Unit - 3

1) Instruction Cycle and Subcycle (Fetch & Execute etc) - Sequence of instructions

2) Micro-operations - All the micro-operations with example.

3) Execution of complete instructions - Sequence of execution of instructions

4) Reduced Instruction Set Computer, Pipelining - Description of RISC and CISC.

5) Hardwired and Microprogrammed control.
- Diagram and description of Hardwired and MP control.

6) Hardwired and Microprogrammed control
- Diagram and description of Hardwired and MP control.

7) Microprogram sequencing - Writing sequence of inst^{n}.

8) Concepts of Horizontal and Vertical Programming - Writing Microprogram.

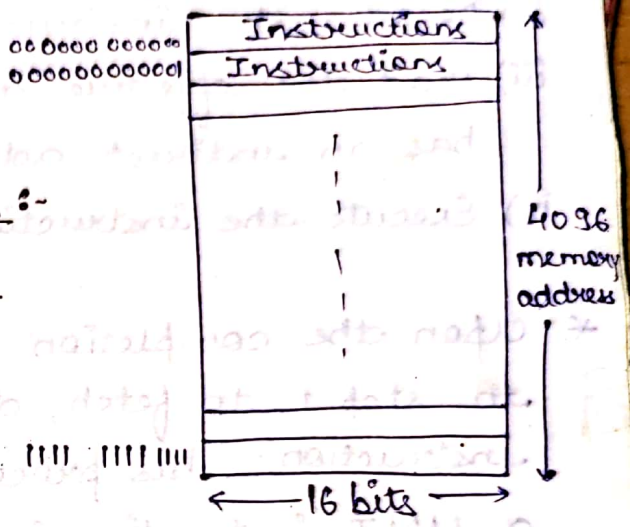
UNIT-3

CONTROL UNIT

Instruction Code :- An instruction code is a group of bits that instruct the computer to perform a specific operation.

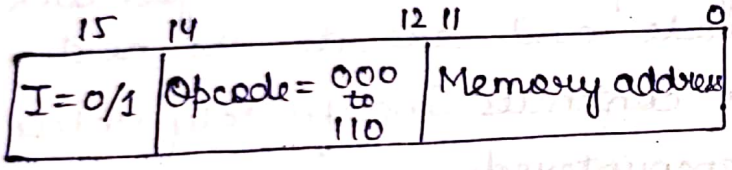
Assume a memory system of size 4096×16
$= 2^{12} \times 16$

No. of address bits = 12 bits
No. of data bits = 16 bits

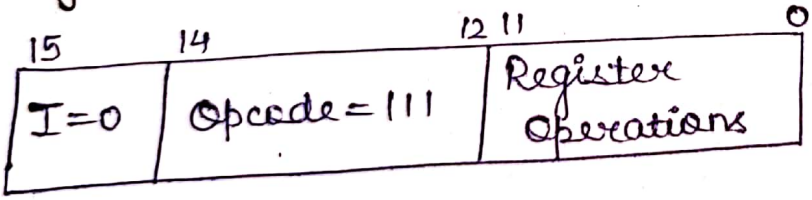


Types of Instruction Code Formats :-

1) Memory Reference Instruction :-

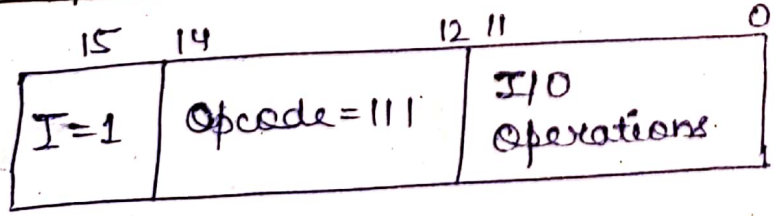


2) Register Reference Instruction :-



→ If opcode = 111
- If I=0; then register (AC) reference.
- If I=1; then I/O instruction

3) Input - Output Instruction :-



→ Else, the instruction is memory reference.
- If I=0; then direct addressing mode.
- If I=1; then indirect addressing mode.

Instruction Cycle :-

- A program residing in the memory unit of the computer consists of a sequence of instructions.
- The program is executed in the computer by going through a cycle for each instruction.
- Each instruction cycle in turn is sub-divided into a sequence of subcycles or phases.
- Each instruction cycle consists of the following phases:-

i) Fetch - an instruction from memory

ii) Decode - the instruction

iii) Read the effective address from memory if the instruction has an indirect address.

iv) Execute the instruction.

* Upon the completion of step 4, the control goes back to step 1 to fetch, decode and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Instruction Set Completeness :-

- A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function that is known to be computable.
- The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:-

1) Arithmetic, logical and shift instructions :-
   - ADD, CMA, INC, CIR, CIL, AND, CLA

2) Instructions for moving information to and from memory and processor registers.
   - LDA, STA

3) Program control instructions together with instructions that check status conditions :-
   - BUN, BSA, ISZ

4) Input and output instructions
   - INP, OUT

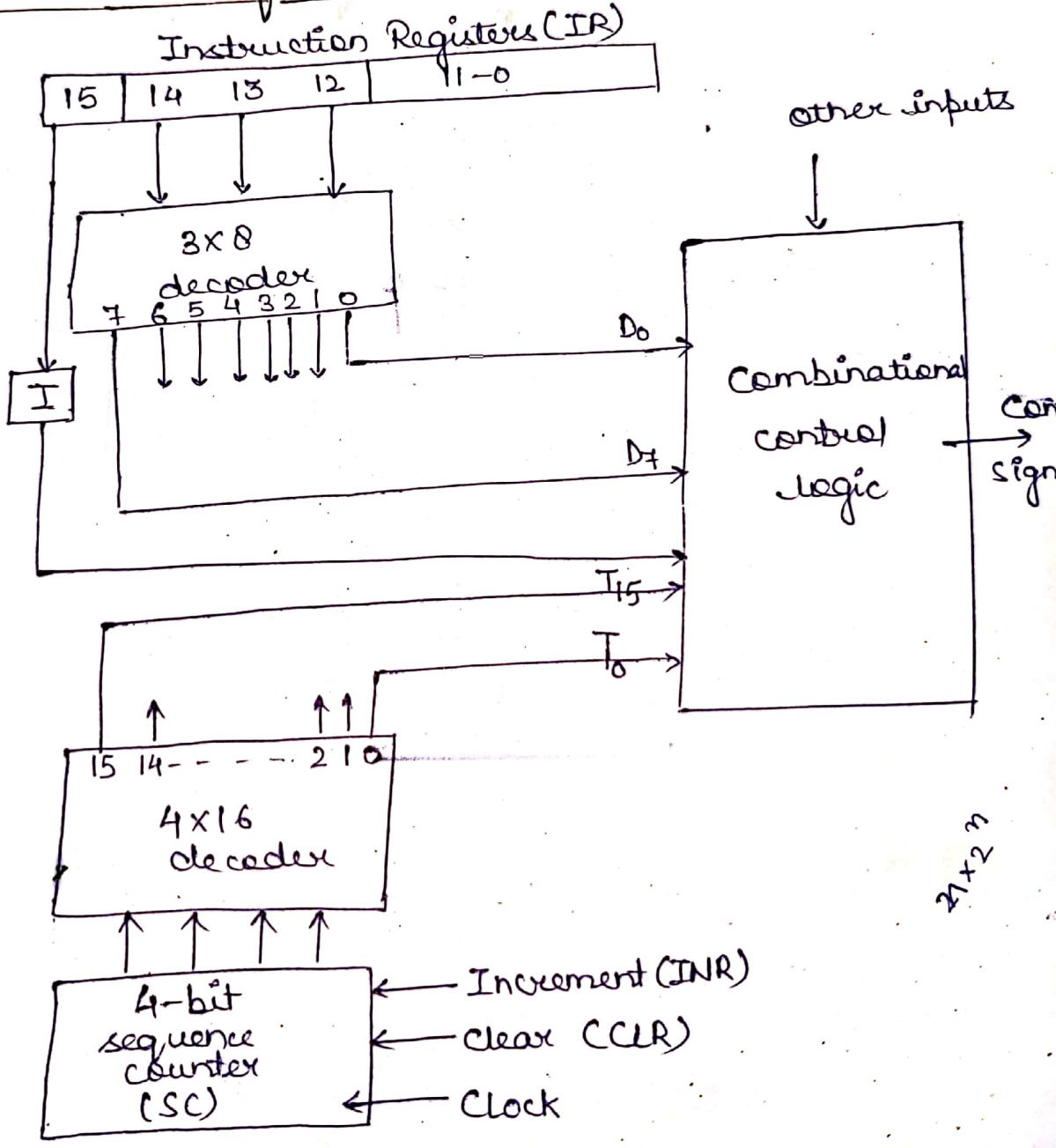
** Hardwired control unit :-

- implemented as logical circuit (gates, flip-flops etc) in the hardware
- faster
- used in RISC
- can't handle complex instructions
- modification is difficult

** Microprogrammed control unit :-

- implemented using programming approach
  - Micro-program, consisting of micro-instructions is stored in the control memory of the control unit.
  - Execution of a micro-instruction is responsible for generation of set of control signals.
- slower
- used in CISC
- easier to handle complete instruction sets
- cheaper and more flexible

Control unit of Basic Computer :-

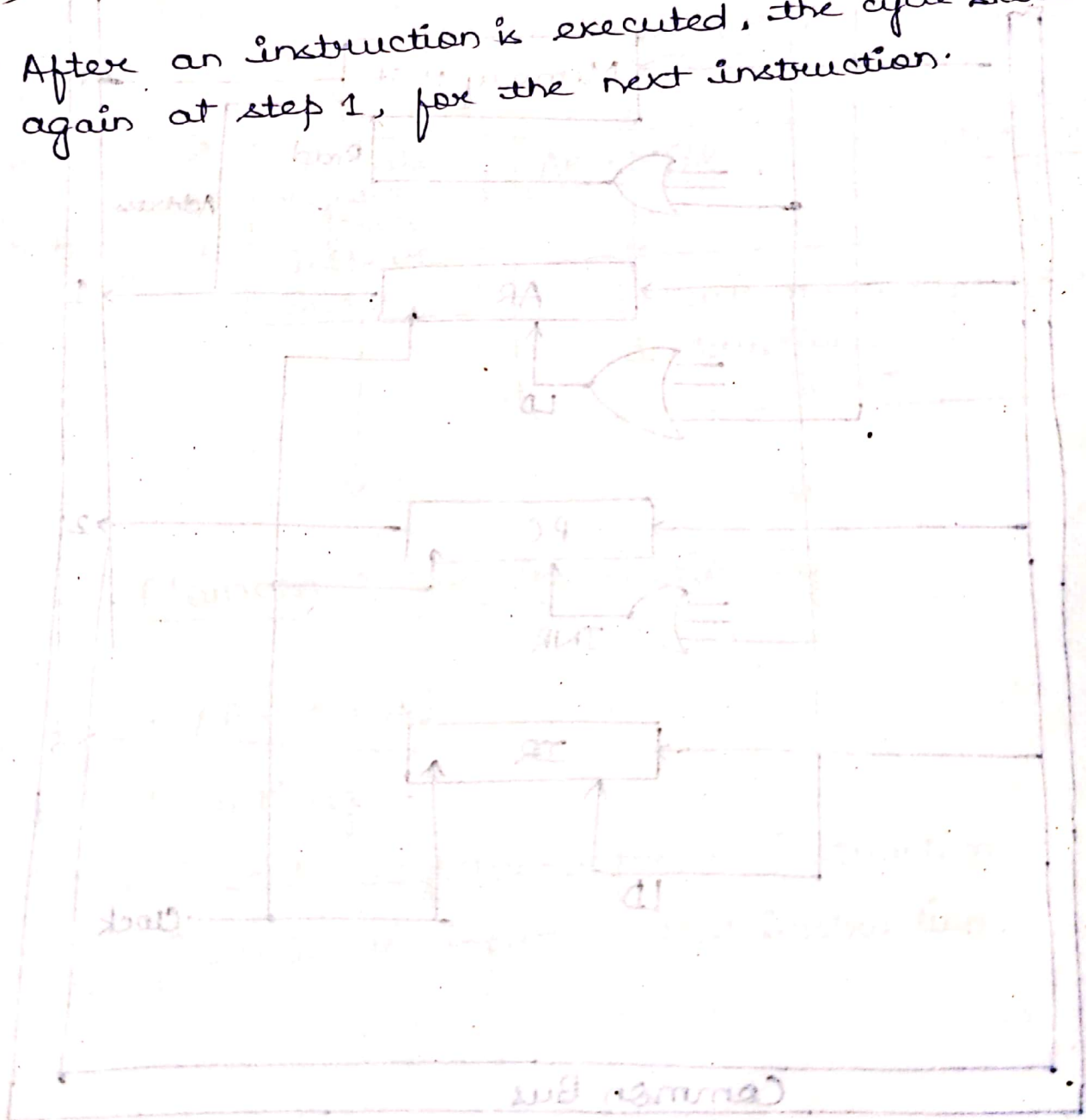


Instruction Cycle :-

- In basic computer, a machine instruction is executed in the following cycle:

