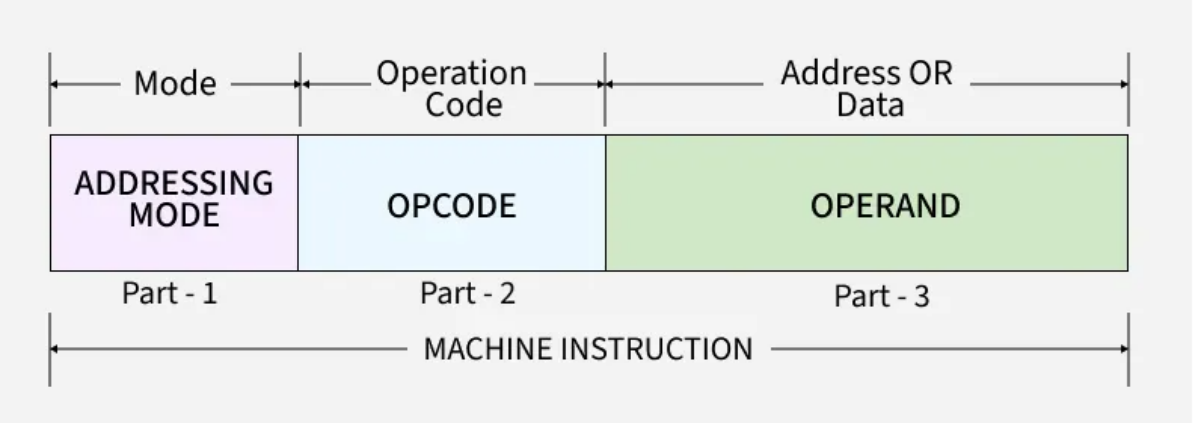
Instruction Formats

Instruction format defines how instructions are represented in a computer's memory. There are different types of instruction formats, including zero, one, two, and three-address instructions.



- **Opcode:** This field specifies the operation to be performed by the CPU, such as addition, subtraction, or data transfer.
- **Operands:** These fields contain the data or references (addresses) to data on which the operation acts.
- **Addressing Mode:** This specifies how to interpret or locate the operand, such as direct, indirect, or immediate addressing.

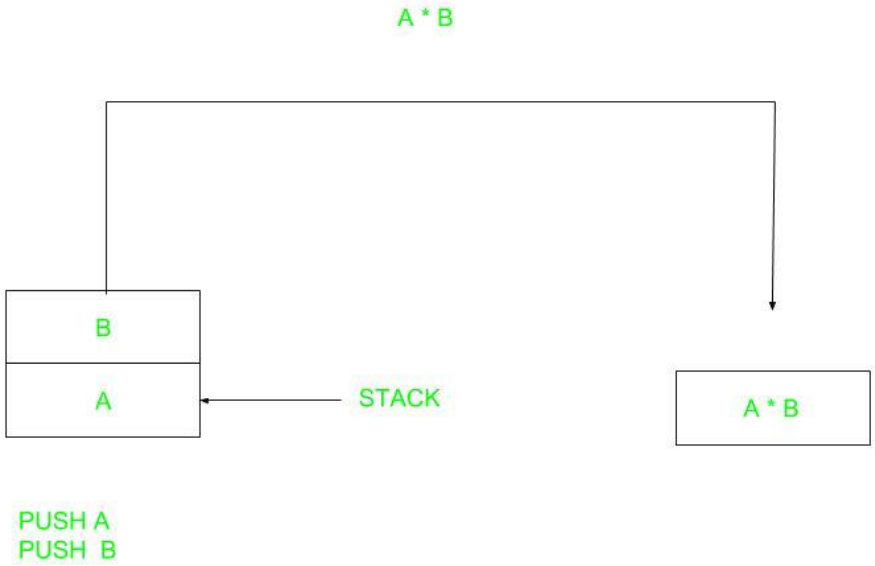
Types of Instruction Formats

Instruction formats are classified into zero, one, two, and three-address types, depending on how many address fields they have. Each type works differently and is used in various ways in computer architecture.

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, it is first converted to reverse Polish Notation i.e. Postfix Notation.

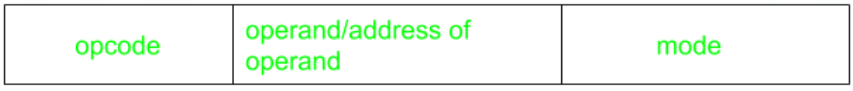
Expression:*X=(A+B)*(C+D)* ***Postfixed:****X=AB+CD+** **Instruction**	**Stack (TOP Value After Execution)**
PUSH A	TOP = A
PUSH B	TOP = B
ADD	TOP = A + B
PUSH C	TOP = C

Expression:*X=(A+B)*(C+D)* ***Postfixed:****X=AB+CD+** **Instruction**	**Stack (TOP Value After Execution)**
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.



One Address Instruction

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

Instruction	Stack / Register (AC / M[])
AC = A	AC = A
AC = AC + B	AC = A + B
M[T] = AC	M[T] = A + B
AC = C	AC = C
AC = AC + D	AC = C + D
M[] = AC	M[] = C + D
AC = AC * M[T]	AC = (A + B) * (C + D)
M[X] = AC	M[X] = (A + B) * (C + D)

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.

Two Address Instruction

Here destination address can also contain an operand.

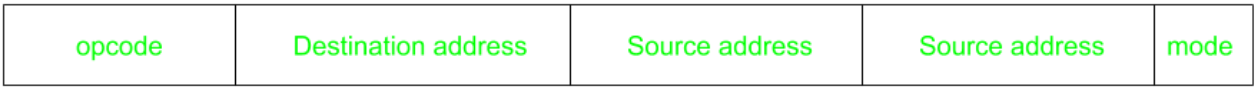
Expression:*X=(A+B)*(C+D)* **Instruction**	**Registers / Memory (R1, R2, M[])**
R1 = A	R1 = A
R1 = R1 + B	R1 = A + B
R2 = C	R2 = C
R2 = R2 + D	R2 = C + D
R1 = R1 * R2	R1 = (A + B) * (C + D)
M[X] = R1	M[X] = (A + B) * (C + D)

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.



Three Address Instruction

Expression:*X=(A+B)*(C+D)* **Instruction**	**Description**
R1 = A	Load value A into R1
R1 = R1 + B	Add B to R1
M[T1] = R1	Store R1 into memory at M[T1]
R2 = C	Load value C into R2
R2 = R2 + D	Add D to R2
M[T2] = R2	Store R2 into memory at M[T2]
R1 = M[T1]	Load M[T1] into R1
R1 = R1 * M[T2]	Multiply R1 by M[T2] and store in R1
M[X] = R1	Store R1 into memory at M[X]