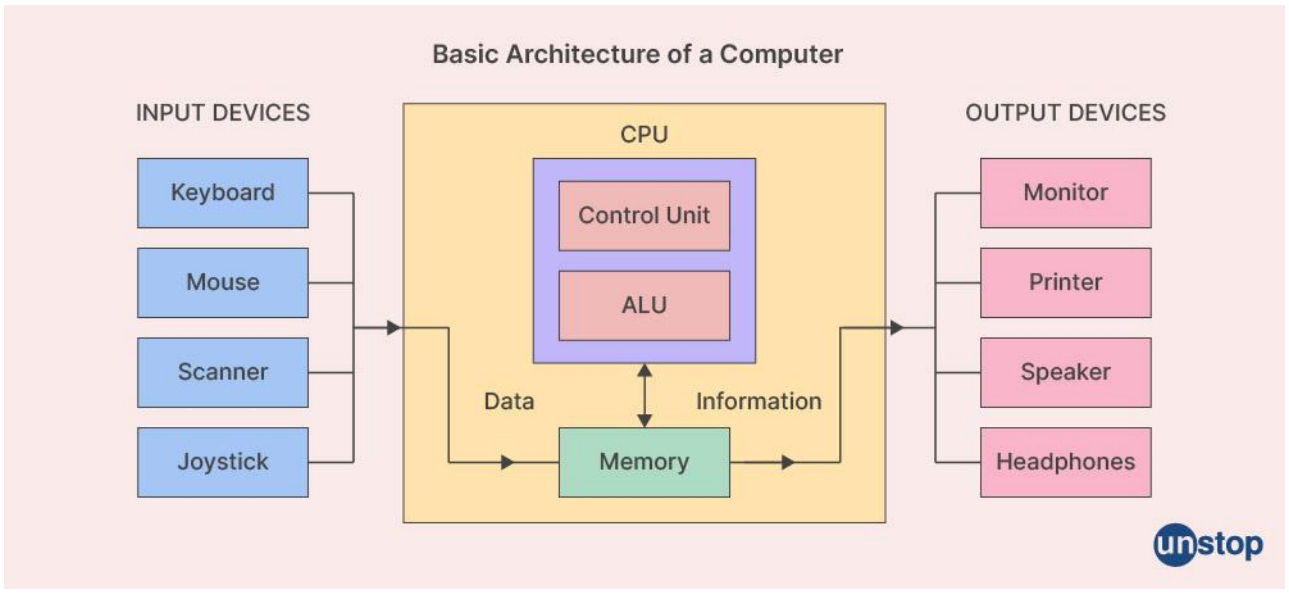
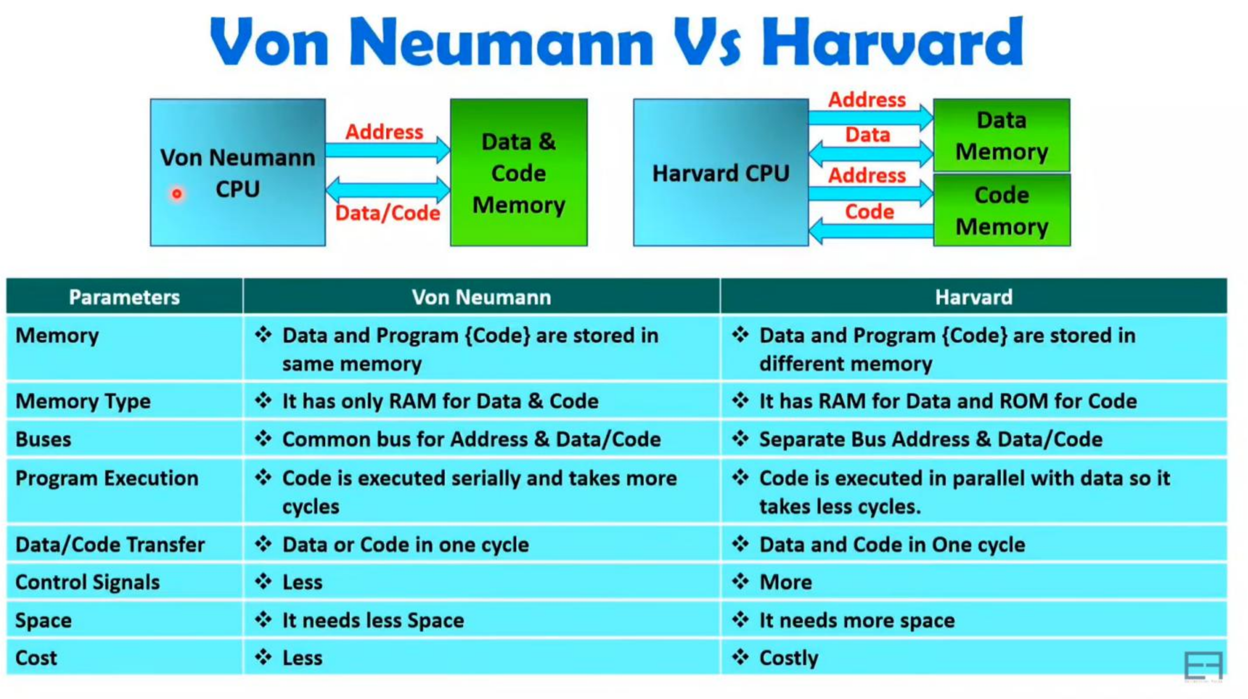
Computer architecture and organization

Computer Architecture is concerned with the structure and behavior of the computer system as seen by the user.

Computer organization is concerned with the way hardware components are connected together to form a computer system.




Von Neumann Vs Harvard
Von Neumann CPU
Address
Data/Code
Data & Code Memory
Harvard CPU
Address
Data
Address
Code
Data Memory
Code Memory

Parameters	Von Neumann	Harvard
Memory	❖ Data and Program {Code} are stored in same memory	❖ Data and Program {Code} are stored in different memory
Memory Type	❖ It has only RAM for Data & Code	❖ It has RAM for Data and ROM for Code
Buses	❖ Common bus for Address & Data/Code	❖ Separate Bus Address & Data/Code
Program Execution	❖ Code is executed serially and takes more cycles	❖ Code is executed in parallel with data so it takes less cycles.
Data/Code Transfer	❖ Data or Code in one cycle	❖ Data and Code in One cycle
Control Signals	❖ Less	❖ More
Space	❖ It needs less Space	❖ It needs more space
Cost	❖ Less	❖ Costly

Difference Between Von Neumann and Harvard Architecture

Parameters	**Von Neumann Architecture**	**Harvard Architecture**	
Definition	The Von Neumann Architecture is an ancient type of computer architecture that follows the concept of a stored-program computer.	Harvard Architecture is a modern type of computer architecture that follows the concept of the relay-based model by Harvard Mark I.	
Physical Address	It uses one single physical address for accessing and storing both data and instructions.	It uses two separate physical addresses for storing and accessing both instructions and data.	
Buses (Signal	One common signal path (bus) helps in	It uses separate buses for the transfer of both	

Paths)	the transfer of both instruction and data.	data and instructions.	
Number of Cycles	It requires two clock cycles for executing a single instruction.	It executes any instruction using only one single cycle.	
Cost	It is comparatively cheaper in cost than Harvard Architecture.	It is comparatively more expensive than the Von Neumann Architecture.	
Access to CPU	The CPU is not able to read/write data and access instructions at the same time.	The CPU can easily read/write data as well as access the instructions at any given time.	
Uses	This method comes to play in the case of small computers and personal computers.	This architecture is best for signal processing as well as microcontrollers.	
Requirement of Hardware	As compared to Harvard Architecture, Von Neumann Architecture requires lesser architecture. It is because it only needs to reach one common memory.	This one requires more hardware. It is because it requires separate sets of data as well as address buses for individual memory.	
Requirement of Space	This architecture basically requires less space.	This architecture comparatively requires more space.	
Usage of Space	This architecture does not waste any space. It is because the	This type of architecture can result in space wastage. It is because	

	instruction memory can utilize the left space of the data memory. It can also happen vice-versa.	the instruction memory cannot utilize the leftover space in the data memory. It also cannot happen vice-versa.	
Execution Speed	The speed of execution of the Von Neumann Architecture is comparatively slower. It is because it is not capable of fetching the instructions and data both at the same time.	The overall speed of execution of Harvard Architecture is comparatively faster. It is because the processor, in this case, is capable of fetching both instructions and data at the very same time.	
Controlling	The process of controlling becomes comparatively simpler with this architecture. It is because it fetches either instructions or data at any given time.	The process of controlling becomes comparatively complex with this architecture. It is because it basically fetches both instructions and data simultaneously at the very same time	

