MODULE – II: ORGANIZATION OF A COMPUTER

Instruction code:

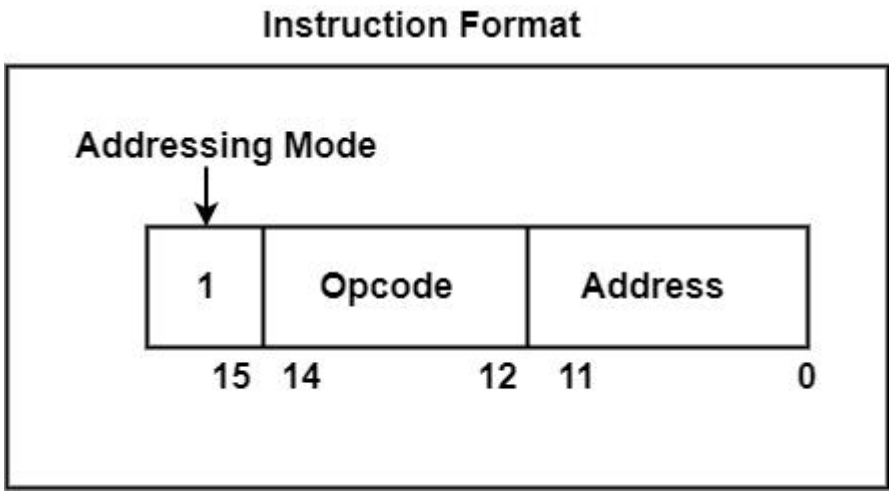
What are Instruction Codes and Operands in Computer Architecture?

A computer instruction is a binary code that determines the micro-operations in a sequence for a computer. They are saved in the memory along with the information. Each computer has its specific group of instructions.

They can be categorized into two elements as Operation codes (Opcodes) and Address. Opcodes specify the operation for specific instructions. An address determines the registers or the areas that can be used for that operation. Operands are definite elements of computer instruction that show what information is to be operated on.

It consists of 12 bits of memory that are required to define the address as the memory includes 4096 words. The 15th bit of the instruction determines the addressing mode (where direct addressing corresponds to 0, indirect addressing corresponds to 1). Therefore, the instruction format includes 12 bits of address and 1 bit for the addressing mode, 3 bits are left for Opcodes.

The following block diagram shows the instruction format for a basic computer.



There are three parts of the Instruction Format which are as follows −

Addressing Modes

Instructions that define the address of a definite memory location are known as memory reference instructions. The method in which a target address or effective address is recognized within the instruction is known as addressing mode.

The address field for instruction can be represented in two different ways are as follows −

- Direct Addressing − It uses the address of the operand.
- Indirect Addressing − It facilitates the address as a pointer to the operand.

The address of the operand or the target address is called the effective address.

Effective Address (EA) − It defines the address that can be executed as a target address for a branch type instruction or the address that can be used directly to create an operand for a computation type instruction, without creating any changes.

Opcodes

An opcode is a collection of bits that represents the basic operations including add, subtract, multiply, complement, and shift. The total number of operations provided through the computer determines the number of bits needed for the opcode. The minimum bits accessible to the opcode should be n for 2n operations. These operations are implemented on information that is saved in processor registers or memory.

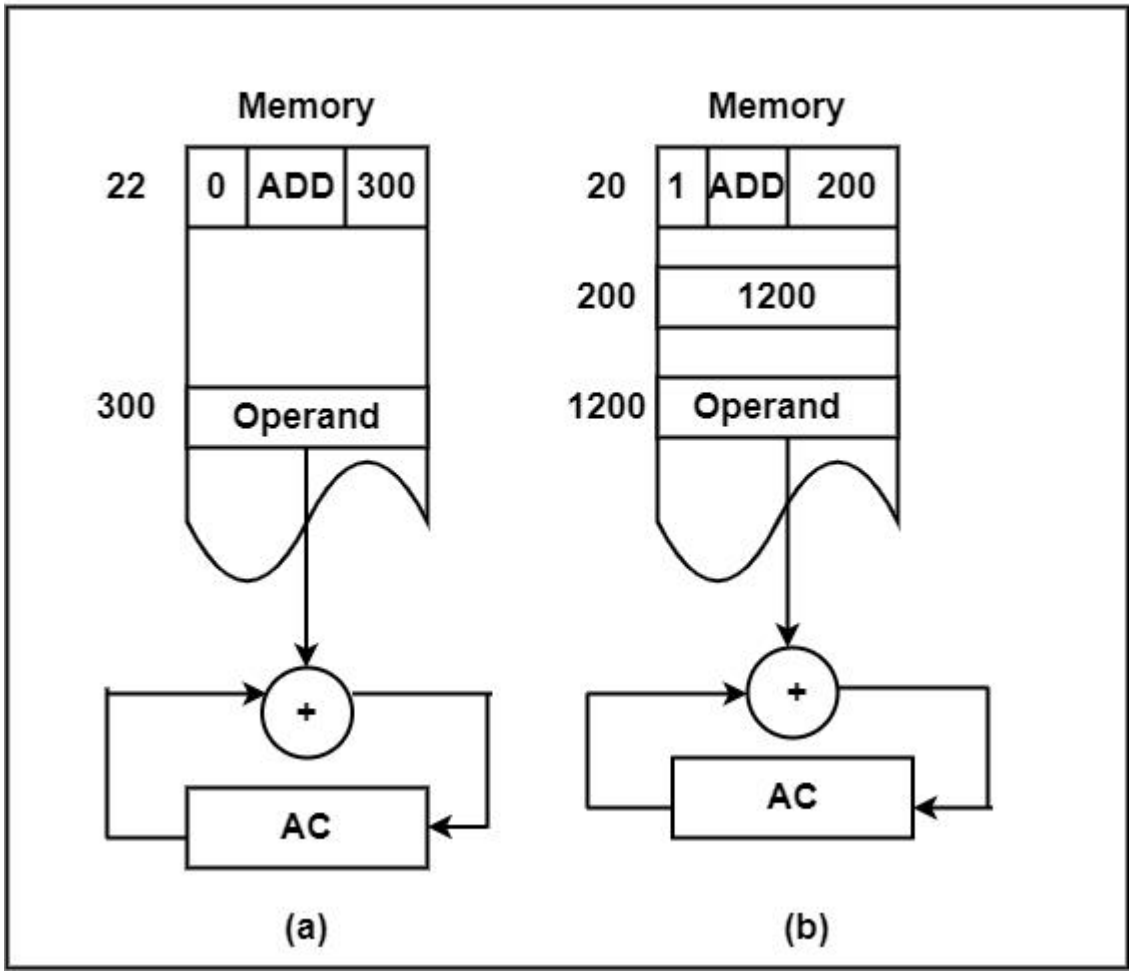
Address

The address is represented as the location where a specific instruction is constructed in the memory. The address bits of an instruction code is used as an operand and not as an address. In such methods, the instruction has an immediate operand. If the second part has an address, the instruction is referred to have a direct address.

There is another possibility in the second part including the address of the operand. This is referred to as an indirect address. In the instruction code, one bit can signify if the direct or indirect address is executed.

The figure shows a diagram showing direct and indirect addresses.

Direct and Indirect Address



Computer registers:

