# **Computer - Components**

Input Unit

This unit contains devices with the help of which we enter data into the computer. This unit creates a link between the user and the computer. The input devices translate the information into a form understandable by the computer.

CPU (Central Processing Unit)

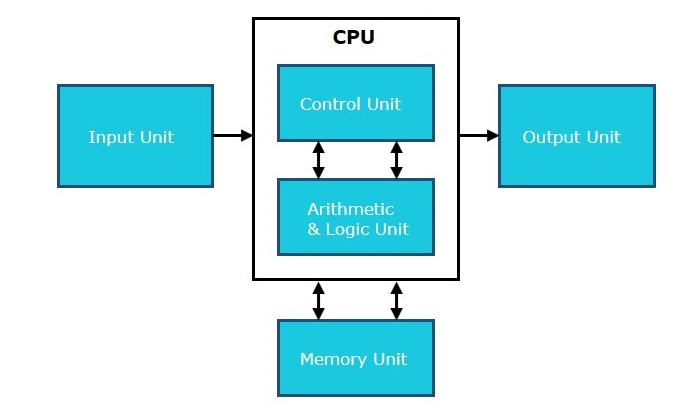
CPU is considered as the brain of the computer. CPU performs all types of data processing operations. It stores data, intermediate results, and instructions (program). It controls the operation of all parts of the computer.

CPU itself has the following three components −

* ALU (Arithmetic Logic Unit)
* Memory Unit
* Control Unit

Output Unit

The output unit consists of devices with the help of which we get the information from the computer. This unit is a link between the computer and the users. Output devices translate the computer's output into a form understandable by the users.



## Working of the CPU

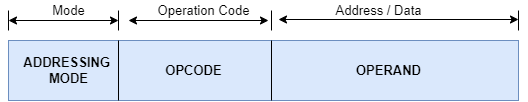
You must be aware that the CPU cannot directly execute the program written in a high-level language. Therefore, all the programs are first compiled to binary format. The compiler converts the high-level program into its equivalent low-level instruction containing 0 and 1. These instructions are machine instructions in computer organization that the processor can directly execute.

The operating system loads the machine instructions into the RAM. The CPU starts the execution by fetching these instructions one by one. These fetched instructions are stored in the instruction registers. In machine instructions, the bits are grouped based on instruction format. Each bit gives specific information to the CPU to decode. Information required by the CPU includes the address of the data and the operation to be performed.

## Instruction Format

The instruction formats are a sequence of bits (0 and 1). These bits, when grouped, are known as fields. Each field of the machine provides specific information to the CPU related to the operation and location of the data.

The instruction format is depicted in a rectangular box, symbolizing the instruction bits in memory words or a control register. The bits grouped are divided into three parts:



1. **Addressing Mode:**The addressing mode indicates how the data is represented.
2. **Opcode:**The opcode part indicates the operation type on the data.
3. **Operand:**The operand part indicates either the data or the address of the data.

The instruction format also defines the layout of the bits for an instruction. It can be of variable lengths with multiple numbers of addresses. These address fields in the instruction format vary as per the organization of the registers in the CPU. The formats supported by the CPU depend upon the Instructions Set Architecture implemented by the processor.

Depending on the multiple address fields, the instruction is categorized as follows:

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

The operations specified by a computer instruction are executed on data stored in memory or processor registers. The operands residing in processor registers are specified with an address. The registered address is a binary number of k bits that defines one of the 2k registers in the CPU. Thus, a CPU with 16 processors registers R0 through R15 and will have a four-bit register address field.

**Example:**The binary number 0011 will designate register R3.

A computer can have instructions of different lengths containing varying numbers of addresses. The number of address fields of a computer depends on the internal design of its registers. Most of the computers fall into one of three types of CPU organizations:

1. Single accumulator organization.
2. General register organization.
3. Stack organization.

### Single Accumulator Organization

All the operations on a system are performed with an implied accumulator register. The instruction format in this type of computer uses one address field.

For example, the instruction for arithmetic addition is defined by an assembly language instruction ‘ADD.’

Where X is the operand’s address, the ADD instruction results in the operation.

