Unit-3

Basic Computer Organization and Design:

⊛ Instruction code → An instruction code is a group of bits that instruct the computer to perform a specific operation.

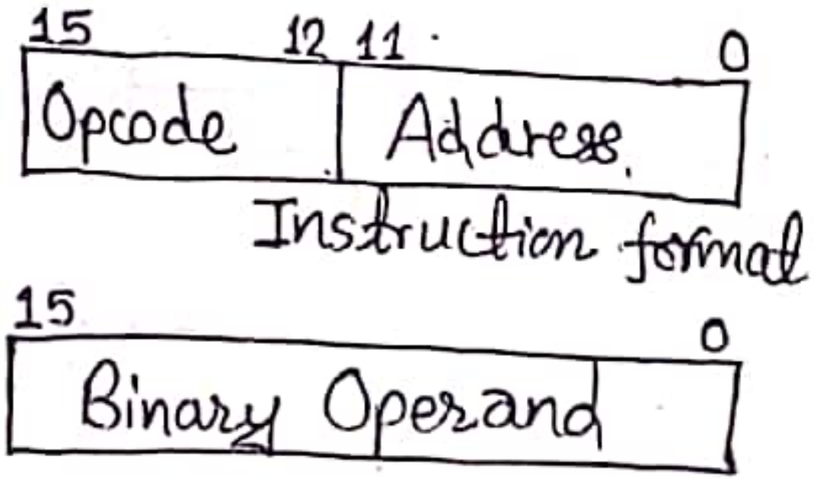
⊛ Operation code → The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift and complement. The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.

⊛ Concept of Instruction format:

The simplest way to organize a computer is to have single processor register called accumulator and an instruction format with two parts.

→ The first part specifies the operation to be performed (i.e, opcode)

→ The second part specifies an address. and the memory address tells the control where to find an operation in memory. i.e. (address).



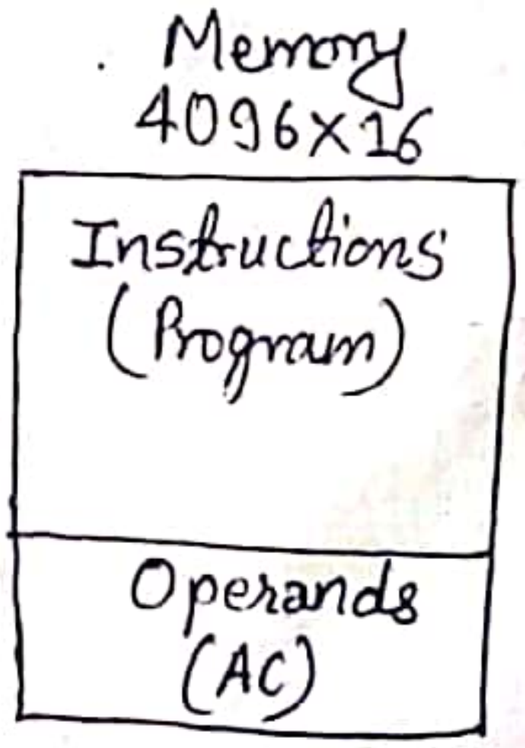


fig. Stored Program Organization

→ Instruction format is divided into two parts (bits 0 to 11 as address and 12 to 15 as opcode). The whole instruction format is of 16-bit (i.e, from 0 to 15). The opcode and address commonly in one called Binary operand.

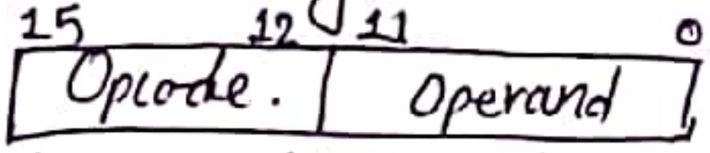
→ Memory consist of Instructions and Operands. Instruction code is having 16-bit so, $2^{16} = 4096$. which specifies address part and $2^4 = 16$ specifies opcode part.

Instruction that comes in instruction format is of three types as:

i) Memory Refrence instruction
ii) Register Refrence instruction
iii) I/O Refrence instruction.

Based on the type of instruction processor will specify either the instruction is from memory or register or any I/O device. Instruction codes for ~~three~~ these instructions are in the following forms:-

i) Immediate addressing → Actual operand will be present in the immediate addressing.

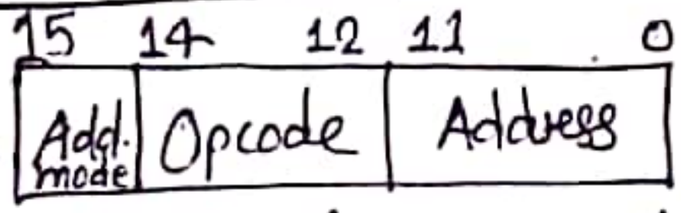


In immediate addressing the first part of instruction format (i.e, operand) specifies the operation to be performed.

(Conceptual part)

ii) Direct addressing →

Instruction format:



0-11 bits represent address
12-14 bits represent opcode
Last 15th bit represents addressing mode.
~~If address~~ 15th bit consist either 0 or 1.
If 1 → Indirect addressing mode
If 0 → Direct addressing mode.