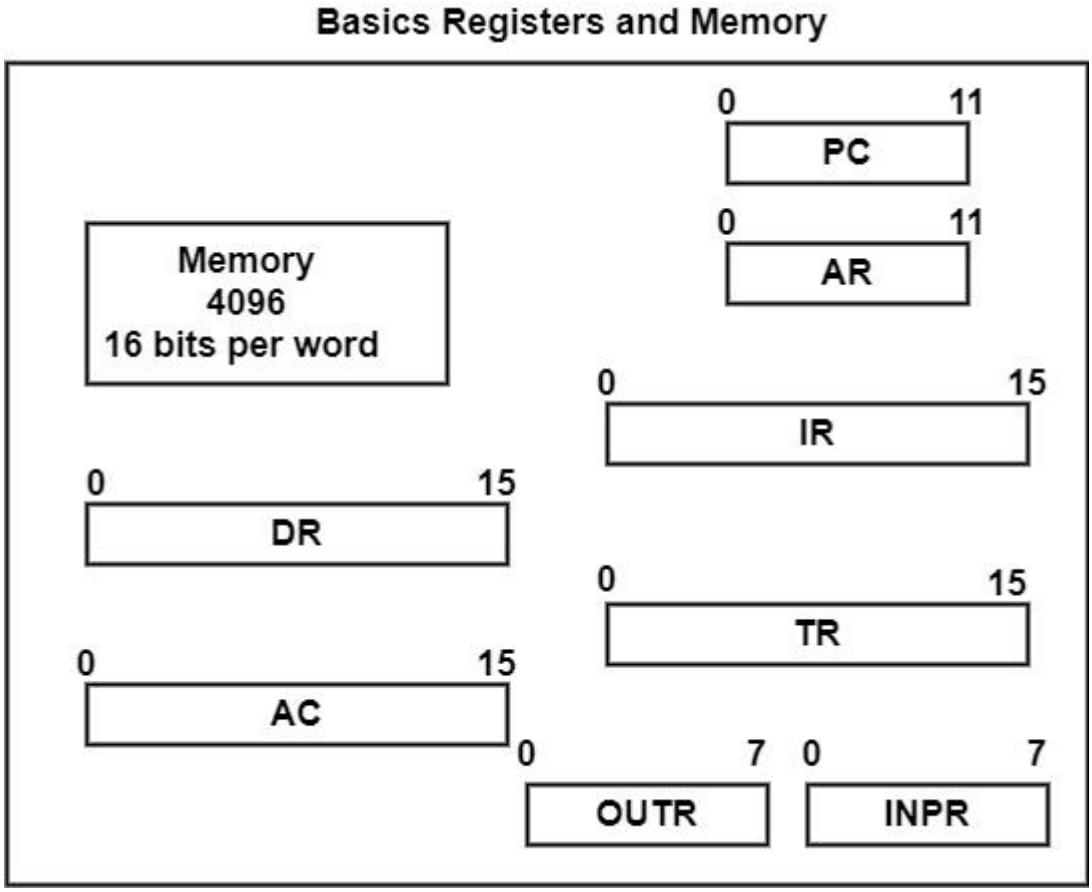
What are Computer Registers in Computer Architecture?

Computer registers are high-speed memory storing units. It is an element of the computer processor. It can carry any type of information including a bit sequence or single data.

A register should be 32 bits in length for a 32-bit instruction computer. Registers can be numbered relies upon the processor design and language rules.

The instructions in a computer are saved in memory locations and implemented one after another at a time. The function of the control unit is to fetch the instruction from the memory and implement it. The control does the similar for all the instructions in the memory in sequential order.

A counter is needed to maintain a path of the next instruction to be implemented and evaluate its address. The figure shows the registers with their memories. The memory addresses are saved in multiple registers. These requirements certainly state the use for registers in a computer.



The following table shows the registers and their functions.

Register Symbol	Number of Bits	Register Name	Function
OUTR	8	Output register	O It holds output character.
INPR	8	Input register	It holds input character.
PC	12	Program Counter	It holds the address of the instruction.
AR	12	Address Register	It holds an address for memory.
DR	16	Data Register	It holds memory operand.
AC	16	Accumulator	It's a processor register.
IR	16	Instruction Register	It holds an instruction code.
TR	16	Temporary Register	It holds temporary data.

The description for each of the registers determined in the figure is as follows −

- The data register holds the operand read from the memory.
- The accumulator is a general-purpose register need for processing.
- The instruction register holds the read memory.
- The temporary data used while processing is stored in the temporary register.
- The address register holds the address of the instruction that is to be implemented next from the memory.
- The Program Counter (PC) controls the sequence of instructions to be read. In case a branch instruction is detected, the sequential execution does not arise. A branch execution calls for a transfer to an instruction that is not in sequence with the instructions in the PC.

- The input register (INPR) and output register (OUTPR) are the registers used for the I/O operations. The INPR receives an 8-bit character from the input device. It is similar to the OUTPR.

Computer instructions:

In computer organization, a computer instruction is a set of commands that tells the computer hardware to perform a specific operation. Therefore, computer instructions are the primary building blocks of a computer program and hence a software, because computer instructions are entirely responsible for execution of the program or software. In this article, we will learn the basic computer instructions in computer organization.

The format in which the computer instructions are to be written is defined by the instruction set architecture (ISA) of the computer. The instruction set architecture defines the set of instructions that are supported by the processor of the computer. It also provides the information about the instruction's syntax, semantics, and encoding.

In a computer, computer instructions are able to perform a variety of operations like data transfer, arithmetic operations, logical operations, control flow, I/O operations, etc. Each computer instruction is specified by an opcode (binary code) that the hardware of the computer can understand.

When a computer program executes, the CPU of the computer fetches the computer instructions from the memory, and processes and decode it to identify the function to be performed.

Overall, computer instruction in computer organization are very important components, as they are responsible for allowing the computer to perform operations. Hence, in computers, instructions are entirely responsible for executing functions like data processing, data manipulation, control flow, hardware-software communication, etc.

Types of Basic Computer Instructions

Based on the functionality, the basic computer instructions can be classified into the following three major types:

- Memory Reference Instructions
- Register Reference Instructions
- Input/Output Instructions

Let us now discuss each type of these basic computer instructions in detail.

Memory Reference Instructions

In computer organization, the memory reference instructions are used for manipulation of data stored in the computer memory. Memory reference instructions enable the computer's CPU to access and process the data stored in a specific memory address.

Therefore, memory reference instructions provide a mean for writing data to memory or reading data from memory. These instructions allow the processor to communicate with the computer's memory system.

The format of memory reference instruction consists of three parts namely, opcode (operation code), operands, and addressing mode. Where, opcode represents the operation to be performed, while the operand represents the memory address where the data is stored.

A memory reference instruction typically uses 12 bits to specify a memory address, 3 bits to specify the opcode, and 1 bit to identify the addressing mode.

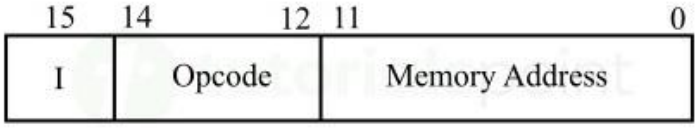


Figure - Memory Reference Instruction

The memory reference instructions are further classified into the following two types −

- **Store Instruction** − The store instruction is used to write data into the memory.
- **Load Instruction** − This instruction is used to retrieve data stored in the memory.

The execution process of memory reference instruction is explained here.

Firstly, the computer's CPU uses the program counter to fetch the instruction from the memory. Then, this instruction is decoded to determine the opcode and operands. If the opcode is for load instruction, the CPU accesses the memory location of the provided memory address to retrieve the stored data. If the opcode is for store instruction, the CPU writes the data into the provided memory address.

Once the execution of the current memory reference instruction is completed, the CPU updates the program counter to execute the next memory reference instruction.

Hence, memory reference instructions are very important for many computing tasks like data processing, file handling, database operations, etc., as these computer instructions allow for accessing and manipulation data stored in the computer memory.

Register Reference Instructions

Register reference instruction is a computer instruction that operates on the data stored in the registers instead of memory addresses. Hence, the register reference instructions are primarily used for manipulation of data stored in the registers of the processors of a computer.

Similar to memory reference instructions, register reference instructions can also perform operation like data manipulation, arithmetic operations, logical operations, etc. on the data stored in registers. In comparison to memory reference instructions, the register reference instructions are faster in execution.

Register reference instruction consists of two parts namely, opcode, and register operation. The opcode (operational code) represents the operation to be performed.

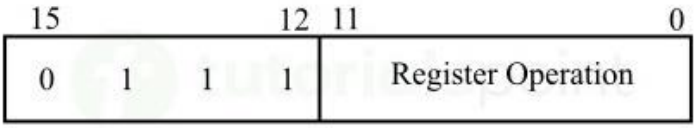


Figure - Register Reference Instruction

In computer organization, the register reference instructions are determined by their opcode 111 with a 0 at the left most of the instruction. In the case of register reference instructions, there is no need of an operand from memory because it uses additional 12 bits to identify the operation to be implemented.

Input-Output Instructions

Those computer instructions that provide a mean for communication between a computer system and its I/O peripherals are called input-output instructions.

Therefore, the input-output instructions enable the data transfer from and to input and output devices like keyboard, memory disks, monitor, etc. With the help of input-output instructions, the computer system receives data input from users and sends output to users. Hence, input-output instructions act as the communication interface between CPU and peripheral devices of the computer system.

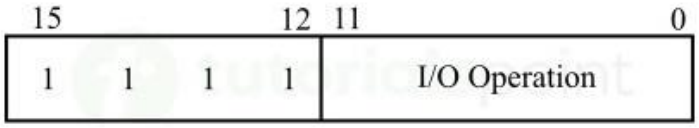


Figure - Input/ Output Instruction

Similar to register reference instructions, the input-output instructions also do not require a memory reference. It is identified by the opcode 111 with a 1 at the leftmost of the instruction. The rest 12-bits are used to represent the type of input/output operation to be performed.

