⇒ **Addressing Modes:-**

The Addressing Mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

→ **Implied mode:-**

- operands are specified implicitly in the definition of the instruction.

  eg:- complement accumulator

- All register reference instructions that use an accumulator are implied mode.
- zero-address instructions are implied-mode

→ **Immediate Mode:-**

- operand is specified in the instruction itself.
- This mode has operand field rather than an address field.

→ **Register mode:-**

- operands are in registers

→ **Register Indirect mode:-**

- The instruction specifies a register in the CPU whose contents give the address of the operand in memory.

→ **Autoincrement or Autodecrement mode:-**

The register is incremented or decremented after (or before) its value is used to access memory.

→ **Direct Address mode:**

Effective address is equal to address part of the instruction.

→ Indirect Address Mode:-
The address field of the instruction gives the address where the effective address is stored in memory.

→ Relative Address:-
The content of the program counter is added to the address part of the instruction in order to obtain the effective address.

→ Indexed Addressing Mode:-
The content of an index register is added to the address part of the instruction to obtain the effective address.

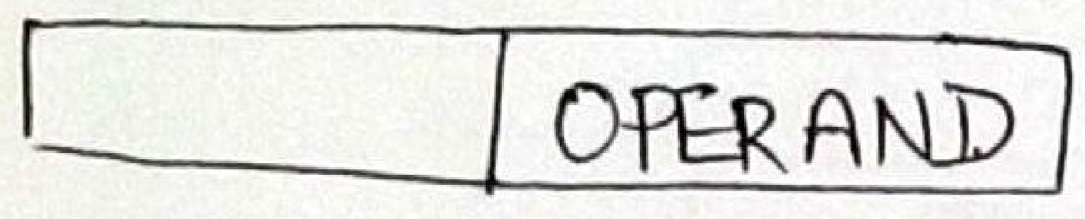
→ Base register Addressing Mode:-
The content of base register is added to the address part of the instruction to obtain the effective address.