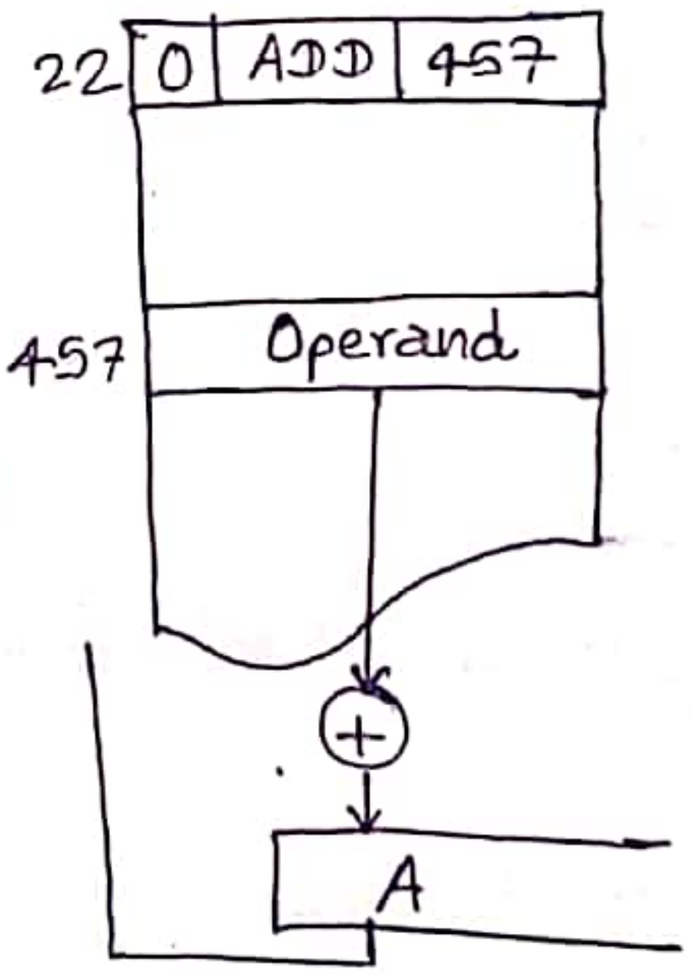


fig. Direct addressing.

→ When the second part of an instruction code specifies the address of an operand, the ~~address~~ instruction is said to have direct address.

→ For instance the instruction MOV R0 00H. R0, when converted to machine language is the physical address of register R0. The instruction moves 0 to R0.

⇒ Indirect Addressing →

→ When the second part of an instruction code specifies the address of a memory word in which the address of the operand, the instruction is said to have indirect address.

→ For instance the instruction MOV @R0 00H, when converted to machine language, @R0 becomes whatever is stored in R0, and what is the address used to move 0 to R0. It can be whatever ~~is~~ stored in R0.

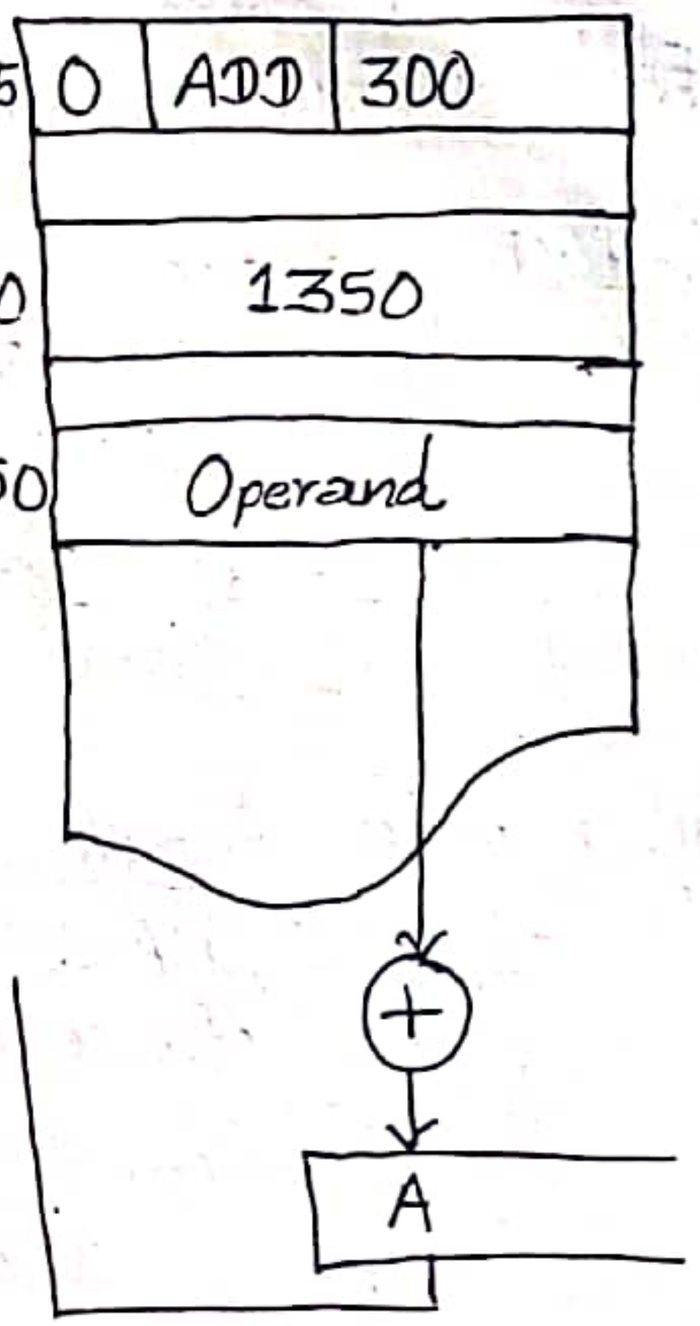


fig. Indirect addressing.

⊛. Basic Computer Registers and Memory :-

Table: List of Registers for Basic Computers:

Register Name	Register Symbol	Bits	Function
Data Register	DR	16	Holds memory operand
Address Register	AR	12	Holds address for memory
Accumulator	AC	16	Processor register
Instruction register	IR	16	Holds instruction code.
Program counter	PC	12	Holds address of instruction.
Temporary register	TR	16	Holds temporary data.
Input register	INPR	8	Holds input character.
Output register	OUTR	8	Holds output character.

⊛ Instruction Format of basic computer:

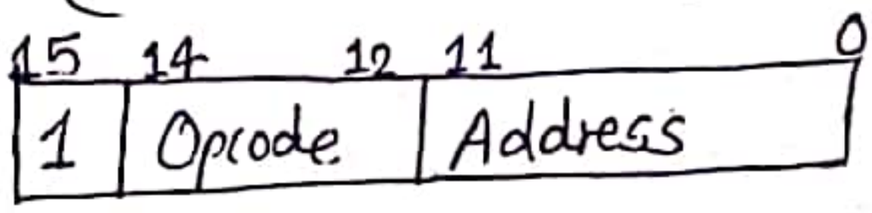
The basic computer has three instruction code formats. Each instruction code format has total of 16-bits.

1) Memory-Refrence instructions

If full forms needed for these instruction refer table of text book page no. 50.