1) Fetch an instruction from memory
2) Decode the instruction and calculate effective address (EA)
3) Fetch operand :: Read the EA from memory if the instruction has an indirect address.
4) Execute the instruction

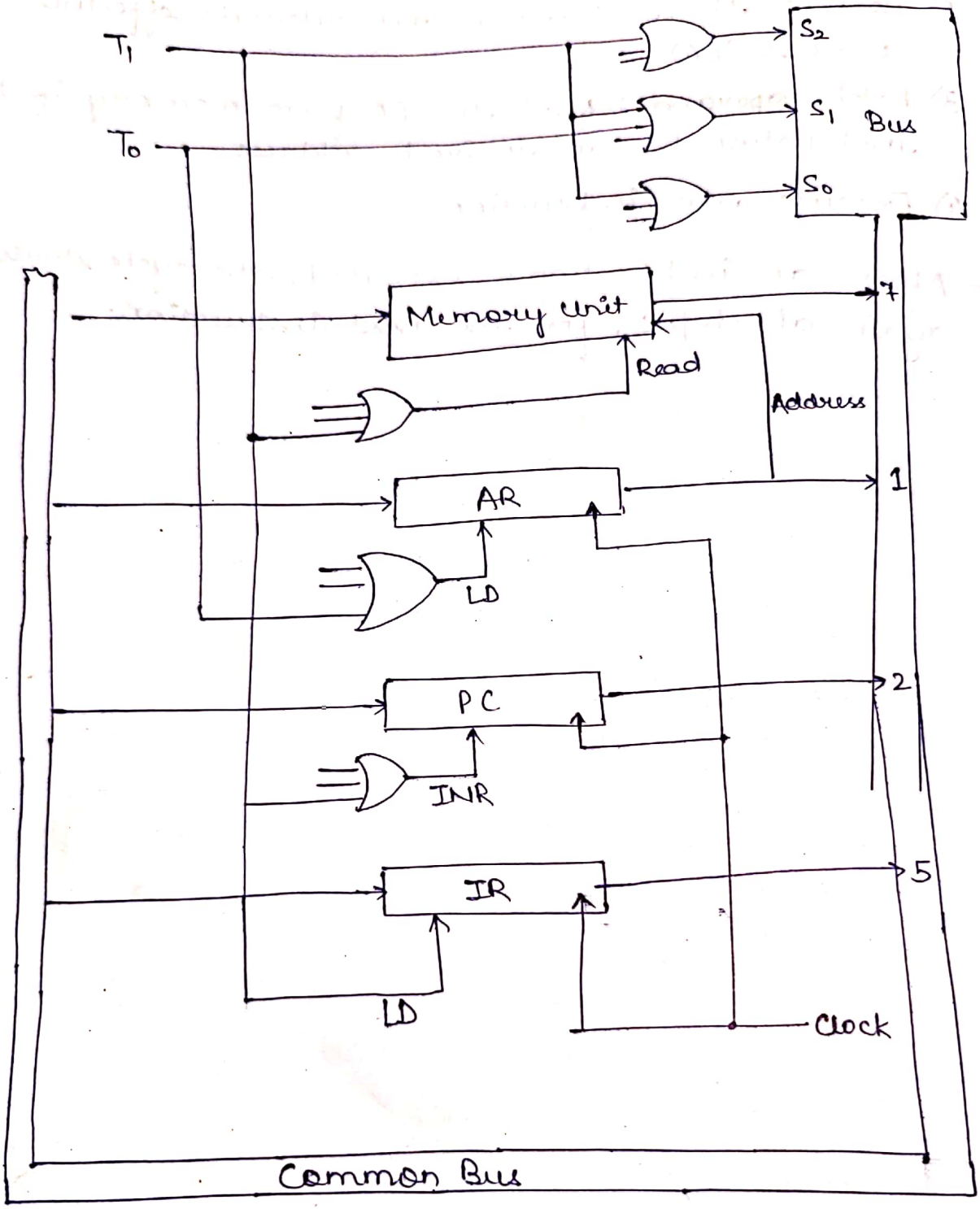
- After an instruction is executed, the cycle starts again at step 1, for the next instruction.



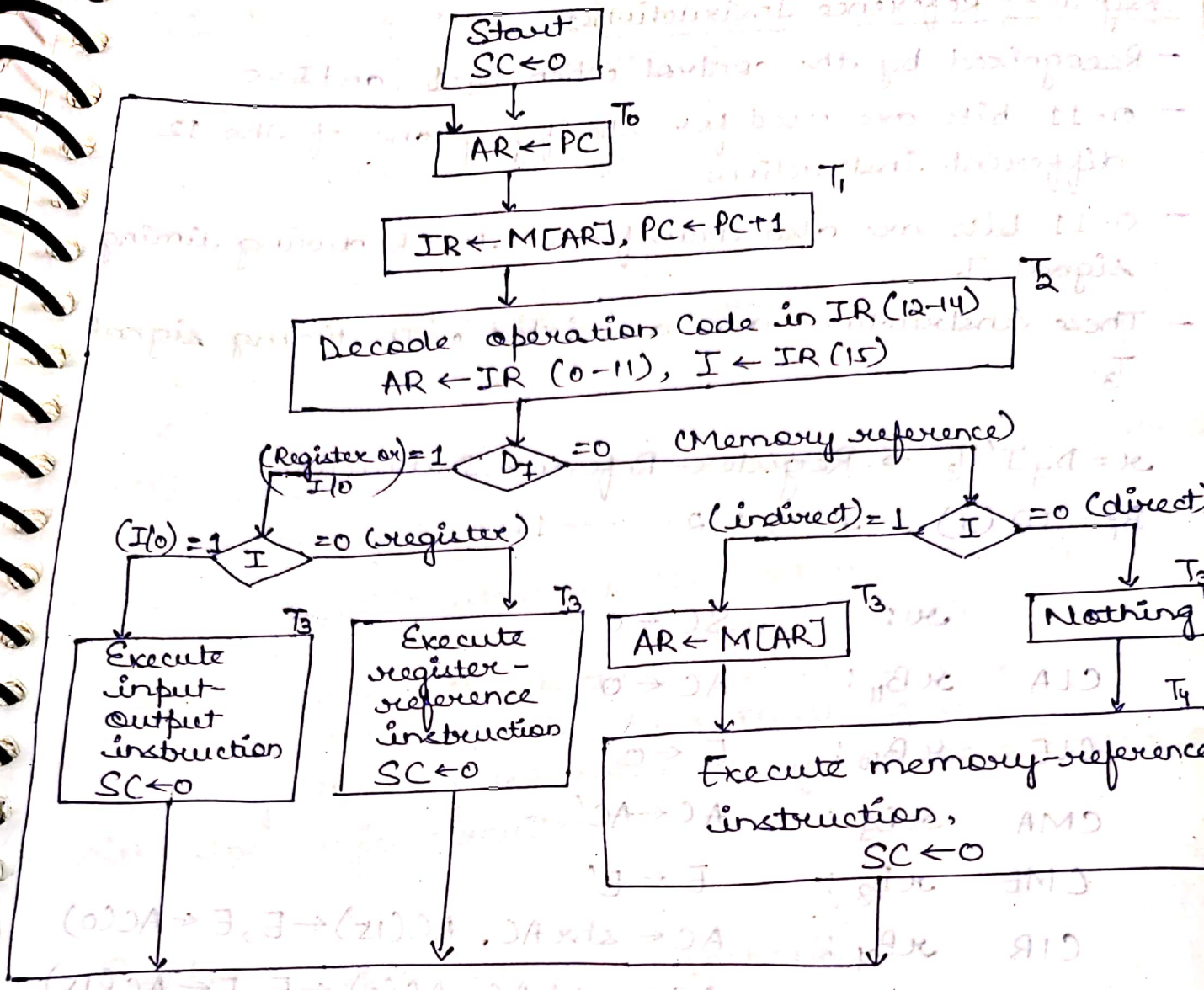
T_0 : AR ← PC ($S_0S_1S_2 = 010$, $T_0 = 1$)

T_1 : IR ← M[AR], PC ← PC + 1 ($S_0S_1S_2 = 111$, $T_1 = 1$)

T_2 : $D_0, \ldots, D_7$ ← Decode IR (12-14), AR ← IR (0-11), I ← IR (15)



Register transfers for fetch phase



Flowchart for Instruction Cycle

$D_7' I T_3$: $AR \leftarrow M[AR]$

$D_7' I' T_3$: Nothing

$D_7 I' T_3$: Execute register-reference instruction

$D_7 I T_3$: Execute an input-output instruction

Register Reference Instructions:-

- Recognized by the control when $D_7 = 1$, and $I = 0$
- 0-11 bits are used for specifying one of the 12 different instructions
- 0-11 bits are also transferred to AR during timing signal T_2.
- These instructions are associated with timing signal T_3.

$r = D_7 I' T_3 \Rightarrow$ Register-Reference Instruction

$B_i = IR(i)$, $i = 0, 1, 2 \cdots 11$

	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	if $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3:	if $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2:	if $(AC = 0)$ then $(PC \leftarrow PC + 1)$
SZE	rB_1:	if $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$ (S is a start-stop flip flop).