AC ← AC + M[X].

AC is the accumulator register, M[X] symbolizes the memory word located at address X.

### General Register Organization

The general register type computers employ two or three address fields in their instruction format. Each address field specifies a processor register or a memory. An instruction symbolized by ADD R1, X specifies the operation R1 ← R + M [X].

This instruction has two address fields: register R1 and memory address X.

### Stack Organization

A computer with a stack organization has PUSH and POP instructions that require an address field.  Hence, the instruction PUSH X pushes the word at address X to the top of the stack. The stack pointer updates automatically. In stack-organized computers, the operation type instructions don’t require an address field as the operation is performed on the two items on the top of the stack.

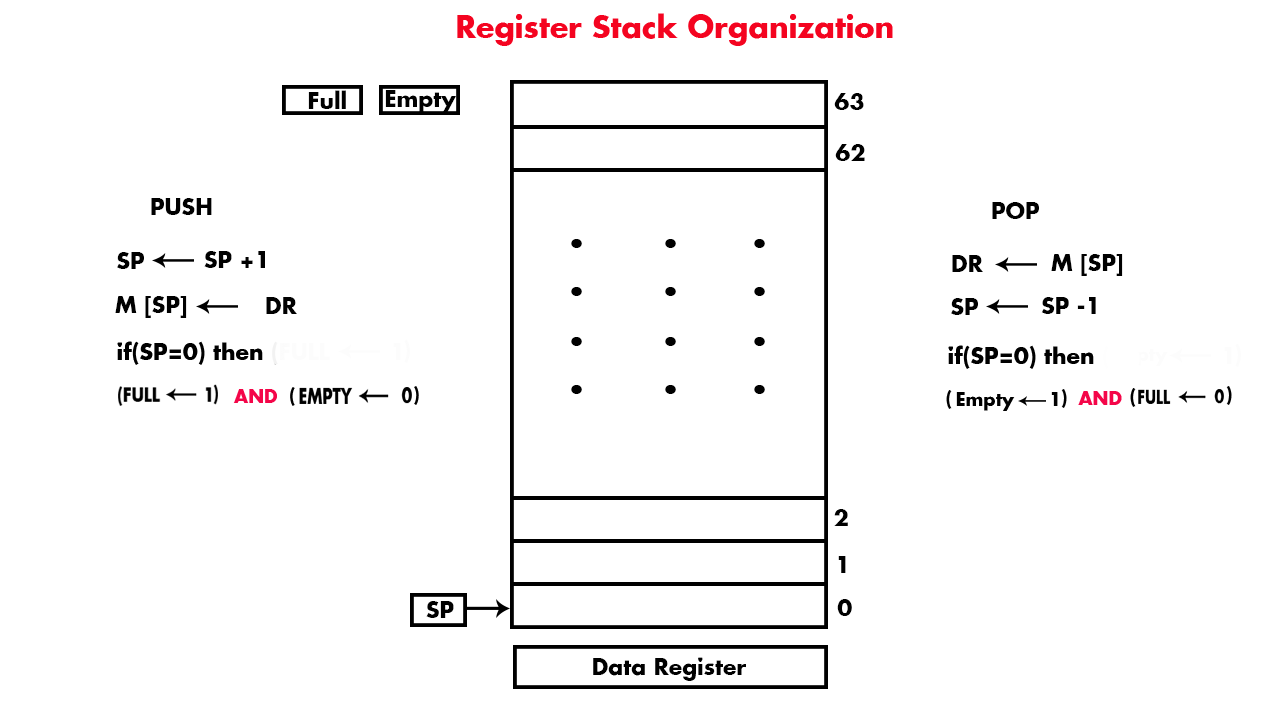
# **Types of CPU Organizations**

The length of instruction depends on the CPU organization. There are four basic types of instructions in CPU Organization which are.

* Zero Address Instruction (Stack Organization)
* Single Address Instructions
* Two address instructions
* Three address instructions

## ****1. Zero Address Instruction (Stack Organization)****

Zero addresses instructions having no operand with OPCODE.



