Computer Organization and Architecture

Architecture

* Refers to attributes of the system, visible to the programmer have direct impact on the logical execution of the program
* low level concept/logical unit
*. What the system do/ functionality "what"
*. Defines the system abstract model
* Instruction type, Addresing modes, datatypes
Ex:

Organization

* Refers to operational units that are interconnected by which that achieves the architectural specification.
* High-level concepts
* How the functionality is implemented
* Realization of abstract model
*Ex:- Circuits like Adders, substractors, Registers etc.

Basic Components of a Computer (or) Functional units of a computer

3 basic components of a computer

1. Central processing unit (CPU) - processor
2. Memory
3: Input/output.

1: Central processing unit.

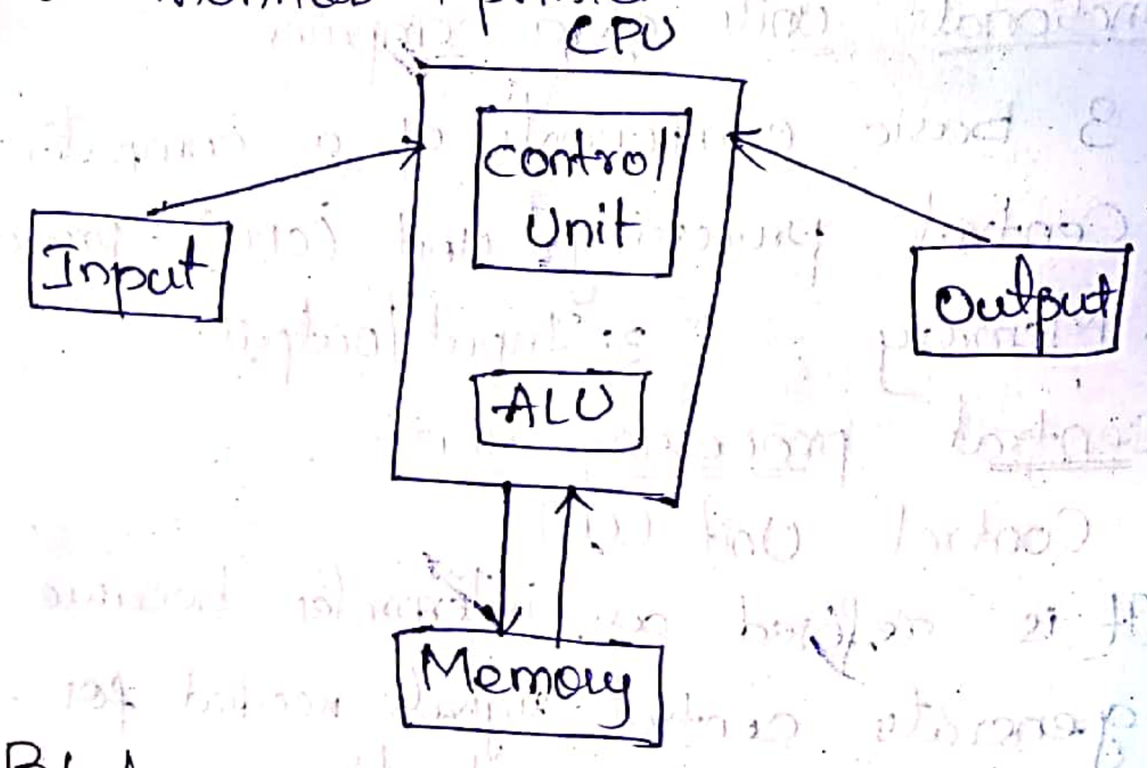
* Control Unit (CU)

It is refered as interpreter because it generates control signals needed for interpreting the instructions

* Set of register
Arthimetic and logical Unit (ALU)
* It performs all Arthmetic operations like -, +, *, /
It performs all logical operations like AND, OR, NOT

2. Memory :- Memory is a location where data or programs are stored.

3. Input/Output :-
Input module :- Ex:- keyboard, mouse
It accepts the input from user, converts to machine understandable.
Output module :- It displays the result or output.
Ex:- monitor, printer



Block diagram of a computer

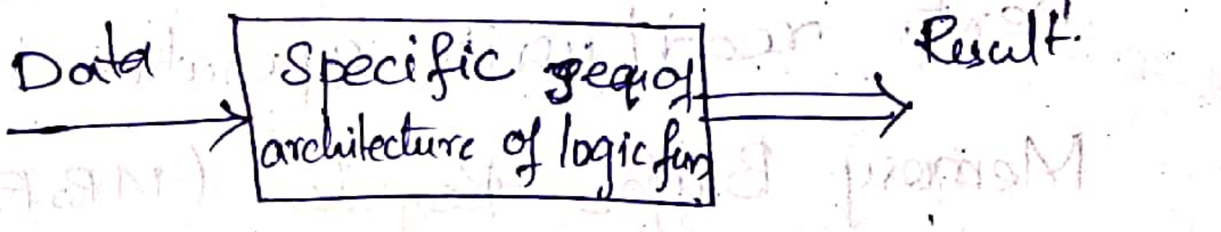
Von Neumann Architecture :-

* Almost all computer designs are based on concepts that are developed by Von Neumann such designs are reffered to as Von-neumann architecture