Memory Reference Instructions :-

- For memory reference instructions the decoder output is D_i for $i = 0, 1, 2, 3, 4, 5, 6$ from operation decoder.
- So there are seven different memory reference instructions
- The effective address of the instruction is in the address register AR and was placed there during timing signal T_2 when $i=0$, or during timing signal T_3 when $i=1$.
- The execution of memory-reference instructions starts with timing signal T_4.
- The actual execution of the instruction in the bus system will require a sequence of micro-operations. This is because the data stored in memory cannot be processed directly. The data must be read from memory to register where they can be operated on with the logic circuits

Symbol	Operation Decoder	Symbolic Description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], E \leftarrow Cout$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, PC \leftarrow AR+1$
ISZ (Increment & Skip if 0).	D_6	$M[AR] \leftarrow M[AR]+1$, if $M[AR]+1=0$ then $PC \leftarrow PC+1$

AND to AC

$D_0 T_4$: $DR \leftarrow M[AR]$ Read operand

$D_0 T_5$: $AC \leftarrow AC \wedge DR, SC \leftarrow 0$ AND with AC

ADD to AC

$D_1 T_4$: $DR \leftarrow M[AR]$ Read operand

$D_1 T_5$: $AC \leftarrow AC + DR, E \leftarrow Cout, SC \leftarrow 0$ ADD to AC and store carry in E

LDA : Load to AC

$D_2 T_4$: $DR \leftarrow M[AR]$

$D_2 T_5$: $AC \leftarrow DR, SC \leftarrow 0$

STA : Store AC

$D_3 T_4$: $M[AR] \leftarrow AC, SC \leftarrow 0$

BUN : Branch unconditionally

$D_4 T_4$: $PC \leftarrow AR, SC \leftarrow 0$

BSA : Branch and Save Return Address

$D_5 T_4$: $M[AR] \leftarrow PC, AR \leftarrow AR + 1$

$D_5 T_5$: $PC \leftarrow AR, SC \leftarrow 0$

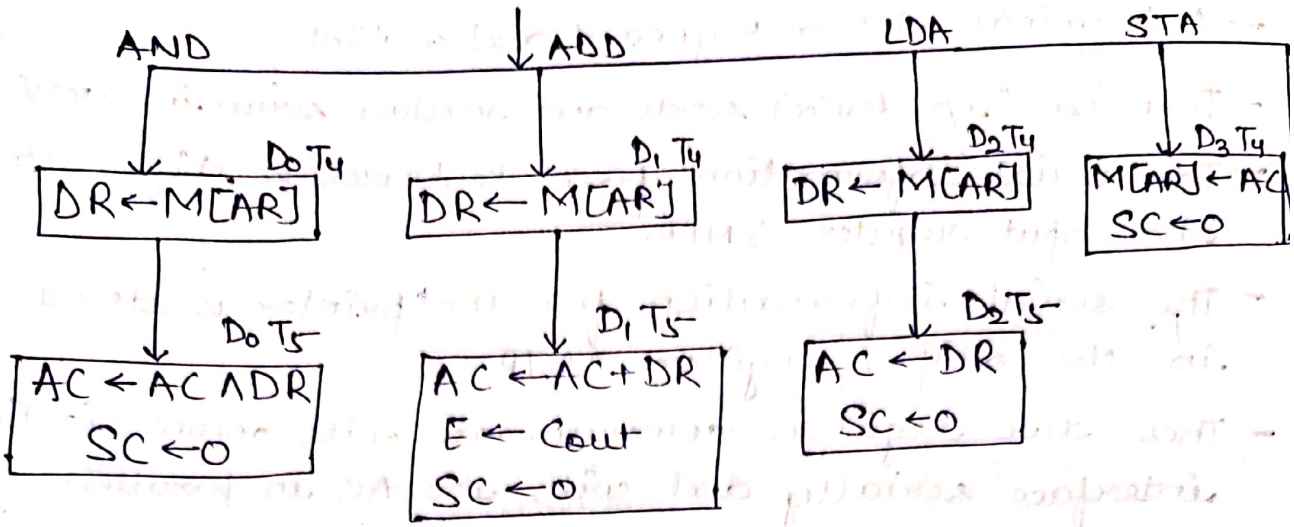
ISZ : Increment and Skip-if-Zero

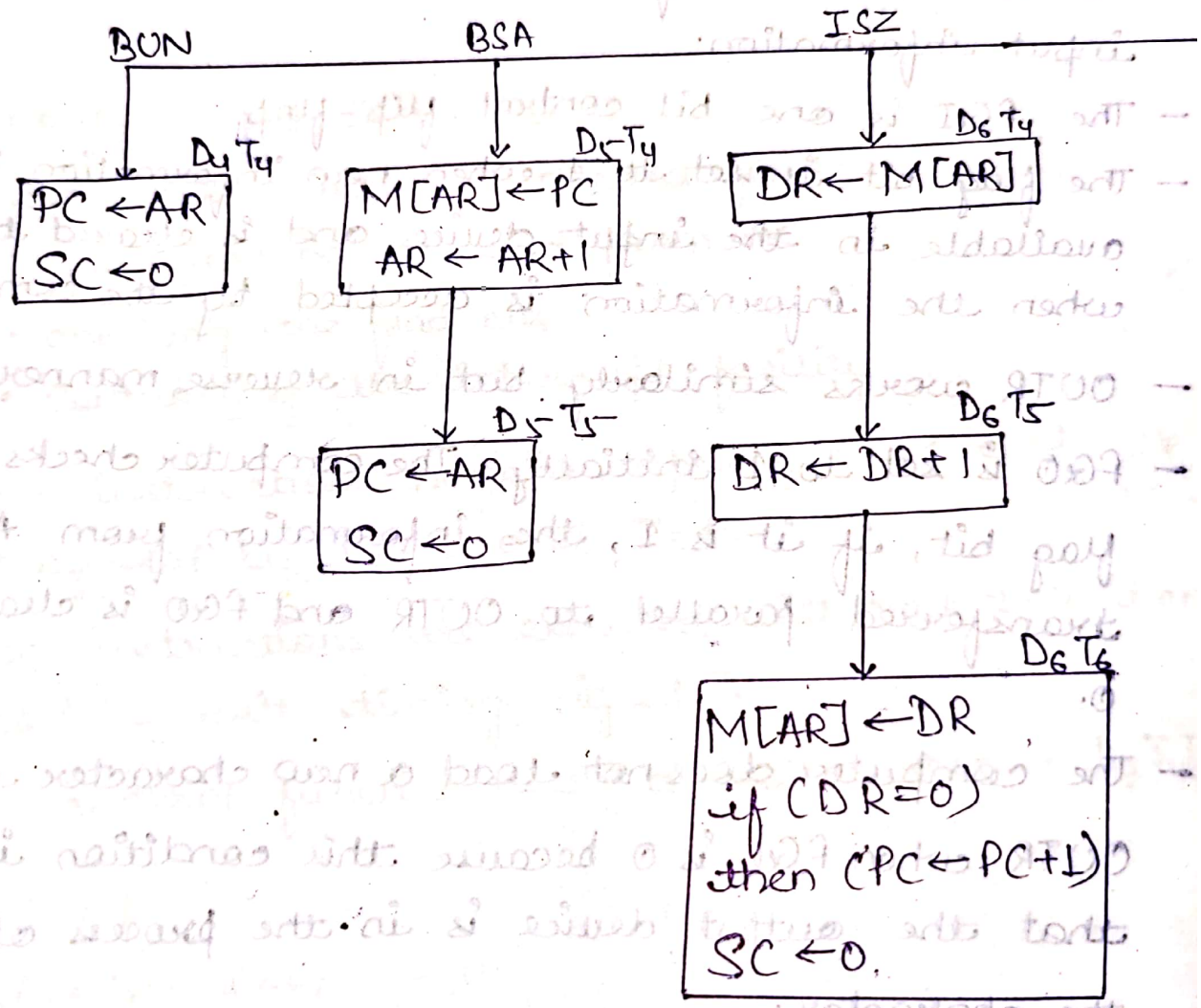
$D_6 T_4$: $DR \leftarrow M[AR]$

$D_6 T_5$: $DR \leftarrow DR + 1$

$D_6 T_6$: $M[AR] \leftarrow DR$, if $(DR = 0)$ then $PC \leftarrow PC + 1, SC \leftarrow 0$

Flowchart for Memory-reference instructions :





Input - Output Reference :-

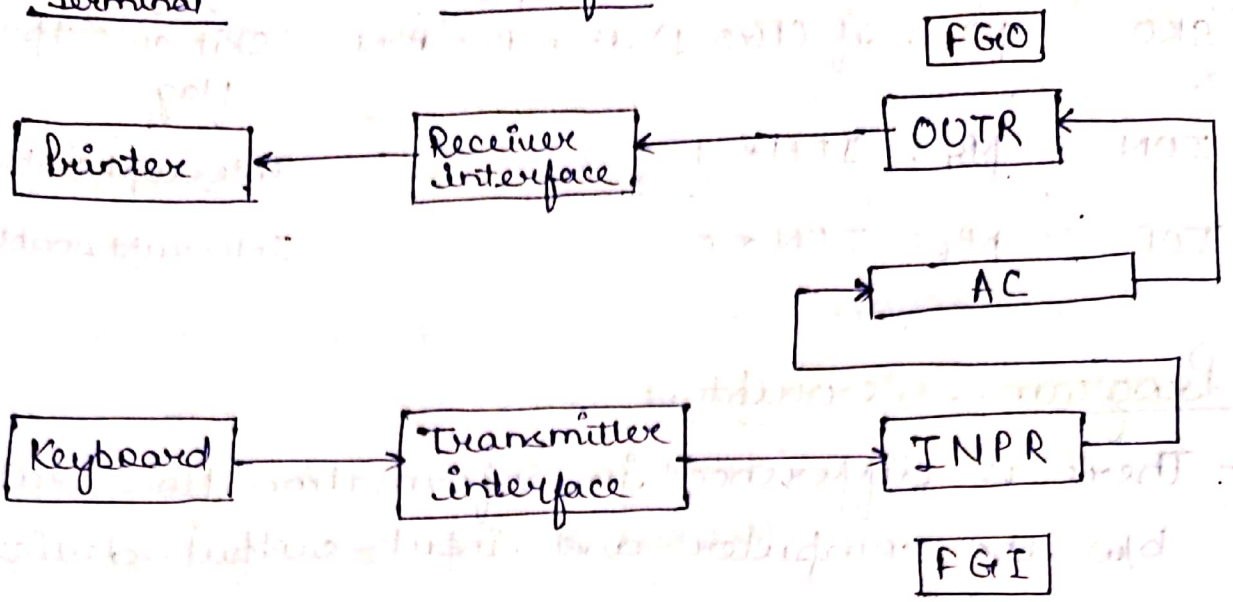
- A terminal with a keyboard and a printer.
- Terminal (i/o device) sends and receives serial information.
- The serial information from keyboard is shifted into the input register INPR.
- The serial information for the printer is stored in the output register OUTR.
- These two registers communicate with communication interface serially and with the AC in parallel.
- The INPR consists of 8 bits and holds alphanumeric input information.
- The FGI is one bit control flip-flop.
- The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.
- OUTR works similarly but in reverse manner.
- FGO is set to 1 initially. The computer checks the flag bit, if it is 1, the information from AC is transferred parallel to OUTR and FGO is cleared to 0.
- The computer does not load a new character into OUTR when FGO is 0 because this condition indicates that the output device is in the process of printing the character.

- Input - Output configuration :-

Input - Output terminal

Serial communication interface

computer registers and flip-flops



- Input - Output Instructions :-

- These instructions are needed for :-
  - transferring to and from AC
  - checking the flag bits
  - Controlling the interrupt facility
- These instructions have an opcode 1111 and recognized by $D_7 = 1$ and $I = 1$.
- These instructions are executed with clock transition associated with timing signal T_3.
- Each control function needs a Boolean relation $D_7 I T_3$

$D_7 I T_3 = p$

$IR(i) = B_i, \quad i = 6, \ldots, 11$

	p:	$SC \leftarrow 0$	Clear SC
INP	$p B_{11}$:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$.	Input char. to AC
OUT	$p B_{10}$:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$	Output char. from

SKI	pB_9 : if (FGI = 1) then PC ← PC + 1	Skip on input flag
SKO	pB_8 : if (FGO = 1) then PC ← PC + 1	Skip on output flag
ION	pB_7 : IEN ← 1	Interrupt enable ON
IOF	pB_6 : IEN ← 0	Interrupt enable OFF

Program Interrupt

- There is difference in information flow rate b/w the computer and input-output device.
- Because of this difference programmed control transfer becomes inefficient.
- An alternative to programmed control transfer is to let the external device inform the computer when it is ready for the transfer.
- In the meantime the computer can be busy with other tasks. This type of transfer uses interrupt facility.
- While the computer is running a program, it does not check the flags. However when the flag is set, the computer is momentarily interrupt from proceeding with the current program and is informed of the fact that flag has been set.