Register Stack Organization Follow the Rule Last In First Out (LIFO).  To add data in stack we use PUSH and to remove we will use POP. Stack organization can be implemented on RAM or Registers.

SP is a stack pointer (top of the stack) which is use to store the address of that location where we have to push or pop any data. As SP stores the address of that location where we PUSH or POP data.

We use two flags registers named as FULL and EMPTY.

If the value of FULL is 1, then Empty will be zero and vice versa.

and If FULL value is 1 then it means there is no more space to enter any new data.

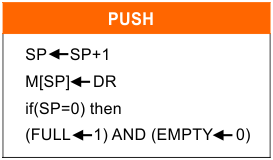
If EMPTY value is 1 than it means we can add new data is stack organized registers.

## ****How to PUSH DATA?****

As in diagram SP points Zero. When we want to add any data from the data register to the stack register.

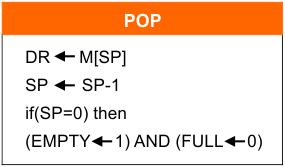
 SP incremented by 1 and Points the first general register having address 1. If we want to add new data then SP incremented to again and data will be pushed to the second general register and so on.

In this way, 63 (1 to 63) data values can be added to different 64 general-purpose registers.  
When 64th data value is add to the 64-register stack organization then SP points to ZERO because (64= 1000000).  As SP size is 6-bits and a higher bit is discarded and least 6-bits are considered. So, SP value becomes (000000=0). When SP=0 then FULL=1 and EMPTY=0 which means there is no more space to PUSH 65th data value in stack.

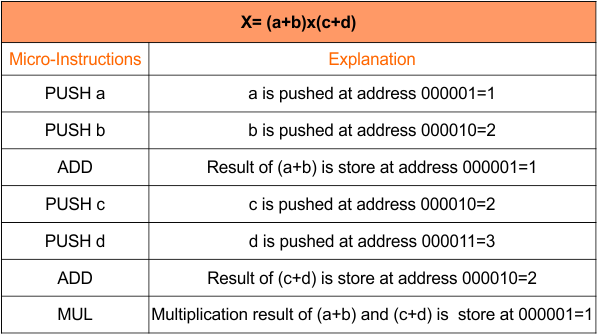


## ****How to POP Data?****

If we want to retrieve data from stack to Data Register then we use POP command. Memory location of SP Address is POP and sends data to Data Register. SP, decremented by 1. In same way next memory location of SP address is POP. We can remove more data from stack until SP points to zero. When SP=0 then EMPTY =1 and FULL=0. It means all stack is empty.



## EXAMPLE: Zero address instruction in register stack organization



When we apply any OPCODE like ADD, MUL etc. then it takes previous two operands from stack apply the operation on them and store the result in the position of first operand. We cannot use any operand with OPCODE because it is a Zero address instruction.

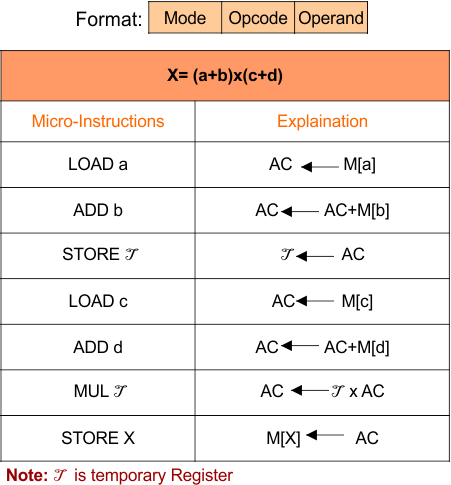
## ****2. Single Address Instructions****

It is also known as Single Accumulator organization because OS use single operand. Here operand may be either the address of memory/ Register or may be a data itself.

It totally depends on addressing mode which tells us how to treat with operand to find actual data.

The format of instruction is: Opcode + Address (Operand)

Some examples of One Address Machine



This technique totally replaced by the introduction of the new general register.