ADD, AND, LDA, STA, BUN, BSA, ISZ are memory refrence instructions.

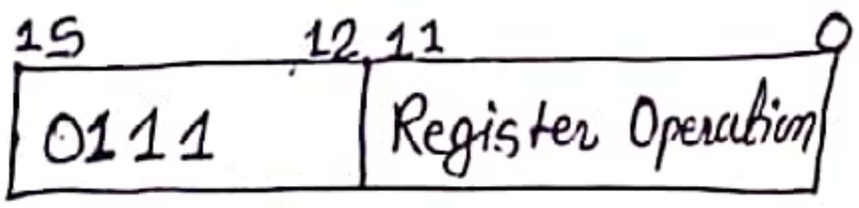
(Op-code = 000~110)



11) Register-Refrence instructions

CLA, CLE, CMA, CME, CIR, INC, SPA, SNA, SZA, SZE, HLT are register-refrence instructions.

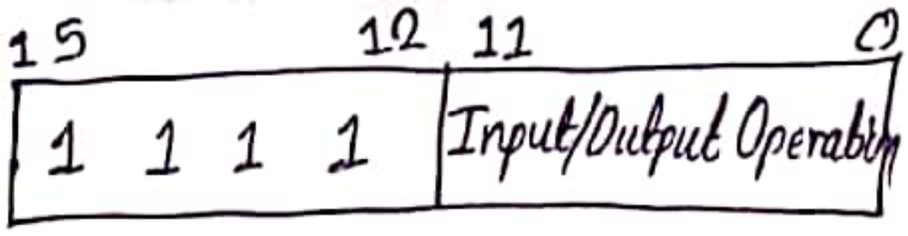
(Op-code = 111, I=0)



111) Input-Output instructions

INP, OUT, SKI, SKO, ION, IOF are input-output instructions.

(Op-code = 111, I=1)



⊛ Instruction Set Completeness: (V.V.I)

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

a) Arithmetic, Logic and shift instructions →

Arithmetic → Addition ⇒ ADD
→ Subtraction ⇒ ADD, CMA, INC
→ Division ⇒ Shifting + Subtraction.
→ Multiplication ⇒ Shifting + addition

Logic → AND + CMA ⇒ NAND

Shift → CIR & CIL

b) For moving information to and from memory and CPU registers.
⇒ LDA, STA.

c) For Branching & testing various conditions:-
⇒ BUN, SPA, SNA, SZA, SZE

d) Input and output instructions:-
⇒ INP, OUT.

⊛. Common Bus System for Basic Computer:

The common bus is required in computer for communication with registers and memory to decrease the hardware complexity and troubleshoot problems.

→ The basic computer has eight registers, a memory unit and a control unit.
→ An efficient way for transfering information in a system with many register is to use a common bus.

Requirements of common bus system:

→ Six registers and a memory are connected to a bus.
→ Which register among seven register and memory to be selected is determined from a binary value of variables S2, S1 and S0.
→ The input register INPR and output register OUTR has 8-bits each.
→ The seven registers, memory, INPR and OUTR are driven by a single phase clock pulse.
→ The particular register whose load (LD) input is enabled receives the data from bus during next clock pulse.

1,2,3,4,6

→ Five registers have three control inputs: LD (load), INR (increment), and CLR (clear). Two registers IR and OUTR ~~and~~ have only LD inputs.

→ The result is transfered to AC and end-carry is transferred to flip-flop E.

