Central Processing Unit

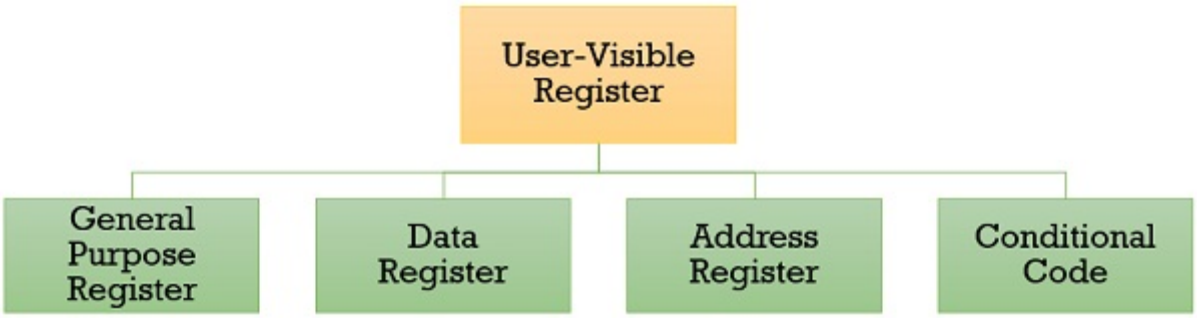
Register organization is the arrangement of the registers in the processor. The processor designers decide the organization of the registers in a processor. Different processors may have different register organization. Depending on the roles played by the registers they can be categorized into two types, user-visible register and control and status register.

Registers are the smaller and the fastest accessible memory units in the central processing unit (CPU). According to memory hierarchy, the registers in the processor function a level above the main memory and cache memory. The registers used by the central unit are also called as processor registers.

A register can hold the instruction, address location, or operands. Sometimes, the instruction has register as a part of itself. Registers can be organized into two main categories i.e. the User-Visible Registers and the Control and Status Registers.

User-Visible Registers

These registers are visible to the assembly or machine language programmers and they use them effectively to minimize the memory references in the instructions. Well, these registers can only be referenced using the machine or assembly language.



The registers that fall in this category are discussed below:

1. **General Purpose Register**

The general-purpose registers detain both the addresses and the data. Although we have separate data registers and address registers. The general purpose register also accepts the intermediate results in the course of program execution.

Well, the programmers can restrict some of the general-purpose registers to specific functions. Like, some registers are specifically used for stack operations or for floating-point operations. The general-purpose register can also be employed for the addressing functions.

2. **Data Register**

The term itself describes that these registers are employed to hold the data. But the programmers can't use these registers for calculating operand address.

3. **Address Register**

Now, the address registers contain the address of an operand or it can also act as a general-purpose register. An address register may be dedicated to a certain addressing mode. Let us understand this with the examples.

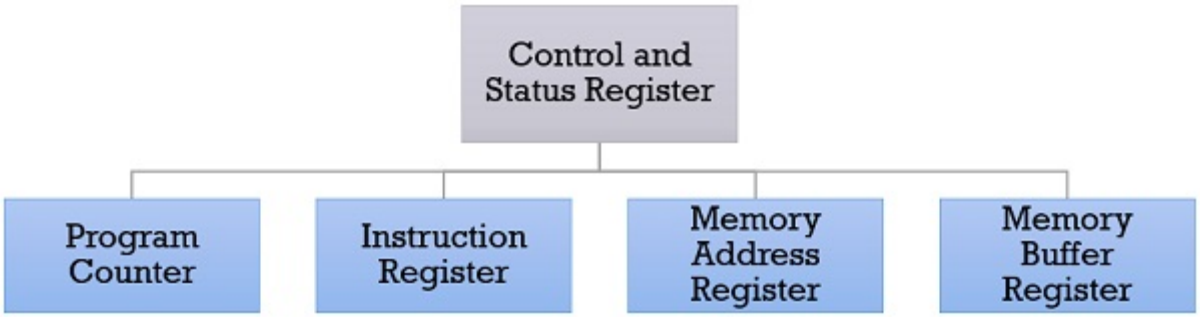
(a) Segment Pointer Register

A memory divided in segments, requires a segment register to hold the base address of the segment. There can be multiple segment registers. As one segment register can be employed to hold the base address of the segment occupied by the operating system. The other segment register can hold the base address of the segment allotted to the processor.

(b) **Index Register**
The index register is employed for indexed addressing and it is initial value is 0. Generally, it used for traversing the memory locations. After each reference, the index register is incremented or decremented by 1, depending upon the nature of the operation.
Sometime the index register may be auto indexed.
(c) Stack Pointer Register
The stack register has the address that points the stack top.

4. **Condition Code**
Condition codes are the flag bits which are the part of the control register. The condition codes are set by the processor as a result of an operation and they are implicitly read through the machine instruction.
The programmers are not allowed to alter the conditional codes. Generally, the condition codes are tested during conditional branch operation.

****Control and Status Registers**
The control and status register holds the address or data that is important to control the processor's operation. The most important thing is that these registers are not visible to the users. Below we will discuss all the control and status registers are essential for the execution of an instruction.



1. **Program Counter**
The program counter is a processor register that holds the address of the instruction that has to be executed next. It is a processor which updates the program counter with the address of the next instruction to be fetched for execution.