### **Advantages**

* One of the operands is always held by the accumulator register. This results in short instructions and less memory space.
* The Instruction cycle requires less time to complete

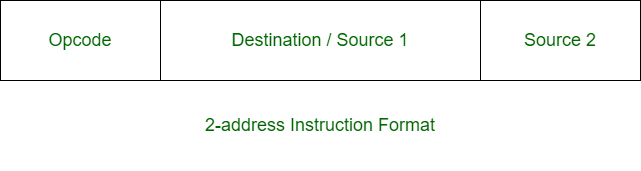
### **Disadvantages**

* When complex expressions are executed then program size will increase due to the execution of many short instructions.
* When the number of instructions increases for a program, then the execution time increases.

## ****3. Two address instructions****

Two and three address instructions are consider as general register organizations.

One address field is common and can be used for either destination or source and other address field for source.



**Example:**

X = (A + B) x (C + D)

## ****Sol:****

MOV R1, A R1 <- M[A]

ADD R1, B R1 <- R1 + M[B]

MOV R2, C R2 <- M[C]

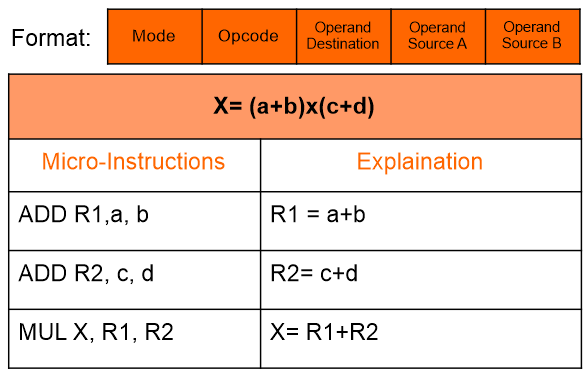
ADD R2, D R2 <- R2 + D

MUL R1, R2 R1 <- R1 x R2

MOV X, R1 M[X] <- R1

## ****4.  Three address instructions****

* Three address instructions use three operands (2 sources and one destination).
* More the number of registers are require as compare to zero, single, two address instructions. So, More space required for registers and buses. That’s why, it is costly than other types of instructions.

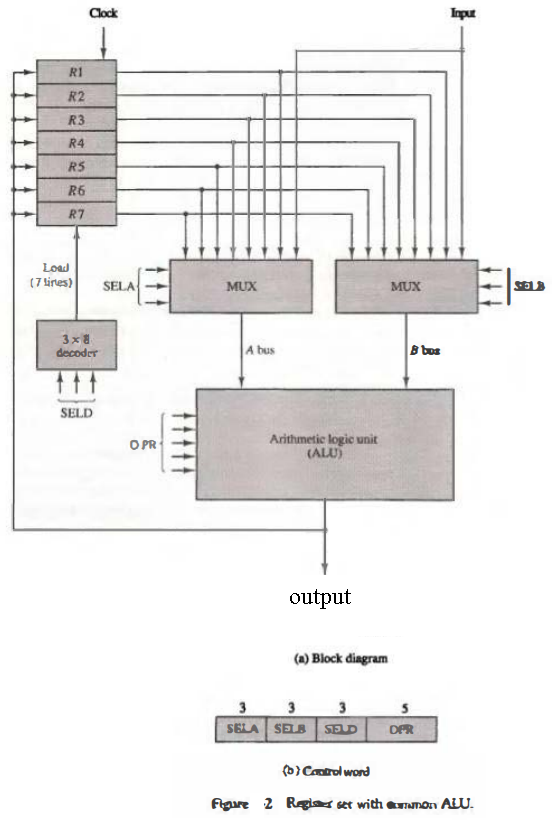


### **General Register Organization**

* **Memory locations** are needed for storing pointers, counters, return addresses, temporary results, and partial products during multiplication.
* **Having to refer to memory locations** for such applications is time consuming because memory access is the most time-consuming, operation in a computer.
* **It is more convenient** and more efficient to store these intermediate values in processor registers.
* **When a large number** of registers are included in the CPU, it is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfers, but also while performing various microoperations.
* **Hence it is necessary** to provide a common unit that can perform all the arithmetic, logic, and shift microoperations in the processor.
* **A bus organization** for seven CPU registers is shown in Fig. 2. The output of each register is connected to two multiplexers (MUX) to form the two buses A and B. The selection lines in each multiplexer select one register or the input data for the particular bus.
* **The A and B buses form** the inputs to a common arithmetic logic unit (ALU).
* **The operation** selected in the ALU determines the arithmetic or logic micro-operation that is to be performed.
* **The result of the microoperation** is available for output data and also goes into the inputs of all the registers.