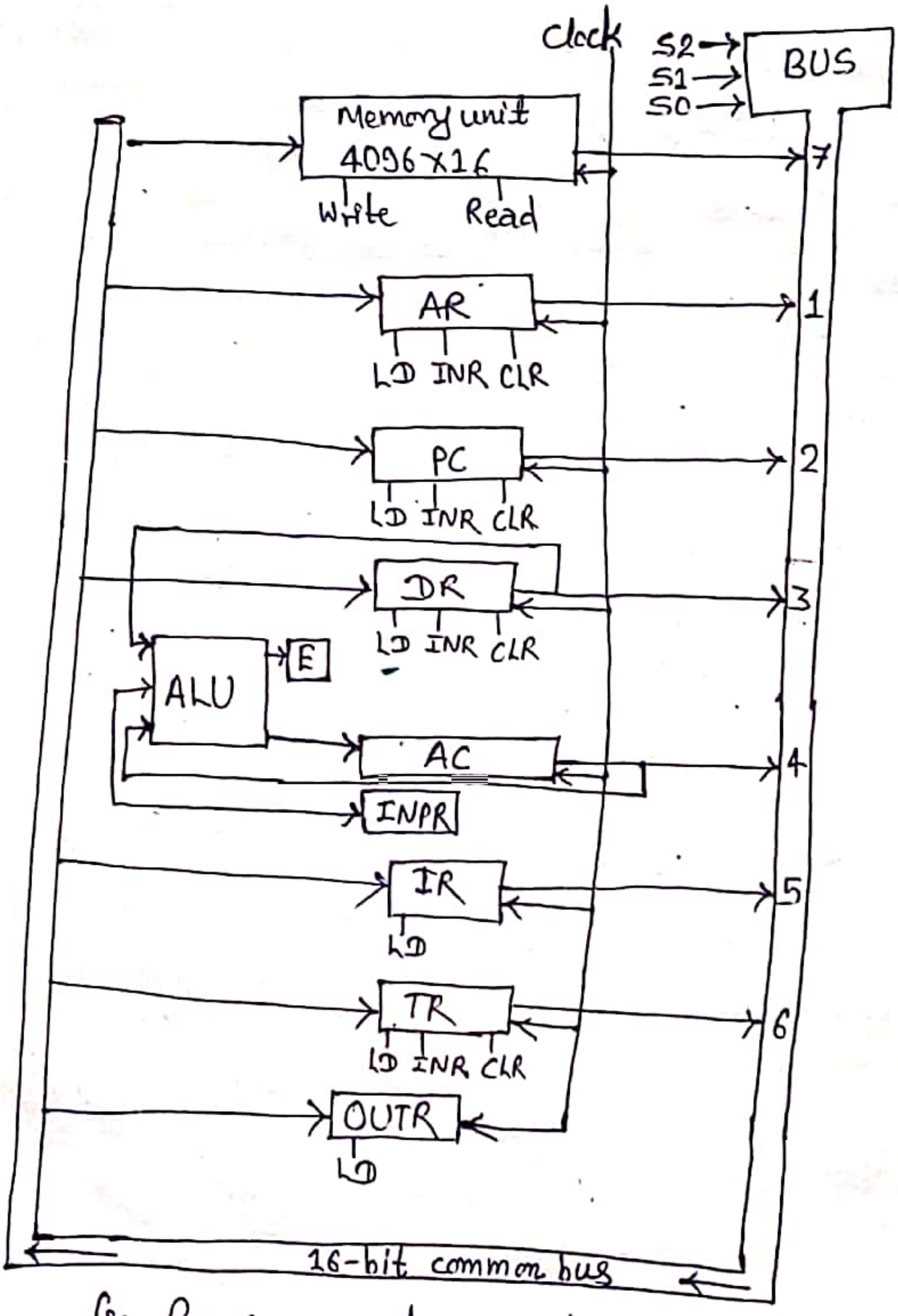


fig. Basic computer registers connected to a common bus.

⊛. Control Unit of Basic Computer:

Control Unit

Microprogrammed Control Unit

→ Control logic is implemented by means of microprogram(software).
→ It is slow & Cheap
→ Modification of logic is easy
→ Used in general purpose computers.

Hardwired Control Unit

→ Control logic is implemented using hardware (flip-flops, logic gates etc.)
→ It is fast and expensive,
→ Modification of logic is difficult.
→ Used in microcontrollers.

Hardwired Control Unit:

(Question maybe asked as:
How instruction is mapped into control ~~of~~ signal OR Explain the block diagram of Hardwired control unit?)

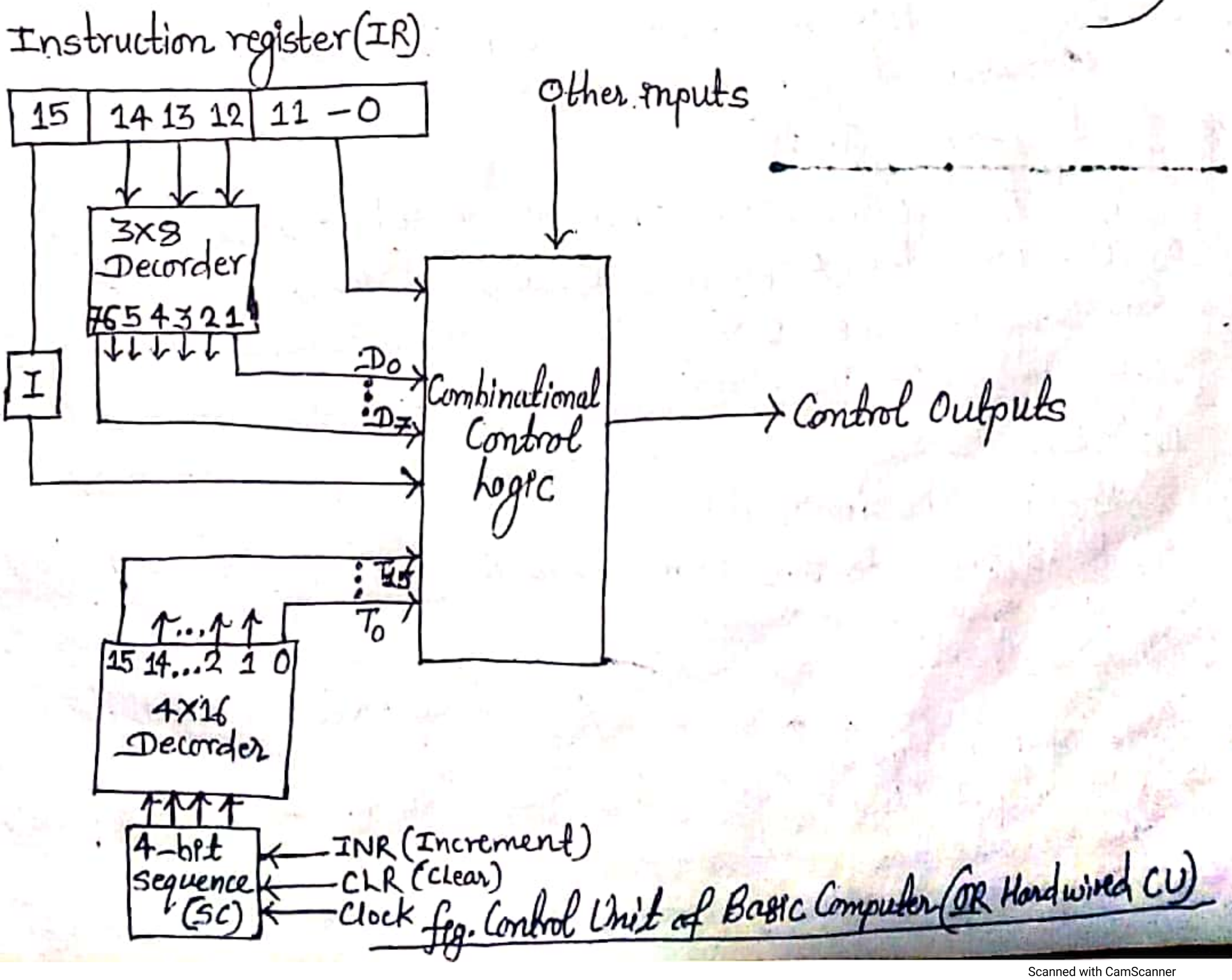


fig. Control Unit of Basic Computer (OR Hardwired CU)

Explanation:

→ An instruction read from memory is placed in the instruction register (IR).

→ In control unit IR is divided into three parts: I-bit as addressing mode, bits 12–14 as opcode and bits 0–11 as address (or operand).

↳ The opcode on bits 12–14 are decoded with a 3×8 decoder and outputs (D_0 to D_7) applied to Combinational control logic.

→ Bit-15 of the instruction is transferred to a flip-flop I.

→ Bits 0 to 11 are applied to the control logic gates (or combinational control logic).

