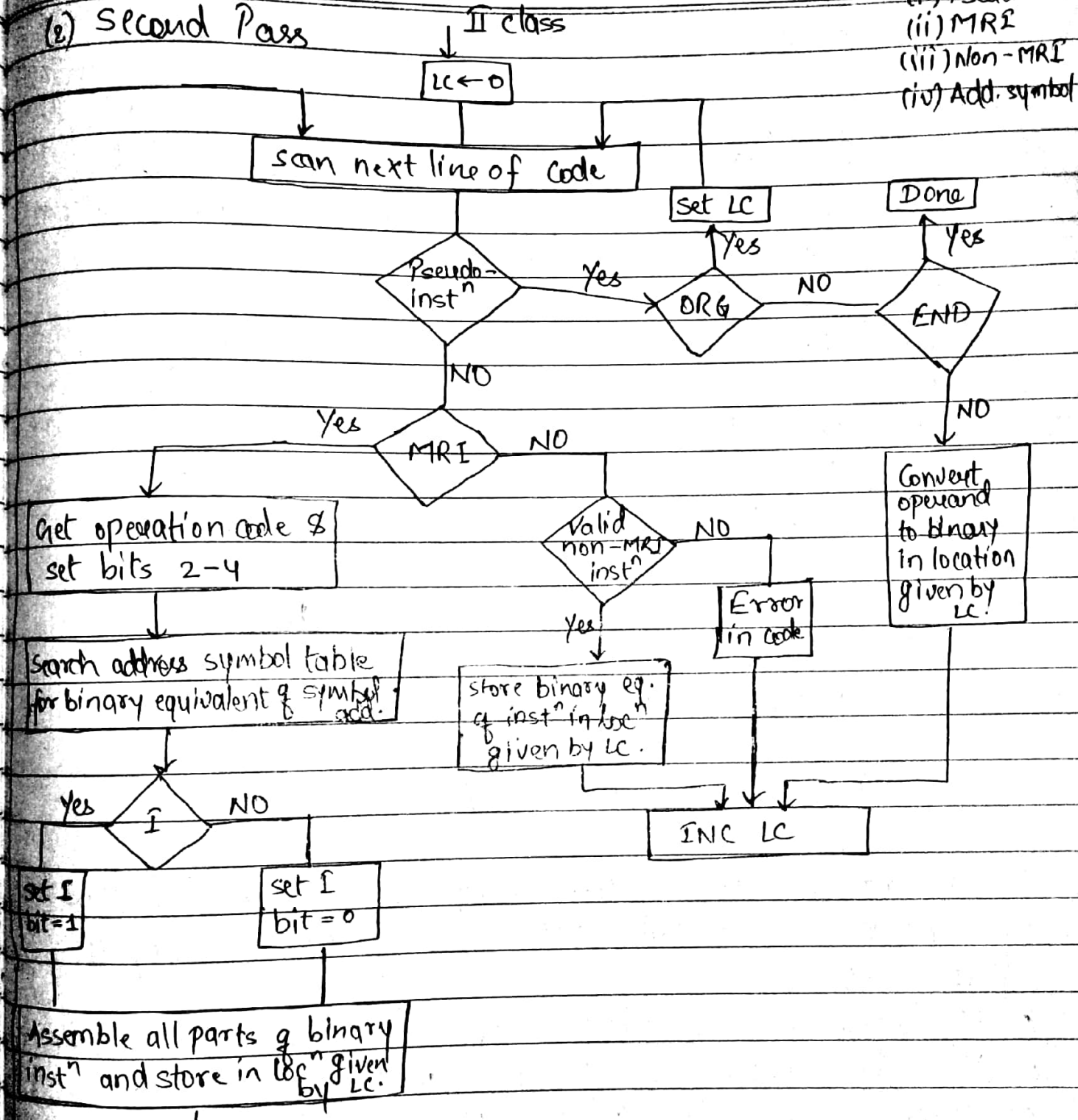


Location counter (LC)
origin (ORG)

Date : Page :