* **The register** that receives the information from the output bus is selected by a decoder.

* **The decoder** activates one of the register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register.

* 

* **The control unit** that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system. For example, to perform the operation R1 ← R2 + R3

* **the control must** provide binary selection variables to the following selector inputs:

* 1. **MUX A selector (SELA):** to place the content of R2 into bus A.

* 1. **MUX B selector (SELB):** to place the content o f R 3 into bus B.

* 1. **ALU operation selector (OPR):** to provide the arithmetic addition A + B.

* 1. **Decoder destination selector (SELD):** to transfer the content of the output bus into R1.
* **The four control** selection variables are generated in the control unit and must be available at the beginning of a clock cycle.

* **The data** from the two source registers propagate through the gates in the multiplexers and the ALU, to the output bus, and into the inputs of the destination register, all during the clock cycle interval.

* **Then**, when the next clock transition occurs, the binary information from the output bus is transferred into R1.

* **To achieve** a fast response time, the ALU is constructed with high-speed circuits.

### **Control Word**

* **There are 14 binary selection** inputs in the unit, and their combined value specifies a control word. The 14-bit control word is defined in Fig. 2(b).

* **It consists of four fields**. Three fields contain three bits each, and one field has five bits.

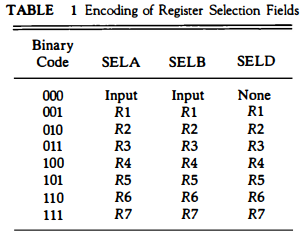
* **The three bits of SELA** select a source register for the A input of the ALU. The three bits of SELB select a register for the B input of the ALU.

* **The three bits of SELD** select a destination register using the decoder and its seven load outputs.

* **The five bits of OPR** select one of the operations in the ALU.

* **The 14-bit control word**when applied to the selection inputs specify a particular microoperation.

* **The encoding of the register**selections is specified in Table 1.

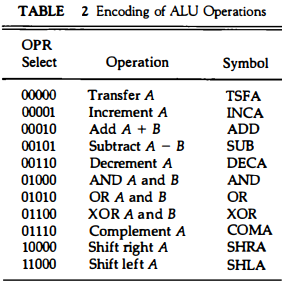
* 

* **The 3-bit binary code** listed in the first column of the table specifies the binary code for each of the three fields.

* **The register** selected by fields SELA, SELB, and SELD is the one whose decimal number is equivalent to the binary number in the code. When SELA or SELB is 000, the corresponding multiplexer selects the external input data.

* **When SELD = 000**, no destination register is selected but the contents of the output bus are available in the external output. The ALU provides arithmetic and logic operations.

* **In addition**, the CPU must provide shift operations. The shifter may be placed in the input of the ALU to provide a preshift capability, or at the output of the ALU to provide postshifting capability. In some cases, the shift operations are included with the ALU.

* **The function table for this ALU** is listed in Table 8. The encoding of the ALU operations for the CPU is specified in Table 2. The OPR field has five bits and each operation is designated with a symbolic name.

## Reverse Polish Notation

Reverse polish notation is also known as postfix notation is defined as: In postfix notation operator is written after the operands. Examples of postfix notation are AB + and CD –. Here A and B are two operands and the operator is written after these two operands. The conversion from infix expression into postfix expression is shown below.

**Question 7: Convert the infix notation {A – B + C × (D × E – F)} / G + H × K into postfix notation?**

**Solution:**

{A – B + C × (D × E – F)} / G + H × K

= {A – B + C × ([DE ×] – F)} / G + [HK ×]

= {A – B + C × [DE × F –]} / [G HK × +]

= {A – B + [C DE × F – ×]} / [G HK × +]

= {[AB –] + [C DE × F – ×]} / [G HK × +]

= [AB – C DE × F – × +] / [G HK × +]

= [AB – C DE × F – × + G HK × + /]

= AB – C DE × F – × + G HK × + /

So the reverse polish notation is AB – C DE × F – × + G HK × + /.