→ We need a 4-bit sequence counter for providing timing signals to combinational control logic. It consists Increment, clear and clok pins.

→ The output of the sequence counter (SC) is applied to 4×16 decorder which consists ~~bits~~ 16-bits (0–15) denoted as T_0 to T_{15} which are also applied to the combirational control logic.

→ Based on this architecture we get some output through control outputs.

⊛. Instruction Cycle:

→ Instruction cycle is basically related with the execution of the instruction. A program residing in the memory unit of the computer consists of sequence of instructions. In the basic computer 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.

Step 1 to step 4 will be executed for all instructions in the program.

→ After step 4 (i.e, iv), the control goes back to step 1 to fetch, decode and execute the next instruction.

→ This process continues until a HALT instruction is encountered.

Fetch & Decode → The microoperations for fetch and decode phases are as follows:-

(i) T_0 : $AR \leftarrow PC$

i.e, At time T_0 the content of Program counter (PC) is transfered to Address Register AR.

(ii) T_1 : $IR \leftarrow M[AR], PC \leftarrow PC+1$

i.e, At time T_1 the content of memory pointed by the address register will be transfered to IR and increment will be done to PC by one.

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

i.e, At time T_2 decoding operations will start to instructions. IR contains the instructions to be decoded.

Determining type of instruction:

Q. How instruction is executed in basic computer? Explain with flowchart.

Ans:

fig. Flowchart for instruction cycle.

⊛ Memory Refrence Instructions:

i) AND: AND to AC → This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of operation is transfered to AC.

$D_0T_4: DR \leftarrow M[AR]$

i.e, At time D_0T_4 the content of memory pointed by address register is transfered to data register.

$D_0T_5: AC \leftarrow AC \wedge DR, SC \leftarrow 0.$

i.e, At time D_0T_5 the content of accumulator is anded with data register and result is placed in accumulator, and once the execution is completed we re-initialize sequence counter (SC) to zero.

ii) ADD: ADD to AC → The instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry Cout is transfered to the E(extented accumulator) flip-flop.

$D_1T_4: DR \leftarrow M[AR]$

$D_1T_5: AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0.$

iii) LDA: Load to AC → This instruction transfers the memory word specified by the effective address to AC.

$D_2T_4: DR \leftarrow M[AR]$

$D_2T_5: AC \leftarrow DR, SC \leftarrow 0.$

iv) STA: Store AC → This instruction stores the content of AC into the memory word specified by the effective address.

$D_3T_4: M[AR] \leftarrow AC, SC \leftarrow 0.$

v) BUN: Branch unconditionally → This instruction transfers the program to instruction specified by the effective address. The BUN instruction allows the programmer to specify an instruction out of sequence and the program branches (i.e jumps) unconditionally.

$D_4T_4: PC \leftarrow AR, SC \leftarrow 0$

vi) Branch and Save Return Address (BSA) →

This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address available in PC into a memory location specified by effective address.

D_5T_4: M[AR] ← PC, PC ← AR+1.

D_5T_5: PC ← AR, SC ← 0.

vii) Increment and Skip if Zero (ISZ) →

This instruction increments the word specified by effective address, and if the increment value is equal to 0, PC is incremented by 1.

D_6T_4: DR ← M[AR]

D_6T_5: DR ← DR+1

D_6T_6: M[AR] ← DR, if (DR=0) then (PC ← PC+1), SC ← 0.

⊛. Input-Output of Basic computer:

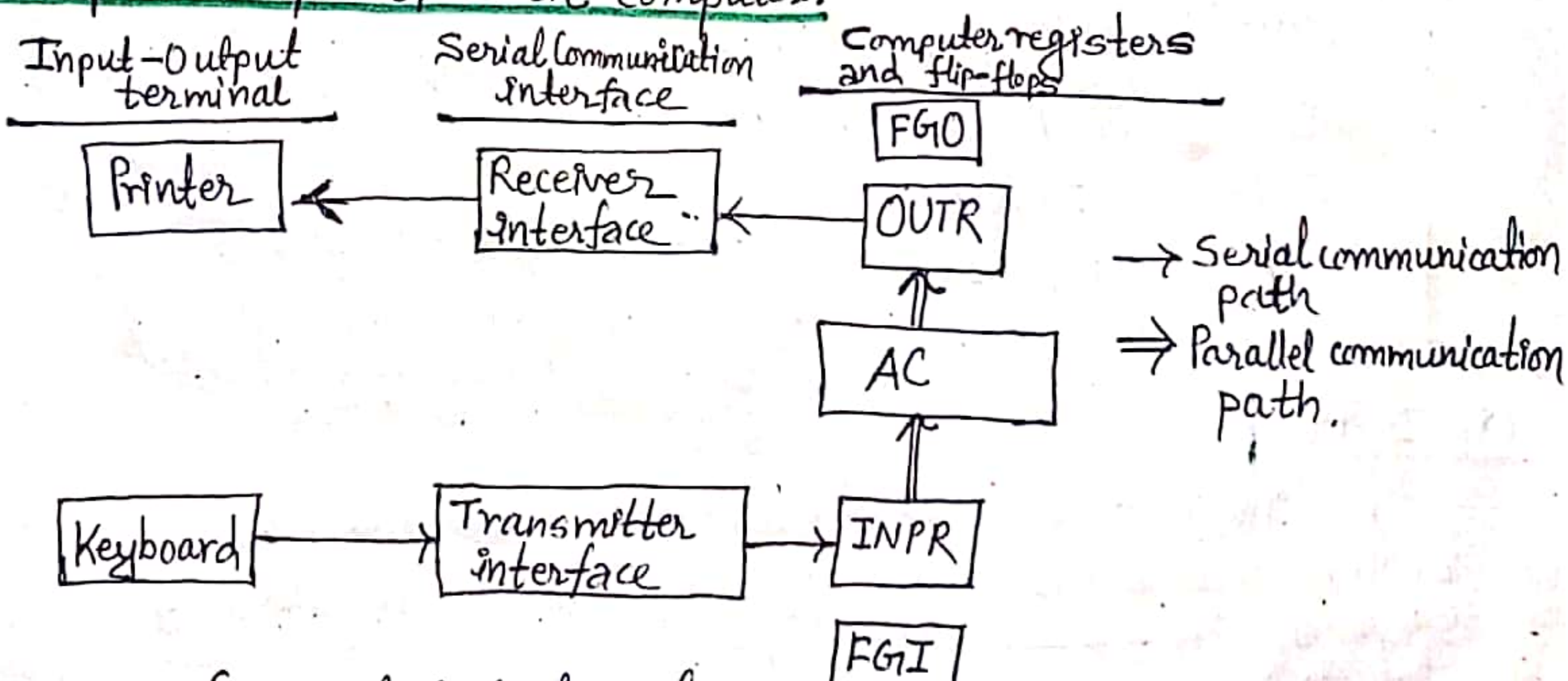


fig. Input-Output configuration of basic computer.