Timing and control:

Control Unit

- CPU is partitioned into Arithmetic Logic Unit (ALU) and Control Unit (CU).
- The function of control unit is to generate relevant timing and control signals to all operations in the computer.
- It controls the flow of data between the processor and memory and peripherals

FUNCTIONS OF CONTROL UNIT

- The control unit directs the entire computer system to carry out stored program instructions.
- The control unit must communicate with both the arithmetic logic unit (ALU) and main memory.
- The control unit instructs the arithmetic logic unit that which logical or arithmetic operation is to be performed.
- The control unit co-ordinates the activities of the other two units as well as all peripherals and auxiliary storage devices linked to the computer.

DESIGN OF CONTROL UNIT

Control unit generates control signals using one of the two organizations:

- Hardwired Control Unit
- Micro-programmed Control Unit

HARDWIRED CONTROL UNIT

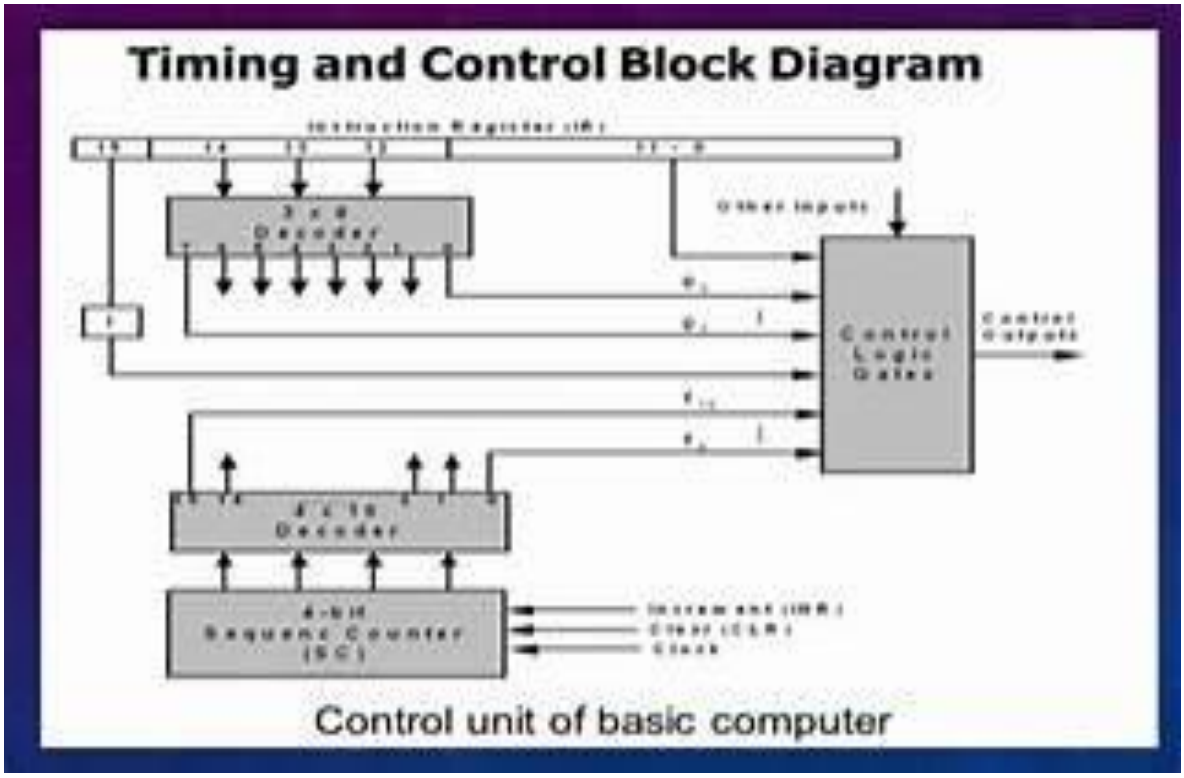
- It is implemented as logic circuits (gates, flip-flops, decoders etc.) in the hardware.
- This organization is very complicated if we have a large control unit.
- In this organization, if the design has to be modified or changed, requires changes in the wiring among the various components. Thus the modification of all the combinational circuits may be very difficult.

ADVANTAGES

- Hardwired Control Unit is fast because control signals are generated by combinational circuits.
- The delay in generation of control signals depends upon the number of gates.

DISADVANTAGES

- More is the control signals required by CPU; more complex will be the design of control unit.
- Modifications in control signal are very difficult. That means it requires rearranging of wires in the hardware circuit.
- It is difficult to correct mistake in original design or adding new feature in existing design of control unit.



Control unit consist of a:

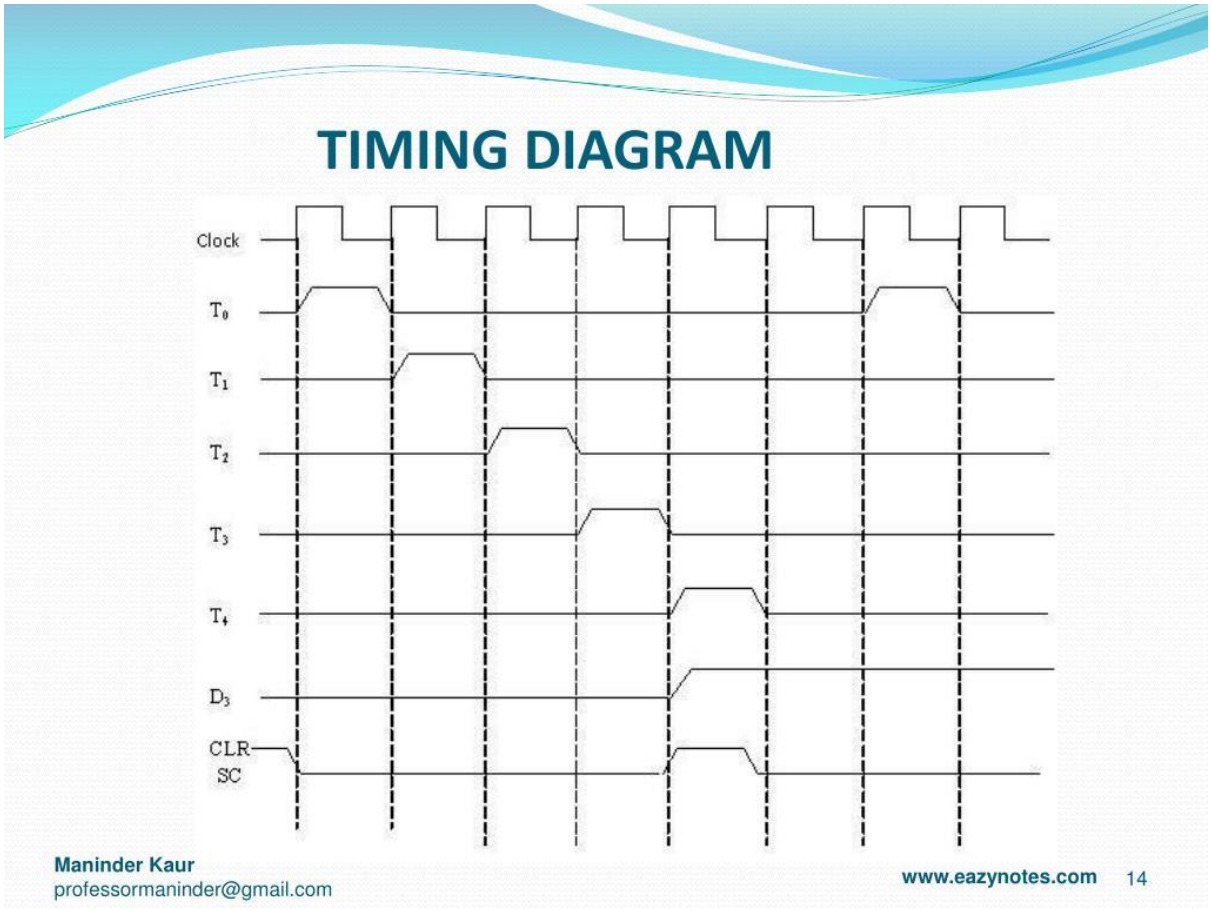
- Instruction Register
- Number of Control Logic Gates,
- Two Decoders
- 4-bit Sequence Counter
- An instruction read from memory is placed in the instruction register (IR).
- The instruction register is divided into three parts: the I bit, operation code, and address part.
- First 12-bits (0-11) to specify an address, next 3-bits specify the operation code (opcode) field of the instruction and last left most bit specify the addressing mode I. I = 0 for direct address I = 1 for indirect address
- First 12-bits (0-11) are applied to the control logic gates.
- The operation code bits (12 – 14) are decoded with a 3 x 8 decoder.
- The eight outputs (D0 through D7) from a decoder goes to the control logic gates to perform specific operation.
- Last bit 15 is transferred to a I flip-flop designated by symbol I.