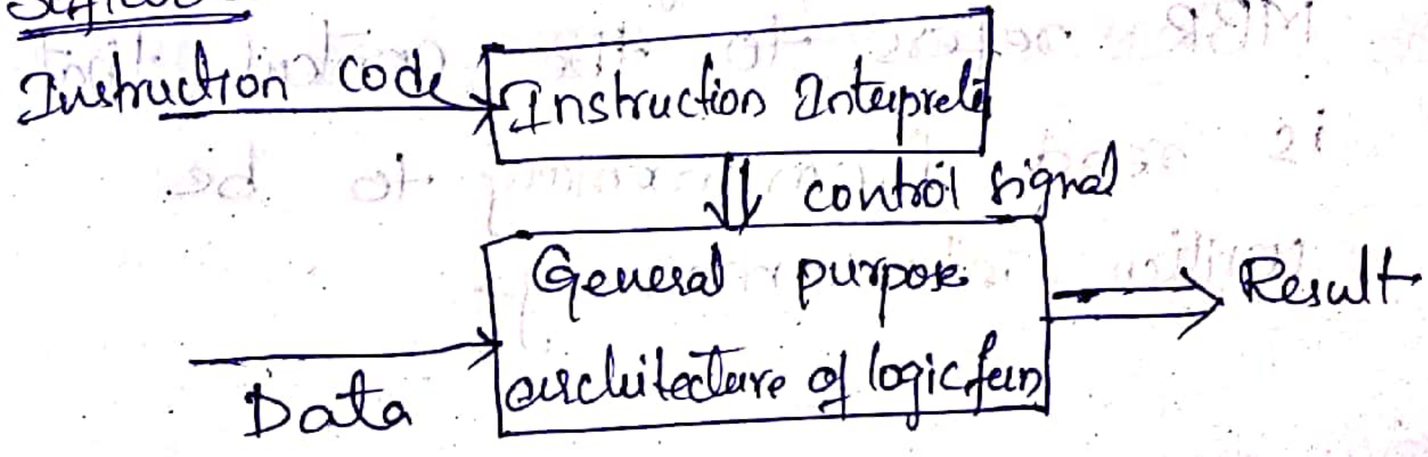
* Concepts
There are mainly 3 concepts to design.
1. Memory :- Data & instructions are stored in a single read-write memory
2. Memory content :- Content of memory is addressable by locations irrespective of type of data instructions.
3. Execution :- Execution will be done in a sequential manner unless it is changed by jumps or goto.

Hardware & Software approach

Hardware



Software



Computer function (or) Interconnection of components

The data has to be exchange between the component in order to execute an instruction. Memory, CPU, I/O are interconnected

Memory & CPU can exchange & CPU & I/O the data has to be exchanged.

* Data has to be exchanged between memory & CPU to execute an instruction With the help of 2 registers data has to be transformed. Those are

1. Memory Address register (MAR)
2. Memory Buffer register (MBR)

Memory Address register (MAR):—
MAR specifies in memory for next read/write operation.

Memory Buffer Register (MBR):—
MBR refers to the content that is read from memory to be written into memory.

2 registers

1. I/O Address register :— (IOAR)
2. I/O Buffer register : (IOBR)

1. I/O Address register :—
It specifies the address of I/O modules, to interact with the CPU

2. I/O Buffer register :—
It specifies the content from the I/O module or content to the I/O module

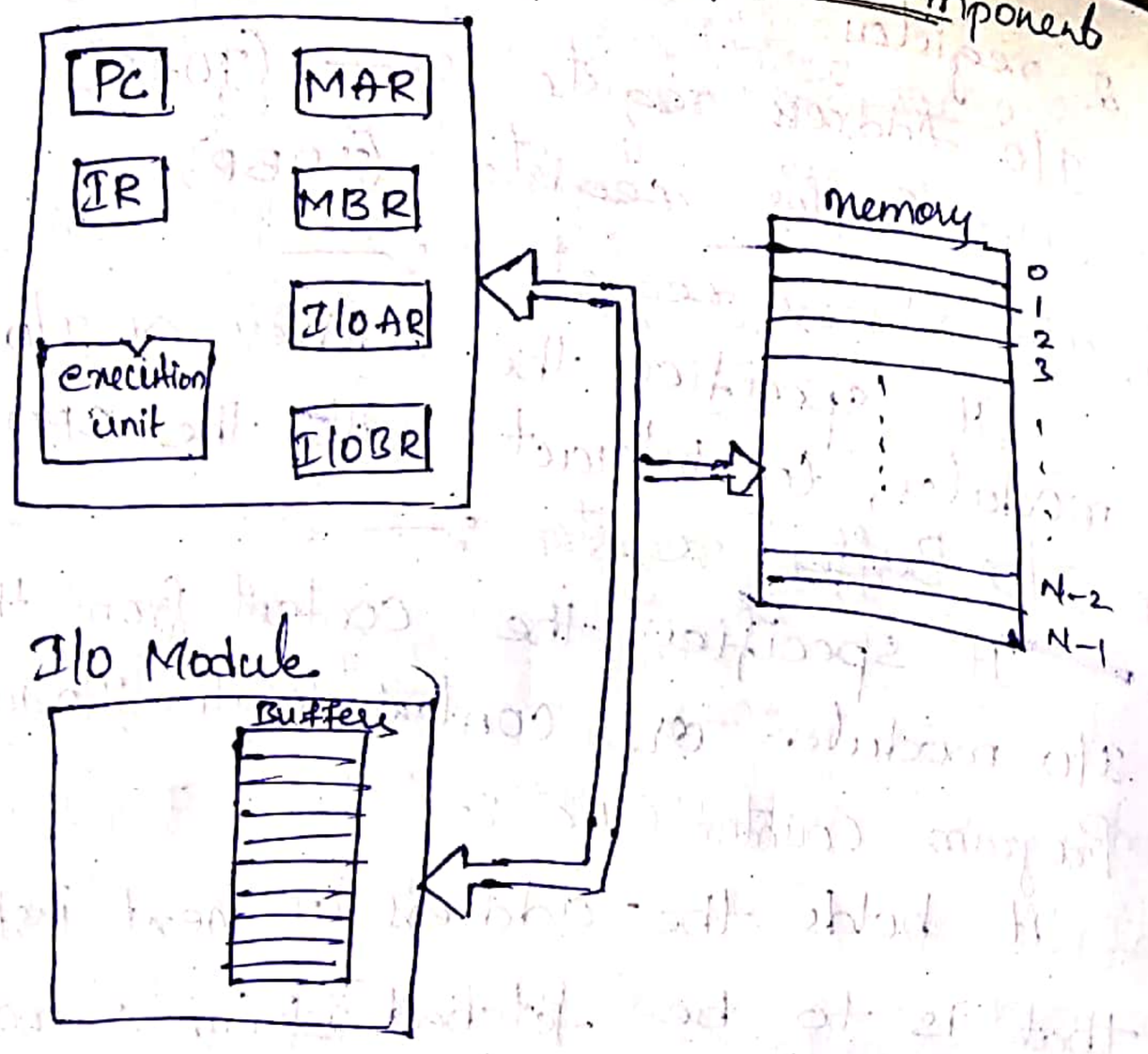
Program Counter (PC) :—
It holds the address of next instruction that is to be fetched from memory

Instruction Register (IR) :—
It holds the instruction code that was read from memory

Accmulator (AC) :—
It holds the data temperarily or it the result after the completion of procer.