2. **Instruction Register**
Instruction register has the instruction that is currently fetched. It helps in analysing the opcode and operand present in the instruction.

3. **Memory Address Register (MAR)**
Memory address register holds the address of a memory location.

4. **Memory Buffer Register (MBR)**
The memory buffer register holds the data that has to be written to a memory location or it holds the data that is recently been read.
The memory address registers (MAR) and memory buffer registers (MBR) are used to move the data between processor and memory.
Apart from the above registers, several processors have a register termed as Program Status Word (PSW). As the word suggests it contains the status information.
The fields included in **Program Status Word (PSW**):

- Sign: This field has the resultant sign bit of the last arithmetic operation performed.
- Zero: This field is set when the result of the operation is zero.
- Carry: This field is set when an arithmetic operation results in a carry into or borrow out.

- Equal: If a logical operation results in, equality the Equal bit is set.
- Overflow: This bit indicates the arithmetic overflow.
- Interrupt: This bit is set to enable or disable the interrupts.
- Supervisor: This bit indicates whether the processor is executing in the supervisor mode or the user mode.

STACK ORGANIZATION

Stack is a storage structure that stores information in such a way that the last item stored is the first item retrieved. It is based on the principle of LIFO (Last-in-first-out). The stack in digital computers is a group of memory locations with a register that holds the address of top of element. This register that holds the address of top of element of the stack is called Stack Pointer.

Stack Operations

The two operations of a stack are:

1. **Push**: Inserts an item on top of stack.
2. **Pop**: Deletes an item from top of stack.

Implementation of Stack

In digital computers, stack can be implemented in two ways:

1. **Register Stack**
2. **Memory Stack**

Register Stack

A stack can be organized as a collection of finite number of registers that are used to store temporary information during the execution of a program. The stack pointer (SP) is a register that holds the address of top of element of the stack.

Memory Stack

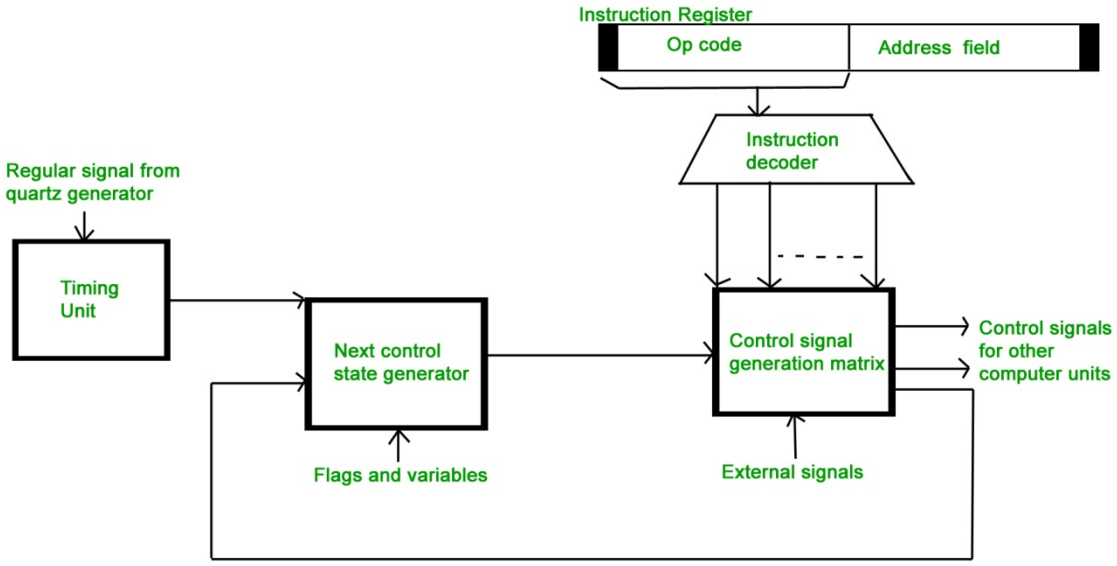
A stack can be implemented in a random access memory (RAM) attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. The starting memory location of the stack is specified by the processor register as stack pointer.

To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. There are two approaches used for generating the control signals in proper sequence as Hardwired Control unit and Micro-programmed control unit.

*****Hardwired Control Unit –**

The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs. The outputs of the state machine are the control signals. The sequence of the operation carried out by this machine is determined by the wiring of the logic elements and hence named as “hardwired”.

- Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.
- Hardwired control is faster than micro-programmed control.
- A controller that uses this approach can operate at high speed.
- RISC architecture is based on hardwired control unit