(i) Pseudo
(ii) MRI
(iii) Non-MRI
(iv) Add. symbol

(2) Second Pass



* Program for division

Date : Page :

```
    LDA   0002H ;
    MOV   C,A
    LDA   0003H                 if |A| < reg | c=1
    MVI   B, 00H                if |A| = reg | z=1
L2: CMP   C                     if |A| > reg | CS & Z are reset
    JC    L1 ———— when c=1
    SUB   C
    INR   B
    JMP   L2
L1: STA   0003H
    MOV   A, B
    STA   0004H
```

Inst^n size — for 8085

(1) 1-Byte inst^n

Ex:- MOV C, A
(MOV: 4 bits, C, A: 4 bits)

ADD, SUB, INR, DCR

Data transfer inst^n that copy the content from one reg. or memory into another reg. or memory are 1-byte inst^n.
A 1-byte inst^n includes the opcode and the operand in the same byte. Arithmetic and logic inst^n without ending with letter 'I' are 1-byte inst^n.

(2) 2-Byte inst^n

In this, the first byte specifies the opcode and the second byte specifies the operand.

Date : Page :

Inst^{n} that load or manipulate 8 bit data directly are 2-byte inst^{n}.
Instructions ending with letter 'I' except LXI are called 2-byte inst^{n}.

Ex:- MVI (8 bit) A, Data (8 bit)

ADI, ANI, ORI

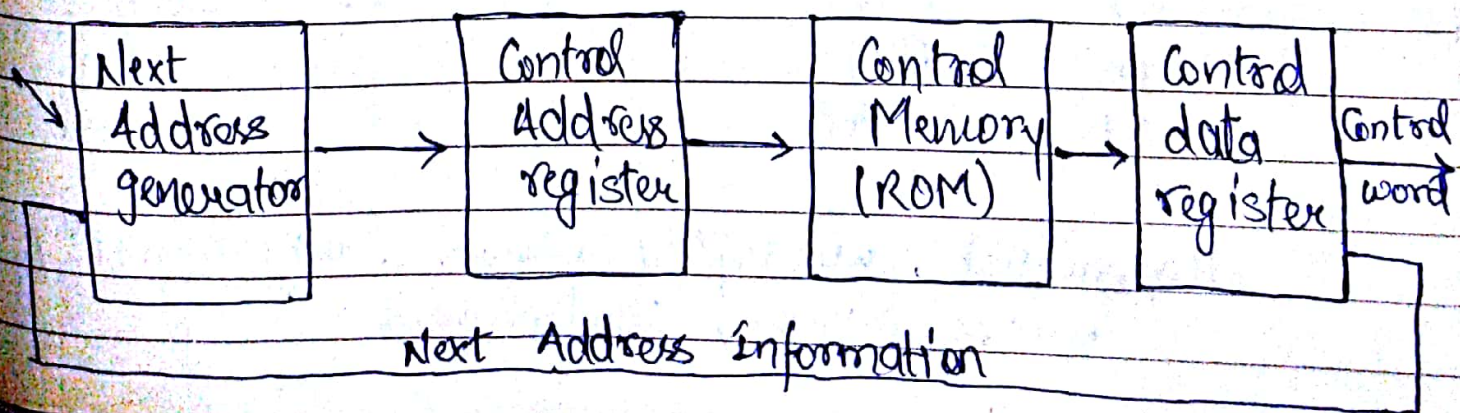
(3) 3-Byte inst^{n}

Inst^{n} that loads 16 bits or refer to memory add. are 3-byte inst^{n}. In 3-byte inst^{n}, the first byte specify the opcode and the following 2 bytes specify the 16 bit address. Second byte is the lower order add. and the third byte is higher order address.