Top-level view of Basic computer compone

Top level View of Basic computer components



Instruction execution (or) Instruction cycle

Basic function of a computer -

Execution of a program

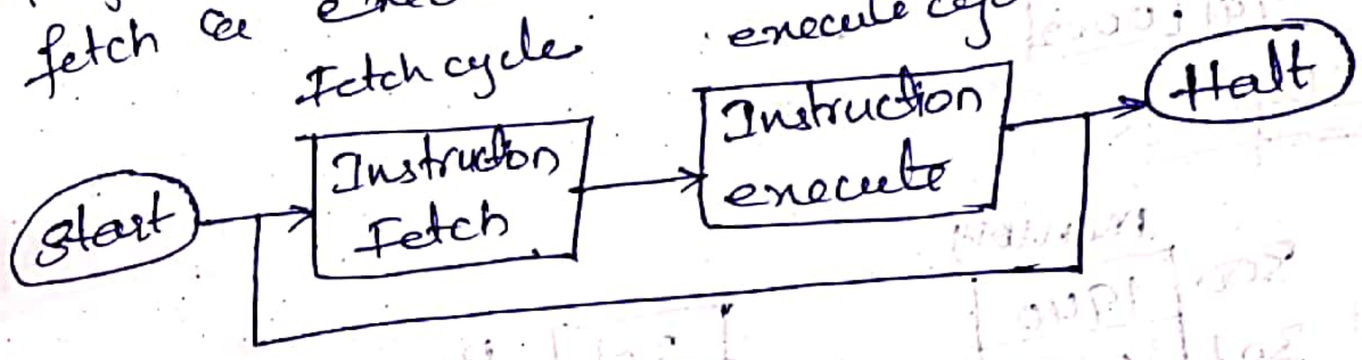
Program:- Set of Instructions

*Processing of a single instruction at a time is called as instruction cycle

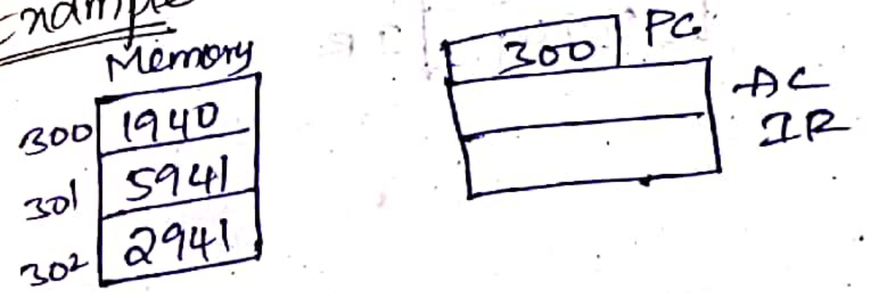
Steps:-

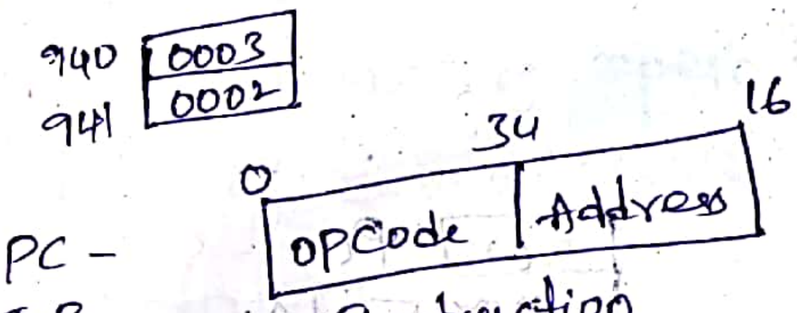
1. Fetching (Reading) the instruction from memory (Fetch cycle)
2. Executes the instruction (Execute cycle)

* Instruction cycle is a combination of fetch cycle & execute cycle
* Program execution is repeated process of fetch & execute cycle