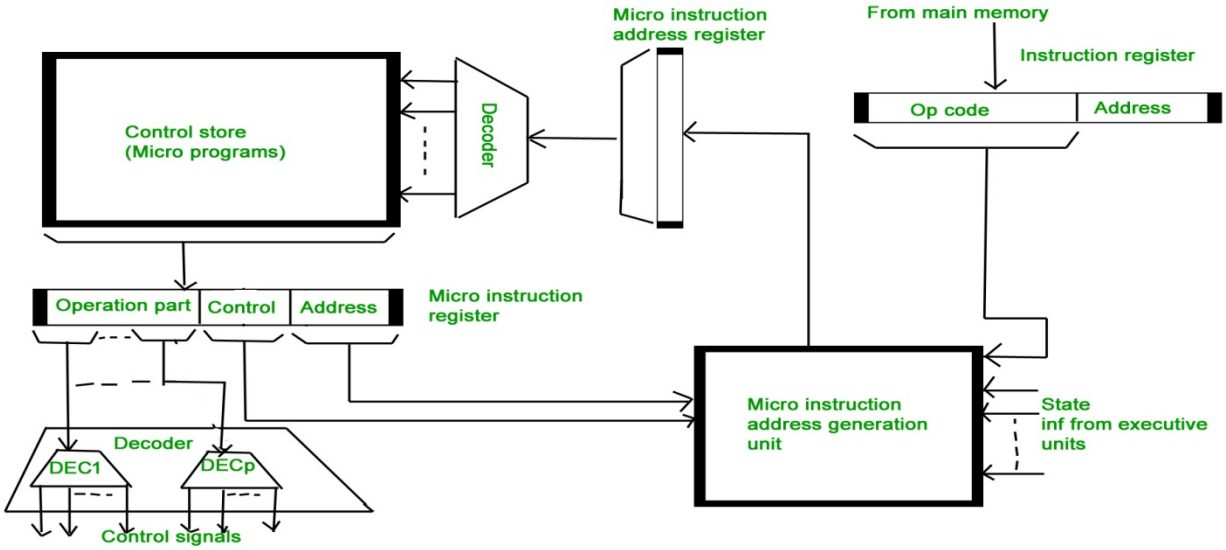


*****Micro-programmed Control Unit –**

- The control signals associated with operations are stored in special memory units inaccessible by the programmer as Control Words.
- Control signals are generated by a program are similar to machine language programs.
- Micro-programmed control unit is slower in speed because of the time it takes to fetch microinstructions from the control memory.

Some Important Terms –

1. Control Word : A control word is a word whose individual bits represent various control signals.
2. Micro-routine : A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro-routine for that instruction.
3. Micro-instruction : Individual control words in this micro-routine are referred to as microinstructions.
4. Micro-program : A sequence of micro-instructions is called a micro-program, which is stored in a ROM or RAM called a Control Memory (CM).
5. Control Store : the micro-routines for all instructions in the instruction set of a computer are stored in a special memory called the Control Store.



Types of Micro-programmed Control Unit – Based on the type of Control Word stored in the Control Memory (CM), it is classified into two types :

1. **Horizontal Micro-programmed control Unit :**
The control signals are represented in the decoded binary format that is 1 bit/CS. Example: If 53 Control signals are present in the processor than 53 bits are required. More than 1 control signal can be enabled at a time.

- It supports longer control word.
- It is used in parallel processing applications.
- It allows higher degree of parallelism. If degree is n, n CS are enabled at a time.
- It requires no additional hardware (decoders). It means it is faster than Vertical Micro programmed.
- It is more flexible than vertical micro programmed

2. **Vertical Micro-programmed control Unit :**
The control signals represented in the encoded binary format. For N control signals- Log2 (N) bits are required.

- It supports shorter control words.
- It supports easy implementation of new control signals therefore it is more flexible.
- It allows low degree of parallelism i.e., degree of parallelism is either 0 or 1.
- Requires an additional hardware (decoders) to generate control signals, it implies it is slower than horizontal microprogrammed.
- It is less flexible than horizontal but more flexible than that of hardwired control unit.

*****Instruction Format:**

Computer perform task on the basis of instruction provided. A instruction in computer comprises of groups called fields. These field contains different information as for computers everything is in 0 and 1 so each field has different significance on the basis of which a CPU decide what so perform. The most common fields are:

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

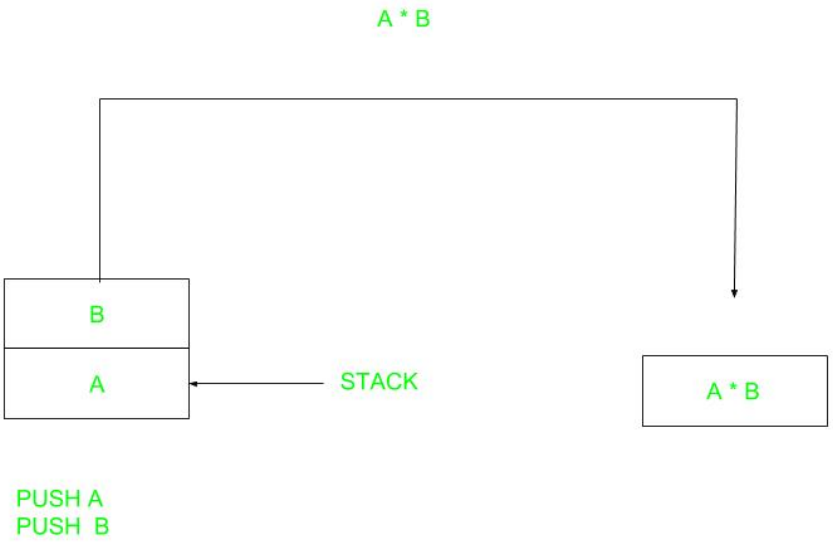
A instruction is of various length depending upon the number of addresses it contain. Generally CPU organization are of three types on the basis of number of address fields:

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

In first organization operation is done involving a special register called accumulator. In second on multiple registers are used for the computation purpose. In third organization the work on stack basis operation due to which it does not contain any address field. It is not necessary that only a single organization is applied a blend of various organization is mostly what we see generally.