The 4-bit sequence counter SC can count in binary from 0 through 15.

- The counter output is decoded into 16 timing pulses T0 through T15.
- The sequence counter can be incremented by INR input or clear by CLR input synchronously.

For example: Consider the case where SC is incremented to provide timing signalsT0, T1, T 2 , T3, and T4 in sequence. At time T4 , SC is cleared to 0 if decoder output D3 is active. This is expressed symbolically by the statement: D3 T4 : SC ← 0

❖ The timing diagram shows the time relationship of the control signals.



MICRO-PROGRAMMED CONTROL UNIT

❖ A micro-programmed control unit is implemented using programming approach. A sequence of microoperations are carried out by executing a program consisting of micro-instructions.
❖ Micro-program, consisting of micro-instructions is stored in the control memory of the control unit. Execution of a micro-instruction is responsible for generation of a set of control signals.

A micro-instruction consists of:

❖ One or more micro-operations to be executed.
❖ Address of next microinstruction to be executed. Micro-Operations: The operations performed on the data stored inside the registers are called micro-operations.
❖ Micro-Programs: Microprogramming is the concept for generating control signals using programs. These programs are called micro-programs.

Micro-Instructions:

The instructions that make micro-program are called micro-instructions.

❖ Micro-Code: Micro-program is a group of micro- instructions. The micro-program can also be termed as micro-code.

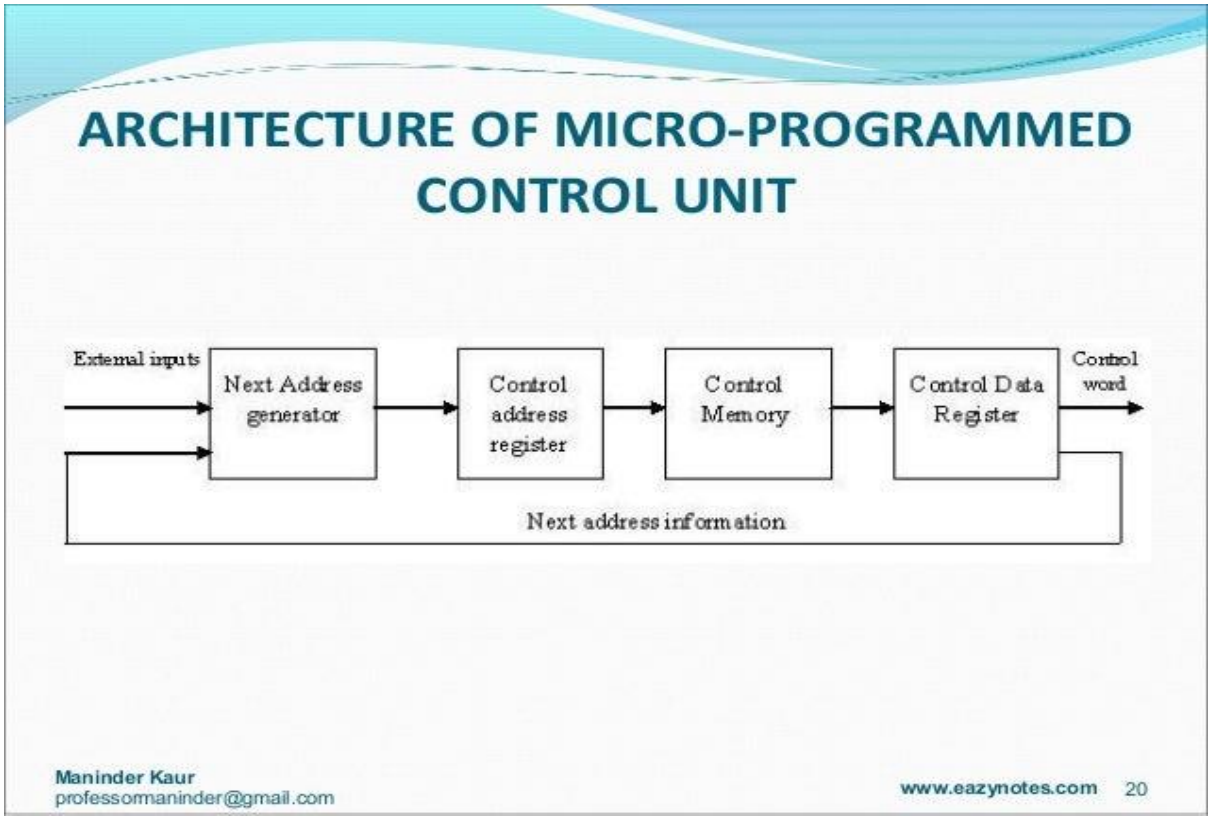
- Control Memory: Micro-programs are stored in the read only memory (ROM). That memory is called control memory.

ADVANTAGES

- The design of micro-program control unit is less complex because micro-programs are implemented using software routines.
- The micro-programmed control unit is more flexible because design modifications, correction and enhancement is easily possible.
- The new or modified instruction set of CPU can be easily implemented by simply rewriting or modifying the contents of control memory.
- The fault can be easily diagnosed in the micro-program control unit using diagnostics tools by maintaining the contents of flags, registers and counters.

DISADVANTAGES

- The micro-program control unit is slower than hardwired control unit. That means to execute an instruction in micro-program control unit requires more time.
- The micro-program control unit is expensive than hardwired control unit in case of limited hardware resources.
- The design duration of micro-program control unit is more than hardwired control unit for smaller CPU.



ARCHITECTURE OF MICRO-PROGRAMMED

- The address of micro-instruction that is to be executed is stored in the control address register (CAR).
- Micro-instruction corresponding to the address stored in CAR is fetched from control memory and is stored in the control data register (CDR).
- This micro-instruction contains control word to execute one or more micro-operations. After the execution of all micro-operations of micro-instruction, the address of next micro-instruction is located.

COMPARISON BETWEEN HARDWIRED AND MICRO-PROGRAMMED CONTROL UNIT

Attributes	Hardwired Control	Micro-programmed Control
Speed	Fast	Slow
Cost of Implementation	More	Cheaper
Flexibility	Not flexible, difficult to modify for new instruction	Flexible, new instructions can easily be added
Ability to Handle Complex Instructions	Difficult	Easier
Decoding	Complex	Easy
Applications	RISC Microprocessor	CISC Microprocessor
Instruction Set Size	Small	Large
Control Memory	Absent	Present
Chip Area Required	Less	More

Maninder Kaur
professormaninder@gmail.com

Instruction cycle

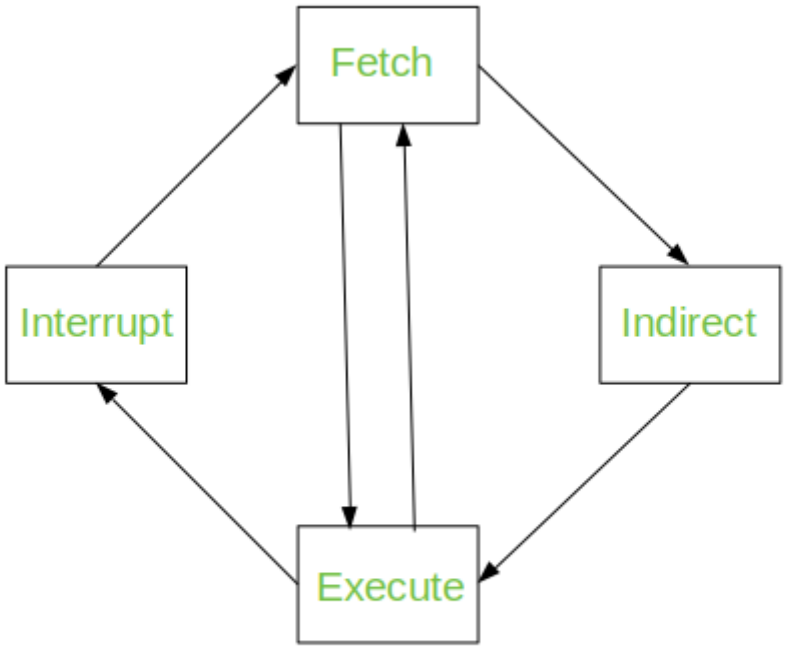
In computer organization, an instruction cycle, also known as a fetch-decode-execute cycle, is the basic operation performed by a central processing unit (CPU) to execute an instruction. The instruction cycle consists of several steps, each of which performs a specific function in the execution of the instruction. The major steps in the instruction cycle are:

1. Fetch: In the fetch cycle, the CPU retrieves the instruction from memory. The instruction is typically stored at the address specified by the program counter (PC). The PC is then incremented to point to the next instruction in memory.
2. Decode: In the decode cycle, the CPU interprets the instruction and determines what operation needs to be performed. This involves identifying the opcode and any operands that are needed to execute the instruction.
3. Execute: In the execute cycle, the CPU performs the operation specified by the instruction. This may involve reading or writing data from or to memory, performing arithmetic or logic operations on data, or manipulating the control flow of the program.
4. There are also some additional steps that may be performed during the instruction cycle, depending on the CPU architecture and instruction set:
5. Fetch operands: In some CPUs, the operands needed for an instruction are fetched during a separate cycle before the execute cycle. This is called the fetch operands cycle.
6. Store results: In some CPUs, the results of an instruction are stored during a separate cycle after the execute cycle. This is called the store results cycle.
7. Interrupt handling: In some CPUs, interrupt handling may occur during any cycle of the instruction cycle. An interrupt is a signal that the CPU receives from an external device or software that requires immediate attention. When an interrupt occurs, the CPU suspends the current instruction and executes an interrupt handler to service the interrupt.