Computer Architecture	**Computer Organization**	
Computer Architecture is concerned with the way hardware components are connected together to form a computer system.	Computer Organization is concerned with the structure and behaviour of a computer system as seen by the user.	
It acts as the interface between hardware and software.	It deals with the components of a connection in a system.	
Computer Architecture helps us to	Computer Organization tells us	

understand the functionalities of a system.	how exactly all the units in the system are arranged and interconnected.	
A programmer can view architecture in terms of instructions, addressing modes and registers.	Whereas Organization expresses the realization of architecture.	
While designing a computer system architecture is considered first.	An organization is done on the basis of architecture.	
Computer Architecture deals with high-level design issues.	Computer Organization deals with low-level design issues.	
Architecture involves Logic (Instruction sets, Addressing modes, Data types, Cache optimization)	Organization involves Physical Components (Circuit design, Adders, Signals, Peripherals)	

Difference between Computer Architecture and Computer Organization:

S. No.	Computer Architecture	Computer Organization
1.	Architecture describes what the computer does.	The Organization describes how it does it.
2.	Computer Architecture deals with the functional behavior of computer systems.	Computer Organization deals with a structural relationship.
3.	In the above figure, it's clear that it deals with high-level design issues.	In the above figure, it's also clear that it deals with low-level design issues.
4.	Architecture indicates its hardware.	Whereas Organization indicates its performance.
5.	As a programmer, you can view architecture as a series of instructions, addressing modes, and registers.	The implementation of the architecture is called organization.
6.	For designing a computer, its architecture is fixed first.	For designing a computer, an organization is decided after its architecture.

S. No.	Computer Architecture	Computer Organization
7.	Computer Architecture is also called Instruction Set Architecture (ISA).	Computer Organization is frequently called microarchitecture.
8.	Computer Architecture comprises logical functions such as instruction sets, registers, data types, and addressing modes.	Computer Organization consists of physical units like circuit designs, peripherals, and adders.
9.	The different architectural categories found in our computer systems are as follows: 1. Von-Neumann Architecture 2. Harvard Architecture 3. Instruction Set Architecture 4. Micro-architecture 5. System Design	CPU organization is classified into three categories based on the number of address fields: 1. Organization of a single Accumulator. 2. Organization of general registers 3. Stack organization
10.	It makes the computer's hardware visible.	It offers details on how well the computer performs.
11.	Architecture coordinates the hardware and software of the system.	Computer Organization handles the segments of the network in a system.
12.	The software developer is aware of it.	It escapes the software programmer's detection.
13.	Examples- Intel and AMD created the x86 processor. Sun Microsystems and others created the SPARC processor. Apple, IBM, and Motorola created the PowerPC.	Organizational qualities include hardware elements that are invisible to the programmer, such as interfacing of computer and peripherals, memory technologies, and control signals.

What is RISC?

RISC stands for Reduced Instruction Set Computers. RISC is a microprocessor and as the name indicates, it performs a smaller number of computer instructions. Thus, it has a high speed to operate. It works on a fixed instruction format that keeps less than 100 instructions to be processed with a register-based instruction used by a few simple addressing modes. The LOAD/STORE is the only instruction used to access the memory as it is a compiler development mechanism.

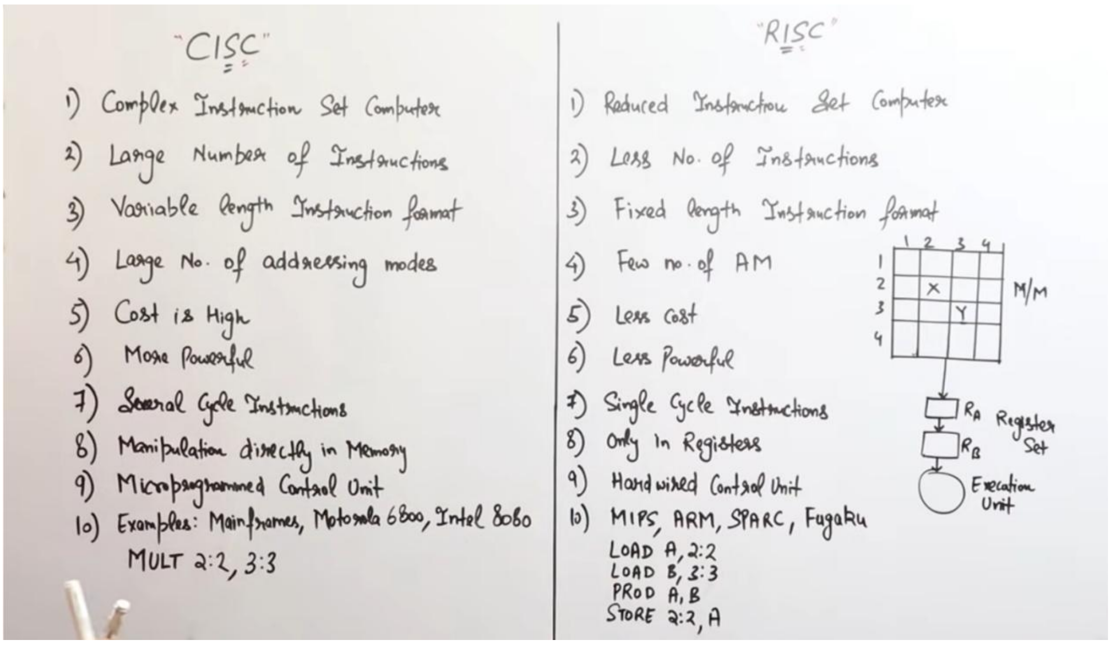
What is CISC?

CICS stands for Complex Instruction Set Computer. CISC is the kind of chip that can be easily programmed and makes the best and efficient use of memory. The main motive of CISC is to make compiler development easy and simple. There is no need for the machine to generate instructions for the processor as CISC eliminates the need.