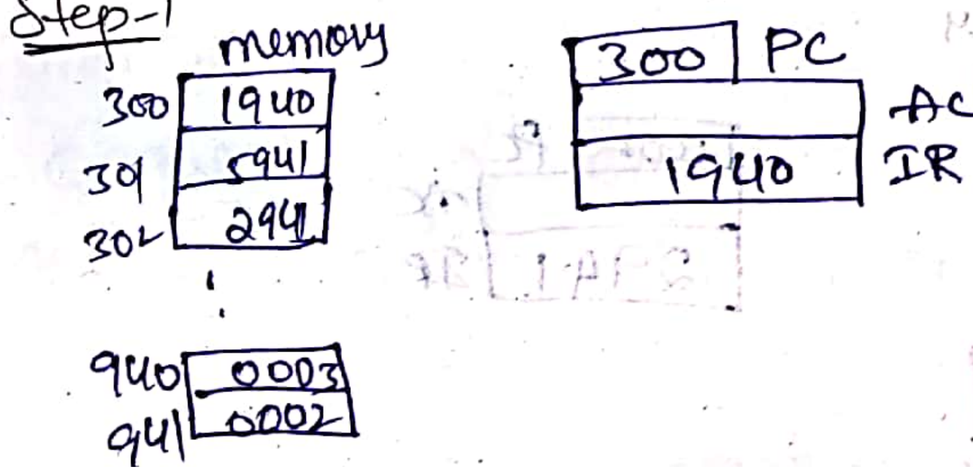
Example



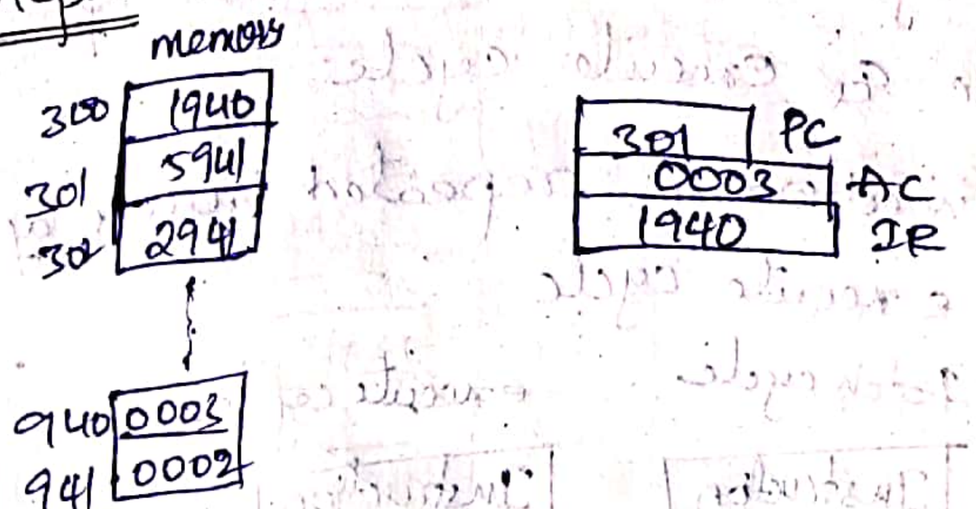


1. 0001 - load AC from memory.
2. 0010 - Store AC to memory
5. 0101 - Add AC to memory

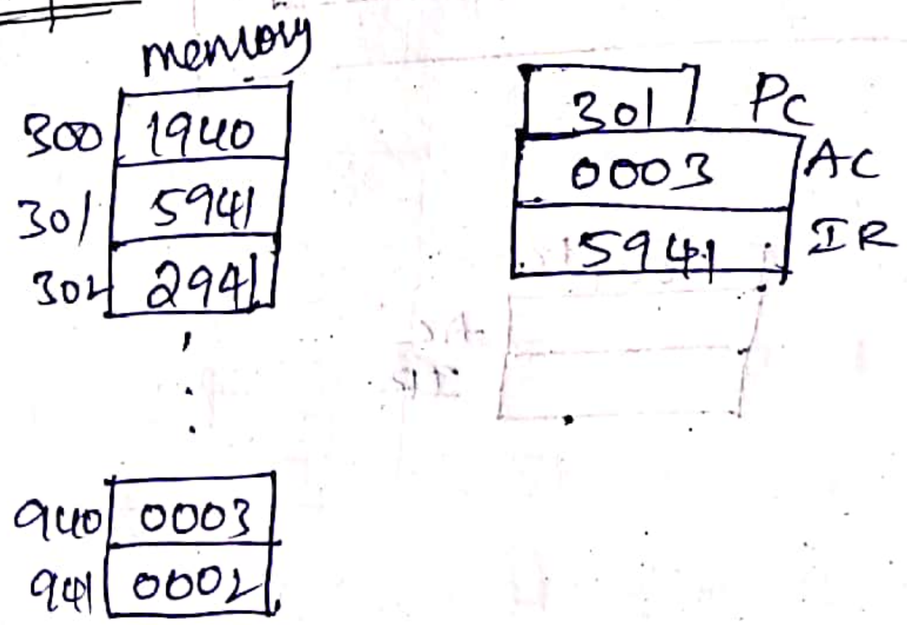
Step-1



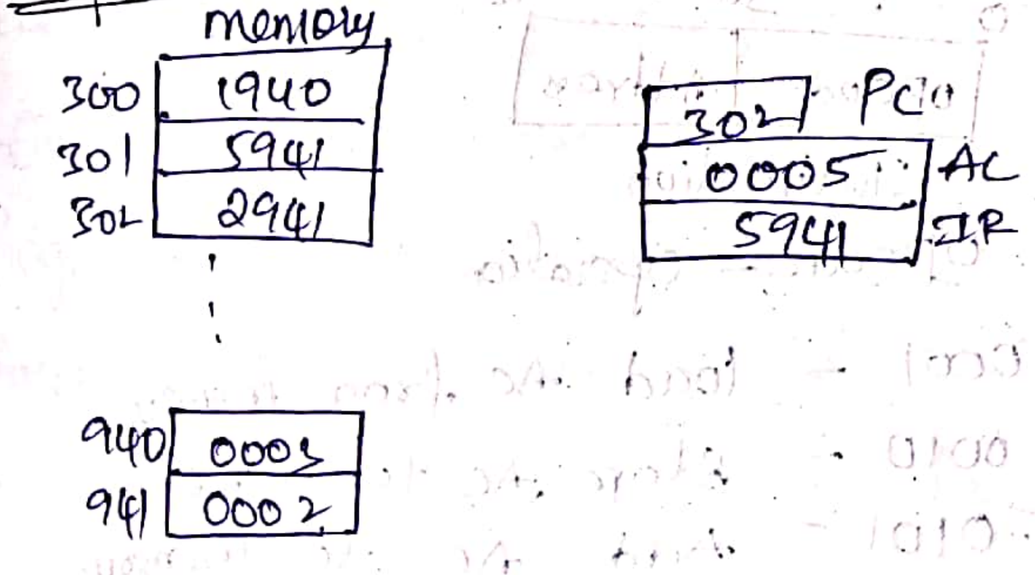
Step-2



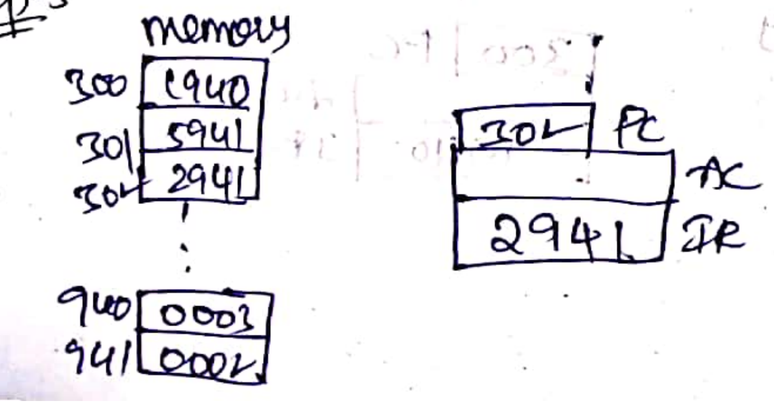
Step-3



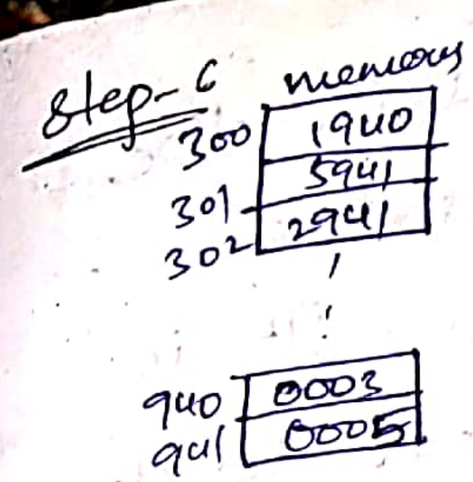
Step-4



Step-5



Step-6



941 location stores 0005

RTL interpretation of instruction

RTL - Register Transfer Language

Digital System :- Interconnection of modul Each module has its specific informat processing task.