IEN (Interrupt Enable Flip-flop)

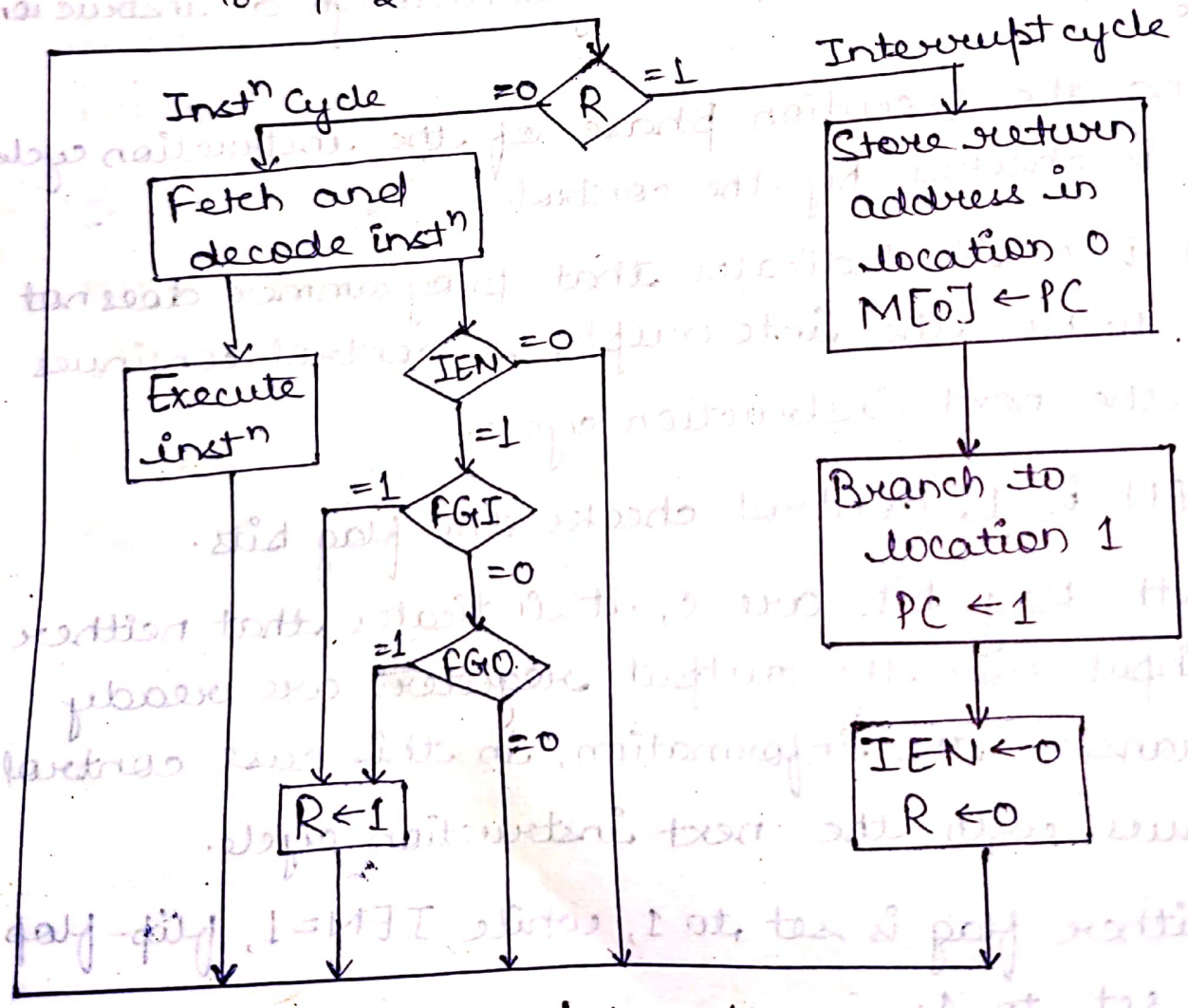
- IEN can be set and cleared with two instructions
- IOf instruction clears the IEN to 0, the flags cannot interrupt the computer.
- ION instruction sets the IEN to 1, the computer can be interrupted now.
- These two instructions provide the programmer with the capability of making a decision as to whether or not to use the interrupt facility.

Interrupt Cycle :-

- An interrupt flip-flop R is included in the computer.
- When R=0, the computer goes through an instruction cycle.
- During the execution phase of the instruction cycle IEN is checked by the control.
- If it is 0, it indicates that programmer does not want to use the interrupt, so control continues with the next instruction cycle.
- If IEN is 1, control checks the flag bits.
- If both flag bits are 0, it indicates that neither the input nor the output registers are ready to transfer the information, in this case control continues with the next instruction cycle.
- If either flag is set to 1, while IEN=1, flip-flop R is set to 1.

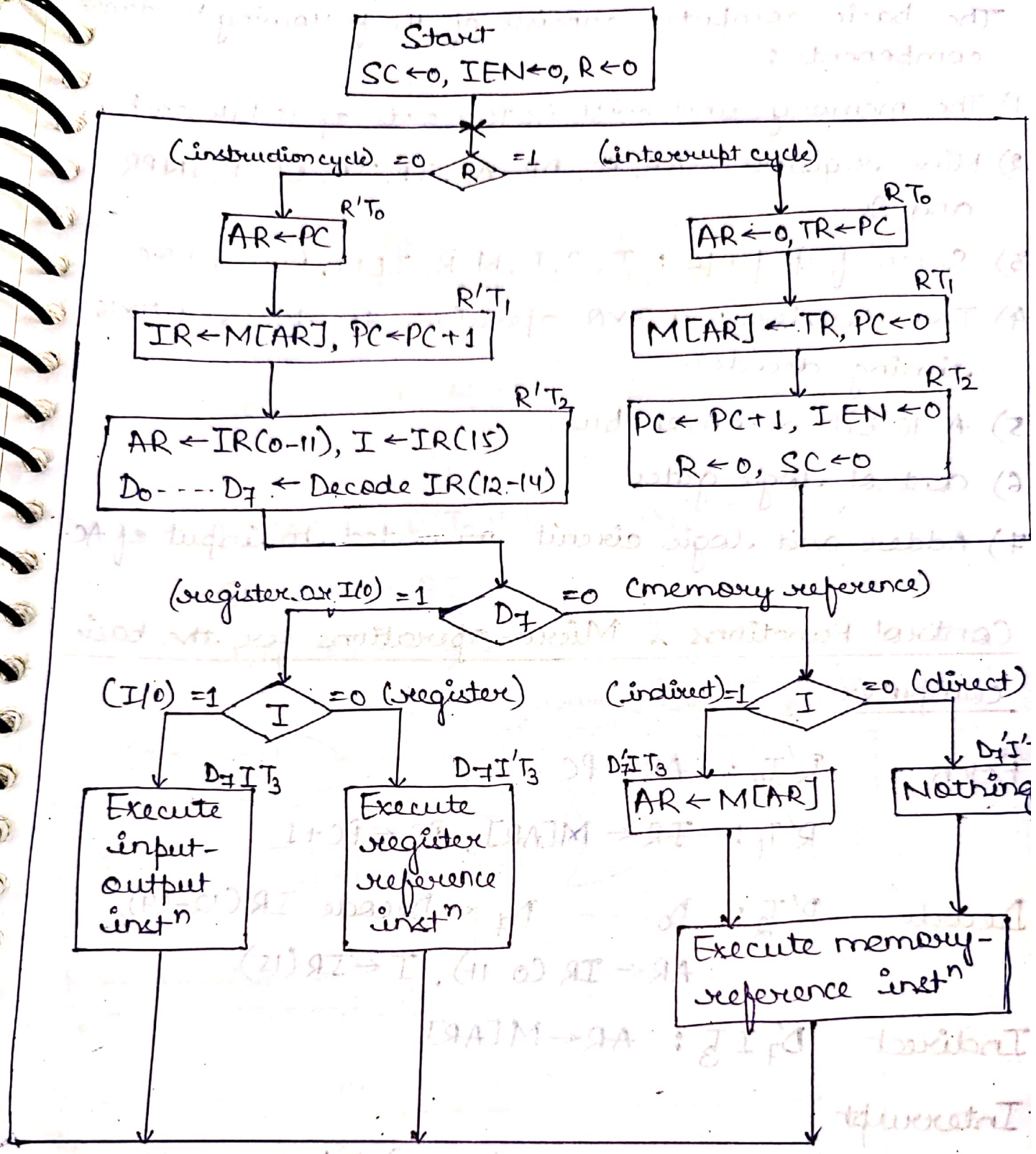
- At the end of the execute phase, control checks the value of R and if it is equal to 1; it goes an interrupt cycle instead of an instruction cycle.
- Interrupt cycle is initiated after the last execute phase if the interrupt flip-flop R is equal to 1.
- This flip-flop is set to 1 if IEN=1 and either FGI or FGO are equal to 1.
- This can happen with any clock transition except when timing signals T_0, T_1, or T_2 are active.
- The condition for setting flip-flop R to 1 can be expressed with the following register transfer statement:

$$T_0' T_1' T_2' (IEN)(FGI + FGO) : R \leftarrow 1$$



Flowchart for interrupt cycle

– Sequence of execution of instructions



Design of Basic Computer

The basic computer consists of the following hardware components :-

1) The memory unit with 4096 words of 16 bits each.
2) Nine registers : AR, PC, DR, AC, IR, TR, OUTR, INPR and SC.
3) Seven flip-flops : I, S, E, R, IEN, FGI & FGO
4) Two decoders : a 3×8 operation decoder and 4×16 timing decoder.
5) A 16-bit common bus.
6) control logic gates.
7) Adder and logic circuit connected to input of AC.

Control functions & Micro-operations for the basic computer

Fetch $R'T_0$: $AR \leftarrow PC$

$R'T_1$: $IR \leftarrow M[AR]$, $PC \leftarrow PC+1$

Decode $R'T_2$: $D_0 \dots D_7 \leftarrow$ Decode $IR(12-14)$, $AR \leftarrow IR(0-11)$, $I \leftarrow IR(15)$

Indirect $D_7 I T_3$: $AR \leftarrow M[AR]$

Interrupt

$T_0' T_1' T_2' (IEN)(FGI + FGO)$: $R \leftarrow 1$

RT_0 : $AR \leftarrow 0$, $TR \leftarrow PC$

RT_1 : $M[AR] \leftarrow TR$, $PC \leftarrow 0$

RT_2 : $PC \leftarrow PC+1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$

Control of Registers and Memory :-

Address Register (AR) :-

Scan all of the register transfer statements that change the content of AR :-

$R'T_0$: $AR \leftarrow PC$ LD (AR)

$R'T_2$: $AR \leftarrow IR(0-11)$ LD (AR)

$D_7'IT_3$: $AR \leftarrow M[AR]$ LD (AR)

RT_0 : $AR \leftarrow 0$ CLR (AR)

