Unit - 3

* Instruction Set Architecture

⇒ It is the design of a computer from the Programmer's Perspective.

⇒ It describes the design of a computer in terms of the basic operations it must support.

⇒ It can also be defined as the interface between software & hardware.

⇒ An ISA contains :

i) The functional definition of storage locations (registers, memory) & operations (add, multiply, branch, load, store, etc)

ii) Precise description of how to invoke & access them.

⇒ Based on where the operands are stored and whether they are named explicitly or implicitly, the ISA can be classified as Stack Architecture, Accumulator Architecture & Register - set Architecture.

Stack Architecture : { 0 - Address instruction }

↳ The operands are put into the stack. Operations take place at the top two locations on the stack, destroying the operands, and leaving the result on the top of the stack.

↳ Stack is a block of memory in RAM, but CPU keeps a stack pointer register point to the top of the stack.

↳ Data in stack are first - in last out.

↳ 2 Operations of Stack : [1 operand]

i) Push → Insert a value at the top of the st

ii) Pop → Remove the top element of the stack to a destination.

↳ To perform an operation, the operands are pushed into the stack first. Once performed an operation, its operand(s) are deleted from the stack, and its result is stored at the top of the stack. Then, it can use pop command to store the result back to the memory.

Eg: C = A + B [A, B, C → Memory addr

```
PUSH A      # push value of at location A
PUSH B      # push value at location B
ADD         # add the top 2 elements of the stack & s
              the result back to the top of the stac
POP C       # store the value and the top of the st
              (result) to location C.
```

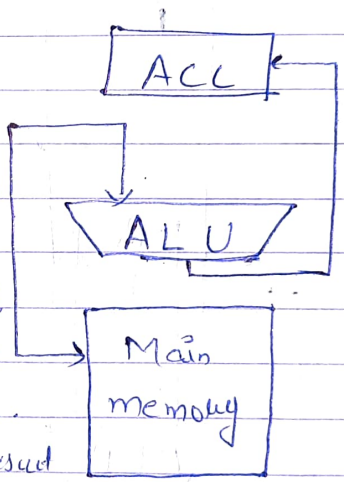
Accumulator Architecture : {1-Address instruction}

↳ It places one operand in the accumulator & one in memory. The one in the accumulator is implicit, and the one in the memory is an explicit memory address.

The result of ALU is written back to the accumulator.

↳ The operand in the accumulator is loaded from memory using the command LOAD, & the result is stored in memory from accumulator using the command STORE.

Eg: C = A+B.




```
LOAD A    # load value at location A from main memory to ACC.
ADD B     # fetch the value at location B, add it with the value of ACC &
STORE C   # save the result back to ACC.
          # store the value of ACC (result of A+B) to location C.
```

Register-Set Architecture : {2-3-address instructions}

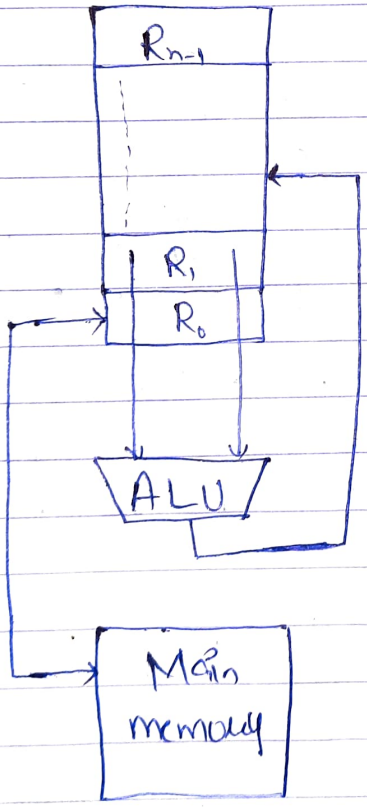
⇒ It's the dominant architecture used by modern computers.

⇒ Modern CPUs have a number of general-purpose registers (GPRs) for internal storage, at least 8 & as many as 32.

↳ This architecture allows fast access to temporary values, permits clever compiler optimization, but the instructions are longer than the accumulator designs.

↳ Operations need to specify all the operands explicitly.

↳ Depending on whether the operands are available in memory or registers, it can be further classified as register - register, register - memory & memory - memory.



Stack Architecture