Describe Module :- Registers, operation on data of register.

Micro Operations: the operations executed on data of a register is called as Micro operations.

~~Digital~~ system

Internal Hardware Organization of a digital computer is best defined by

1. set of registers it contains and their functions
2. Sequence of micro operations performe on information stored on a register

3. The control that initiates the sequence of micro operations.

Register transfer language :- It is a symbolic notation used to describe the micro operations transfers among the registers.

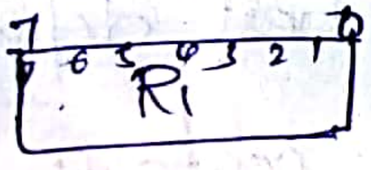
Ex: Transfer of content from R_1 to R_2 is represented as $R_2 \leftarrow R_1$

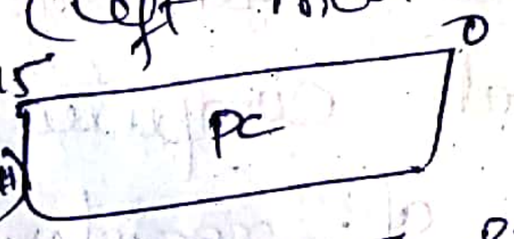
Register :- It is a collection of flipflops used to store data or information

A register is designated by letters followed by numerals MAR, MBR, PC, R_1, R_2

A register is represented by rectangular box

Register bits of number from 0 (right most bit) to n-1 (left most bit)

Ex:- 



PC(L) 0-7, 8 to 15

if (P==1) then $R_2 \leftarrow R_1$ language
↓ control function

P: $R_2 \leftarrow R_1$

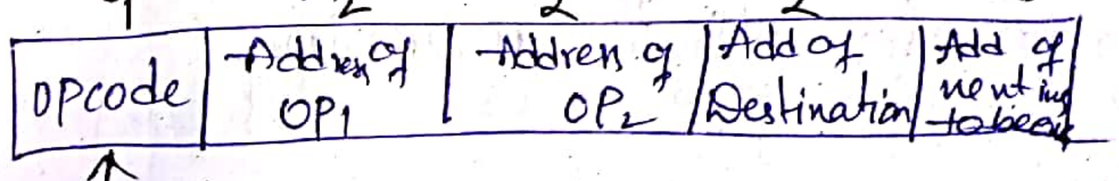
* Control function is a boolean variable that is equal to 1 or zero, and which controls the micro operation.

Ex: P: $R_2 \leftarrow R_1$

Symbol	Description	Example
letters followed by numerals	Denotes the Register	MAR, PC, R_1, R_2 - - - -
Parenthesis ( )	Denotes part of a register	PC(L), PC(H)
Arrow	Denotes transfer of information	$R_2 \leftarrow R_1$
Comma (,)	It is used to seperate two micro operations	$R_2 \leftarrow R_1$, MAR $\leftarrow$ PC

Instruction Set Architecture (ISA)

Instruction :- It is a considered as a word of Machine's Language convey : what operation Processor its has to do.



↑ Explicitly

Explicitly

1. It is lengthy
2. More memory
3. more execution time

Implicitly

By using stack, pointers we can reduce to implicity

Explicitly Specifying the instruction

By Explicitly specifying the instruction

1. We have long instruction
2. More memory spaces
3. More execution time.

The Size of an instruction can be reduced, by implicitly specifying the instruction

Ex:- PC, AC

Instruction Set Architecture :-

It is a structure of a computer which machine language programmer should understand. to write a correct program to that machine

* An ISA defines the operations the processor performs the data transfer mechanism & how to transfer data different control mechanism & it is an essential contact between software & Hardware
* ISA is important not only for programmer but also for designer who design & implement the processor.

ISA - visible for programmes

* Register (Store the data)
* Addressing modes (how to acces the data
* Instruction format (how to specify instructio
* Exceptional conditions
* Instruction set (what operation)

Classifications of ISA's

Based on means of storage

All of

1. Single Accumulate Architecture →
2. Stack Architecture
3. General purpose register

Single Accumulator Architecture

Operands stored in the accumulator

one register architecture

Stack Operands on the top of the

Architecture Stack

General purpose register :-

Operands are in specified register

or in memory

Evolution of ISA's :-

Single Accumulator

(1953, IBM 700 series)

Stack

(1960-1970's

HP-3000,

Borroughs)

General purpose Register

(1987 - - -

IBM RS 6000 - - -)

Architecture	Source operands	Destination
1. Stack	Two top elements of the stack	Top
2. Single Accumulator	Accumulator (or) Memory	Accumulator
3. General purpose Register	Register (or) Memory	Register or memory

* Consider $C = a + b$ & perform operations

$C = a + b$

Stack	Single Accumulator
push a	load a
push b	Add b · AC ← {AC} + M[5]
add	store c
popc	

General purpose register

load R1 a

Add R1 b

store C R1

Instruction code

Program - Set of instruction

Instruction code : Processor's language coord group of bits that will specify sequence of microoperations that are to be executed by a processor.

- Instruction code :- Divided into 2 fields.