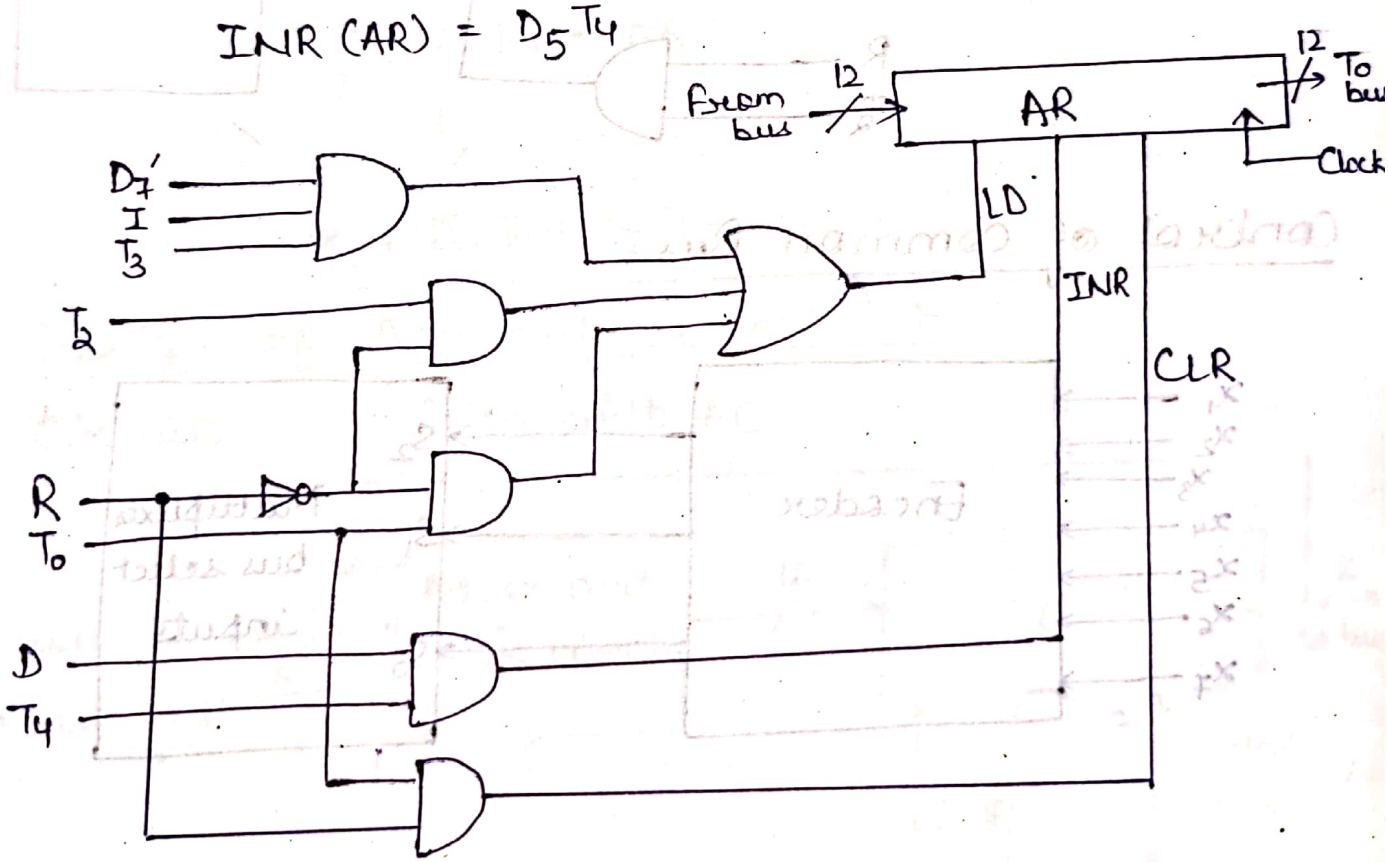
D_5T_4 : $AR \leftarrow AR+1$ INR (AR)

$$\Downarrow$$

$$LD(AR) = R'T_0 + R'T_2 + D_7'IT_3$$

$$CLR(AR) = RT_0$$

$$INR(AR) = D_5T_4$$



Control of Flags :-

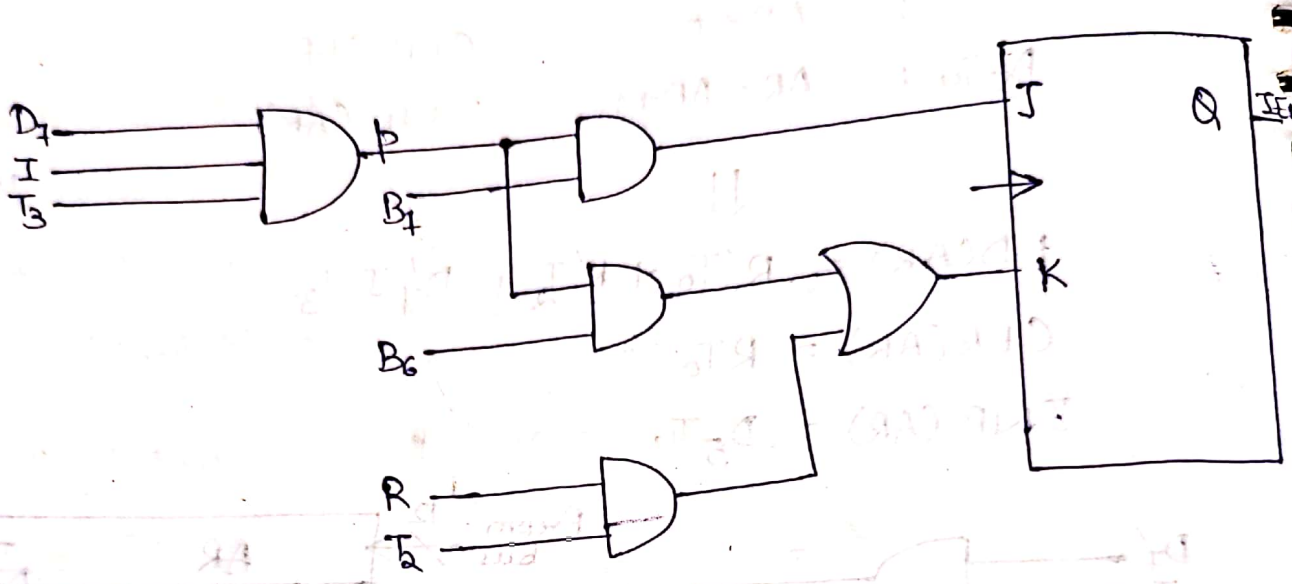
IEN: Interrupt Enable Flag

$p B_7$: IEN ← 1 (I/O Instruction)

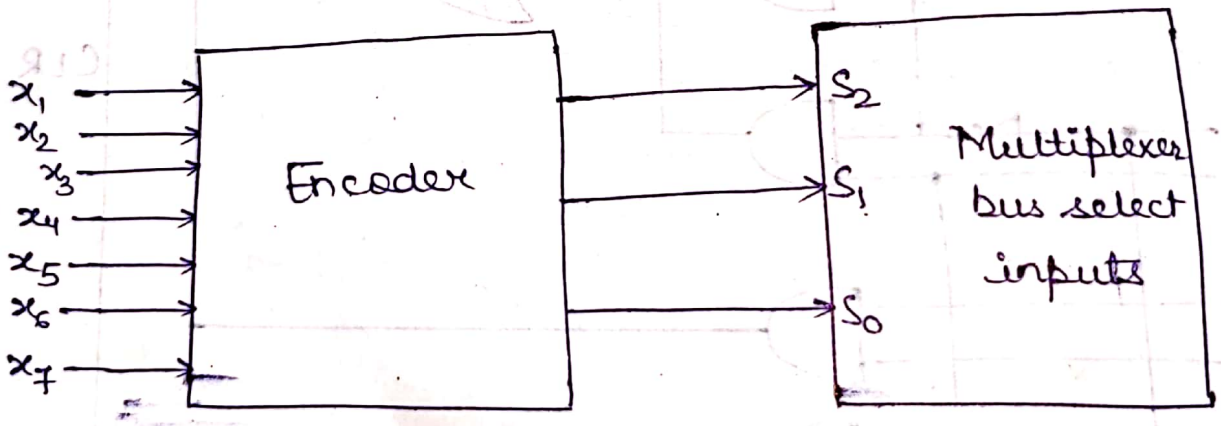
$p B_6$: IEN ← 0 (I/O Instruction)

RT_2 : IEN ← 0 (Interrupt)

$p = D_7 I T_3$ (Input/Output Instruction)



Control of Common Bus



x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	Selected register
0	0	0	0	0	0	0	0	0	0	none
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