Implied

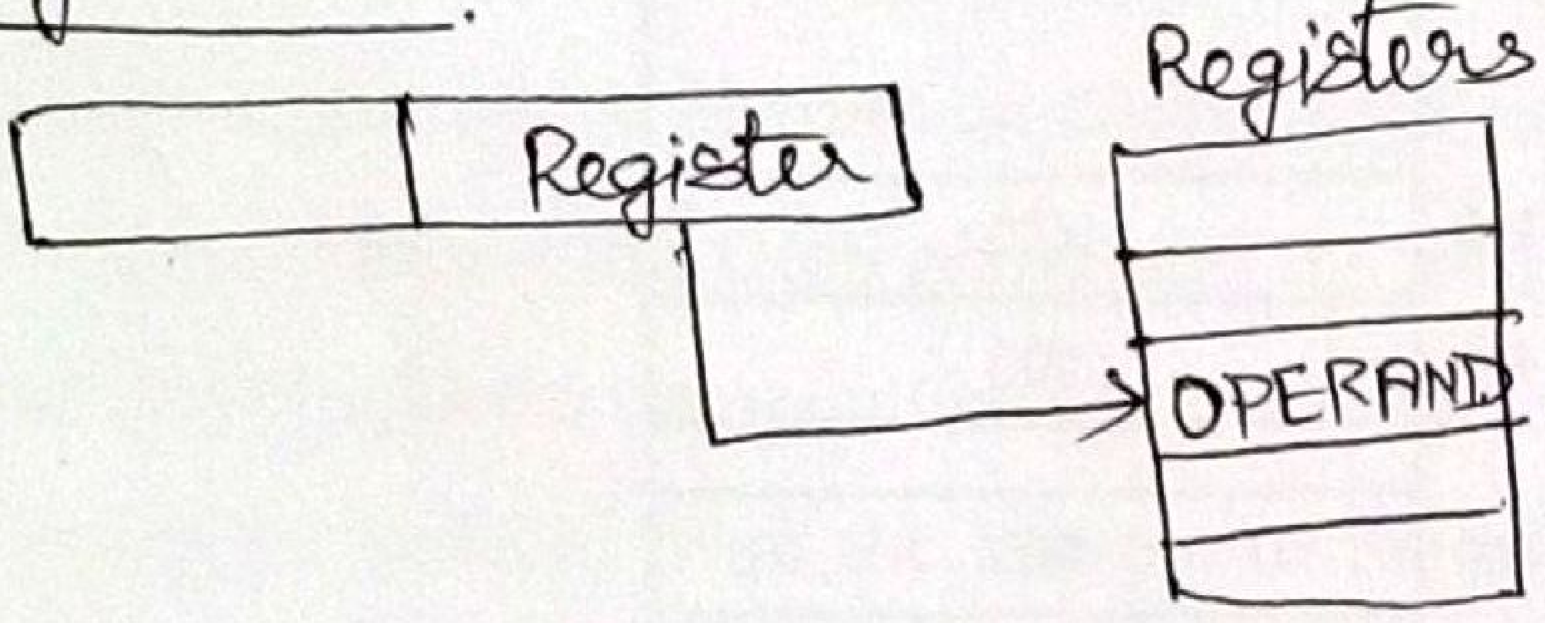
complement Accumulator

Zero Address Instructions.