Opcode	Address

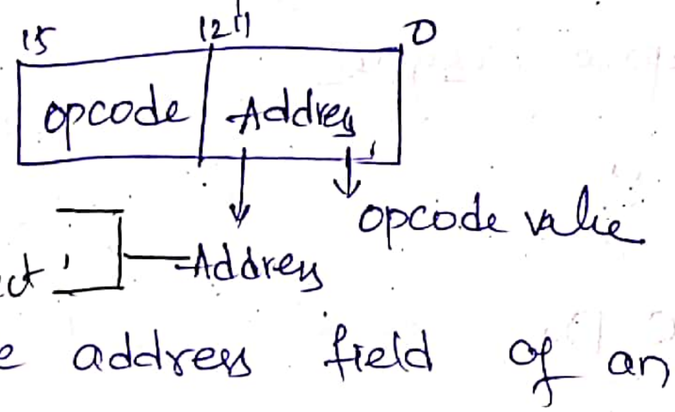
1. Opcode field
2. Address field

Opcode :- Operation code

* An instruction code is a group of bits that specify the sequence of micro operation

opcode field:- It stands for operation code opcode specifies what operations the processor is supposed to do.
Ex:- add, sub, mul, shift, complement

Address field :- Address field of an instruction code where the operation is for the processor

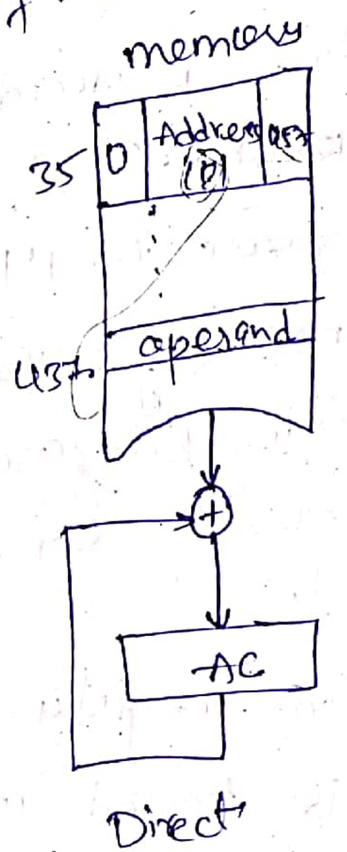


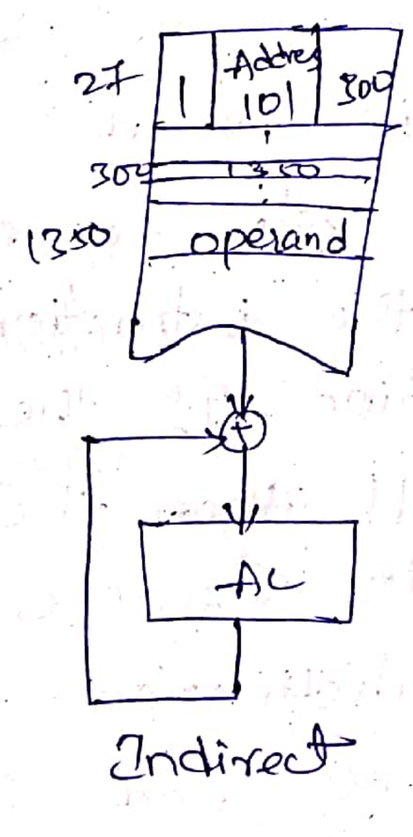
The address field of an instruction code can have operand value or address
If the Instruction have operand value then we call that instruction as immediate instruction. If we have the address then the address can be direct or indirect address.

Direct addressing mode :- If the address field of the instruction code have the effective address of the operand then it is called direct addressing mode.

Indirect addressing mode :- If the address field of an instruction code have some address where the effective address of an operand is present it is called as indirect addressing mode
The 15th bit of the instruction code is designated for this purpose and it is represented by I
If I=0 then it is direct addressing
If I=1 then it is indirect addressing.

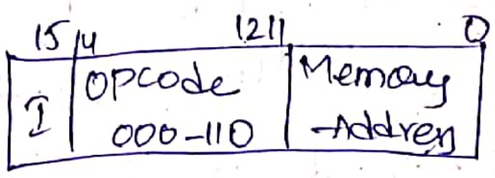
Ex



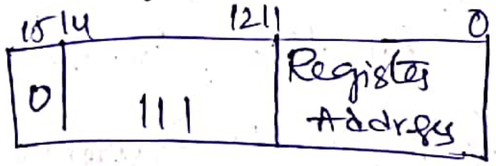


Instruction types :- 3 types.
1. Memory - Reference - Instruction
2. Register - Reference Instruction
3. I/O instruction

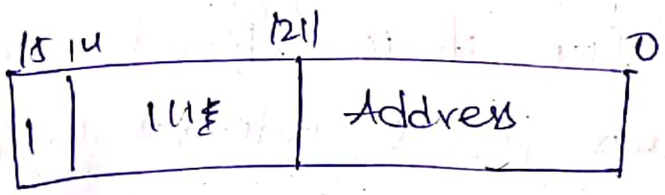
Memory- Reference Instruction.



2. Register Reference Instruction



3. I/O Instruction.



* The type of the instruction is decided by the opcode bits (12-14). If the opcode bits are not equals to 111 then the instruction is memory reference instruction. If the opcode bits are equal to 111 then the instruction can be register reference or I/O.
* The distinction between register refrence and I/O is based upon the 15th bit. If the 15th bit = 0 Register reference.
If the 15th bit = 1 then it is I/O.

Hex code

Instruction	I=0	I=1	Description
AND	0xxx	8xxx	AND → Accumulation of memory
ADD	1xxx	9xxx	ADD operation of AC of memory content
LDA	2xxx	Axxx	load AC from memory
STA	3xxx	Bxxx	store from AC to memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch & save the writen address
ISZ	6xxx	Exxx	increment & zero (0)

BUN = Branch unconditionally

Register Reference Instruction

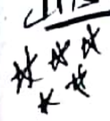
Instruction	codes	Description
CLA	7800	clear Accumulator
CLE	7400	clear Enable
CMA	7200	Compliment Accumulator
CME	7100	Compliment Enable
CIR	7080	Circular accumulator Enable to right

Instruction	codes	Description
CIL	F040	circulate AC & enable to left
INC	F020	Increment AC
SPA	F010	skip instruction if AC is +ve
SNA	F008	skip instruction AC is -ve
SZA	F004	skip instruction of AC = 0
SZE	F002	skip enable if = 0
HLT	F001	Halt the instruction