x_1 for placing AR onto bus

$D_4 T_4$: $PC \leftarrow AR$

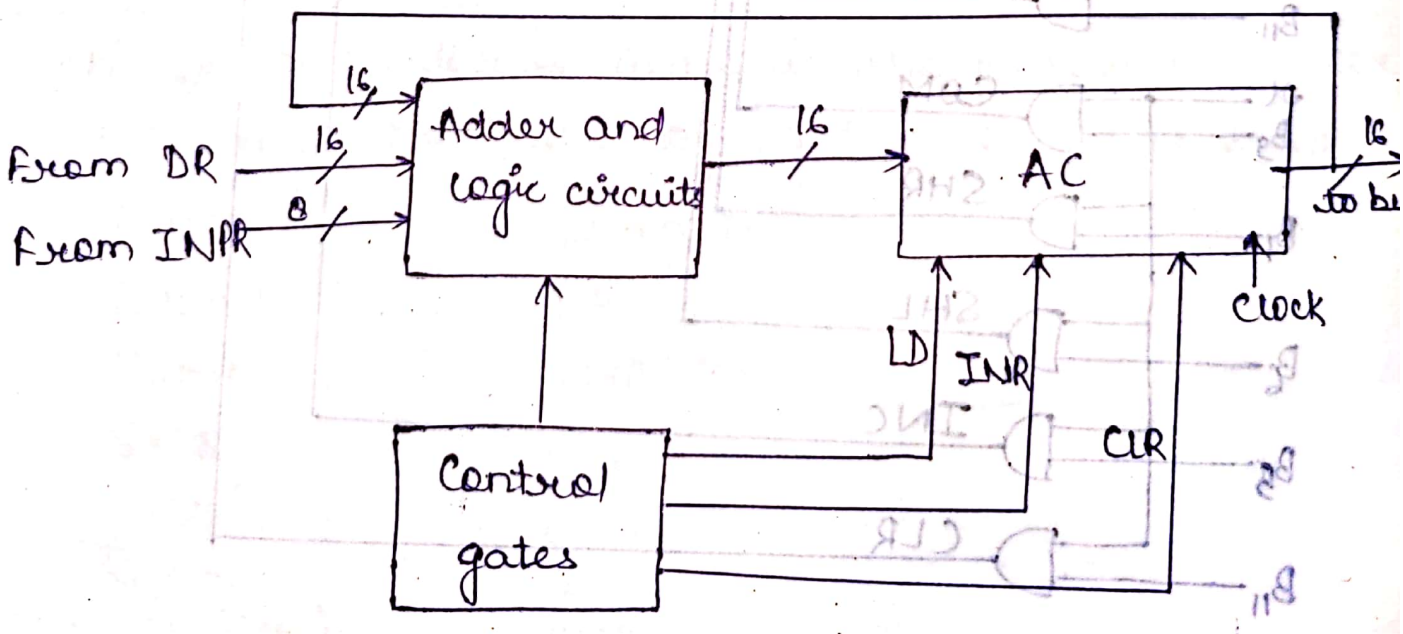
$D_5 T_5$: $PC \leftarrow AR$

$\Downarrow$

$x_1 : D_4 T_4 + D_5 T_5$

- Design of Accumulator logic :-

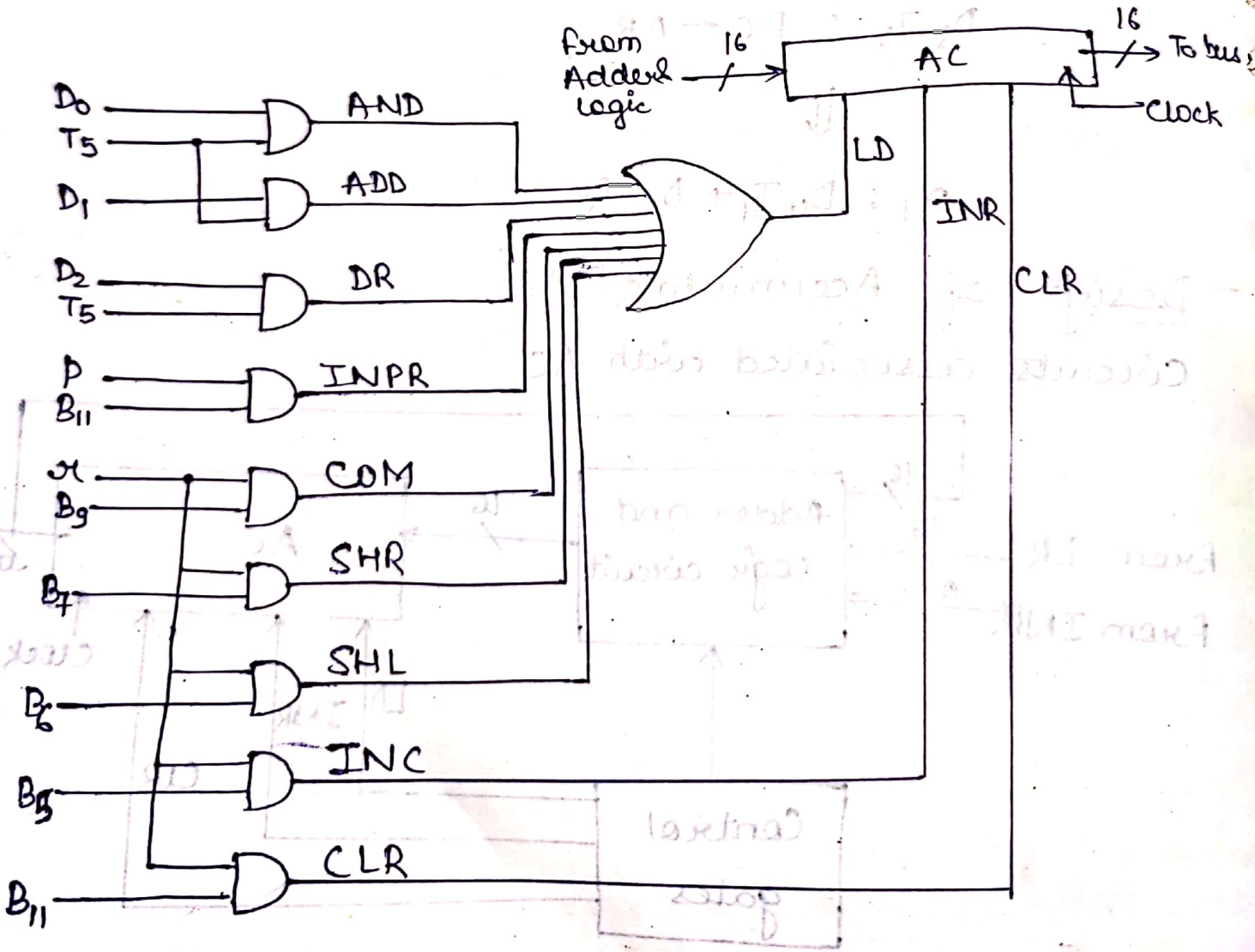
Circuits associated with AC



All the statements that changes the content of AC

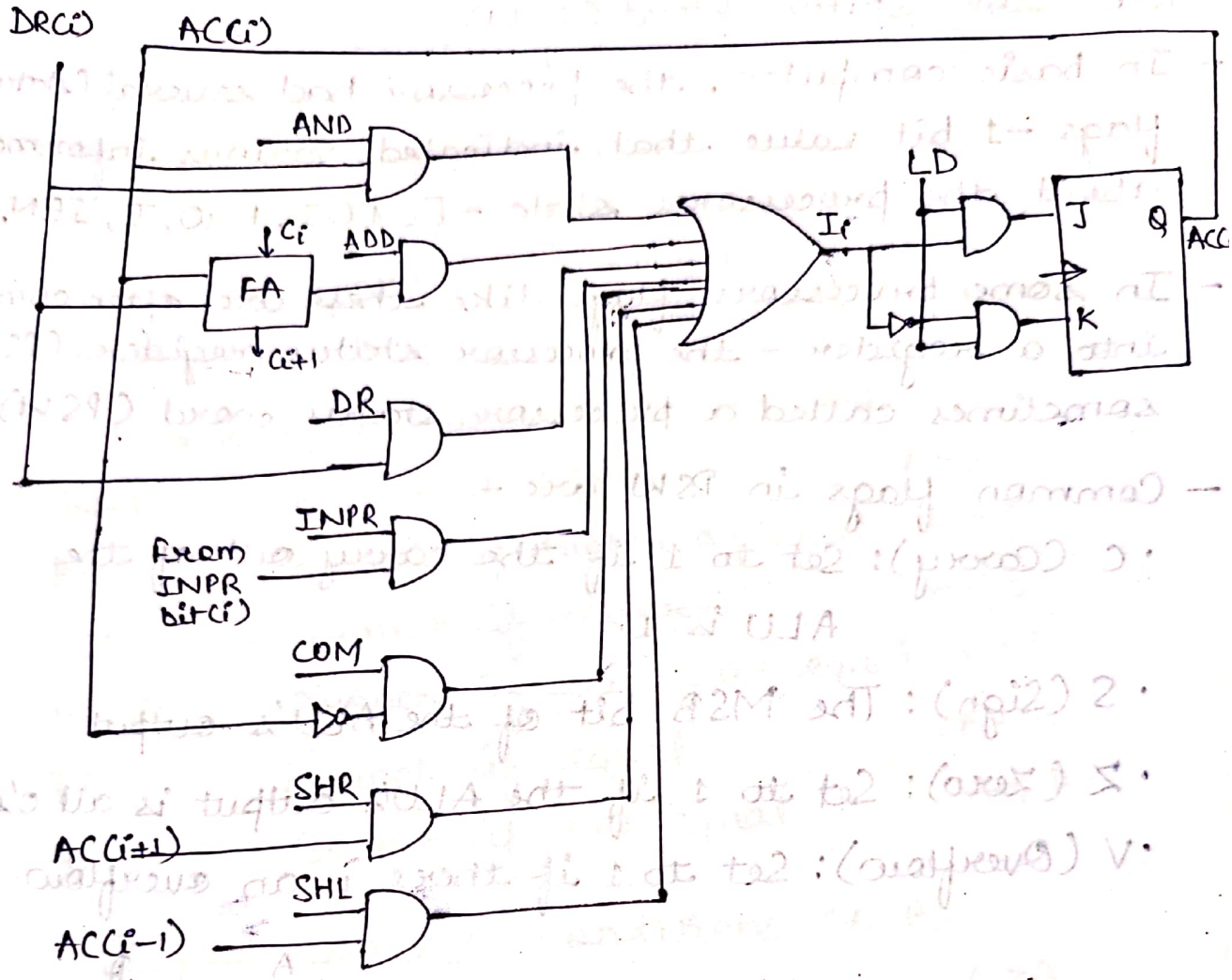
D_0T_5 :	$AC \leftarrow AC \wedge DR$	AND with DR
D_1T_5 :	$AC \leftarrow AC + DR$	ADD with DR
D_2T_5 :	$AC \leftarrow DR$	Transfer from DR
pB_{11} :	$AC(0\text{-}7) \leftarrow INPR$	Transfer from INPR
rB_9 :	$AC \leftarrow AC'$	Complement
rB_7 :	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E$	Shift right
rB_6 :	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E$	Shift left
rB_{11} :	$AC \leftarrow 0$	Clear
rB_5 :	$AC \leftarrow AC + 1$	Increment

Control of AC Register :-



ALU (Adder & Logic Circuits)

- One stage of Adder and Logic Circuit



Program Control :-

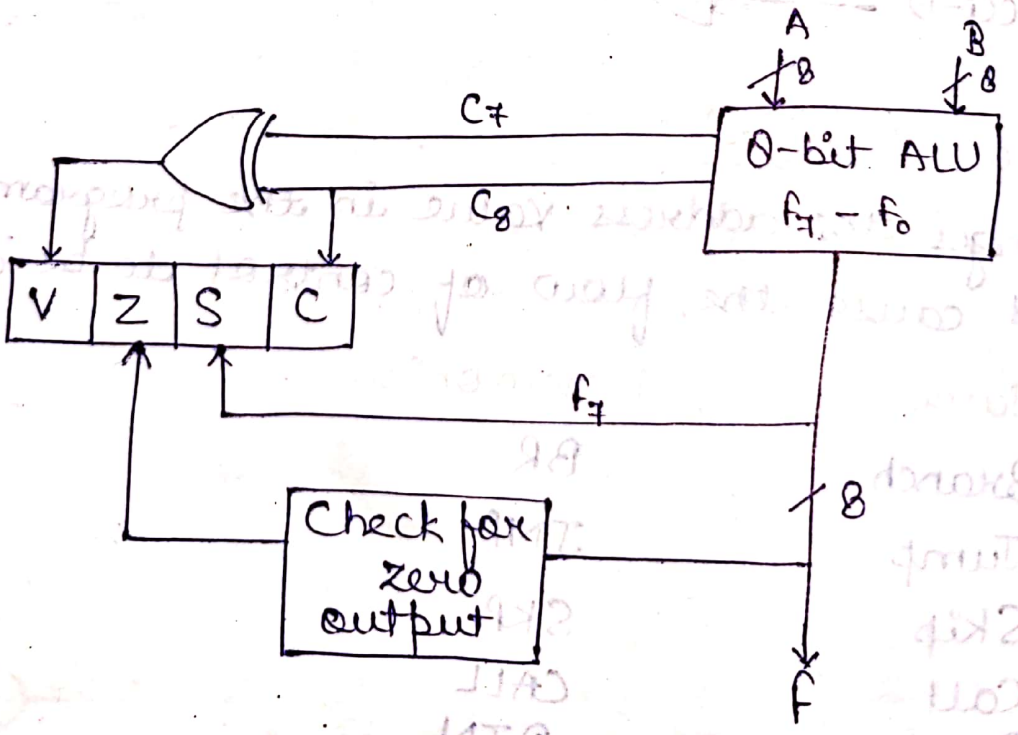
- changes the address value in the program counter and cause the flow of control to be altered.

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare (by -)	CMP
Test (by AND)	TST

- CMP and TST instructions do not retain their results of operations (- and AND respectively). They only set or clear certain flags.

Processor Status Word (PSW)

- In basic computer, the processor had several (status) flags - 1 bit value that indicated various information about the processor's state - E, FGI, FGO, I, IEN,
- In some processors, flags like these are often combined into a register - the processor status register (PSR), sometimes called a processor status word (PSW).
- Common flags in PSW are -
  - C (Carry): Set to 1 if the carry out of the ALU is 1.
  - S (Sign): The MSB bit of the ALU's output.
  - Z (Zero): Set to 1 if the ALU's output is all 0's.
  - V (Overflow): Set to 1 if there is an overflow.



Conditional Branch Instructions :-

Mnemonic	Branch Condition	Tested condition
BZ	Branch if zero	Z=1
BNZ	Branch if not zero	Z=0
BC	Branch if carry	C=1
BNC	Branch if no carry	C=0
BP	Branch if plus	S=0
BM	Branch if minus	S=1
BV	Branch if overflow	V=1
BNV	Branch if no overflow	V=0

Unsigned compare conditions (A-B)

BHI	Branch if higher	A>B
BHE	Branch if higher or equal	A≥B
BLO	Branch if lower	A<B
BLOE	Branch if lower or equal	A≤B
BE	Branch if equal	A=B
BNE	Branch if not equal	A≠B

Signed compare conditions (A-B)

BGT	Branch if greater than	A>B
BGE	Branch if greater or equal	A≥B
BLT	Branch if less than	A<B
BLE	Branch if less than or equal	A≤B
BE	Branch if equal	A=B
BNE	Branch if not equal	A≠B

- Subroutine Call and Return :-

The instruction known by different names :-

- call subroutine
- jump to subroutine
- branch to subroutine
- branch and save address

→ Operation code + Address that specifies the beginning of the subroutine.