Input Process → Initially FGI is cleared to 0. When key is pressed on keyboard 8-bit alphanumeric code is shifted into INPR and input flag FGI is set to 1.

→ As long as the flag is set to 1, the information in INPR cannot be ~~can~~ changed by striking key. During this time the information from INPR is transferred parallel in AC and FGI is cleared to 0.

Output process → The output flag FGIO is set to 1 initially. The computer checks flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGIO is cleared to 0.
→ The output device accepts the coded information then prints the information and sets FGIO to 1.

⊛. Input – Output Instructions:–

Input and output instructions are needed for transferring information to and from AC register, for checking the flag bits and for controlling interrupt facility. The control functions and micro-operations for the input-output instructions are listed below:

INP	AC(0-7) ← INPR, FGI ← 0 (AC: Accumulator; 8-bits)	Input char. to AC
OUT	OUTR ← AC(0-7), FGO ← 0	Output char. from AC.
SKI	if (FGI=1) then (PC ← PC+1)	Skip on input flag.
SKO	if (FGO=1) then (PC ← PC+1)	Skip on output flag.
ION	IEN ← 1	Interrupt enable on.
IOF	IEN ← 0	Interrupt enable off.

(Interrupt Enable flags: 1 = set, 0 = reset)

⊛ Interrupt Cycle:-

The interrupt cycle is a hardware implementation of a branch and save return address operation. Interrupt cycle occurs ~~when~~ during input or output process. Input/Output operation are asynchronous, they are not fixed. i.e, the time is not fixed for them that when they are going to occur, because computer do not know when we are giving input to the computer. So, we consider Input/Output operation as interrupt since we are changing control of execution of our program. Now we see how it occurs from following flow chart.

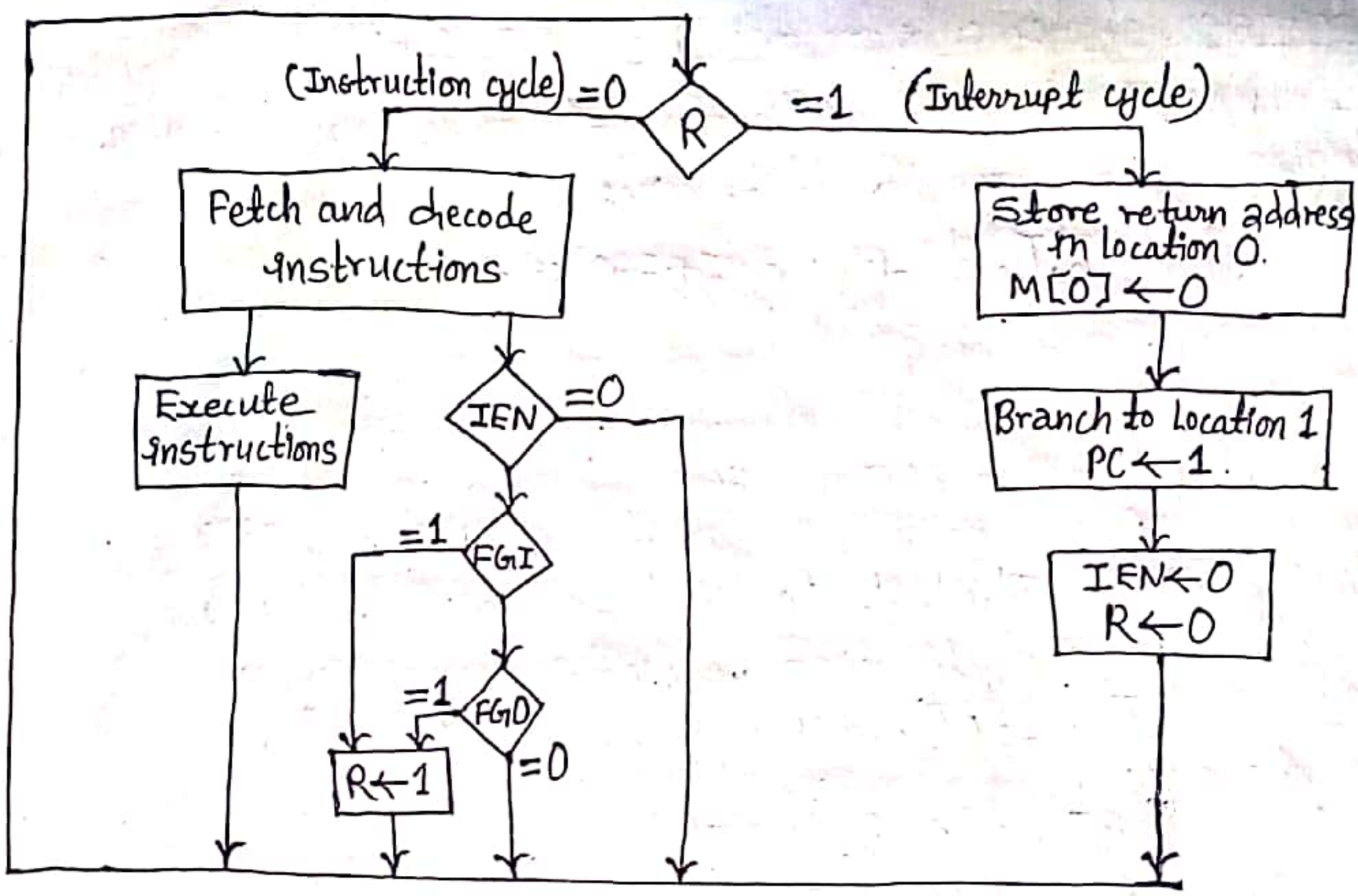


fig. flowchart for interrupt cycle

Working & Occurance → For this we have one flip-flop R. This flip-flop R will decide whether it is interrupt or it is normal execution of our program. If R value is set to 0 then it is our normal execution cycle, in which we will do fetching and decoding of instructions and executing instructions. But during fetch and decode if we encounter IEN flag then, following conditions occur:

→ If IEN is set to 1 then we check FGI.
→ If FGI is 1 then interrupt is occured we need to set flip-flop R to 1.
& If FGI is 0 We go to FGO, if FGO is 1 then interrupt is occured we need to set flip-flop R to 1.
→ Otherwise it is normal execution.