I/O Instructions

INT	F800	Input a character to the accumulator
OOT	F400	output a character from the accumulator
SKI	F200	skip input flag
SKO	F100	skip on output flag
ION	F080	Interrupt ON
IOF	F040	Interrupt OFF

Instruction format



1. Three-address instruction format
2. Two-address instruction format
3. One " " "
4. Zero " "

Three address instruction format :-

$X = (A+B) * (C+D)$ — 3 addreses each → memory, re

```
ADD  R1  A  B ; R1 ← M[A] + M[B]
ADD  R2  C  D ; R2 ← M[C] + M[D]
MUL  X   R1 R2; M[X] ← R1 * R2
```

Advantage :- less program

Disadvantage :- bits will be more

Two-address instruction format :-

Consists two addreses from that one is source and it also acts as destination

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

```
MOV  R1  A ; R1 ← M[A]
ADD  R1  B ; R1 ← R1 + M[B]
MOV  R2  C ; R2 ← M[C]
ADD  R2  D ; R2 ← R2 + M[D]
MUL  R1  R2; R1 ← R1 * R2
MOV  X   R1; M[X] ← R1
```

Adva :- Bits will be less

Disadv :- More program.

One-Address Instruction format

One address - Memory, register

Implicit Register - AC

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

```
LOAD   A ;  AC ← M[A]
ADD    B ;  AC ← AC + M[B]
STORE  T ;  M[T] ← AC   {T = A+B}
LOAD   C ;  AC ← M[C]
ADD    D ;  AC ← AC + M[D]
MUL    T ;  AC ← AC * M[T]
STORE  X ;  M[X] ← AC
```

Zero-Address Instruction format:-

NO address field in the instruction

Stack organization uses this instruction format

For Push & pop operations we use single address

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

```
PUSH   A ;  Top ← M[A]
PUSH   B ;  Top ← M[B]
ADD
PUSH   C ;  Top ← M[C]
PUSH   D ;  Top ← M[D]
ADD
MUL
POP    X ;
```

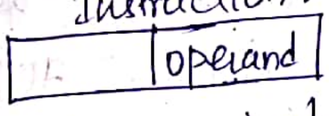
Addressing Modes

Types of addressing modes

1. Implied (or) Stack.
2. Immediate
3. Direct
4. Indirect
5. Register
6. Register Indirect
7. Displacement
8. Autoincrement or autodecrement
9. Relative
10. Indexed
11. Base Register

* An addressing mode specifies how to fetch an operand for an operation.

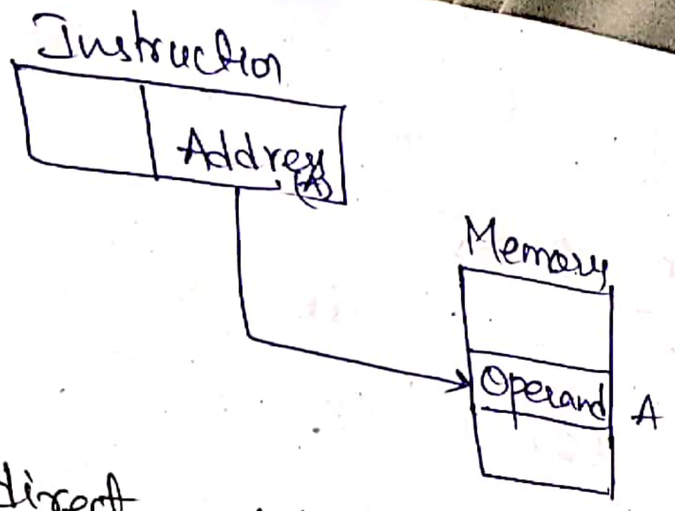
1. Immediate Addressing mode:-



If the address instruction directly has the operand then it is immediate Addressing mode

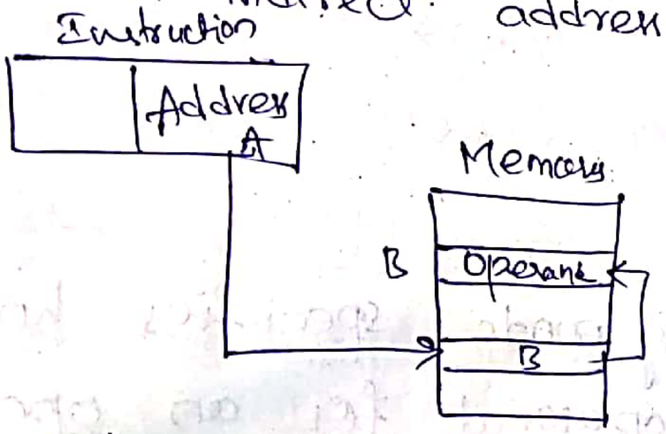
2. Direct Addressing mode:-

If the address field of the instruction has the effective address of the operand then it is called as direct Addressing mode



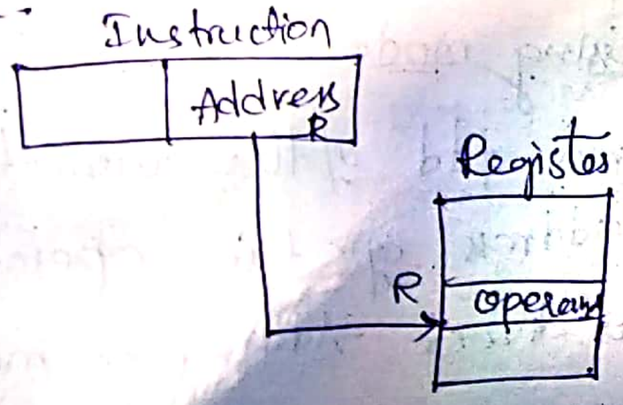
Indirect Addressing mode

If the address field of the instruction has some address where the effective address of the operand is present that is called indirect addressing mode



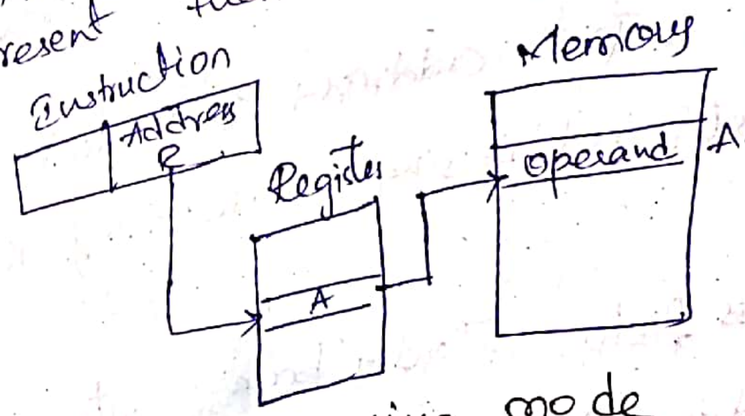
Register Addressing mode :-

If the address field of the instruction has register address where the operand is present then it is register addressing mode.