These cycles are the basic building blocks of the CPU's operation and are performed for every instruction executed by the CPU. By optimizing these cycles, CPU designers can improve the performance and efficiency of the CPU, allowing it to execute instructions faster and more efficiently.

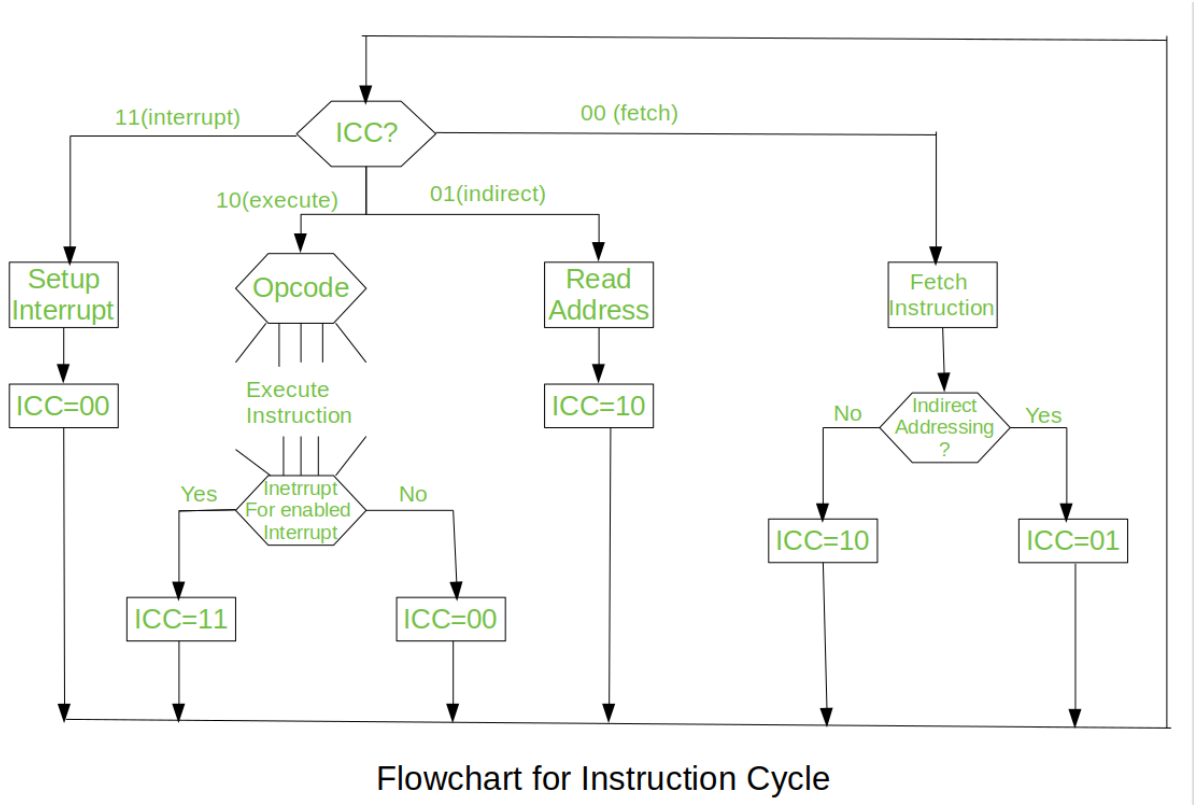
The Instruction Cycle –

Each phase of Instruction Cycle can be decomposed into a sequence of elementary micro-operations.



The Instruction Cycle

The Indirect Cycle is always followed by the Execute Cycle. The Interrupt Cycle is always followed by the Fetch Cycle. For both fetch and execute cycles, the next cycle depends on the state of the system.



Flowchart for Instruction Cycle

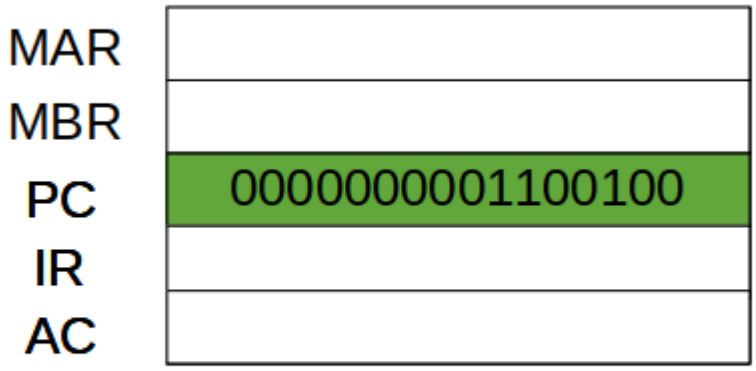
We assumed a new 2-bit register called Instruction Cycle Code (ICC). The ICC designates the state of processor in terms of which portion of the cycle it is in:

00 : Fetch Cycle
01 : Indirect Cycle
10 : Execute Cycle
11 : Interrupt Cycle

At the end of the each cycles, the ICC is set appropriately. The above flowchart of Instruction Cycle describes the complete sequence of micro-operations, depending only on the instruction sequence and the interrupt pattern (this is a simplified example). The operation of the processor is described as the performance of a sequence of micro-operation.

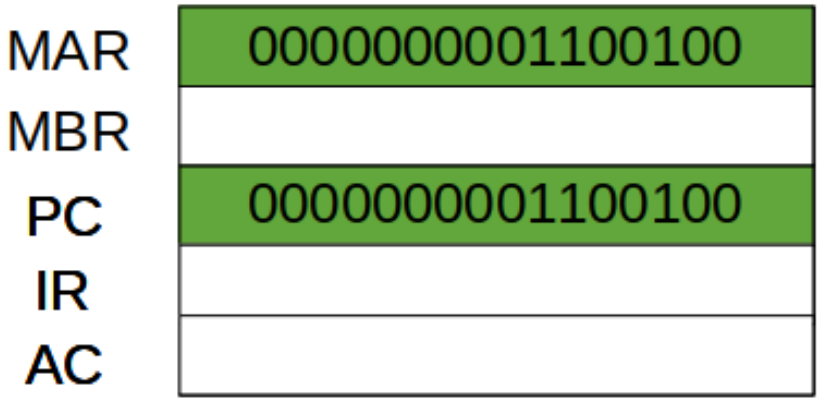
Different Instruction Cycles:

- **The Fetch Cycle –**
  At the beginning of the fetch cycle, the address of the next instruction to be executed is in the Program Counter(PC).



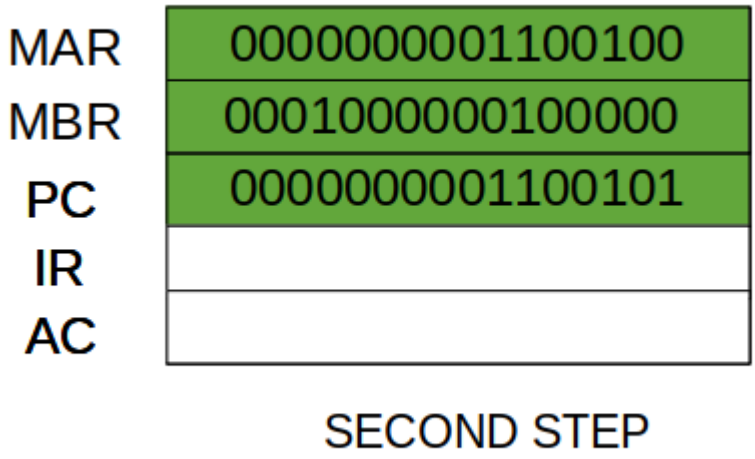
BEGINNING

- Step 1: The address in the program counter is moved to the memory address register(MAR), as this is the only register which is connected to address lines of the system bus.