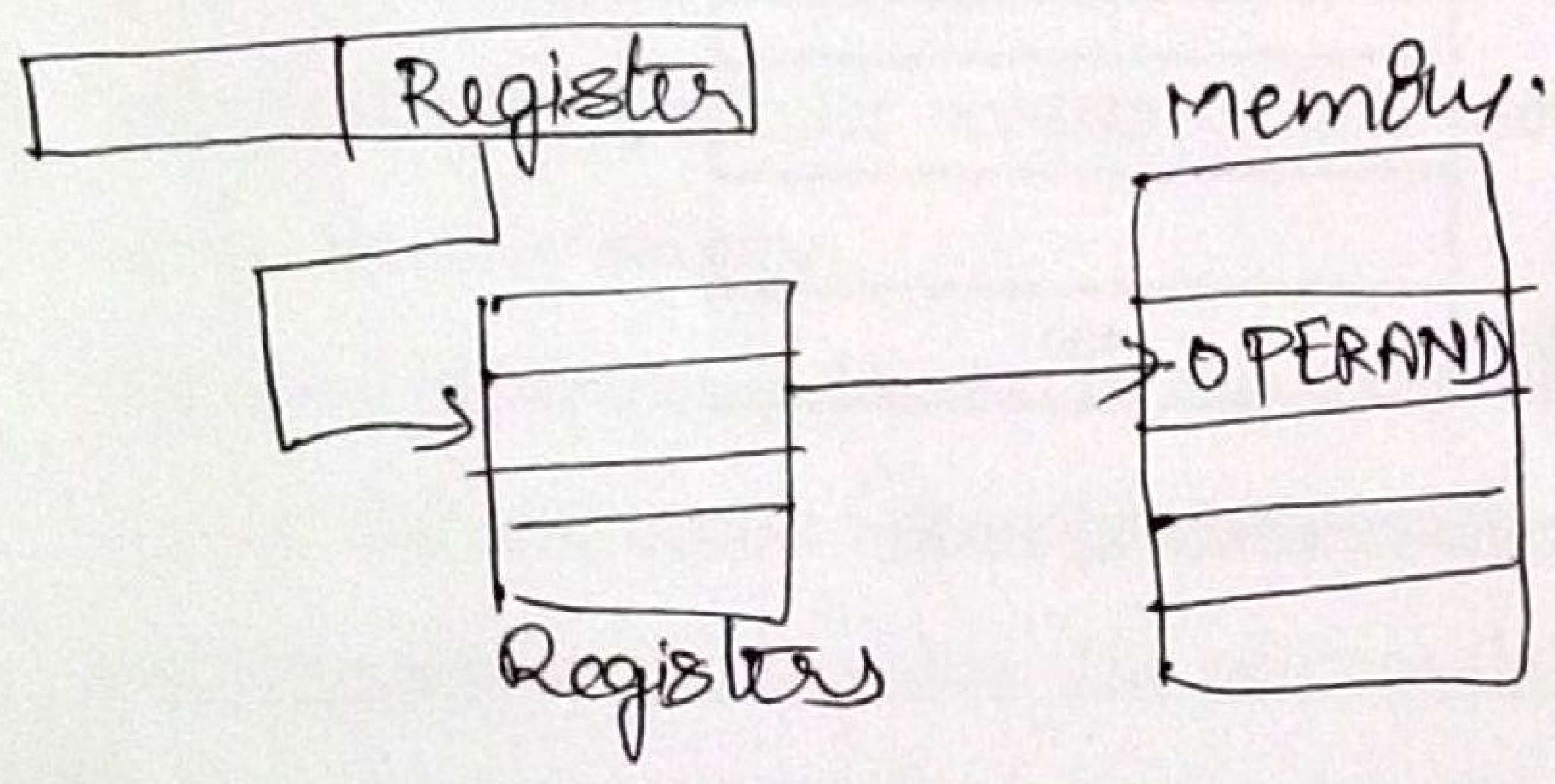
Immediate



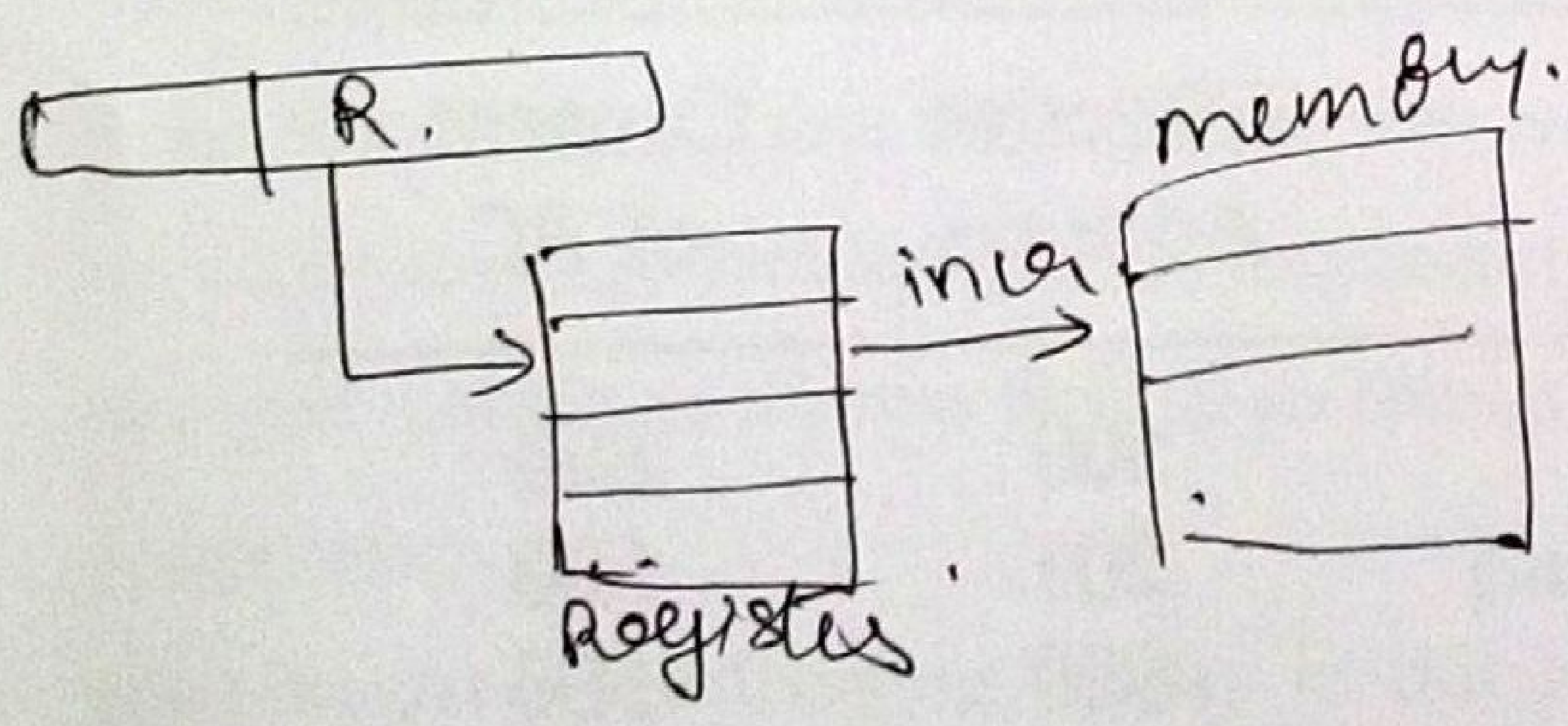
Register



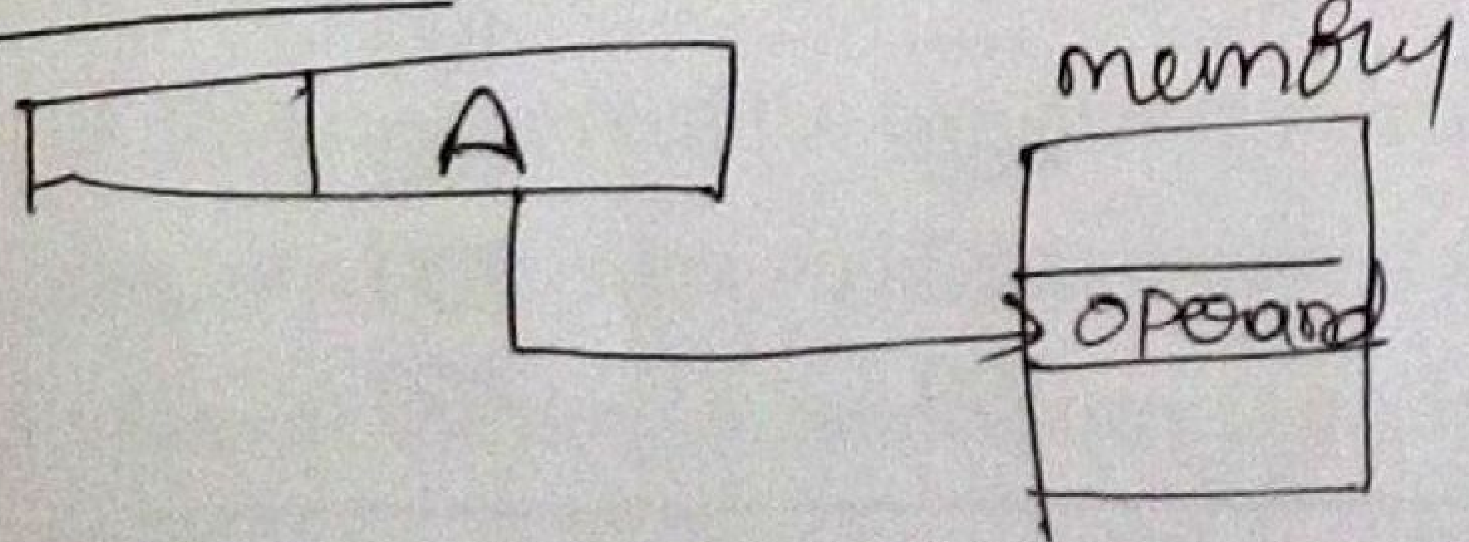
Register Indirect



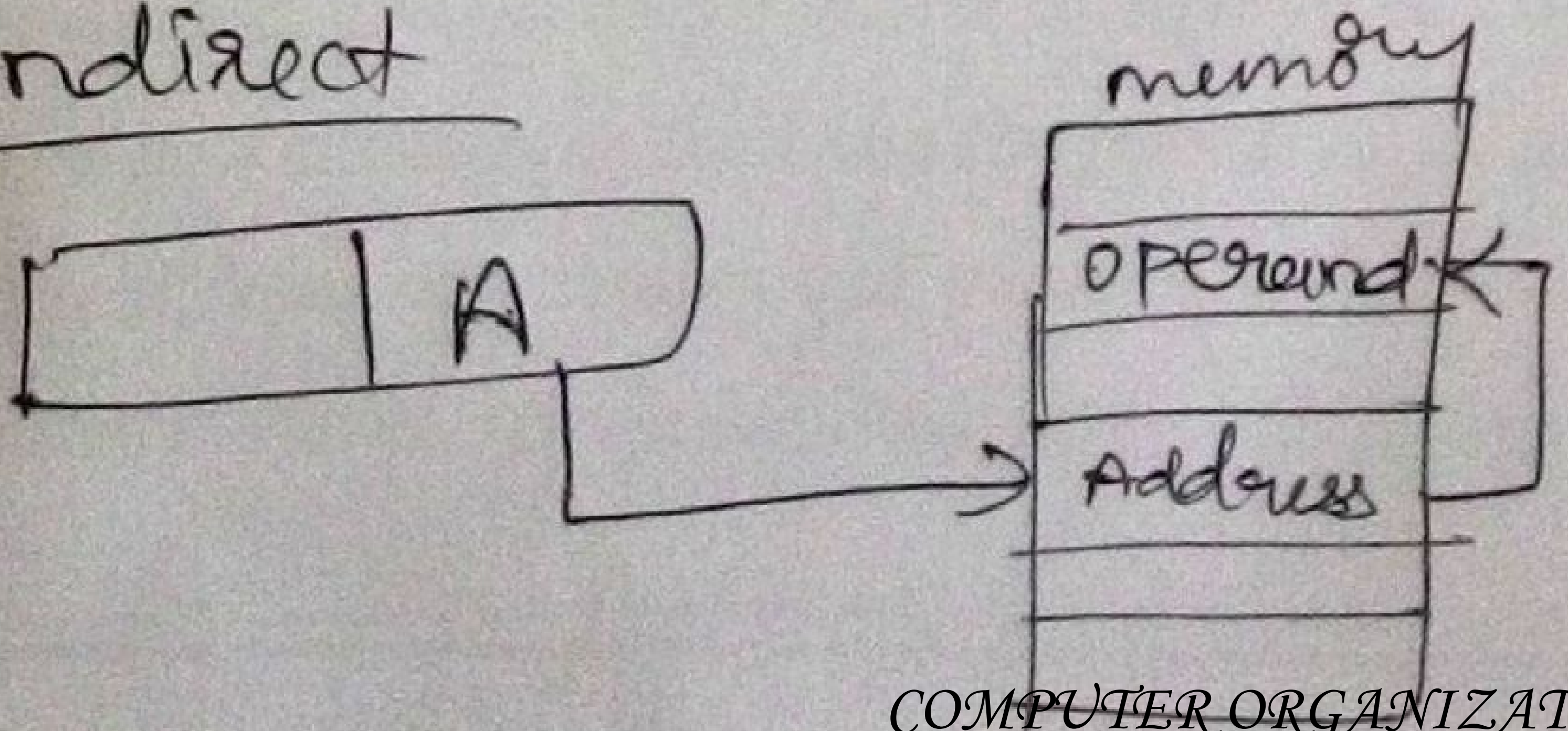
Autoincrement /decrement



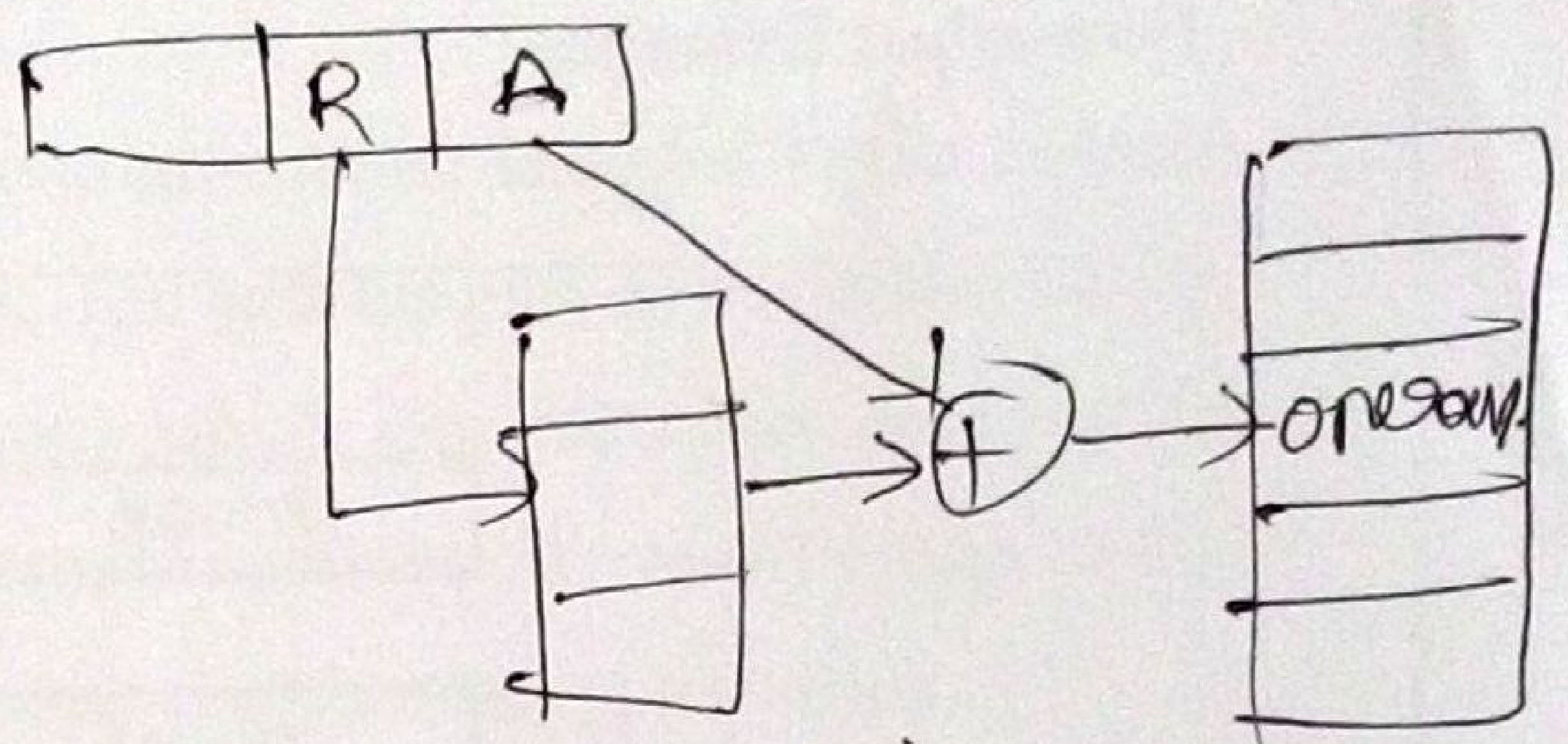
Direct



Indirect



Displacement



EA = A + (R).

Relative

R = PC

Indexed

R = index

A = Address

Base register

A = displacement

R = Address

⇒ Instruction Set Architecture Design :-

- The design of an instruction set architecture is arguably the most important step in the design of a microprocessor.
- A good way to start designing an instruction set architecture is to decide what should the instruction set architecture and its processor be able to do?