Register Indired addressing mode :-

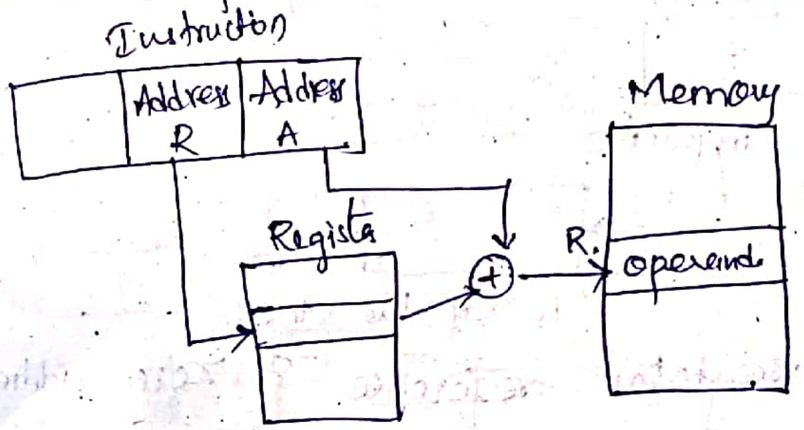
If the address field of the instruction has some register address where the effective memory address of the operand is present then it is Register Indirect addressing mode



Displacement addressing mode

Displacement is a combination of register indirect and direct addressing modes.

In displacement addressing mode, the address field has 2 addresses among which one is implicit address.



There are 3 common uses of displacement mode. They are

1. Relative
2. Base Register
3. Indered.

Relative Addressing mode: If the address field content of an instruction is added with the program Counter register content to get the affective operand address of the object then it is relative addressing mode

* EA = field content address + PC content

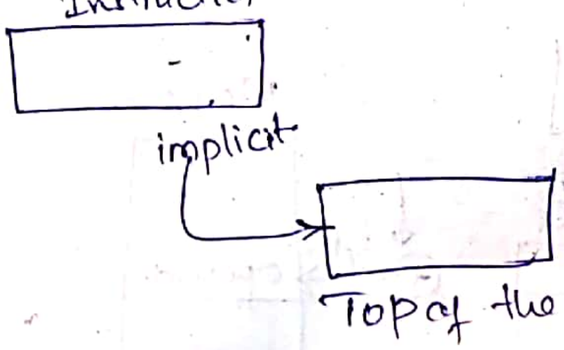
Base Register addressing mode:- If the Affective.

EA = address field content + base Register content

Inden Rs addressing mode:-

EA = Address field content + inden register content

Implied or stack:- If the operand is implicitly provided for then it is Implied addressing mode



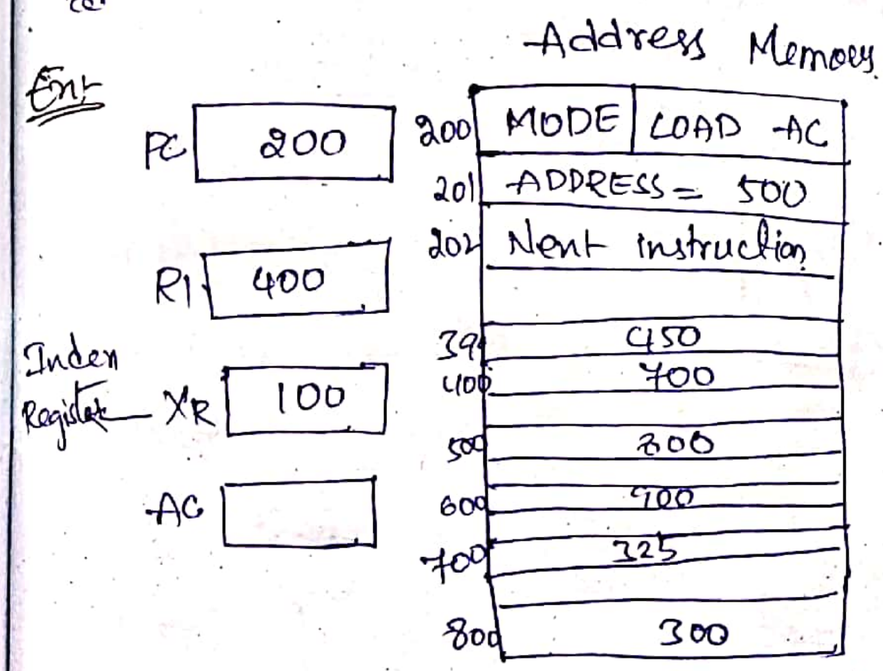
All the register reference & zero address instructions are implied.

Autoincrement (or) Auto decrement

Autoincrement (or) Autodecrement is similar to register indirect except that the register content is autoincremented or autodecremented after (before) the instruction is executed.

Addressing mode	Advantage	Disadvantage
1. Immediate	1. NO memory reference	1. Limited operand magnitude
2. Direct	2. Simple	2. Limited address phase
3. Indirect	3. large address phase	3. Multiple memory reference
4. Register	4. NO memory reference	4. Limited address space
5. Register Indirect	5. Large address space	5. More reference of memory
6. Displacement	6. Flexibility	6. Complexity
7. Implied (or) stack	7. No memory reference	7. limited Applicability

Ex.



Addressing mode	Effective address	Opcode width
1. Direct	500	800
2. Indirect	800	300
4 Register	No	400
3. Immediate	201	500
5. Register Indirect	400	700
6. Displacement Relative	500+200 =700	325
7. Index	500+100 =600	900
8. Auto increment	400	700
9. Auto decrement	399	450