On the basis of number of address instruction are classified as:
Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.

1. **Zero Address Instructions** –



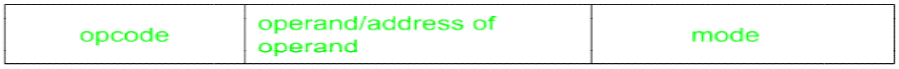
A stack based computer do not use address field in instruction.To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

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

2. **One Address Instructions –**

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



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

3. **Two Address Instructions –**

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

Here destination address can also contain operand.

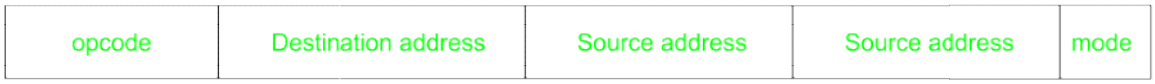
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location

MOV	R1, A	R1 = M[A]
ADD	R1, B	R1 = R1 + M[B]

MOV	R2, C	R2 = C
ADD	R2, D	R2 = R2 + D
MUL	R1, R2	R1 = R1 * R2
MOV	X, R1	M[X] = R1

4. **Three Address Instructions –**

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



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

The operation field of an instruction specifies the operation to be performed. This operation will be executed on some data which is stored in computer registers or the main memory. The way any operand is selected during the program execution is dependent on the addressing mode of the instruction. **The purpose of using addressing modes is as follows:**

1. **To give the programming versatility to the user.**
2. **To reduce the number of bits in addressing field of instruction.**

******Types of Addressing Modes:**

Below we have discussed different types of addressing modes one by one:

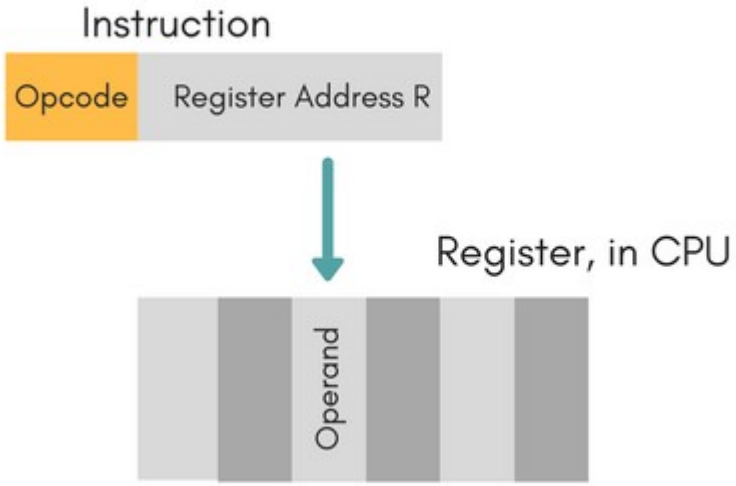
Immediate Mode

In this mode, the operand is specified in the instruction itself. An immediate mode instruction has an operand field rather than the address field.

For example: ADD 7, which says Add 7 to contents of accumulator. 7 is the operand here.

Register Mode

In this mode the operand is stored in the register and this register is present in CPU. The instruction has the address of the Register where the operand is stored.



Advantages

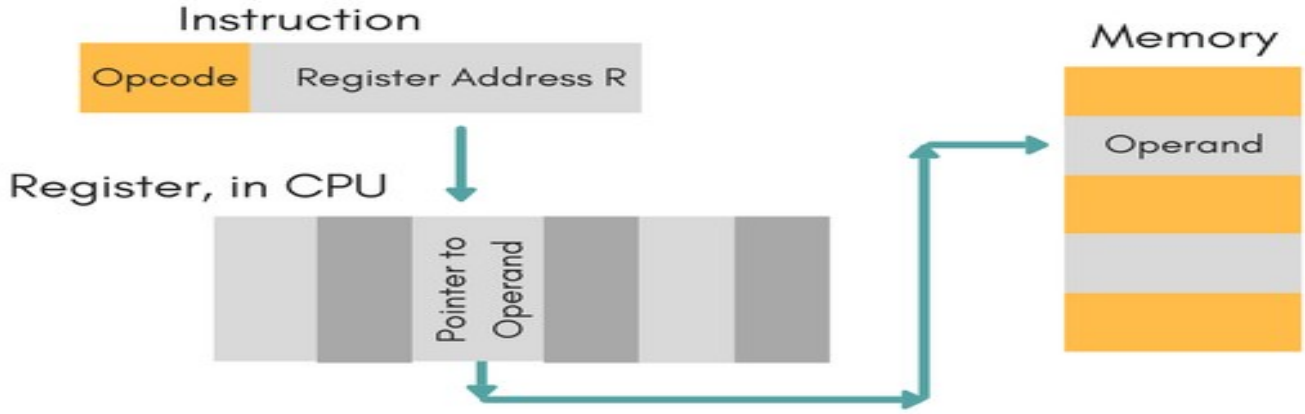
- Shorter instructions and faster instruction fetch.
- Faster memory access to the operand(s)

Disadvantages

- Very limited address space
- Using multiple registers helps performance but it complicates the instructions.