**Question 6: Convert the infix notation A × B + C × D + E × F into Postfix Notation or Reverse Polish Notation?**

**Solution:**

A × B + C × D + E × F

= [AB ×] + [CD ×] + [EF ×]

= [AB × CD × +] + [EF ×]

= [AB × CD × + EF × +]

= AB × CD × + EF × +

So the postfix notation or reverse polish notation of (A × B + C × D + E × F) is AB × CD × + EF × +.

**Addressing Modes**– The term addressing modes refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

**IMPORTANT TERMS**

* **Starting address** of memory segment.
* **Effective address or Offset**: An offset is determined by adding any combination of three address elements: **displacement, base and index.**
  + **Displacement:**It is an 8 bit or 16 bit immediate value given in the instruction.
  + **Base**: Contents of base register, BX or BP.
  + **Index**: Content of index register SI or DI.

**Types of Addressing Modes-**

In computer architecture, there are following types of addressing modes-

1. Implied / Implicit Addressing Mode
2. Immediate Addressing Mode
3. Direct Addressing Mode
4. Indirect Addressing Mode
5. Register Direct Addressing Mode
6. Register Indirect Addressing Mode
7. Relative Addressing Mode
8. Indexed Addressing Mode
9. Base Register Addressing Mode
10. Auto-Increment Addressing Mode
11. Auto-Decrement Addressing Mode

**1. Implied Addressing Mode-**

In this addressing mode,

* The definition of the instruction itself specify the operands implicitly.
* It is also called as **implicit addressing mode**.

**Examples-**

* The instruction “Complement Accumulator” is an implied mode instruction.
* In a stack organized computer, Zero Address Instructions are implied mode instructions.

(since operands are always implied to be present on the top of the stack)

**2. Stack Addressing Mode-**

In this addressing mode,

* The operand is contained at the top of the stack.

**Example-**

ADD

* This instruction simply pops out two symbols contained at the top of the stack.
* The addition of those two operands is performed.
* The result so obtained after addition is pushed again at the top of the stack.

**3. Immediate Addressing Mode-**

In this addressing mode,

* The operand is specified in the instruction explicitly.
* Instead of address field, an operand field is present that contains the operand.

**Examples-**

* ADD 10 will increment the value stored in the accumulator by 10.
* MOV R #20 initializes register R to a constant value 20.

**4. Direct Addressing Mode-**

In this addressing mode,

* The address field of the instruction contains the effective address of the operand.
* Only one reference to memory is required to fetch the operand.
* It is also called as **absolute addressing mode**.

**Example-**

* ADD X will increment the value stored in the accumulator by the value stored at memory location X.

AC ← AC + [X]

**5. Indirect Addressing Mode-**

In this addressing mode,

* The address field of the instruction specifies the address of memory location that contains the effective address of the operand.
* Two references to memory are required to fetch the operand.

**Example-**

* ADD X will increment the value stored in the accumulator by the value stored at memory location specified by X.

AC ← AC + [[X]]

**6. Register Direct Addressing Mode-**

In this addressing mode,

* The operand is contained in a register set.
* The address field of the instruction refers to a CPU register that contains the operand.
* No reference to memory is required to fetch the operand.

**Example-**

* ADD R will increment the value stored in the accumulator by the content of register R.

AC ← AC + [R]

**NOTE-**

It is interesting to note-

* This addressing mode is similar to direct addressing mode.
* The only difference is address field of the instruction refers to a CPU register instead of main memory.

**7. Register Indirect Addressing Mode-**

In this addressing mode,

* The address field of the instruction refers to a CPU register that contains the effective address of the operand.
* Only one reference to memory is required to fetch the operand.

**Example-**

* ADD R will increment the value stored in the accumulator by the content of memory location specified in register R.