Ex:- JMP (8 bits) 2085 H (8 bits)

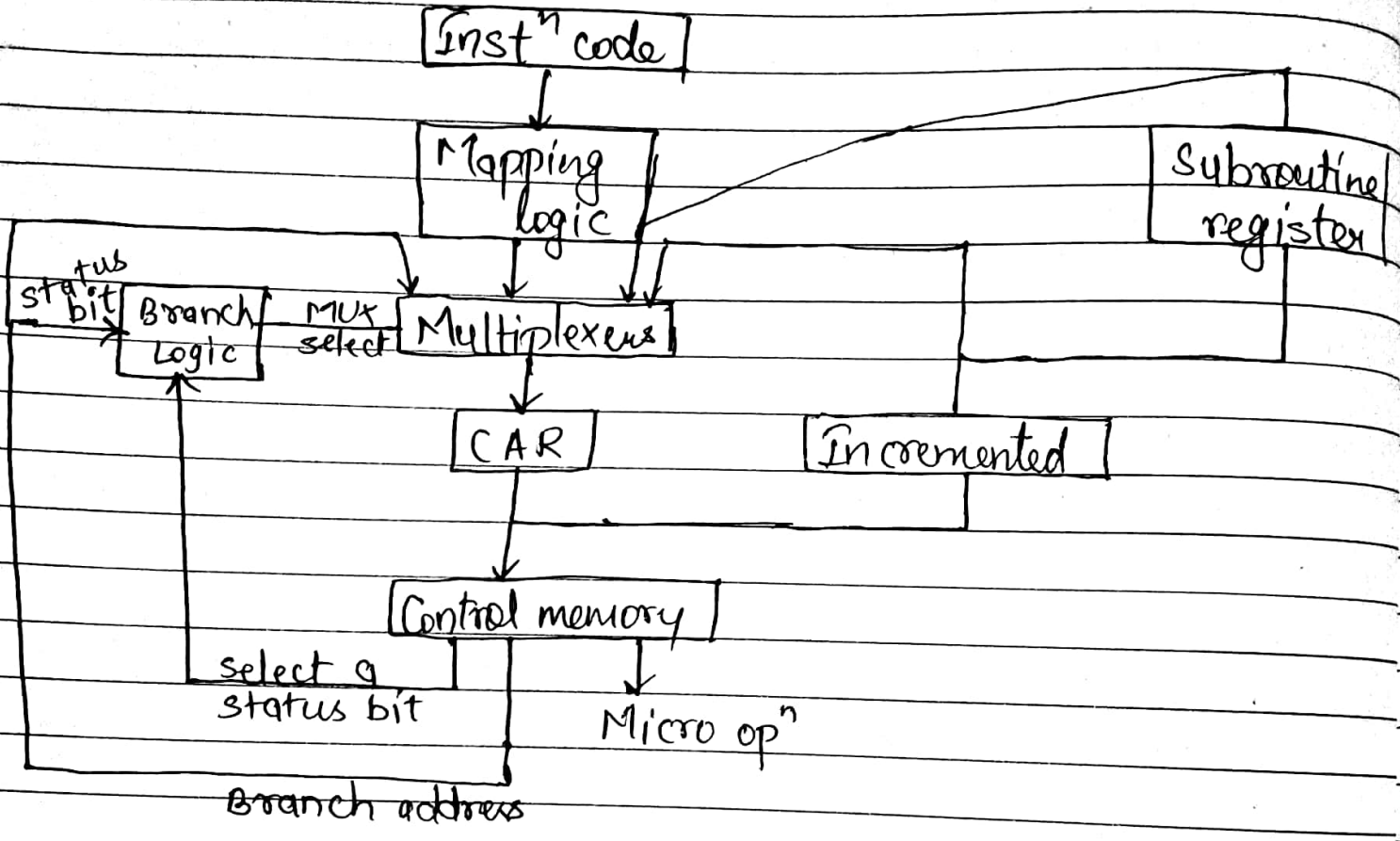
LXI, JMP, Conditional jump, CALL.

Microprogrammed Control Organization –



Date : Page :

Address sequencing -



Micro-inst^n format

f1	f2	f3	CD	BR	AD
3	3	3	2	2	7

20 bits

CD

00 - unconditional
01 - Indirect add.
10 - sign bit of AC.
11 - Zero

BR

00	JMP
01	CALL
10	RET
11	MAP

Hardwired, microprogrammed, micro-inst^n, microprogram, control word