Difference between RISC and CISC Processor

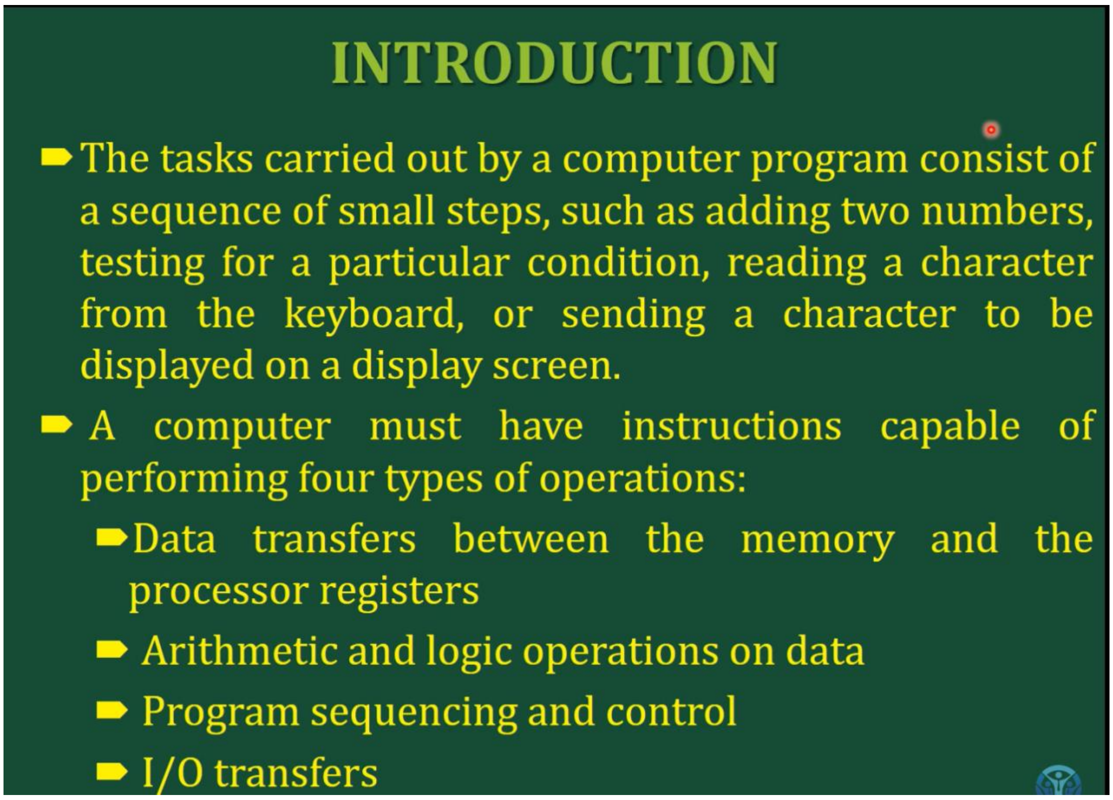
The RISC and CISC both are part of system architecture and set of computer instruction. The important difference between RISC and CISC is the number of computing cycles.

o.	**RISC**	**CISC**
1.	**RISC is a reduced instruction set.**	**CISC is a complex instruction set.**
2.	**The number of instructions is less as compared to CISC.**	**The number of instructions is more as compared to RISC.**
3.	**The addressing modes are less.**	**The addressing modes are more.**
4.	**It works in a fixed instruction format.**	**It works in a variable instruction format.**
5.	**The RISC consumes low power.**	**The CISC consumes high power.**
6.	**The RISC processors are highly pipelined.**	**The CISC processors are less pipelined.**
7.	**It optimizes the performance by focusing on software.**	**It optimizes the performance by focusing on hardware.**
8.	**Requires more RAM.**	**Requires less RAM.**



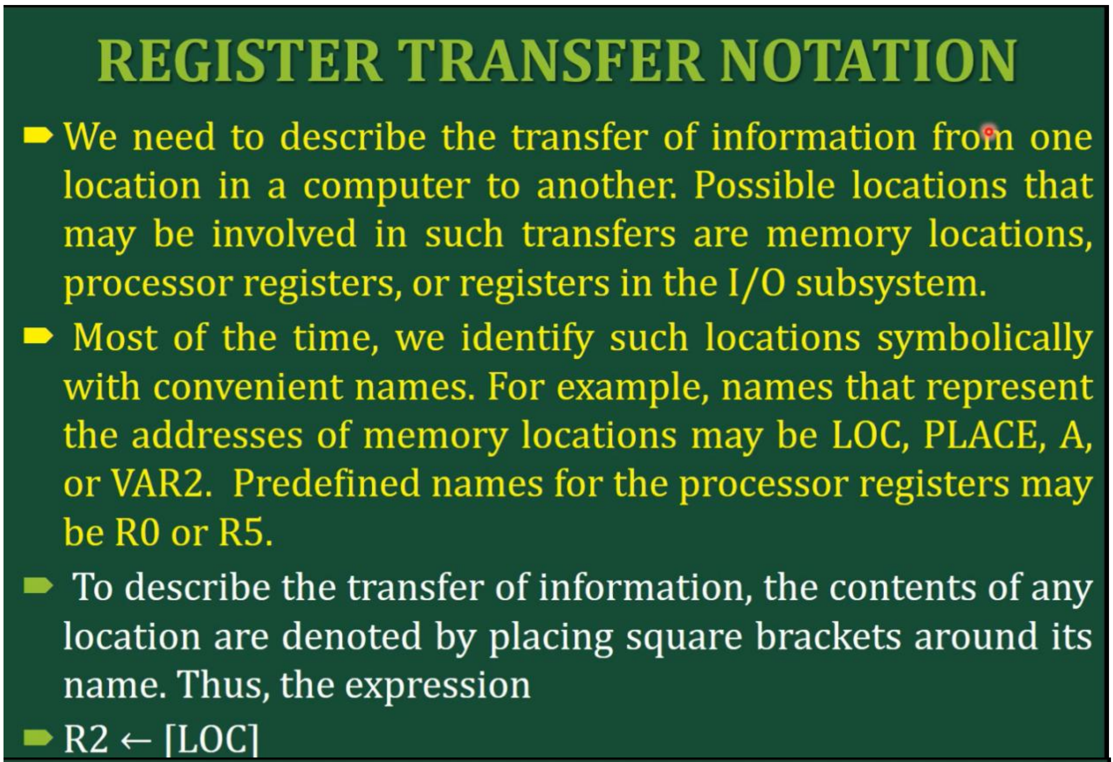
"CISC"	"RISC"
1) Complex Instruction Set Computer	1) Reduced Instruction Set Computer
2) Large Number of Instructions	2) Less No. of Instructions
3) Variable length Instruction format	3) Fixed length Instruction format
4) Large No. of addressing modes	4) Few no. of AM
5) Cost is High	5) Less Cost
6) More Powerful	6) Less Powerful
7) Several Cycle Instructions	7) Single Cycle Instructions
8) Manipulation directly in Memory	8) Only In Registers
9) Microprogrammed Control Unit	9) Hard wired Control Unit
10) Examples: Mainframes, Motorola 6800, Intel 8080	10) MIPS, ARM, SPARC, Fugaku
MULT 2:2, 3:3	LOAD A, 2:2
	LOAD B, 3:3
	PROD A, B
	STORE 2:2, A

Memory operations, Instruction and instruction sequencing



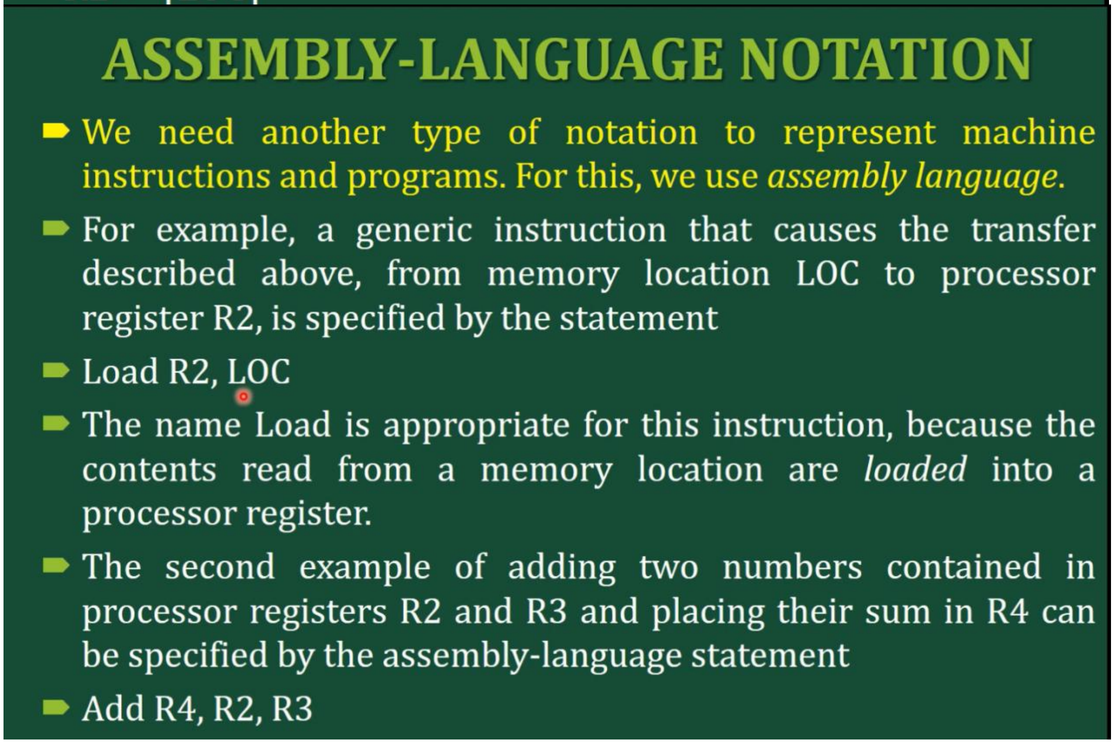
INTRODUCTION

- The tasks carried out by a computer program consist of a sequence of small steps, such as adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen.
- A computer must have instructions capable of performing four types of operations:
  - Data transfers between the memory and the processor registers
  - Arithmetic and logic operations on data
  - Program sequencing and control
  - I/O transfers