- A processor used for general purpose computing such as in a PC, requires large instruction set, But for specialized processor, the tasks ~~are~~ that are performed by this processor must known well in advance.
- Completeness of the instruction set architecture means the instruction set have all of the instruction a program needs to perform its required tasks?

$\boxed{\text{orthogonality}}$

- A good instruction set minimizes the overlap between instructions; this provides the programmer with the necessary functions in a minimum of instructions.
- Another area in which the designer can optimize the instruction set architecture is the register set. Registers. have a large effect on the performance of a CPU.

Having two few registers causes a program to make more references to memory, thus reducing performance.

Some other design issues for ISA are:

- Does this processor have to be backword compatible with other microprocessor.

  Backward compatibility is much more important for general-purpose microprocessors.

  For ex; if ISA of pentium II microprocessor were not backward compatible with ISA of pentium microprocessor, it would not be able to run code written for the pentium.

- What types and sizes of data will the microprocessor deal with?

- Are interrupts needed?

  If interrupts are included, the ~~IS~~ ISA must include the instructions and registers needed to process the interrupts.

- Are conditional instructions needed?

  Jump, flag - 1bit registers
  - zero flag.
  - carry flag.
  - sign flag

⇒ A Relatively Simple Instruction set Architecture:-

- Let's consider this microprocessor can access 64K ($=2^{16}$) bytes of memory, with each byte having 8 bits, or 64K×8 of memory.
- There are two types of Input/output Interactions.
  - Isolated I/O
  - memory mapped I/O.
- There are three registers in the ISA of this processor.
  - Accumulator (AC) - 8 bit Register
  - Register R is an 8 bit general purpose register.
  - 1-bit zero flag Z.
- The final component of the instruction set architecture for this Relatively simple CPU is its instruction set.