AC ← AC + [[R]]

**NOTE-**

It is interesting to note-

* This addressing mode is similar to indirect addressing mode.
* The only difference is address field of the instruction refers to a CPU register.

**8. Relative Addressing Mode-**

In this addressing mode,

* Effective address of the operand is obtained by adding the content of program counter with the address part of the instruction.

|  |
| --- |
| **Effective Address**  **= Content of Program Counter + Address part of the instruction** |

**NOTE-**

* **Program counter** (PC) always contains the address of the next instruction to be executed.
* After fetching the address of the instruction, the value of program counter immediately increases.
* The value increases irrespective of whether the fetched instruction has completely executed or not.

**9. Indexed Addressing Mode-**

In this addressing mode,

* Effective address of the operand is obtained by adding the content of index register with the address part of the instruction.

|  |
| --- |
| **Effective Address**  **= Content of Index Register + Address part of the instruction** |

**10. Base Register Addressing Mode-**

In this addressing mode,

* Effective address of the operand is obtained by adding the content of base register with the address part of the instruction.

|  |
| --- |
| **Effective Address**  **= Content of Base Register + Address part of the instruction** |

**11. Auto-Increment Addressing Mode-**

* This addressing mode is a special case of Register Indirect Addressing Mode where-

|  |
| --- |
| **Effective Address of the Operand**  **= Content of Register** |

In this addressing mode,

* After accessing the operand, the content of the register is automatically incremented by step size ‘d’.
* Step size ‘d’ depends on the size of operand accessed.
* Only one reference to memory is required to fetch the operand.

**Example-**

Assume operand size = 2 bytes.

Here,

* After fetching the operand 6B, the instruction register RAUTO will be automatically incremented by 2.
* Then, updated value of RAUTO will be 3300 + 2 = 3302.
* At memory address 3302, the next operand will be found.

**NOTE-**

In auto-increment addressing mode,

* First, the operand value is fetched.
* Then, the instruction register RAUTO value is incremented by step size ‘d’.

**12. Auto-Decrement Addressing Mode-**

* This addressing mode is again a special case of Register Indirect Addressing Mode where-

|  |
| --- |
| **Effective Address of the Operand**  **= Content of Register – Step Size** |

In this addressing mode,

* First, the content of the register is decremented by step size ‘d’.
* Step size ‘d’ depends on the size of operand accessed.
* After decrementing, the operand is read.
* Only one reference to memory is required to fetch the operand.

**Example-**

Assume operand size = 2 bytes.

Here,

* First, the instruction register RAUTO will be decremented by 2.
* Then, updated value of RAUTO will be 3302 – 2 = 3300.
* At memory address 3300, the operand will be found.

**NOTE-**

In auto-decrement addressing mode,

* First, the instruction register RAUTO value is decremented by step size ‘d’.
* Then, the operand value is fetched.

Difference between Hardwired and Microprogrammed Control Unit

| **Hardwired Control Unit** | **Microprogrammed Control Unit** |
| --- | --- |
| Hardwired control unit generates the control signals needed for the processor using logic circuits | Microprogrammed control unit generates the control signals with the help of micro instructions stored in control memory |
| Hardwired control unit is faster when compared to microprogrammed control unit as the required control signals are generated with the help of hardwares | This is slower than the other as micro instructions are used for generating signals here |
| Difficult to modify as the control signals that need to be generated are hard wired | Easy to modify as the modification need to be done only at the instruction level |
| More costlier as everything has to be realized in terms of logic gates | Less costlier than hardwired control as only micro instructions are used for generating control signals |
| It cannot handle complex instructions as the circuit design for it becomes complex | It can handle complex instructions |
| Only limited number of instructions are used due to the hardware implementation | Control signals for many instructions can be generated |
| Used in computer that makes use of Reduced Instruction Set Computers(RISC) | Used in computer that makes use of Complex Instruction Set Computers(CISC) |