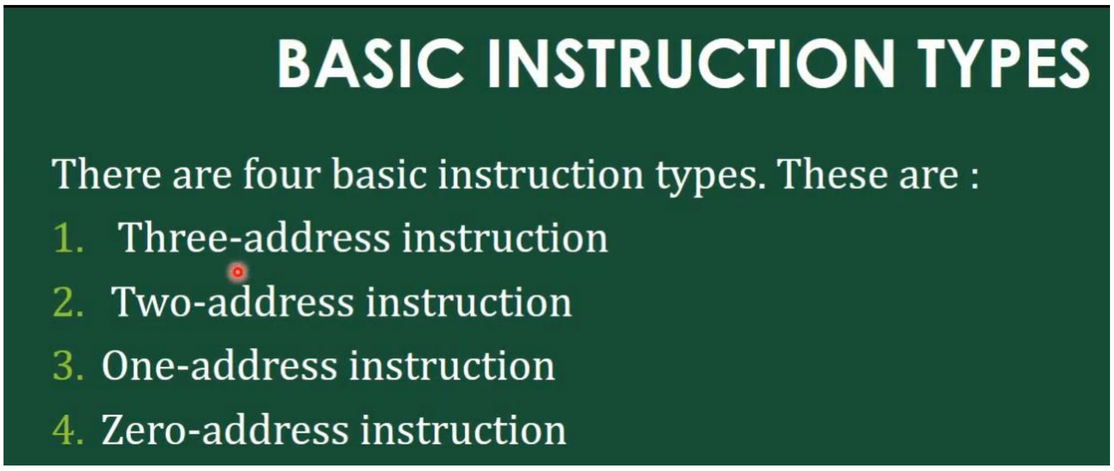
REGISTER TRANSFER NOTATION

- We need to describe the transfer of information from one location in a computer to another. Possible locations that may be involved in such transfers are memory locations, processor registers, or registers in the I/O subsystem.
- Most of the time, we identify such locations symbolically with convenient names. For example, names that represent the addresses of memory locations may be LOC, PLACE, A, or VAR2. Predefined names for the processor registers may be R0 or R5.
- To describe the transfer of information, the contents of any location are denoted by placing square brackets around its name. Thus, the expression
- $R2 \leftarrow [LOC]$



ASSEMBLY-LANGUAGE NOTATION

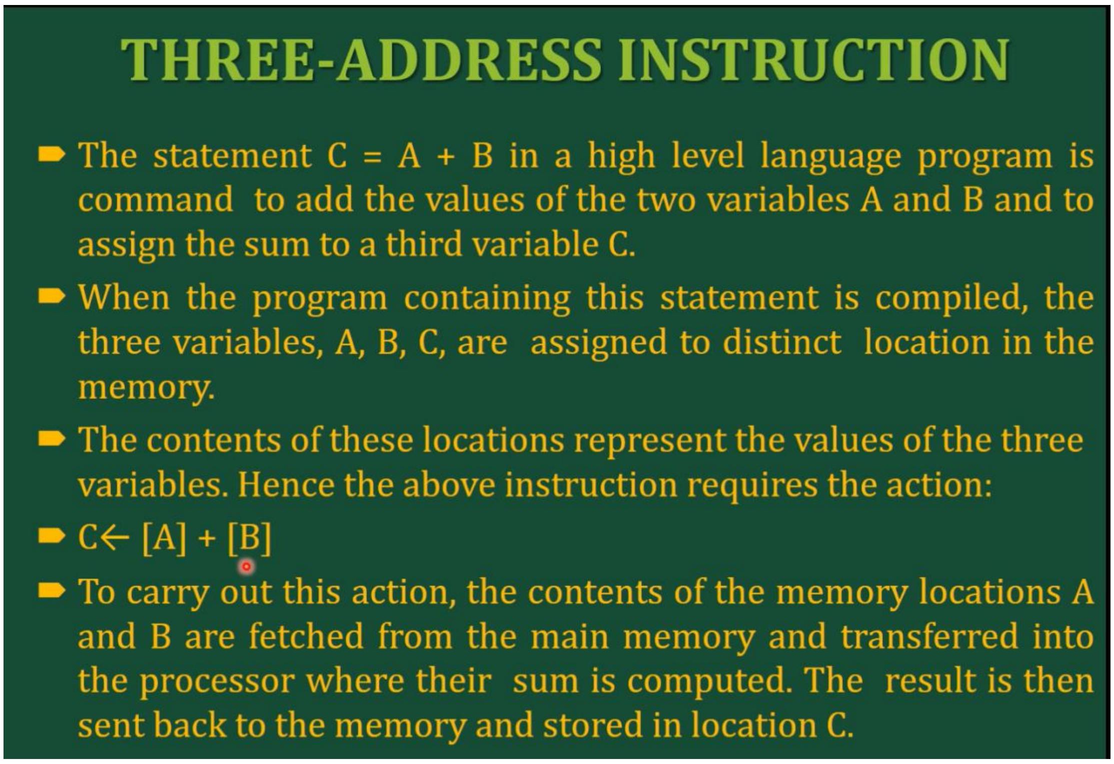
- We need another type of notation to represent machine instructions and programs. For this, we use *assembly language*.
- For example, a generic instruction that causes the transfer described above, from memory location LOC to processor register R2, is specified by the statement
- Load R2, LOC
- The name Load is appropriate for this instruction, because the contents read from a memory location are *loaded* into a processor register.
- The second example of adding two numbers contained in processor registers R2 and R3 and placing their sum in R4 can be specified by the assembly-language statement
- Add R4, R2, R3



BASIC INSTRUCTION TYPES

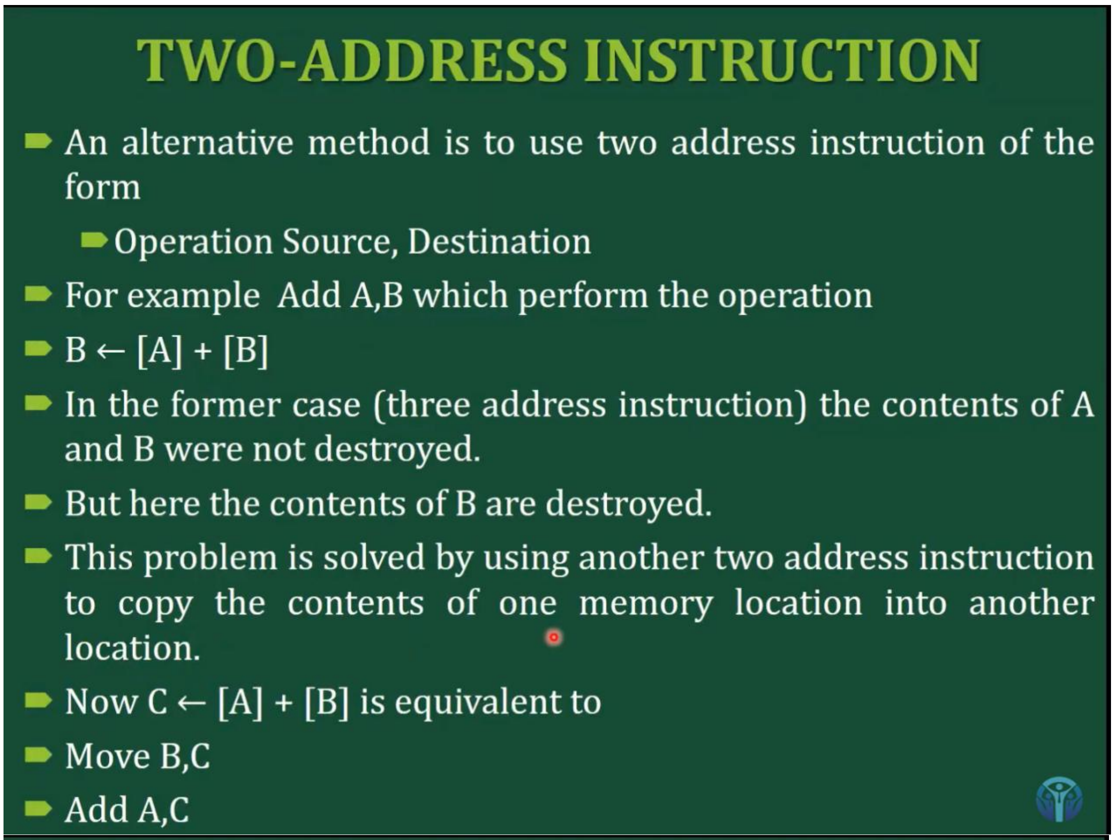
There are four basic instruction types. These are :