Instruction	Instruction code	operation.
NOP	0000 0000	No operation
LDAC	0000 0001 Γ	AC = M[Γ]
STAC	0000 0010 Γ	M[Γ] = AC
MVAC	0000 0010	R = AC
MOVR	0000 0100	AC = R
JUMP	0000 0101 Γ	GOTO Γ
~~JU~~JMPZ	0000 0110 Γ	IF (Z=1) then GOTO Γ
JPNZ	0000 0111 Γ	IF (Z=0) then GOTO Γ
ADD	0000 1000	AC=AC+R, if(AC+R=0) then Z=1 Else Z=0
SUB	0000 1001	AC=AC-R, if(AC-R=0) then Z=1 Else Z=0
INAC	0000 1010	AC=AC+1, if(AC+1=0) then Z=1 Else Z=0
CLAC	0000 1011	AC=0, Z=1;
AND	0000 1100	AC=AC∧R, if(AC∧R=0) then Z=1 Else Z=0
OR	0000 1101	AC=AC∨R, if(AC∨R=0) then Z=1 Else Z=0
XOR	0000 1110	AC=AC⊕R, if(AC⊕R=0) then Z=1 Else Z=0
NOT	0000 1111	AC=AC', if(AC'=0) Then Z=1 Else Z=0

44 - The instruction set contains 16 instruction, each having an 8-bit instruction code.

- The LDAC, STAC, JUMP, JMPZ, & JPNZ instructions all require a 16-bit memory address. These instructions require 3 bytes.

25: JUMP 1234H.

```
25: 0000 0101 (JUMP)
26: 0011 0100 (34H)
27: 0001 0010 (12H
```

Instruction code	byte 1
Low-order 8 bits of Γ	byte 2
high-order 8 bits of Γ	byte 3

<u>Data transfer instructions</u>:-
NOP, LDAC, STAC, MVAC, MOVR.

<u>Program control instructions</u>:-
JUMP, JMPZ, JPNZ.

<u>Data operation instructions</u>:-
ADD, SUB, INAC, CLAC → Arithmetic Instructions
AND, OR, XOR, NOT → Logical Instructions.

- consider a simple program using this instruction set to calculate the sum

$$1+2+\cdots+n, \text{ or } \sum_{i=1}^{n} i$$

high-level language code as

```
total = 0
FOR i=1 to n
DO
{
    total = total + i };
```

Most ISAs include many general purpose registers for storing data.

Alg:-

```
1: total = 0; i = 0
2: i = i+1
3: Total = total + i
4: IF i ≠ n THEN GOTO 2.
```

Assembly code:-

```
      CLAC          }
      STAC total.   }  total = 0,  i = 0.
      STAC i        }

      ~~LOAD~~
Loop: LDAC i        }
      INAC          }  i = i+1
      STAC i        }

      MVAC          }
      LDAC Total    }  total = total + i
      ADD           }
      STAC total.   }

      LDAC n        }
      SUB           }  IF i ≠ n THEN GOTO Loop.
      JPNZ Loop     }
```

Execution trace of the loop summation program:-

Instruction	1st loop	2nd loop	3rd Loop	4th Loop	5th Loop.
CLAC	AC = 0				
STAC total	total = 0				
STAC i	i = 0				
LDAC i	AC = 0	AC = 1	AC = 2	AC = 3	AC = 4
INAC	AC = 1	AC = 2	AC = 3	AC = 4	AC = 5
STAC i	i = 1	i = 2	i = 3	i = 4	i = 5
MVAC	R = 1	R = 2	R = 3	R = 4	R = 5
LDAC total	AC = 0	AC = 1	A = 3	AC = 6	AC = 10
ADD	AC = 1	AC = 3	AC = 6	AC = 10	AC = 15
STAC total	total = 1	total = 3	total = 6	total = 10	total = 15
LDAC n	AC = 5	AC = 5	AC = 5	AC = 5	AC = 5
SUB	AC = 4 Z = 0	AC = 3 Z = 0	AC = 2 Z = 0	AC = 1 Z = 0	AC = 0 Z = 1
JPNZ Loop	JUMP	JUMP	JUMP	JUMP	NO JUMP