Now, if R value is set to 1 then the interrupt is occured. Now in this case we store return address in the location 0 (fixed location) ie, return address is always in PC, so it is stored from PC to memory location pointed by zero. After that we jump branch to location 1. Finally we set IEN to 0 and R to 0.

Register transfer statements for the interrupt cycle:-

The condition for setting flip-flop R=1 can be expressed with the following register transfer statement:

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

- T_0', T_1', T_2' → Timing signals with ' represent initialized to zero
- (IEN) → Interrupt Enable Flags
- (FGI + FGO) → control functions
- + → Logic OR

The symbol + between FGI and FGO in the control functions shows a logic OR operation. This operation is AND with IEN and timing signals T_0', T_1', T_2'. The fetch and decode phases of the instruction cycle must be modified and replace T_0', T_1', T_2' with RT_0, RT_1, RT_2. Therefore the interrupt cycle statements are:-

(Address register)

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

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

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

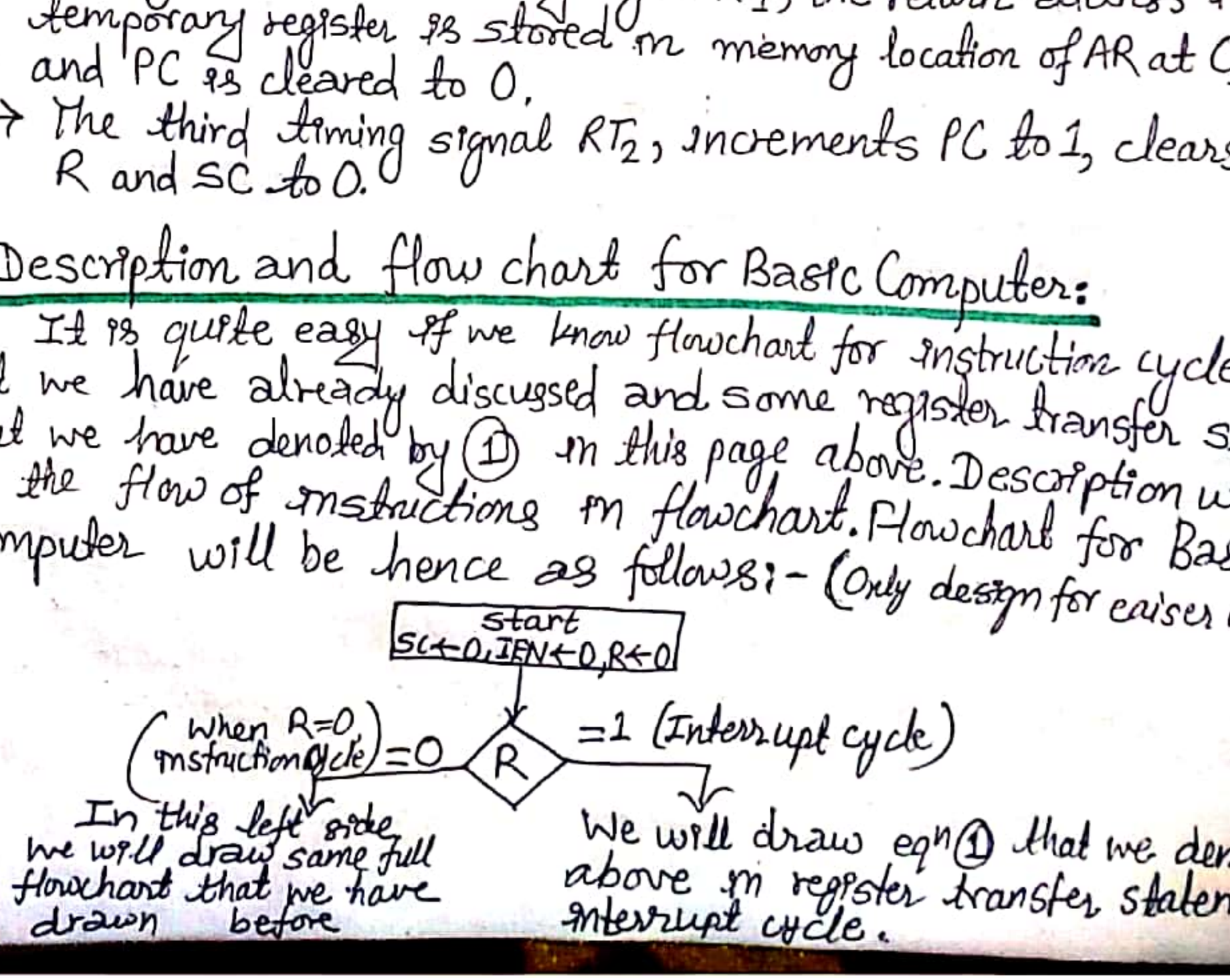
i.e→ During the first timing signal RT_0, AR is cleared to 0 and the content of PC is transfered to temporary register(TR).

→ During the second timing signal RT_1, the return address in temporary register is stored in memory location of AR at 0, and PC is cleared to 0.

→ The third timing signal RT_2, increments PC to 1, clears IEN, R and SC to 0.