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



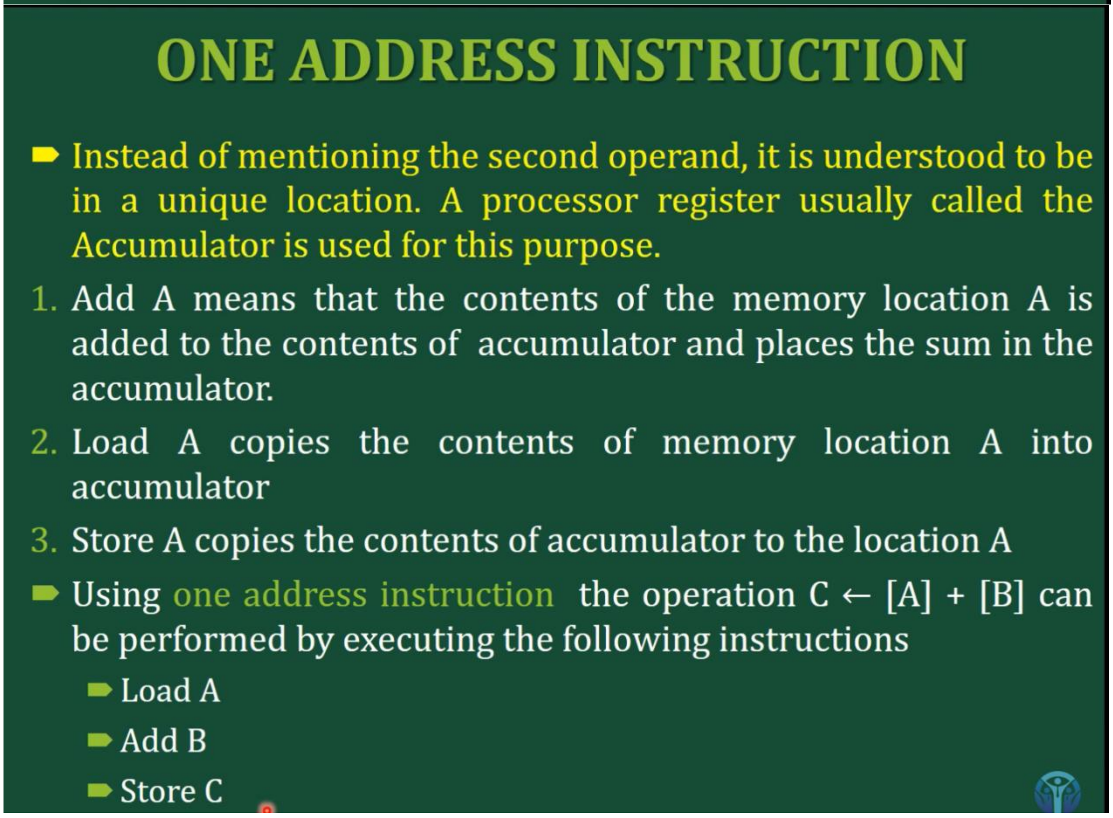
THREE-ADDRESS INSTRUCTION

- The statement C = A + B in a high level language program is command to add the values of the two variables A and B and to assign the sum to a third variable C.
- When the program containing this statement is compiled, the three variables, A, B, C, are assigned to distinct location in the memory.
- The contents of these locations represent the values of the three variables. Hence the above instruction requires the action:
- $C \leftarrow [A] + [B]$
- To carry out this action, the contents of the memory locations A and B are fetched from the main memory and transferred into the processor where their sum is computed. The result is then sent back to the memory and stored in location C.



TWO-ADDRESS INSTRUCTION

- An alternative method is to use two address instruction of the form
  - Operation Source, Destination
- For example Add A,B which perform the operation
- B ← [A] + [B]
- In the former case (three address instruction) the contents of A and B were not destroyed.
- But here the contents of B are destroyed.
- This problem is solved by using another two address instruction to copy the contents of one memory location into another location.
- Now C ← [A] + [B] is equivalent to
- Move B,C
- Add A,C



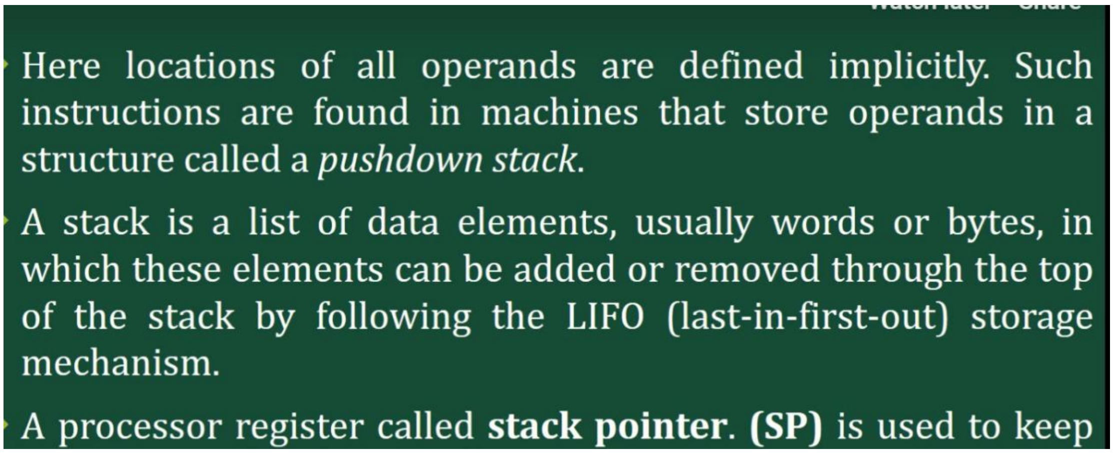
ONE ADDRESS INSTRUCTION

- Instead of mentioning the second operand, it is understood to be in a unique location. A processor register usually called the Accumulator is used for this purpose.