Register Indirect Mode

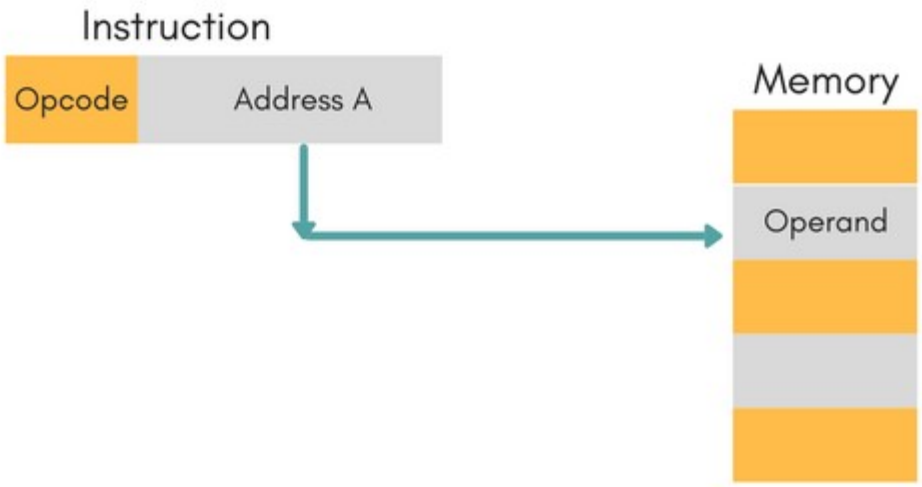
In this mode, the instruction specifies the register whose contents give us the address of operand which is in memory. Thus, the register contains the address of operand rather than the operand itself.



Auto Increment/Decrement Mode
In this the register is incremented or decremented after or before its value is used.

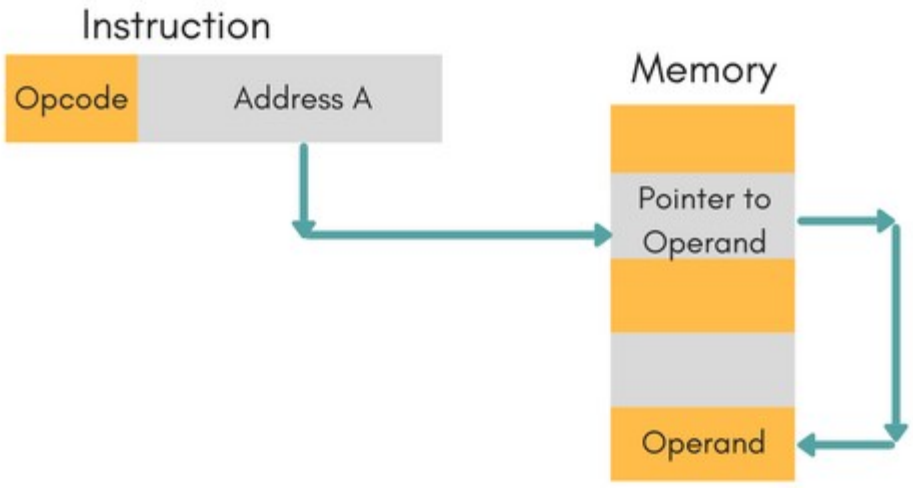
Direct Addressing Mode
In this mode, effective address of operand is present in instruction itself.

- Single memory reference to access data.
- No additional calculations to find the effective address of the operand.



For Example: ADD R1, 4000 - In this the 4000 is effective address of operand.
NOTE: Effective Address is the location where operand is present.

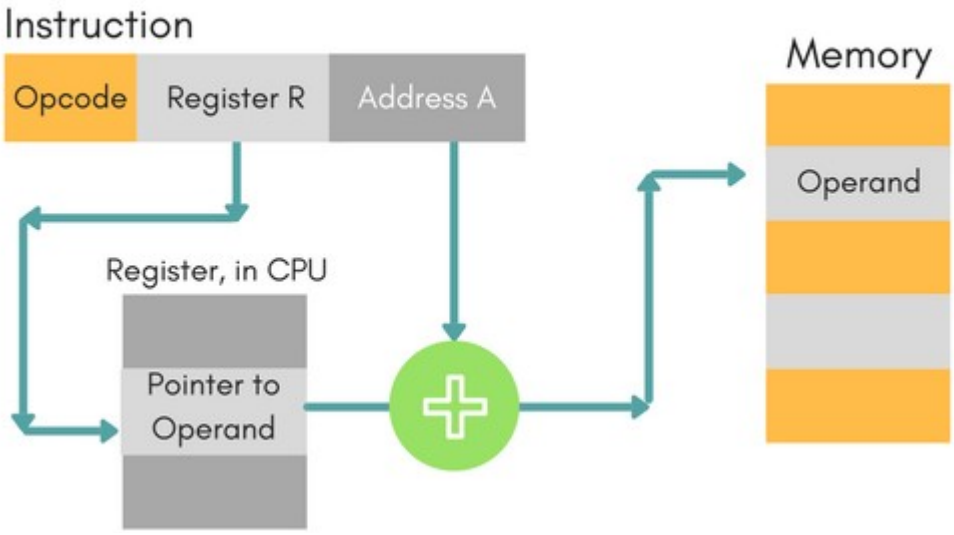
Indirect Addressing Mode
In this, the address field of instruction gives the address where the effective address is stored in memory. This slows down the execution, as this includes multiple memory lookups to find the operand.