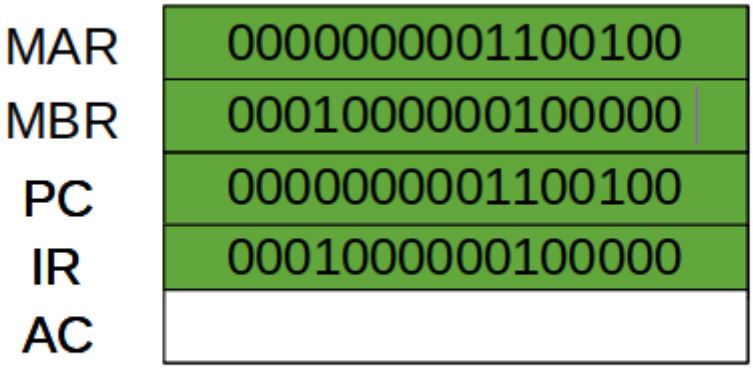
FIRST STEP

- Step 2: The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register (MBR). Program counter is incremented by one, to get ready for the next instruction. (These two action can be performed simultaneously to save time)



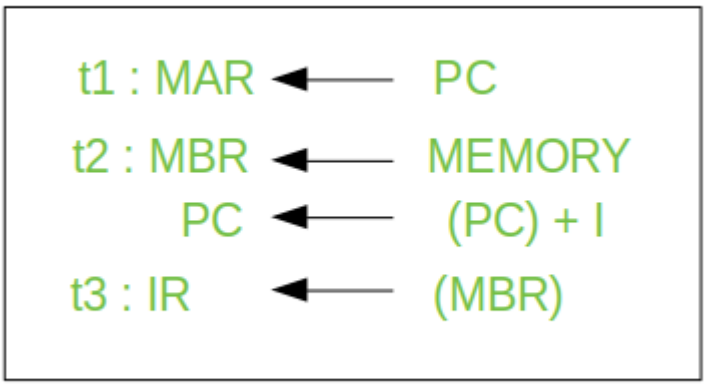
SECOND STEP

- Step 3: The content of the MBR is moved to the instruction register(IR).



FIRST STEP

- Thus, a simple Fetch Cycle consist of three steps and four micro-operation. Symbolically, we can write these sequence of events as follows:



- Here 'I' is the instruction length. The notations (t1, t2, t3) represents successive time units. We assume that a clock is available for timing purposes and it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit.
  First time unit: Move the contents of the PC to MAR.
  Second time unit: Move contents of memory location specified by MAR to MBR. Increment content of PC by I.
  Third time unit: Move contents of MBR to IR.
  Note: Second and third micro-operations both take place during the second time unit.

- **The Indirect Cycles –**

Once an instruction is fetched, the next step is to fetch source operands. Source Operand is being fetched by indirect addressing (it can be fetched by any addressing mode, here its done by indirect addressing). Register-based operands need not be fetched. Once the opcode is executed, a similar process may be needed to store the result in main memory. Following micro-operations takes place:

- Step 1: The address field of the instruction is transferred to the MAR. This is used to fetch the address of the operand.
  Step 2: The address field of the IR is updated from the MBR.(So that it now contains a direct addressing rather than indirect addressing)
  Step 3: The IR is now in the state, as if indirect addressing has not been occurred.

Note: Now IR is ready for the execute cycle, but it skips that cycle for a moment to consider the Interrupt Cycle .

- **The Execute Cycle**

The other three cycles (Fetch, Indirect and Interrupt) are simple and predictable. Each of them requires simple, small and fixed sequence of micro-operation. In each case same micro-operation are repeated each time around.
Execute Cycle is different from them. Like, for a machine with N different opcodes there are N different sequence of micro-operations that can occur.
Lets take an hypothetical example :
consider an add instruction:

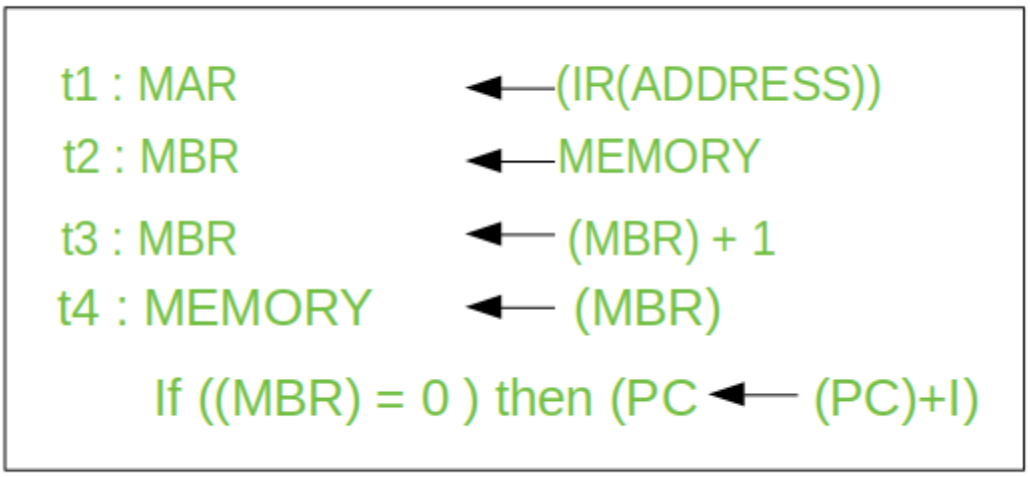
- Here, this instruction adds the content of location X to register R. Corresponding micro-operation will be:-

- We begin with the IR containing the ADD instruction.
  Step 1: The address portion of IR is loaded into the MAR.
  Step 2: The address field of the IR is updated from the MBR, so the reference

memory location is read.
Step 3: Now, the contents of R and MBR are added by the ALU.

Lets take a complex example :-

- Here, the content of location X is incremented by 1. If the result is 0, the next instruction will be skipped. Corresponding sequence of micro-operation will be :-



- Here, the PC is incremented if (MBR) = 0. This test (is MBR equal to zero or not) and action (PC is incremented by 1) can be implemented as one micro-operation.
Note : This test and action micro-operation can be performed during the same time unit during which the updated value MBR is stored back to memory.

- **The Interrupt Cycle**:
At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not. If an enabled interrupt has occurred then Interrupt Cycle occurs. The nature of this cycle varies greatly from one machine to another.
Lets take a sequence of micro-operation:-

- Step 1: Contents of the PC is transferred to the MBR, so that they can be saved for return.
  Step 2: MAR is loaded with the address at which the contents of the PC are to be saved.
  PC is loaded with the address of the start of the interrupt-processing routine.
  Step 3: MBR, containing the old value of PC, is stored in memory.

Note: In step 2, two actions are implemented as one micro-operation. However, most processor provide multiple types of interrupts, it may take one or more micro-operation to obtain the save address and the routine address before they are transferred to the MAR and PC respectively.

Uses of Different Instruction Cycles :

Here are some uses of different instruction cycles:

1. **Fetch cycle:** This cycle retrieves the instruction from memory and loads it into the processor's instruction register. The fetch cycle is essential for the processor to know what instruction it needs to execute.
2. **Decode cycle:** This cycle decodes the instruction to determine what operation it represents and what operands it requires. The decode cycle is important for the processor to understand what it needs to do with the instruction and what data it needs to retrieve or manipulate.
3. **Execute cycle:** This cycle performs the actual operation specified by the instruction, using the operands specified in the instruction or in other registers. The execute cycle is where the processor performs the actual computation or manipulation of data.
4. **Store cycle:** This cycle stores the result of the operation in memory or in a register. The store cycle is essential for the processor to save the result of the computation or manipulation for future use.

The advantages and disadvantages of the instruction cycle depend on various factors, such as the specific CPU architecture and the instruction set used. However, here are some general advantages and disadvantages of the instruction cycle:

Advantages:

1. Standardization: The instruction cycle provides a standard way for CPUs to execute instructions, which allows software developers to write programs that can run on multiple CPU architectures. This standardization also makes it easier for hardware designers to build CPUs that can execute a wide range of instructions.
2. Efficiency: By breaking down the instruction execution into multiple steps, the CPU can execute instructions more efficiently. For example, while the CPU is performing the execute cycle for one instruction, it can simultaneously fetch the next instruction.
3. Pipelining: The instruction cycle can be pipelined, which means that multiple instructions can be in different stages of execution at the same time. This improves the overall performance of the CPU, as it can process multiple instructions simultaneously.