1. Add A means that the contents of the memory location A is added to the contents of accumulator and places the sum in the accumulator.
2. Load A copies the contents of memory location A into accumulator
3. Store A copies the contents of accumulator to the location A

- Using one address instruction the operation C ← [A] + [B] can be performed by executing the following instructions
  - Load A
  - Add B
  - Store C

Zero Address Instruction

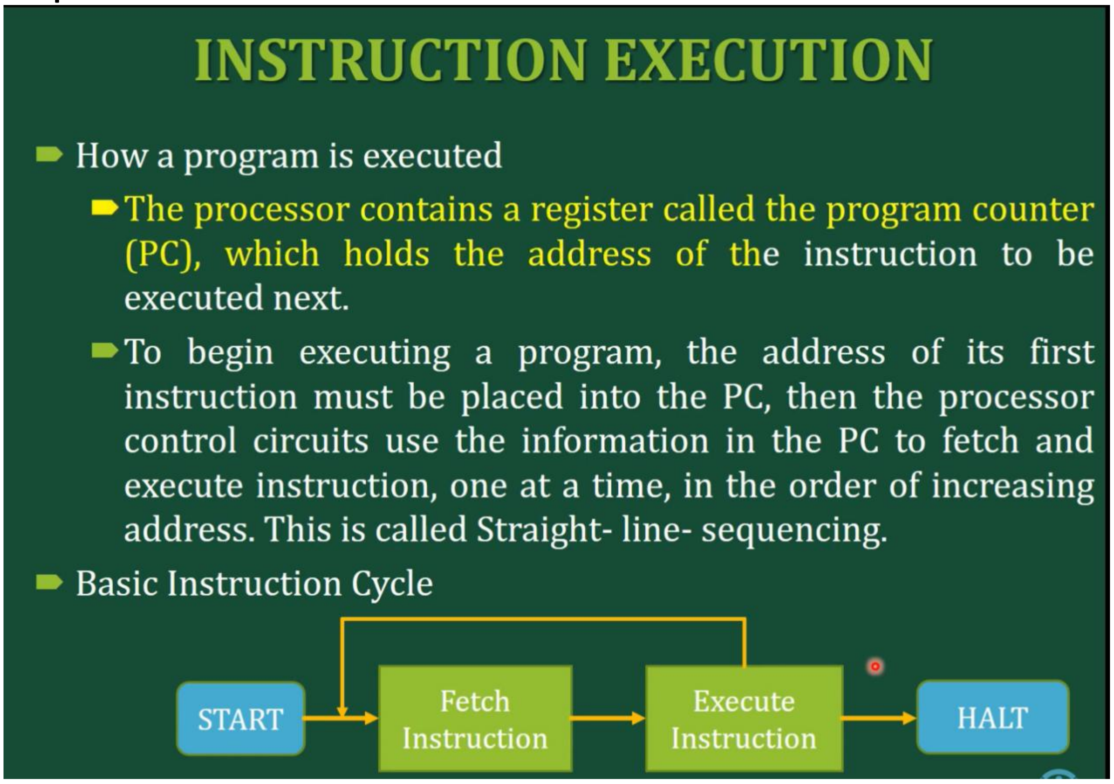


Here locations of all operands are defined implicitly. Such instructions are found in machines that store operands in a structure called a *pushdown stack*.

A stack is a list of data elements, usually words or bytes, in which these elements can be added or removed through the top of the stack by following the LIFO (last-in-first-out) storage mechanism.