- Executed by performing two operations :

1) The return address is stored in a temporary location -
   - In a fixed location in memory
   - Or in a processor register.
   - Or in memory stack (most efficient way)

2) Control is transferred to the beginning of the subroutine.

CALL :-

$SP \leftarrow SP - 1$

$M[SP] \leftarrow PC$

$PC \leftarrow EA$

RTN :-

$PC \leftarrow M[SP]$

$SP \leftarrow SP + 1$

Subroutine :- A subroutine (function) is a subprogram in the main program that perform a specific task.

Types of interrupt :-

1) External interrupts
Initiated from the outside of CPU and Memory
- I/O Device
- Timing Device → Timeout
- Power failure
- Operator

2) Internal interrupts
Caused by the currently running program.
- Register, Stack Overflow
- Divide by zero.
- Opcode Violation
- Protection Violation

3) Software interrupts
Initiated by the executing an instruction
- Supervisor Call
  - Switching from a user mode to the supervisor mode
  - Allows to execute a certain class of operations which are not allowed in the user mode.

Interrupt Procedure :-

Quite similar to a subroutine call except for the variations :-

1) Usually initiated by an internal or an external signal rather than from the execution of an instruction (except for the software interrupt)
2) The address of the interrupt service program is determined by the hardware rather than from

the address field of an instruction.

3) An interrupt procedure usually stores all the information necessary to define the state of CPU rather than storing only the PC.

- The state of the CPU is determined from:-
  - Content of the PC
  - Content of all processor registers
  - Content of status bits:
- Many ways of saving the CPU state depending on the CPU architectures

- RISC and CISC

RISC (emerged in early 1980's)

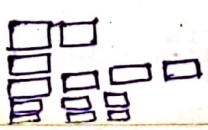
- Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instn.
- All operations done within the registers of the CPU.
- fixed length, easily decoded instruction format.
- Single-cycle instruction execution
- Hardwired rather than microprogrammed control.
- Efficient instruction pipeline.
- LOAD/STORE instructions
- Examples :- ARC, Alpha, ARC, ARM, AVR, PA-RISC and SPARC.

CISC (the original microprocessor ISA) :-

- A large number of instructions - typically from 100 to 250 instructions.
- Some instructions that perform specialized tasks and are used infrequently.
- A large variety of addressing modes - typically from 5 to 20 different modes.
- Variable length instruction formats
- Instructions that manipulates operands in memory
- Examples :- System/360, VAX, AMD and Intel x86 CPUs.

Architecture Characteristics	CISC	RISC
Instruction Size	Varies	One size, usually 32 bits
Instruction format	Field Placement varies	Regular, consistent placement of fields
Instruction Semantics	Varies from simple to complex, possibly many dependent operations per inst^n.	Almost always on simple operation
Control unit	Microprogrammed C.U.	Hardwired control uni
Registers	Few, sometimes special	Many, general-purpose
Memory references	Bundled with operations in many different types of instructions	Not bundled with operations i.e. load store architecture
Hardware Design focus.	Exploit microcoded implementations	Exploit implementati with one pipeline an no microcode

- Picture of 5 typical inst^n. □ = 1 Byte



Pipelining :-

- To improve the performance of CPU we have 2 options:-
  1) Improve the hardware by introducing faster circuits.
  2) Arrange the hardware such that more than one operation can be performed at the same time.

- Pipelining is a process of arrangement of hardware elements of the CPU such that its overall performance is increased.
- Simultaneous execution of more than one instruction takes place in a pipelined processor.

Non-overlapped execution :-

Stage/Cycle	1	2	3	4	5	6	7	8
S_1	I_1				I_2			
S_2		I_1				I_2		
S_3			I_1				I_2	
S_4				I_1				I_2

Overlapped execution :-

Stage/Cycle	1	2	3	4	5
S_1	I_1	I_2			
S_2		I_1	I_2		
S_3			I_1	I_2	
S_4				I_1	I_2

- RISC processor has 5 stage instruction pipeline :-
  1) Stage 1 : (Inst^n Fetch)
  2) Stage 2 : (Inst^n Decode)
  3) Stage 3 : (Inst^n Execute)
  4) Stage 4 : (Memory Access)
  5) Stage 5 : (Write Back)

- Performance

- Non-pipelined execution time :-

= Total number of instructions x Time taken to execute one instn.

= n x k clock cycles

- Pipelined execution time :-

= (k+n-1) clock cycles

- Speed up

= Non-pipelined execution time / Pipelined execution time

= k / {1 + (k-1)/n}

- For very large number of instructions, $n \to \infty$.
  Thus speed up = k.
- Practically, total number of instructions never tend to infinity. Therefore speed up is always less than k.

Dependencies and Data Hazard :-

1) Structural Dependency:
   - due to the resource conflict in the pipeline

2) Data Dependency :-
   - whenever there are two instructions one of which depends on the data obtained from the other.

   i) RAW (Read after Writing) [Flow Dependency]

   ii) WAR (Write after reading) [Anti-Data Dependen

   iii) WAW (Write after Writing) [Output Dependency

3) Control Dependency :-
   - when one of the instructions is a branching instruct
   - "branch penalty" (for the invalid fetches that are

needed to be removed)

- These dependencies may introduce stalls in the pipeline.

Stall :- A stall is a cycle in the pipeline without new input.

- Other possible situations :-
  Operand forwarding (for data dependency).
  1- adding more hardware - Code memory and Data memory (for structural dependency).

- Ideal CPI of the pipelined processor is '1'. But due to stalls, it becomes greater than '1'.

- Speed up (S) = $CPI_{non\text{-}pipeline}$ / (1 + Number of stalls per inst^{n}.)

Types of pipeline :

- Uniform delay pipeline :-
  - all the stages will take same time.
  - Cycle Time (T_p) = Stage Delay
  - If buffers are included between the stages then
    Cycle Time (T_p) = Stage Delay + Buffer Delay.

- Non-Uniform delay Pipeline :-
  - Different stages take different time to complete an operation.
  - Cycle Time (T_p) = Maximum (Stage Delay)

Example:- If there are 4 stages with delays (2ns, 8ns, 3ns, 10ns), find the time taken to execute 100 tasks in the above pipeline.

- As the above pipeline is a non-linear pipeline.

$T_p = \max(2, 8, 3, 10) = 10 \text{ ns}$

We know that ET pipeline $= (k+n-1)\,T_p$

$= (4+100-1) \times 10$

$= 1030 \text{ ns}$

Hardwire and micro-programmed control :-

- The function of the control unit in a digital computer is to initiate sequences of micro-operati The complexity of the digital system is derived from the number of sequences of micro-operation that are performed.

- When the control signals are generated by hardwar using conventional logic design techniques, the control unit is said to be hardwired.

- Micro-programming is a second alternative for designing the control unit of a digital computer. The principle of micro-programming is an elegant and systematic method for controlling the micro-operation sequences in a digital computer.

Difference between Hardwired Control and Micro-programmed control :-

Hardwired Control	Micro-programmed Control
- Technology is circuit based.	- Technology is software based.
- It is implemented through flip-flops, gates, decoders etc.	- Micro instructions generate signals to control the execution of instructions.
- Fixed instruction format.	- Variable instruction format (16-64 bits per instruction)
- Instructions are register based.	- Instructions are not register based.
- ROM is not used.	- ROM is used.
- It is used in RISC.	- It is used in CISC.
- faster decoding	- Slower decoding
- Difficult to modify	- Easily modified.
- Chip area is less	- Chip area is large.

Micro-programmed control :-

- Control Word (CW) :- The control variables at any given time can be represented by a string of 1's and 0's called a control word.
- Micro-instruction :- Each word in control memory contains within it a micro-instruction.

- Microprogram:-
- Sequence of microinstructions constitutes a microprogram.
- The microprogram controls the function of the CPU.
- Alterations in microprograms are not needed once the control unit is in operation. The control memory can be read-only memory (ROM).
- The contents of words of in ROM are fixed and cannot be altered; since there is no writing capacity in ROM.

- Control Memory:-
- Memory that is a part of the control unit is called control memory.
- Control memory holds microprograms that cannot be altered by the user. The micro-program consists of microinstructions to execute register micro-operations.

- Machine instruction initiates a series of micro-instructions in the control memory.
- The micro-instruction generates micro-operations to fetch instructions from main memory, to evaluate effective address, to execute the operation specified by instructions or to repeat the cycle for the next instructions.

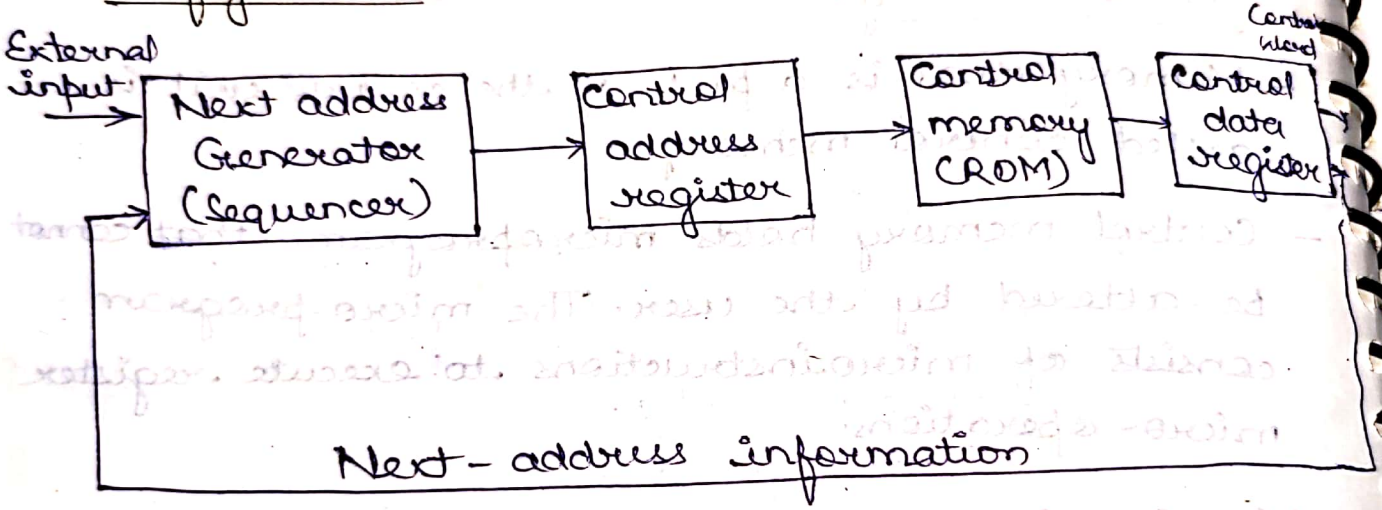
Control Address Register :-

- The control memory address register specifies the address of the microinstruction and the control data register holds the microinstruction read from memory.

Sequencer :-

- The next address generator is sometimes called a microprogram sequencer, as it determines the address sequence that is read from control memory.