Displacement Addressing Mode

In this the contents of the indexed register is added to the Address part of the instruction, to obtain the effective address of operand.

EA = A + (R), In this the address field holds two values, A(which is the base value) and R(that holds the displacement), or vice versa.



Relative Addressing Mode

It is a version of Displacement addressing mode.

In this the contents of PC(Program Counter) is added to address part of instruction to obtain the effective address.

EA = A + (PC), where EA is effective address and PC is program counter.

The operand is A cells away from the current cell(the one pointed to by PC)

Base Register Addressing Mode

It is again a version of Displacement addressing mode. This can be defined as EA = A + (R), where A is displacement and R holds pointer to base address.

Stack Addressing Mode

In this mode, operand is at the top of the stack. For example: ADD, this instruction will POP top two items from the stack, add them, and will then PUSH the result to the top of the stack.

*****Instruction Cycle**

An instruction cycle, also known as fetch-decode-execute cycle is the basic operational process of a computer. This process is repeated continuously by CPU from boot up to shut down of computer.

Following are the steps that occur during an instruction cycle:

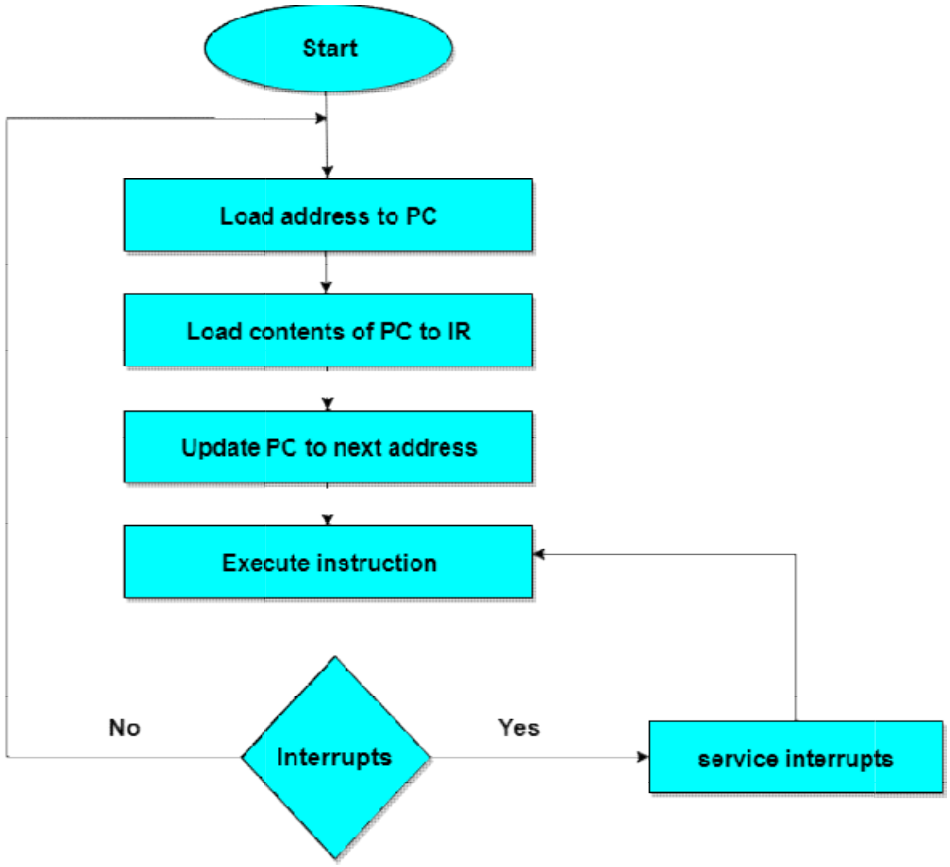
1. **Fetch the Instruction**

The instruction is fetched from memory address that is stored in PC(Program Counter) and stored in the instruction register IR. At the end of the fetch operation, PC is incremented by 1 and it then points to the next instruction to be executed.

2. **Decode the Instruction**
The instruction in the IR is executed by the decoder.

3. **Read the Effective Address**
If the instruction has an indirect address, the effective address is read from the memory. Otherwise operands are directly read in case of immediate operand instruction.

4. **Execute the Instruction**
The Control Unit passes the information in the form of control signals to the functional unit of CPU. The result generated is stored in main memory or sent to an output device.
The cycle is then repeated by fetching the next instruction. Thus in this way the instruction cycle is repeated continuously.



Assembly Language:

Each personal computer has a microprocessor that manages the computer's arithmetical, logical, and control activities.

Each family of processors has its own set of instructions for handling various operations such as getting input from keyboard, displaying information on screen and performing various other jobs. These set of instructions are called 'machine language instructions'.

A processor understands only machine language instructions, which are strings of 1's and 0's. However, machine language is too obscure and complex for using in software development. So, the low-level assembly language is designed for a specific family of processors that represents various instructions in symbolic code and a more understandable form.