Disadvantages:

1. Overhead: The instruction cycle adds overhead to the execution of instructions, as each instruction must go through multiple stages before it can be executed. This overhead can reduce the overall performance of the CPU.
2. Complexity: The instruction cycle can be complex to implement, especially if the CPU architecture and instruction set are complex. This complexity can make it difficult to design, implement, and debug the CPU.
3. Limited parallelism: While pipelining can improve the performance of the CPU, it also has limitations. For example, some instructions may depend on the results of previous instructions, which limits the amount of parallelism that can be achieved. This can reduce the effectiveness of pipelining and limit the overall performance of the CPU.

Issues of Different Instruction Cycles :

Here are some common issues associated with different instruction cycles:

1. **Pipeline hazards:** Pipelining is a technique used to overlap the execution of multiple instructions by breaking them into smaller stages. However, pipeline hazards occur when one instruction depends on the completion of a previous instruction, leading to delays and reduced performance.
2. **Branch prediction errors:** Branch prediction is a technique used to anticipate which direction a program will take when encountering a conditional branch instruction. However, if the prediction is incorrect, it can result in wasted cycles and decreased performance.
3. **Instruction cache misses:** Instruction cache is a fast memory used to store frequently used instructions. Instruction cache misses occur when an instruction is not found in the cache and needs to be retrieved from slower memory, resulting in delays and decreased performance.

4. **Instruction-level parallelism limitations:** Instruction-level parallelism is the ability of a processor to execute multiple instructions simultaneously. However, this technique has limitations as not all instructions can be executed in parallel, leading to reduced performance in some cases.
5. **Resource contention:** Resource contention occurs when multiple instructions require the use of the same resource, such as a register or a memory location. This can lead to delays and reduced performance if the processor is unable to resolve the contention efficiently.

Input-Output and Interrupt:

- A Terminal with a keyboard and a Printer.

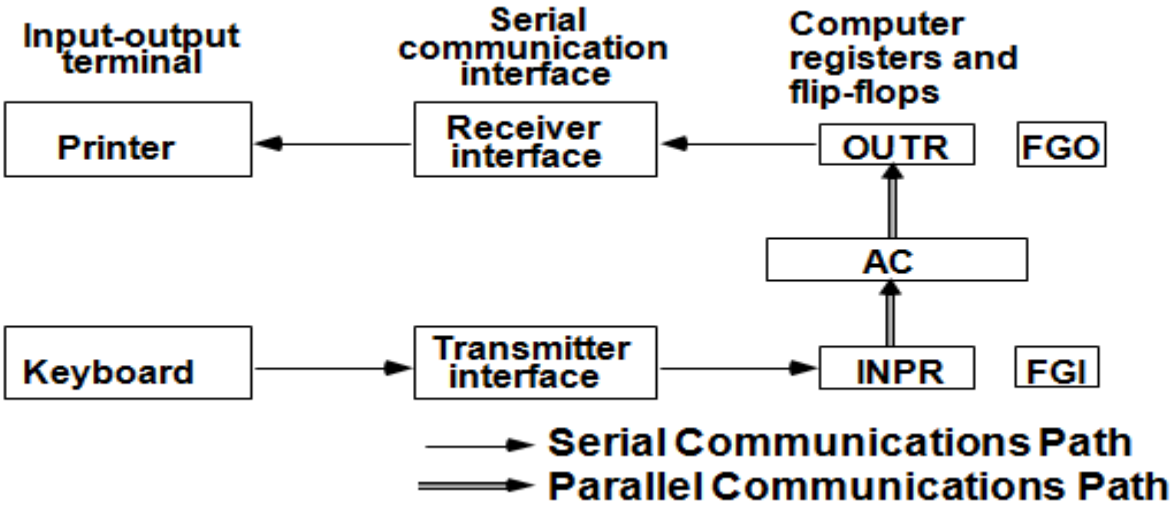


Fig: Input-Output Configuration

- The terminal sends and receives serial information.
- The serial info. from the keyboard is shifted into INPR .
- The serial info. for the printer is stored in the OUTR.
- INPR and OUTR communicate with the terminal serially and with the AC in parallel.
- The flags are needed to synchronize the timing difference between I/O device and the computer.

$D_7IT_3 = p$
$IR(i) = B_i$, $i = 6, \ldots, 11$

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$	Input char. to AC
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$	Output char. from AC
SKI	pB_9:	$if(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	$if(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

Fig: Input-Output Instruction

- The way that the interrupt is handled by the computer can be explained by means of the flowchart of Fig. An interrupt flip-flop R is included in the computer. When R = 0, the computer goes through an instruction cycle. During the execute phase of the instruction cycle IEN is checked by the control. If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle.
- If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information. In this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN = 1, flip-flop R is set to 1. At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

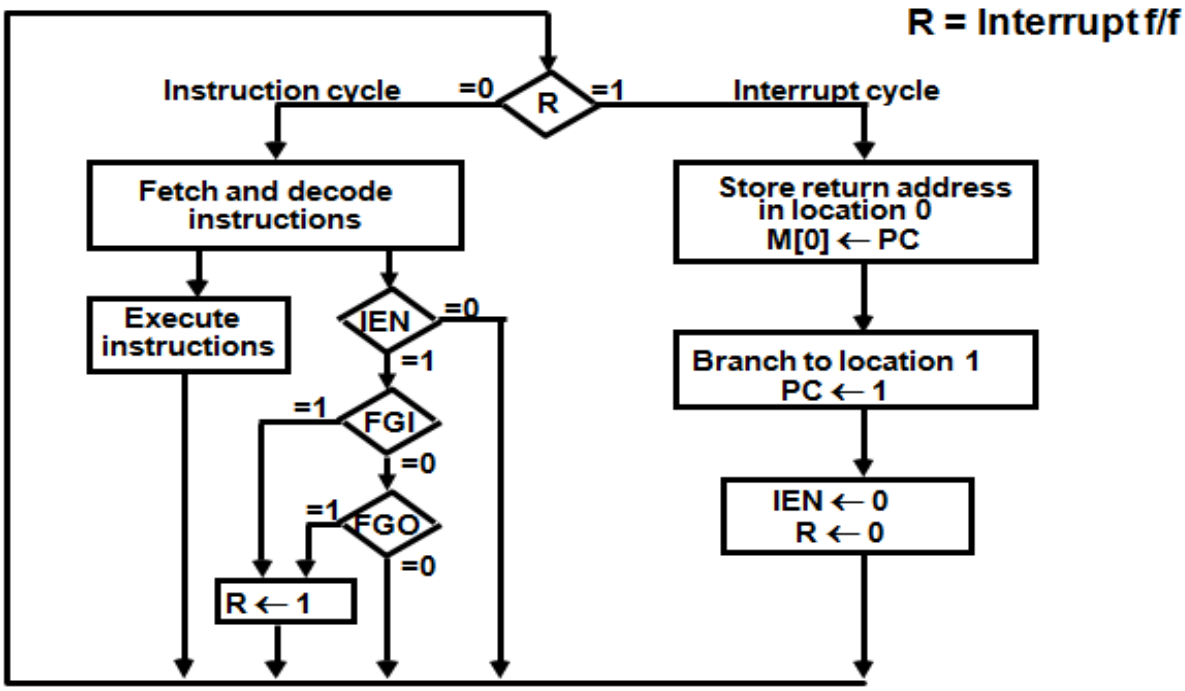
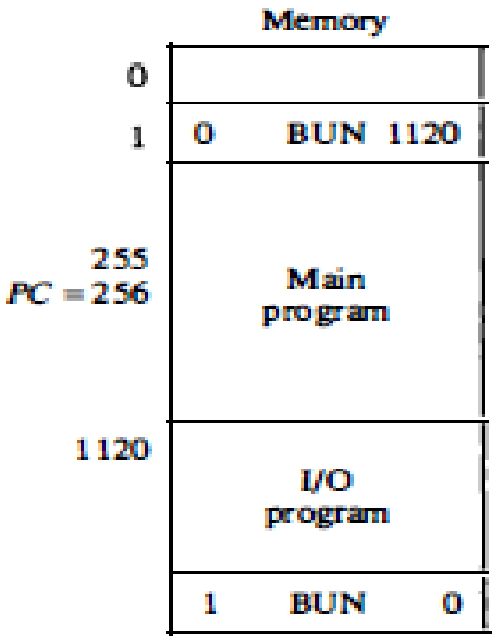
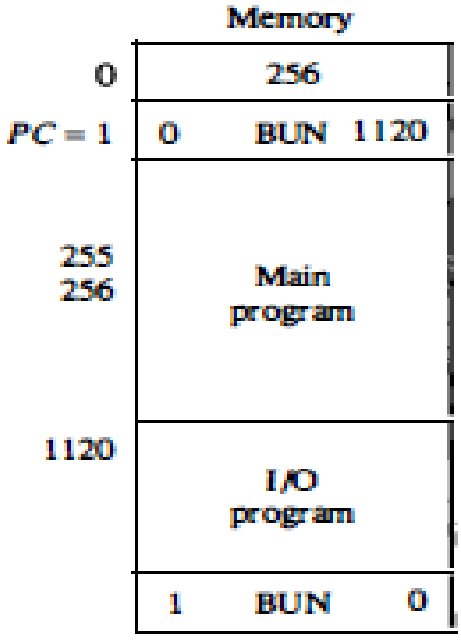


Fig: Flow chart for Interrupt Cycle

- An example that shows what happens during the interrupt cycle is shown in Fig. Suppose that an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255. At this time, the return address 256 is in PC. The programmer has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1. This is shown in Fig(a).
- When control reaches timing signal T_0 and finds that R = 1, it proceeds with the interrupt cycle. The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0. At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC. The branch instruction at address 1 causes the program to transfer to the input-output service program at address 1120.
- This program checks the flags, determines which flag is set, and then transfers the required input or output information. One this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted. This is shown in Fig. (b).