→ Configuration :-

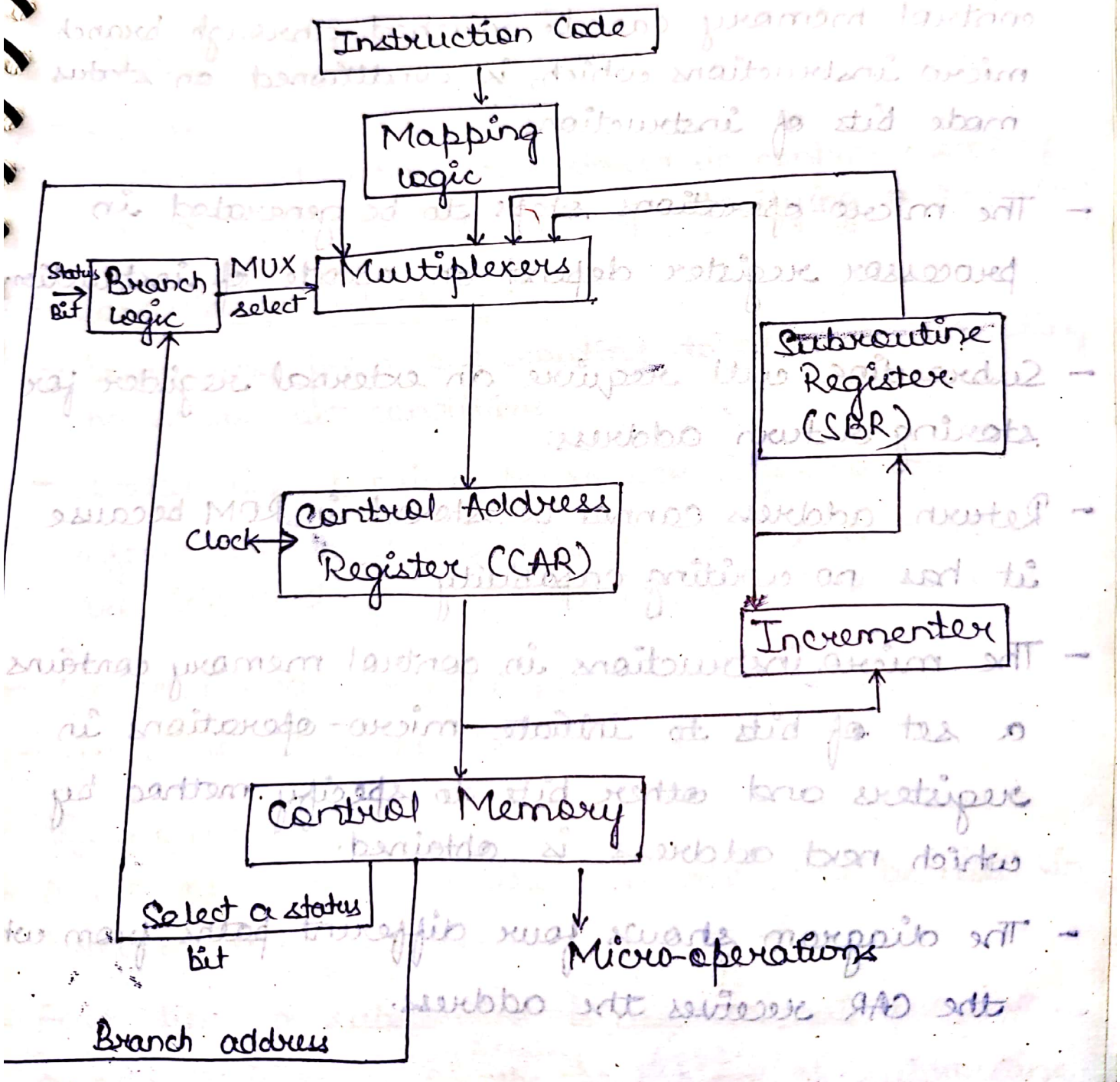


Address Sequencing and Mapping

- Sequencer :- Micro-instructions sequencer is a combination of all hardware for selecting the next micro-instruction address.
- The micro-instruction in control memory contains a set of bits to initiate micro-operations in

computer registers and other bits to specify the method by which the address is obtained.

- Addressing Sequencing capabilities include :-
  - Incrementing of the control address register.
  - Unconditional branch or conditional branch, depending on status bit conditions.
  - A mapping process from the bits of the instⁿ to an address for control memory.
  - A facility for subroutine call and return.



Selection of address from control memory

Address Sequencing :-

- Micro instructions are stored in control memory in group with each group specifying a routine.
- An initial address is located into the control address register when power is turned on. The fetch routine may be sequenced by incrementing CAR. At end of fetch routine instructions is in IR.
- The effective address computation routine in control memory can be reached through branch micro instructions which is conditioned on status mode bits of instructions.
- The micro-operations steps to be generated in processor register depend on opcode of instructio
- Subroutine will require an external register for storing return address.
- Return address cannot be stored in ROM because it has no writing capability.
- The microinstructions in control memory contains a set of bits to initiate micro-operations in registers and other bits to specify method by which next address is obtained.
- The diagram shows four different paths from whic the CAR receives the address.

- The incrementer increments the CAR content by one.
- Branching is achieved by specifying branch address in one of the fields of micro-instructions.
- Conditional branching is obtained by status bit in order to determine its condition.
- The return address for a subroutine is stored in special register (SBR) whose value is then used when microprogram wishes to return from the subroutine.

- Selection of Address :-

- Routine:-
  - Micro instructions are stored in control memory in groups, with each group specifying a routine.

- Conditional Branching :-
  - branching from one routine to another depending on status bit conditions.
  - status bits provide parameter information.
  - information in status bits are tested and actions are initiated based on their conditions : 1 or 0.

- Unconditional Branching :-
  - fix value of status bit to 1.

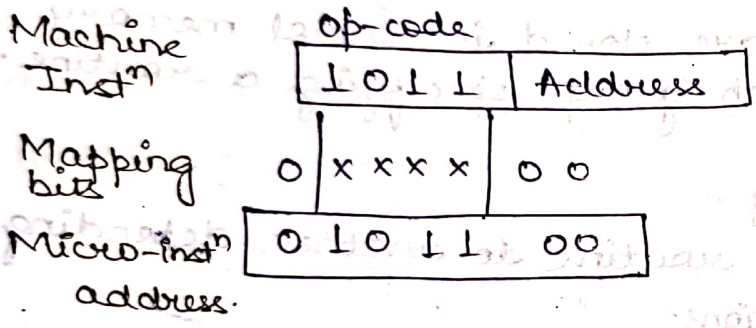
- Subroutines :-
  - A set of common instructions that can be used in program many times.
  - Each time a subroutine is used in main program, a branch is made to the beginning of subroutine

- A subroutine register is used for storing the return address during a subroutine call and restoring the address during subroutine return.

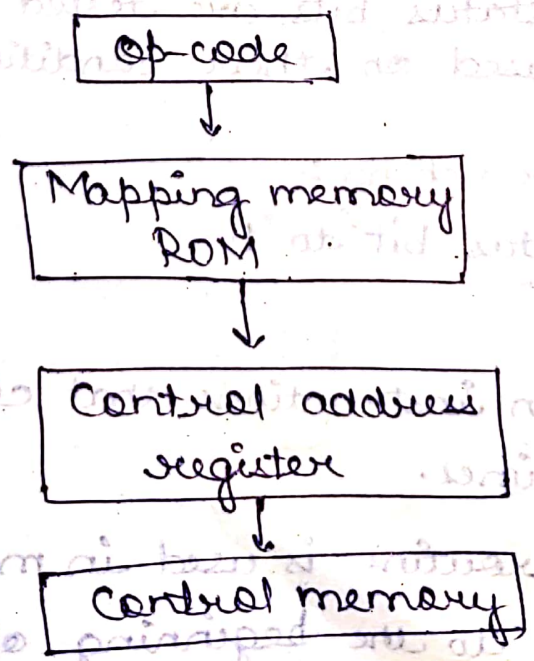
Mapping of instruction:-

- Each computer instruction has its own microprogram.
- Routine stored in a given location of the control memory.

Mapping :- transformation from instruction code bits to microinstruction address in control memory where routine is located.

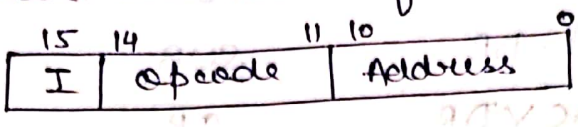


Mapping function implemented by ROM



Microinstruction Code format (20 bits)

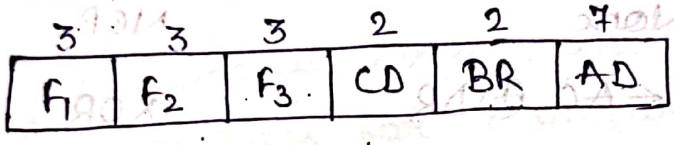
Computer instruction format



Four computer instructions

Symbol	Opcode	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC < 0) then PC ← EA
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

Microinstruction format



F_1, F_2, F_3 : Micro-operation fields

CD : Condition for branching

BR : Branch field

AD : Address field

F_1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F_2	Micro-operation	Symbol
000	None	NOP
001	AC ← AC − DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR.

F_3	Micro-operation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC′	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	DR (15)	I	Indirect address bit
10	AC (15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1 CAR ← CAR+1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR+1 if condition=1 CAR ← CAR+1, if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Symbolic Microprogram :-

- Control Memory : 128 20-bits words
- First 64 words : Routines for 16 machine instructions
- Last 64 words : Used for other purpose (eg. fetch routine and other subroutine)
- Mapping : Op-code XXXX into 0XXXX00, first address for 16 routines are 0(0 0000 0 4(0 0001 00), 8, 12, 16, 20 ---- 60

Partial Symbolic Microprogram :-

Label	Micro-ops	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER :	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH

```
        ORG 8
STORE:  NOP       I   CALL    INDRCT
        ACTDR     U   JMP     NEXT
        WRITE     U   JMP     FETCH

        ORG 12
EXCHANGE: NOP     I   CALL    INDRCT
        READ      U   JMP     NEXT
        ACTDR, DRTAC U JMP    NEXT
        WRITE     U   JMP     FETCH

        ORG 64
FETCH:  PCTAR     U   JMP     NEXT
        READ, INCPC U JMP     NEXT
        DRTAR     U   MAP
INDRCT: READ      U   JMP     NEXT
        DRTAR     U   RET
```

- Horizontal and Vertical Microprogramming :-

- In Horizontal micro-programmed control unit, the control signals are represented in the decoded binary format i.e. 1 bit/CS. Here 'n' control signals require n-bit encoding.

- In Vertical micro-programmed control unit, the control signals are represented in the encoded binary format. Here 'n' control signals require $\log_2 n$ bit encoding.

Horizontal Microprogrammed Control Unit	Vertical Microprogrammed Control Unit
- It supports longer control word.	- It supports shorter control word.
- It allows higher degree of parallelism. If degree is n, then n control signals are enabled at a time.	- It allows low degree of parallelism, i.e. degree of parallelism is either 0 or 1.
- No additional hardware is required.	- Additional hardware in the form of decoders are required to generate control signals.
- It is faster than Vertical microprogrammed control unit.	- It is slower than horizontal microprogrammed control unit.

- It is less flexible than vertical microprogrammed control unit.
- Horizontal microprogrammed control unit uses horizontal microinstruction, where every bit in the control field attaches to a control line.

- It is more flexible than horizontal microprogrammed control unit.
- Vertical microprogrammed control unit uses vertical microinstruction, where a code is used for each action to be performed and the decoder translate this code into individual control signals

Example:- Consider a hypothetical control unit which supports 4k words. The hardware contains 64 control signals and 16 flags. What is the size of control word used in bits and control memory in byte using i) Horizontal Programming ii) Vertical Programming

Here there is no branch condition so neglect it.

$\log_2 16 = 4$

For horizontal = 64 bits/CS

For vertical = $\log_2 64$ = 6 bits/CS

$\log_2 2^{12} = 12$ bits

i) For Horizontal

- 64 bits for 64 signals

Control Word size = 4+64+12 = 80 bits

Control Memory = 4 KW = 4 * 80/8 = 40 KByte

- For Vertical

- 6 bits for 64 signals i.e. $\log_2 64$

Control word size = 4 + 6 + 12 = 22 bits

Control Memory = 4 kW = $4 \times 22/8$ = 11 kByte