Advantages of Assembly Language

Having an understanding of assembly language makes one aware of −

- How programs interface with OS, processor, and BIOS;
- How data is represented in memory and other external devices;
- How the processor accesses and executes instruction;
- How instructions access and process data;
- How a program accesses external devices.

Other advantages of using assembly language are −

- It requires less memory and execution time;
- It allows hardware-specific complex jobs in an easier way;
- It is suitable for time-critical jobs;
- It is most suitable for writing interrupt service routines and other memory resident programs.

Basic Features of PC Hardware

The main internal hardware of a PC consists of processor, memory, and registers. Registers are processor components that hold data and address. To execute a program, the system copies it from the external device into the internal memory. The processor executes the program instructions.

The fundamental unit of computer storage is a bit; it could be ON (1) or OFF (0) and a group of 8 related bits makes a byte on most of the modern computers.

So, the parity bit is used to make the number of bits in a byte odd. If the parity is even, the system assumes that there had been a parity error (though rare), which might have been caused due to hardware fault or electrical disturbance.

The processor supports the following data sizes −

- **Word: a 2-byte data item**
- **Doubleword: a 4-byte (32 bit) data item**
- **Quadword: an 8-byte (64 bit) data item**
- **Paragraph: a 16-byte (128 bit) area**
- **Kilobyte: 1024 bytes**
- **Megabyte: 1,048,576 bytes**

Programmed input–output (also programmed input/output, programmed I/O, PIO) is a method of transferring data between the CPU and a peripheral, such as a network adapter or an ATA storage device. Each data item transfer is initiated by an instruction in the program, involving the CPU for every transaction. In contrast, in direct memory access (DMA) operations, the CPU is not involved in the data transfer.

The term can refer to either memory-mapped I/O (MMIO) or port-mapped I/O (PMIO). PMIO refers to transfers using a special address space outside of normal memory, usually accessed with dedicated instructions, such as IN and OUT in x86 architectures. MMIO refers to transfers to input–output (I/O) devices that are mapped into the normal address space available to the program. PMIO was very useful for early microprocessors with small address spaces, since the valuable resource was not consumed by the I/O devices.

The best known example of a PC device that uses programmed I/O is the ATA interface; however, this interface can also be operated in any of several DMA modes. Many older devices in a PC also use PIO, including legacy serial ports, legacy parallel ports when not in ECP mode, the PS/2 keyboard and mouse ports, legacy MIDI and joystick ports, the interval timer, and older network interfaces.

***Reduced Set Instruction Set Architecture (RISC) –**
The main idea behind 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, 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 increase in number of cycles per instruction.

$$CPU\ Time = \frac{Seconds}{Program} = \frac{Instructions}{Program} X \frac{Cycles}{Instructions} X \frac{Seconds}{Cycle}$$

Earlier when programming was done using assembly language, a need was felt to make instruction do more task because programming in assembly was tedious and error prone due to which CISC architecture evolved but with uprise of high level language dependency on assembly reduced RISC architecture prevailed.

Characteristic of RISC –

1. Simpler instruction, hence simple instruction decoding.
2. Instruction come under size of one word.
3. Instruction take single clock cycle to get executed.
4. More number of general purpose register.
5. Simple Addressing Modes.
6. Less Data types.
7. Pipeling can be achieved.

Characteristic of CISC –

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

Example – Suppose we have to add two 8-bit number:

- CISC approach: There will be a single command or instruction for this like ADD which will perform the task.

- RISC approach: Here programmer will write first load command to load data in registers then it will use suitable operator and then it will store result in desired location.

So, add operation is divided into parts i.e. load, operate, store due to which RISC programs are longer and require more memory to get stored but require less transistors due to less complex command.

Difference –

RISC	CISC
Focus on software	Focus on hardware
Uses only Hardwired control unit	Uses both hardwired and micro programmed control unit
Transistors are used for more registers	Transistors are used for storing complex Instructions
Fixed sized instructions	Variable sized instructions
Can perform only Register to Register Arthmetic operations	Can perform REG to REG or REG to MEM or MEM to MEM
Requires more number of registers	Requires less number of registers
Code size is large	Code size is small
A instruction execute in single clock cycle	Instruction take more than one clock cycle
A instruction fit in one word	Instruction are larger than size of one word

Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing.
Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.
Pipelining increases the overall instruction throughput.
In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment.

Types of Pipeline
It is divided into 2 categories:

1. Arithmetic Pipeline
2. Instruction Pipeline

Arithmetic Pipeline

Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc

Instruction Pipeline

In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system.
An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

Pipeline Conflicts

There are some factors that cause the pipeline to deviate its normal performance. Some of these factors are given below:

1. **Timing Variations**

All stages cannot take same amount of time. This problem generally occurs in instruction processing where different instructions have different operand requirements and thus different processing time.

2. **Data Hazards**

When several instructions are in partial execution, and if they reference same data then the problem arises. We must ensure that next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results.

3. **Branching**

In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its result will lead us to the next instruction, then the next instruction may not be known until the current one is processed.

4. **Interrupts**

Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.

5. **Data Dependency**

It arises when an instruction depends upon the result of a previous instruction but this result is not yet available.