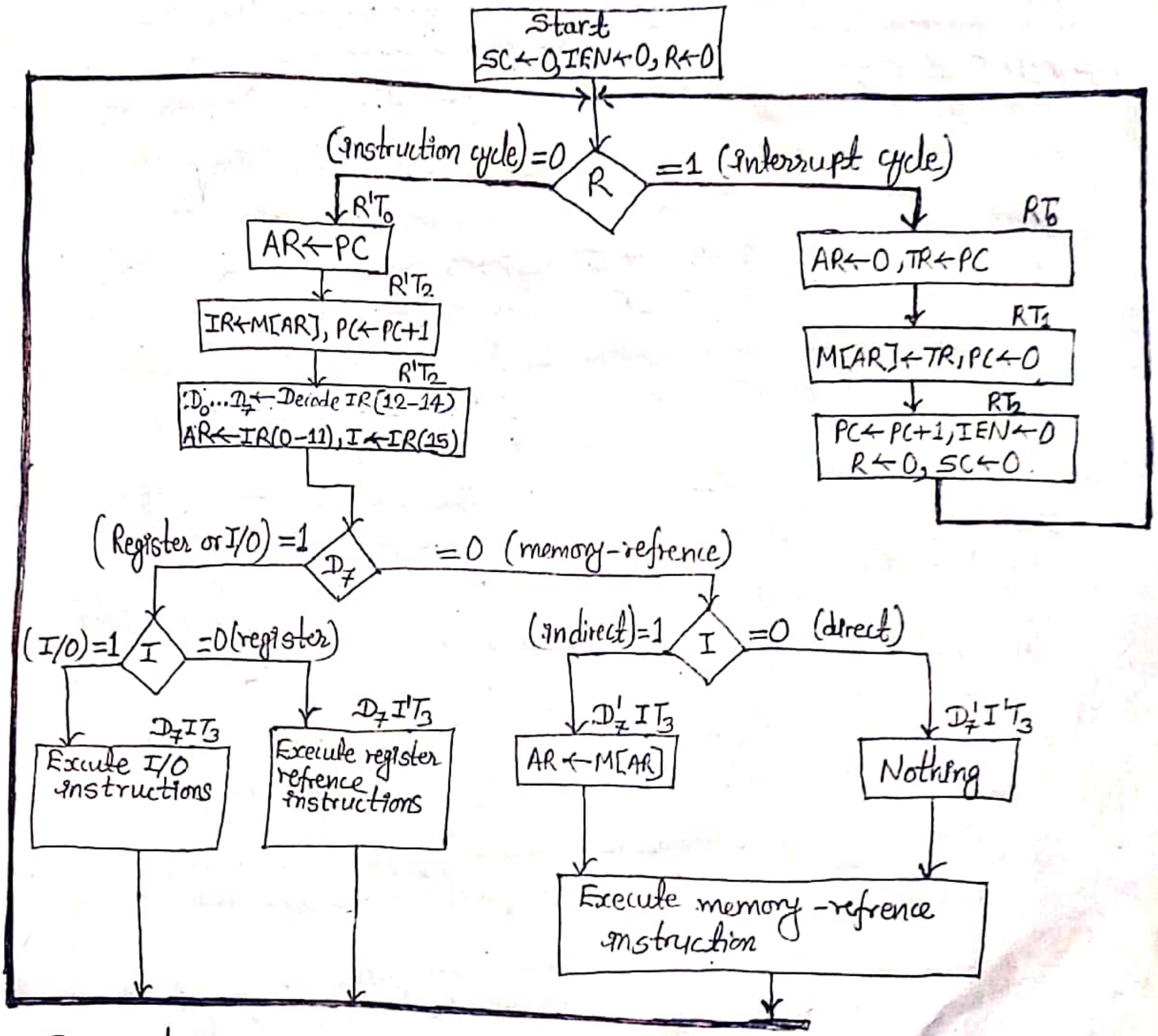
⊛. Description and flow chart for Basic Computer:

It is quite easy if we know flowchart for instruction cycle that we have already discussed and some register transfer statements that we have denoted by (1) in this page above. Description will be on the flow of instructions in flowchart. Flowchart for Basic Computer will be hence as follows:- (Only design for easier understanding)



This is a rough way for beller and short understanding for how to draw flow chart for Basic computer. The Actual flowchart and description are as follows:—

Flow chart:



Description:

→ Firstly the sequence counter is initialized to zero, interrupt enable (IEN) is initialzed to zero and R flip-flop is also initialized to zero.

→ R flip-flop will decide whether it is instruction cycle or interrupt cycle.

→ If R=0, then it will be instruction cycle & if R=1, it will be interrupt cycle.

PC → holds address of next instruction.

In instruction cycle before was only T_0, but now we are combining interrupt and instruction cycle so we use $R'T_0$ which will decide interrupt or instruction cycle
$R'=0$ = instruction cycle
$R=1$ = interrupt cycle.
dash (') represents zero.

Instruction cycle → At time $R'T_0$, the address of next instruction is to be copied to address register.

→ At time $R'T_1$ the instruction is fetched from memory pointed by address register and stored in instruction register and increment is done in program counter by one.

→ This ($R'T_0$ & $R'T_1$) completes our fetching part of instruction now we need to decode it.

→ From 12 to 14 bits of IR register decoding will take place and corresponding D_0 to D_7 bits will be generated along with that 0 to 11 bits of IR register are also copied to AR register and 15th bit of IR register is copied into I.

→ After decoding now we are going to check D_7 bit. If D_7 bit is 1 then it is either register or I/O instruction & if D_7 bit is 0 then it is memory-refrence instruction.

(a) If it is register or I/O instruction then we check I-bit. If I bit is 1 then we need to execute input output instruction and initialize sequence counter to zero.
But if I=0, then it is register refrence instruction so we need to execute register refrence instruction and initialize sequence counter to zero.

(b) If it is memory-refrence instruction (i.e, $D_7=0$) then we follow two parts direct and indirect addressing according to I-bit.
If I-bit is 1 then it is indirect addressing so at time T_3 effective address is fetched from memory and stored in AR register. After some time memory refrence instruction get executed and sequence counter becomes zero.
But if I-bit is 0 then direct addressing so at time T_3 we are supposed to do nothing and simply memory-refrence ~~int~~ instruction get executed with sequence counter zero.

→ Finally the execution again goes to fetch part (i.e, R).

Interrupt cycle → Interrupt cycles RT_0, RT_1, RT_2 are discussed below (1) same thing we write here. It is short.