67

Difference between the RISC and CISC Processors

|  |  |
| --- | --- |
| **RISC** | **CISC** |
| It is a Reduced Instruction Set Computer. | It is a Complex Instruction Set Computer. |
| It emphasizes on software to optimize the instruction set. | It emphasizes on hardware to optimize the instruction set. |
| It is a hard wired unit of programming in the RISC Processor. | Microprogramming unit in CISC Processor. |
| It requires multiple register sets to store the instruction. | It requires a single register set to store the instruction. |
| RISC has simple decoding of instruction. | CISC has complex decoding of instruction. |
| Uses of the pipeline are simple in RISC. | Uses of the pipeline are difficult in CISC. |
| It uses a limited number of instruction that requires less time to execute the instructions. | It uses a large number of instruction that requires more time to execute the instructions. |
| It uses LOAD and STORE that are independent instructions in the register-to-register a program's interaction. | It uses LOAD and STORE instruction in the memory-to-memory interaction of a program. |
| RISC has more transistors on memory registers. | CISC has transistors to store complex instructions. |
| The execution time of RISC is very short. | The execution time of CISC is longer. |
| RISC architecture can be used with high-end applications like telecommunication, image processing, video processing, etc. | CISC architecture can be used with low-end applications like home automation, security system, etc. |
| It has fixed format instruction. | It has variable format instruction. |
| The program written for RISC architecture needs to take more space in memory. | Program written for CISC architecture tends to take less space in memory. |
| Example of RISC: ARM, PA-RISC, Power Architecture, Alpha, AVR, ARC and the SPARC. | Examples of CISC: VAX, Motorola 68000 family, System/360, AMD and the Intel x86 CPUs. |

**Reduced Instruction Set Architecture (RISC) –**   
The main idea behind this is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating, and storing operations just like a load command will load data, a store command will store the data.

**Complex Instruction Set Architecture (CISC) –**   
The main idea is that a single instruction will do all loading, evaluating, and storing operations just like a multiplication command will do stuff like loading data, evaluating, and storing it, hence it’s complex.

Both approaches try to increase the CPU performance

* **RISC:** Reduce the cycles per instruction at the cost of the number of instructions per program.
* **CISC:** The CISC approach attempts to minimize the number of instructions per program but at the cost of an increase in the number of cycles per instruction.

**Characteristic of RISC –**

1. Simpler instruction, hence simple instruction decoding.
2. Instruction comes undersize of one word.
3. Instruction takes a single clock cycle to get executed.
4. More general-purpose registers.
5. Simple Addressing Modes.
6. Fewer Data types.
7. A pipeline can be achieved.

**Characteristic of CISC –**

1. Complex instruction, hence complex instruction decoding.
2. Instructions are larger than one-word size.
3. Instruction may take more than a single clock cycle to get executed.
4. Less number of general-purpose registers as operations get performed in memory itself.
5. Complex Addressing Modes.
6. More Data types.

### Advantages of RISC:

**Simpler instructions:**RISC processors use a smaller set of simple instructions, which makes them easier to decode and execute quickly. This results in faster processing times.  
**Faster execution:**Because RISC processors have a simpler instruction set, they can execute instructions faster than CISC processors.  
**Lower power consumption:** RISC processors consume less power than CISC processors, making them ideal for portable devices.

### Disadvantages of RISC:

**More instructions required:**RISC processors require more instructions to perform complex tasks than CISC processors.  
**Increased memory usage:**RISC processors require more memory to store the additional instructions needed to perform complex tasks.  
**Higher cost:** Developing and manufacturing RISC processors can be more expensive than CISC processors.

### Advantages of CISC:

**Reduced code size:** CISC processors use complex instructions that can perform multiple operations, reducing the amount of code needed to perform a task.  
**More memory efficient:**Because CISC instructions are more complex, they require fewer instructions to perform complex tasks, which can result in more memory-efficient code.  
**Widely used:** CISC processors have been in use for a longer time than RISC processors, so they have a larger user base and more available software.

### Disadvantages of CISC:

**Slower execution:**CISC processors take longer to execute instructions because they have more complex instructions and need more time to decode them.  
**More complex design:**CISC processors have more complex instruction sets, which makes them more difficult to design and manufacture.  
**Higher power consumption:**CISC processors consume more power than RISC processors because of their more complex instruction sets.