Advantages of Pipelining

1. The cycle time of the processor is reduced.
2. It increases the throughput of the system
3. It makes the system reliable.

Disadvantages of Pipelining

1. The design of pipelined processor is complex and costly to manufacture.

2. The instruction latency is more.

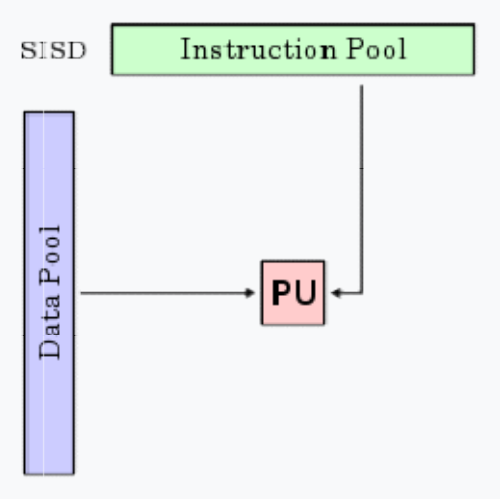
Parallel Architecture is a computing where the jobs are broken into discrete parts that can be executed concurrently. Each part is further broken down to a series of instructions. Instructions from each part execute simultaneously on different CPUs. Parallel systems deal with the simultaneous use of multiple computer resources that can include a single computer with multiple processors, a number of computers connected by a network to form a parallel processing cluster or a combination of both.
Parallel systems are more difficult to program than computers with a single processor because the architecture of parallel computers varies accordingly and the processes of multiple CPUs must be coordinated and synchronized.

The four classifications defined by Flynn are based upon the number of concurrent instruction (or control) streams and data streams available in the architecture.

Single instruction stream, single data stream (SISD)

A sequential computer which exploits no parallelism in either the instruction or data streams. Single control unit (CU) fetches single instruction stream (IS) from memory. The CU then generates appropriate control signals to direct single processing element (PE) to operate on single data stream (DS) i.e., one operation at a time.

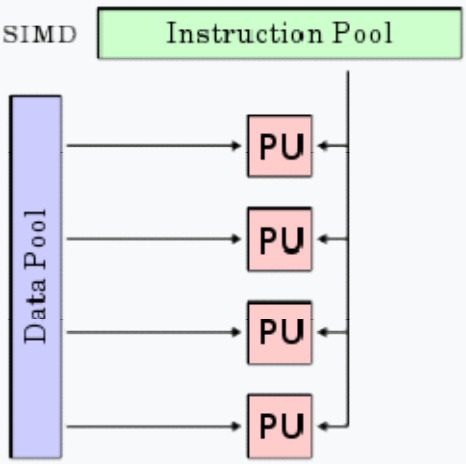
Examples of SISD architecture are the traditional uniprocessor machines like older personal computers



Single instruction stream, multiple data streams (SIMD)

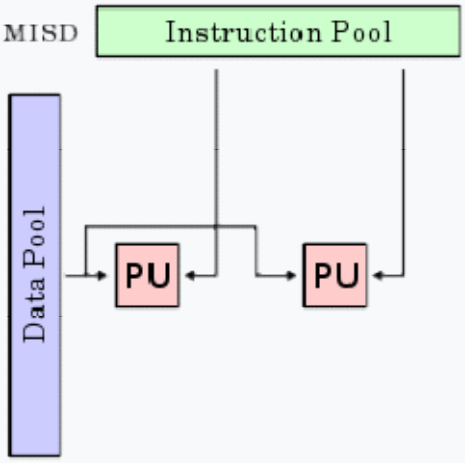
A single instruction operates on multiple different data streams. Instructions can be executed sequentially, such as by pipelining, or in parallel by multiple functional units.

Single instruction, multiple threads (SIMT) is an execution model used in parallel computing where single instruction, multiple data (SIMD) is combined with multithreading. This is not a distinct classification in Flynn's taxonomy, where it would be a subset of SIMD. Nvidia commonly uses the term in its marketing materials and technical documents where it argues for the novelty of Nvidia architecture.



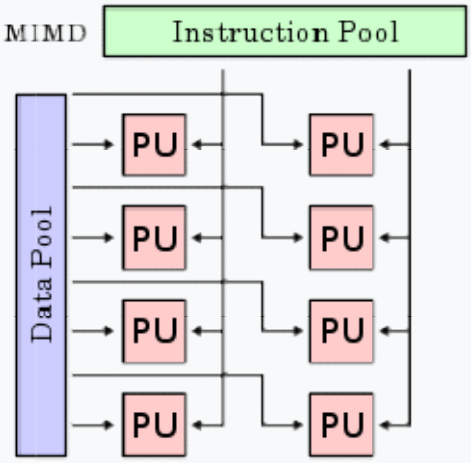
Multiple instruction streams, single data stream (MISD)

Multiple instructions operate on one data stream. This is an uncommon architecture which is generally used for fault tolerance. Heterogeneous systems operate on the same data stream and must agree on the result. Examples include the Space Shuttle flight control computer.



Multiple instruction streams, multiple data streams (MIMD)

Multiple autonomous processors simultaneously executing different instructions on different data. MIMD architectures include multi-core superscalar processors, and distributed systems, using either one shared memory space or a distributed memory space.


MIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
PU
PU
PU
PU