A processor register called **stack pointer**. **(SP)** is used to keep

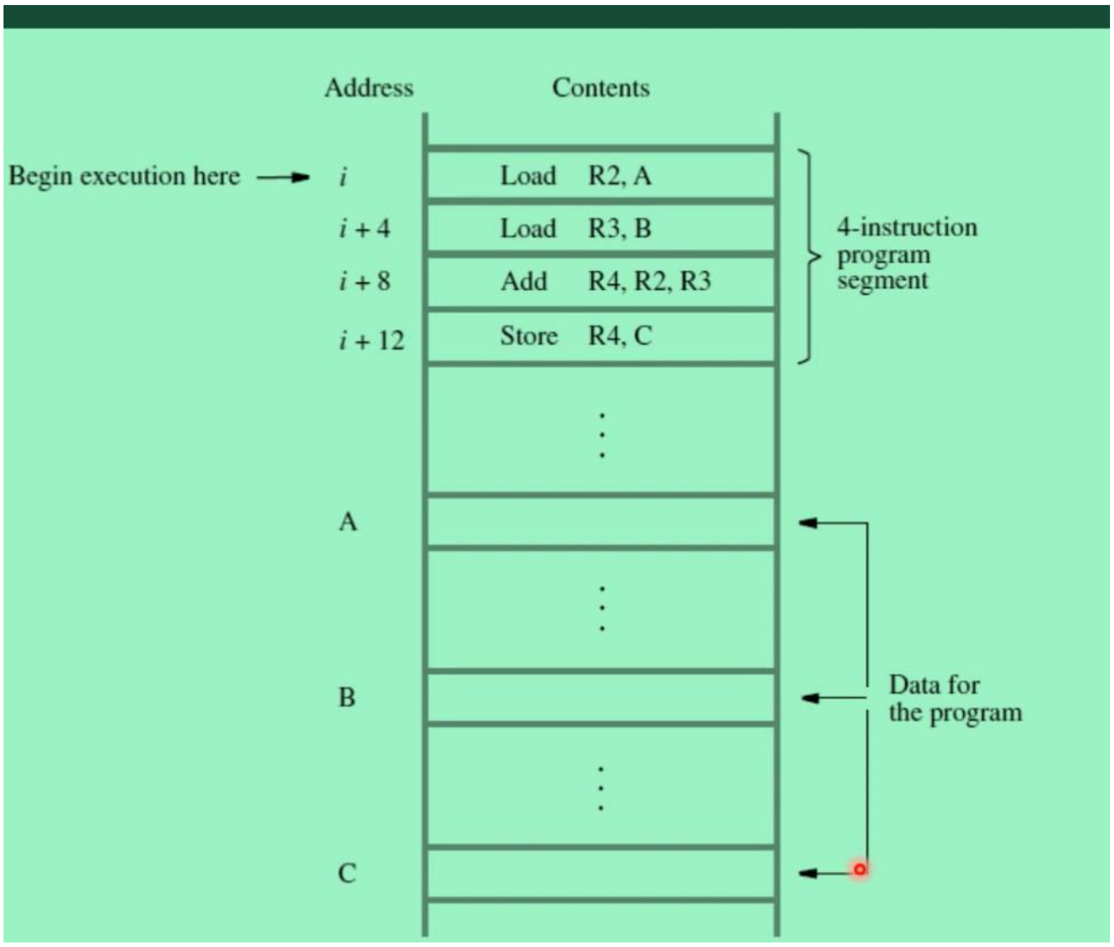
Keep the track



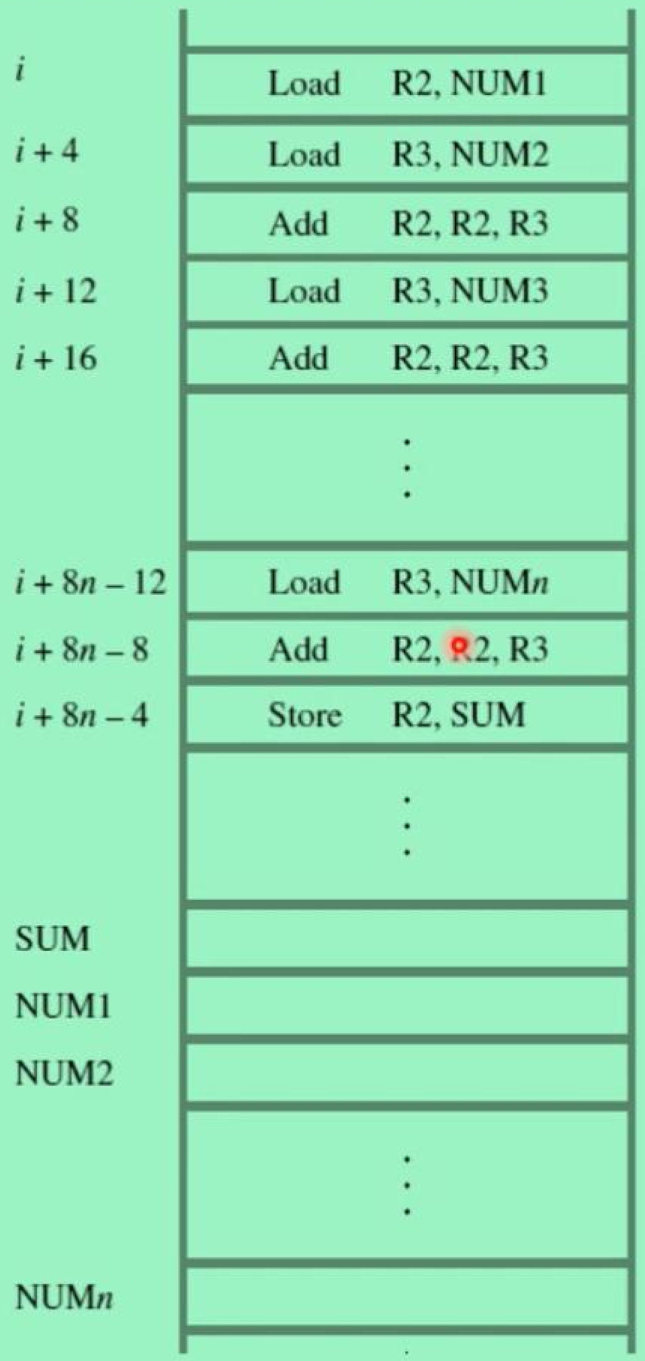
INSTRUCTION EXECUTION

- How a program is executed
  - The processor contains a register called the program counter (PC), which holds the address of the instruction to be executed next.
  - To begin executing a program, the address of its first instruction must be placed into the PC, then the processor control circuits use the information in the PC to fetch and execute instruction, one at a time, in the order of increasing address. This is called Straight- line- sequencing.
- Basic Instruction Cycle

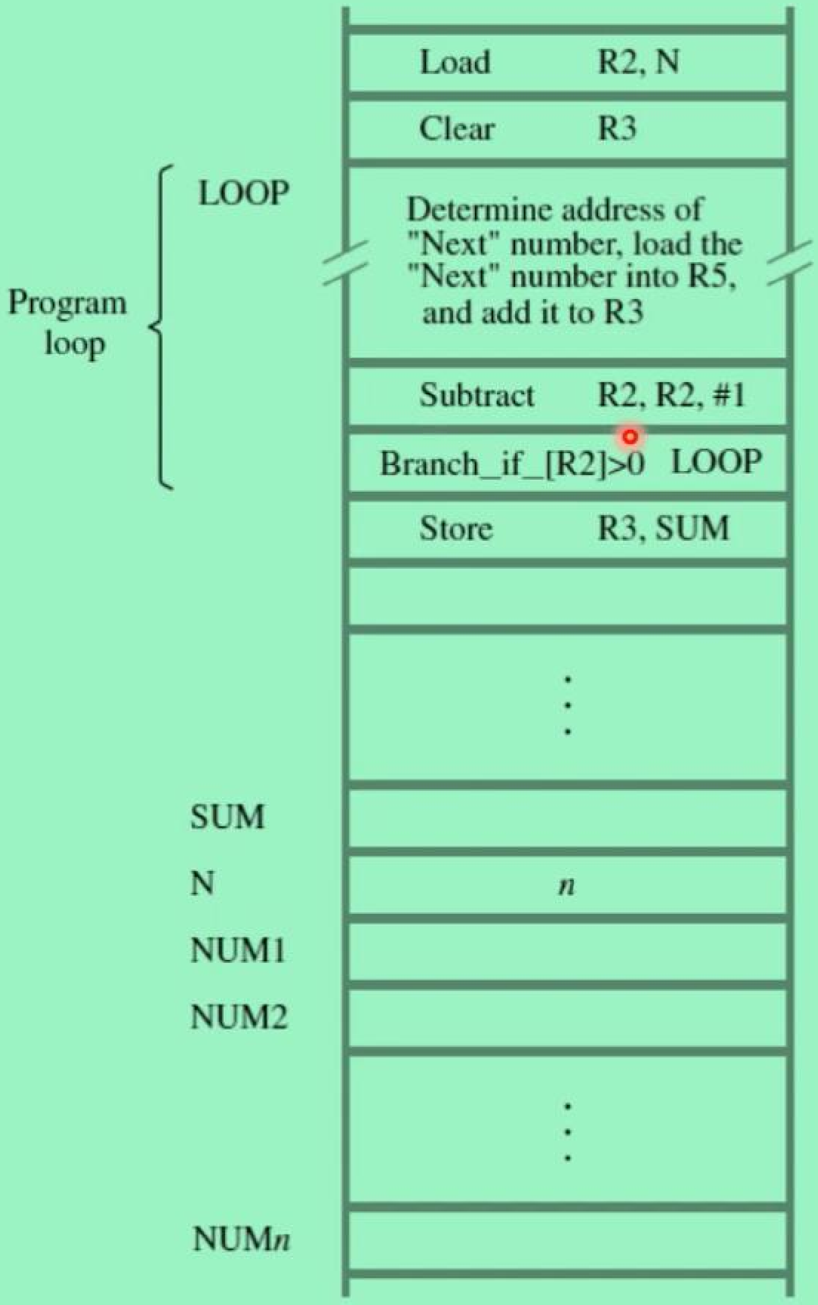
Straight Line Sequencing



Address
Contents
Begin execution here
i
Load R2, A
i + 4
Load R3, B
i + 8
Add R4, R2, R3
i + 12
Store R4, C
4-instruction program segment
A
B
C
Data for the program


```
i            Load   R2, NUM1
i + 4        Load   R3, NUM2
i + 8        Add    R2, R2, R3
i + 12       Load   R3, NUM3
i + 16       Add    R2, R2, R3
                  ⋮
i + 8n – 12  Load   R3, NUMn
i + 8n – 8   Add    R2, R2, R3
i + 8n – 4   Store  R2, SUM
                  ⋮
SUM
NUM1
NUM2
                  ⋮
NUMn
```

BRANCHING


Load R2, N
Clear R3
LOOP
Determine address of "Next" number, load the "Next" number into R5, and add it to R3
Program loop
Subtract R2, R2, #1
Branch_if_[R2]>0 LOOP
Store R3, SUM
SUM
N
n
NUM1
NUM2
NUMn

Computer Organization | Instruction Formats (Zero, One, Two and Three Address Instruction)

In computer organization, instruction formats refer to the way instructions are encoded and represented in machine language. There are several types of instruction formats, including zero, one, two, and three-address instructions.

Each type of instruction format has its own advantages and disadvantages in terms of code size, execution time, and flexibility. Modern computer architectures typically use a combination of these formats to provide a balance between simplicity and power.

What are the Different Types of Filed in Instruction?

A computer performs a task based on the instruction provided. Instruction in computers comprises groups called fields. These fields contain different information for computers everything is in 0 and 1 so each field has different significance based on which a CPU decides what to perform. The most common fields are:

- The operation field specifies the operation to be performed like addition.
- Address field which contains the location of the operand, i.e., register or memory location.
- Mode field which specifies how operand is to be founded.

Instruction is of variable length depending upon the number of addresses it contains.

Generally, CPU organization is of three types based on the number of address fields:

- Single Accumulator organization

- General register organization
- Stack organization

In the first organization, the operation is done involving a special register called the accumulator. In the second multiple registers are used for the computation purpose. In the third organization the work on stack basis operation due to which it does not contain any address field.

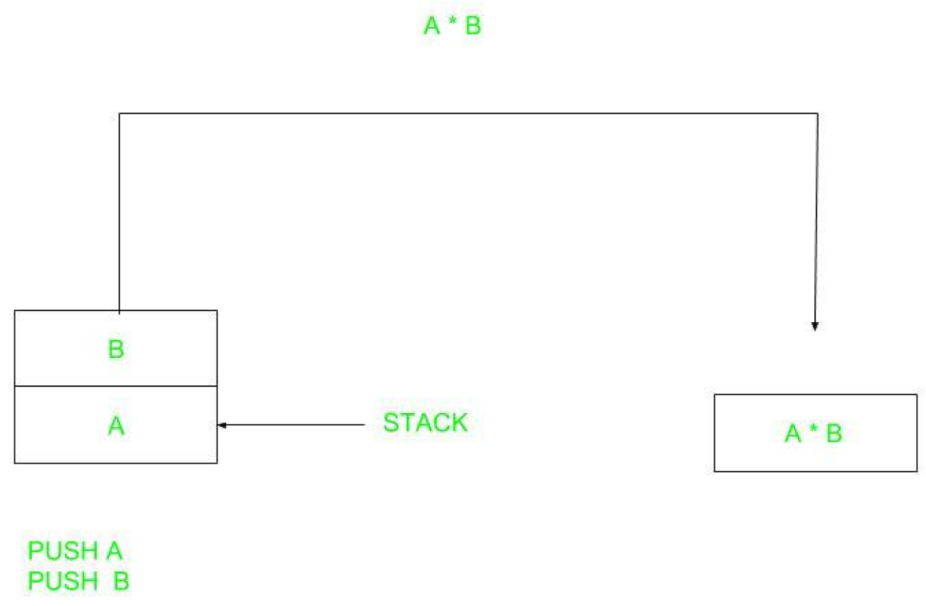
Types of Instructions

Based on the number of addresses, instructions are classified as:

NOTE: We will use the X = (A+B)*(C+D) expression to showcase the procedure.

Zero Address Instructions

These instructions do not specify any operands or addresses. Instead, they operate on data stored in registers or memory locations implicitly defined by the instruction. For example, a zero-address instruction might simply add the contents of two registers together without specifying the register names.



Zero Address Instruction

A stack-based computer does not use the address field in the instruction. To evaluate an expression first it is converted to reverse Polish Notation i.e. Postfix Notation.

Expression: X = (A+B)*(C+D)
Postfixed : X = AB+CD+*
TOP means top of stack
M[X] is any memory location

```
PUSH  A TOP = A
PUSH  B TOP = B
ADD     TOP = A+B
PUSH  C TOP = C
PUSH  D TOP = D
ADD     TOP = C+D
MUL     TOP = (C+D)*(A+B)
POP   X M[X] = TOP
```

One Address Instructions

These instructions specify one operand or address, which typically refers to a memory location or register. The instruction operates on the contents of that operand, and the result may be stored in the same or a different location. For example, a one-address instruction might load the contents of a memory location into a register.

This uses an implied ACCUMULATOR register for data manipulation. One operand is in the accumulator and the other is in the register or memory location. Implied means that the CPU already knows that one operand is in the accumulator so there is no need to specify it.

opcode	operand/address of operand	mode

One Address Instruction

Expression: X = (A+B)*(C+D)
AC is accumulator
M[] is any memory location
M[T] is temporary location

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

Two Address Instructions

These instructions specify two operands or addresses, which may be memory locations or registers. The instruction operates on the contents of both operands, and the result may be stored in the same or a different location. For example, a two-address instruction might add the contents of two registers together and store the result in one of the registers.

This is common in commercial computers. Here two addresses can be specified in the instruction. Unlike earlier in one address instruction, the result was stored in the accumulator, here the result can be stored at different locations rather than just accumulators, but require more number of bit to represent the address.

opcode	Destination address	Source address	mode

Two Address Instruction

Here destination address can also contain an operand.

Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location

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**

Three Address Instructions

These instructions specify three operands or addresses, which may be memory locations or registers. The instruction operates on the contents of all three operands, and the result may be stored in the same or a different location. For example, a three-address instruction might multiply the contents of two registers together and add the contents of a third register, storing the result in a fourth register.

This has three address fields to specify a register or a memory location. Programs created are much short in size but number of bits per instruction increases. These instructions make the creation of the program much easier but it does not mean that program will run much faster because now instructions only contain more information but each micro-operation (changing the content of the register, loading address in the address bus etc.) will be performed in one cycle only.

opcode	Destination address	Source address	Source address	mode

Three Address Instruction

Expression: X = (A+B)*(C+D)

R1, R2 are registers

M[] is any memory location

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**

Advantages of Zero-Address, One-Address, Two-Address and Three-Address Instructions

Zero-address instructions

- They are simple and can be executed quickly since they do not require any operand fetching or addressing. They also take up less memory space.

One-address instructions

- They allow for a wide range of addressing modes, making them more flexible than zero-address instructions. They also require less memory space than two or three-address instructions.

Two-address instructions

- They allow for more complex operations and can be more efficient than one-address instructions since they allow for two operands to be processed in a single instruction. They also allow for a wide range of addressing modes.

Three-address instructions

- They allow for even more complex operations and can be more efficient than two-address instructions since they allow for three operands to be processed in a single instruction. They also allow for a wide range of addressing modes.

Disadvantages of Zero-Address, One-Address, Two-Address and Three-Address Instructions

Zero-address instructions

- They can be limited in their functionality and do not allow for much flexibility in terms of addressing modes or operand types.

One-address instructions

- They can be slower to execute since they require operand fetching and addressing.

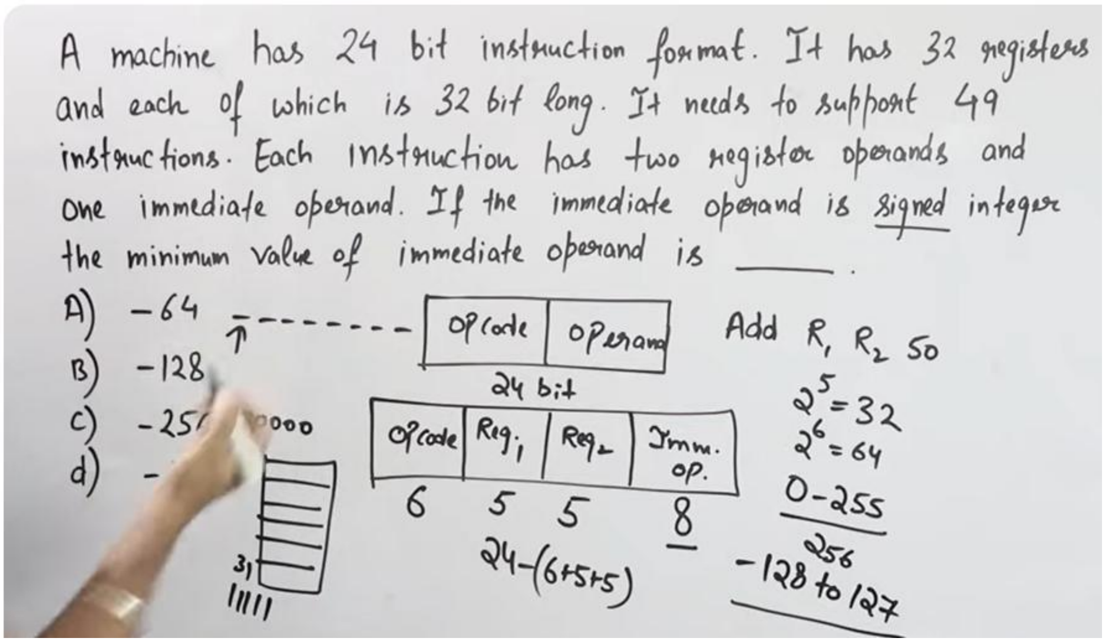
Two-address instructions

- They require more memory space than one-address instructions and can be slower to execute since they require operand fetching and addressing.

Three-address instructions

- They require even more memory space than two-address instructions and can be slower to execute since they require operand fetching and addressing.

Overall, the choice of instruction format depends on the specific requirements of the computer architecture and the trade-offs between code size, execution time, and flexibility.



A machine has 24 bit instruction format. It has 32 registers and each of which is 32 bit long. It needs to support 49 instructions. Each instruction has two register operands and one immediate operand. If the immediate operand is <u>signed</u> integer the minimum value of immediate operand is _____.

A) −64

B) −128

c) −25[illegible]

d) −[illegible]

Opcode	Operand

24 bit

Add R_1 R_2 50

Opcode	Reg.₁	Reg₂	Imm. op.
6	5	5	$\underline{8}$

$24-(6+5+5)$

$2^5=32$

$2^6=64$

$\underline{0-255}$

256

$\underline{-128 \text{ to } 127}$