(a) Before interrupt



(b) After interrupt cycle

Addressing Modes:

addressing modes in computer architecture plays a vital role in enabling efficient and flexible memory access and operand manipulation. Addressing modes define the rules and mechanisms by which the processor calculates the effective memory address or operand location for data operations. By understanding the different addressing modes available, programmers and system designers can optimize memory utilization and enhance overall performance.

In this article, we embark on a comprehensive exploration of addressing modes in computer architecture. We delve into the intricacies of various addressing modes, unraveling their functionalities, benefits, and use cases.

There are two types of addressing modes for 8086 instructions:
1) Addressing modes for data
2) Addressing modes for branch

You may quickly access variables, arrays, records, pointers, and other complicated data types thanks to the 8086 memory addressing modes, which offer flexible access to memory. The correct application of memory addressing modes is the secret to effective assembly language programming.

An assembly language program instruction consists of two parts

The memory address of an operand consists of two components:

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.
  The 8086 microprocessor uses a variety of addressing modes depending on how an operand is specified.

Addressing modes used by 8086 microprocessor are discussed below:
Implied mode: The operand is mentioned in the instruction itself in inferred addressing. Data is a component of instructions in this mode and might be 8 bits or 16 bits long. Insinuated addressing is used in the construction of zero address instructions.

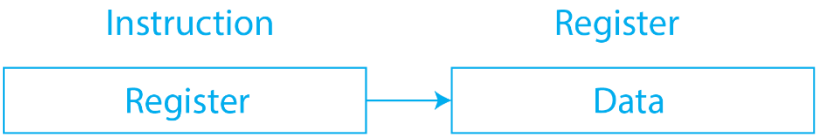
Example: CLC (used to reset Carry flag to 0)

- Immediate addressing mode (symbol #): Data is present in the instruction's address field in this manner. Like a single format for address instructions.

Note: Limitation in the immediate mode is that the range of constants are restricted by the size of the address field.

Example: MOV AL, 35H (move the data 35H into AL register)

- Register mode: The operand is put in either an 8 bit or 16 bit general purpose register when using register addressing. The instruction's designated register for the data is present.
- Here one register reference is required to access the data.



Example: MOV AX,CX (move the contents of CX register to AX register)

- Register Indirect mode: According to the instruction, the operand's offset is put in any one of the registers BX, BP, SI, or DI in this addressing. The base register or an index register that the instruction specifies is where the data's actual address is located.

Here two register references are required to access the data.

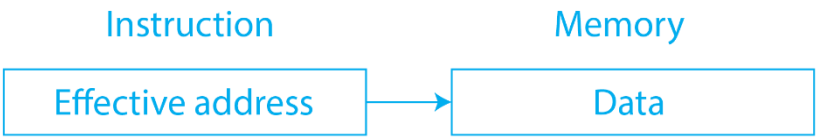
Using the register indirect addressing modes, the 8086 CPUs enable indirect memory access via a register.
MOV AX, [BX](move the contents of memory location s
addressed by the register BX to the register AX)

Auto Indexed (increment mode): Effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.(R1)+.
Here one register reference, one memory reference and one ALU operation is required to access the data.
Example:
Add R1, (R2)+ // OR
R1 = R1 +M[R2]
R2 = R2 + d
Useful for stepping through arrays in a loop. R2 – start of array d – size of an element.

Auto indexed (decrement mode):The information in a register designated by the instruction serves as the operand's effective address. The values of this register are automatically decremented to point to the preceding consecutive memory location before accessing the operand. –(R1)
Here one register reference, one memory reference and one ALU operation is required to access the data.
Example:
Add R1,-(R2) //OR
R2 = R2-d
R1 = R1 + M[R2]
The auto increment and auto decrement modes are identical. Both can be used to the push and pop implementation of a stack. For the implementation of "Last-In-First-Out" data structures, auto increment and auto decrement modes are helpful.

- Direct addressing/ Absolute addressing Mode (symbol [ ]): The operand's offset is given in the instruction as an 8 bit or 16 bit displacement element. In this addressing mode the 16 bit effective address of the data is the part of the instruction.

Here only one memory reference operation is required to access the data.



Example: ADD AL,[0301] //add the contents of offset address 0301 to AL

- Indirect addressing Mode (symbol @ or ()): In this mode address field of instruction contains the address of effective address. Here two references are required.
  1st reference to get effective address.
  2nd reference to access the data.
  Based on the availability of Effective address, Indirect mode is of two kind:

1. Register Indirect: In this method, the register holds the effective address, while the address field of an instruction maintains the name of the relevant register. In this case, one memory reference and one register reference are needed to access the data.
2. Memory Indirect: In this mode, the instruction's address field will always preserve the associated memory address even while the effective address is in the memory. Here, accessing the data requires two memory references.

- Indexed addressing mode: The content of an index register SI or DI and an 8 bit or 16 bit displacement are added to determine the operand's offset.
  Example: MOV AX, [SI +05]
- Based Indexed Addressing: The content of a base register (BX or BP) and an index register (SI or DI) are added to determine the operand's offset.
  Example: ADD AX, [BX+SI]

Based on Transfer of control, addressing modes are:

- PC relative addressing mode: PC relative addressing mode is used to implement intra segment transfer of control, In this mode effective address is obtained by adding displacement to PC.
- EA= PC + Address field value
- PC= PC + Relative value.
- Base register addressing mode: Base register addressing mode is used to implement inter segment transfer of control. In this mode effective address is obtained by adding base register value to address field value.
- EA= Base register + Address field value.
- PC= Base register + Relative value.

Note:

1. Program relocation at runtime is possible using both PC relative and based register addressing modes.
2. Writing position independent codes is best done using the based register addressing technique.

Advantages of Addressing Modes

1. The provision of programming tools like Pointers, counters for loop controllers, data indexing, and program relocation.
2. To decrease the number of bits in the instruction's addressing field.

Data Transfer and Manipulation

Most computer instructions can be classified into three categories:

1) Data transfer, 2) Data manipulation, 3) Program control instructions
2) Data transfer instruction cause transfer of data from one location to another
3) Data manipulation performs arithmatic, logic and shift operations.
4) Program control instructions provide decision making capabilities and change the path taken by the program when executed in computer. Data Transfer Instruction

   Typical Data Transfer Instruction:

   LD » Load : transfer from memory to a processor register, usually an AC (memory read)
   ST » Store : transfer from a processor register into memory (memory write)
   MOV » Move : transfer from one register to another register
   XCH » Exchange : swap information between two registers or register and a memory word
   IN/OUT » Input/Output : transfer data among processor registers and input/output device
   PUSH/POP » Push/Pop : transfer data between processor registers and a memory stack

Program Control Instructions

instructions of the computer are always stored in consecutive memory locations. These instructions are fetched from successive memory locations for processing and executing.

When an instruction is fetched from the memory, the program counter is incremented by 1 so that it points to the address of the next consecutive instruction in the memory. Once a data transfer and data manipulation instruction are executed, the program control along with the program counter, which holds the address of the next instruction to be fetched, is returned to the fetch cycle.

Data transfer and manipulation instructions specify the conditions for data processing operations, whereas the program control instructions specify the conditions that can alter the content of the program counter.

The change in the content of the program counter can cause an interrupt/break in the instruction execution. However, the program control instructions control the flow of program execution and are capable of branching to different program segments.

Some of the program control instructions are listed in the table.

Program Control Instructions

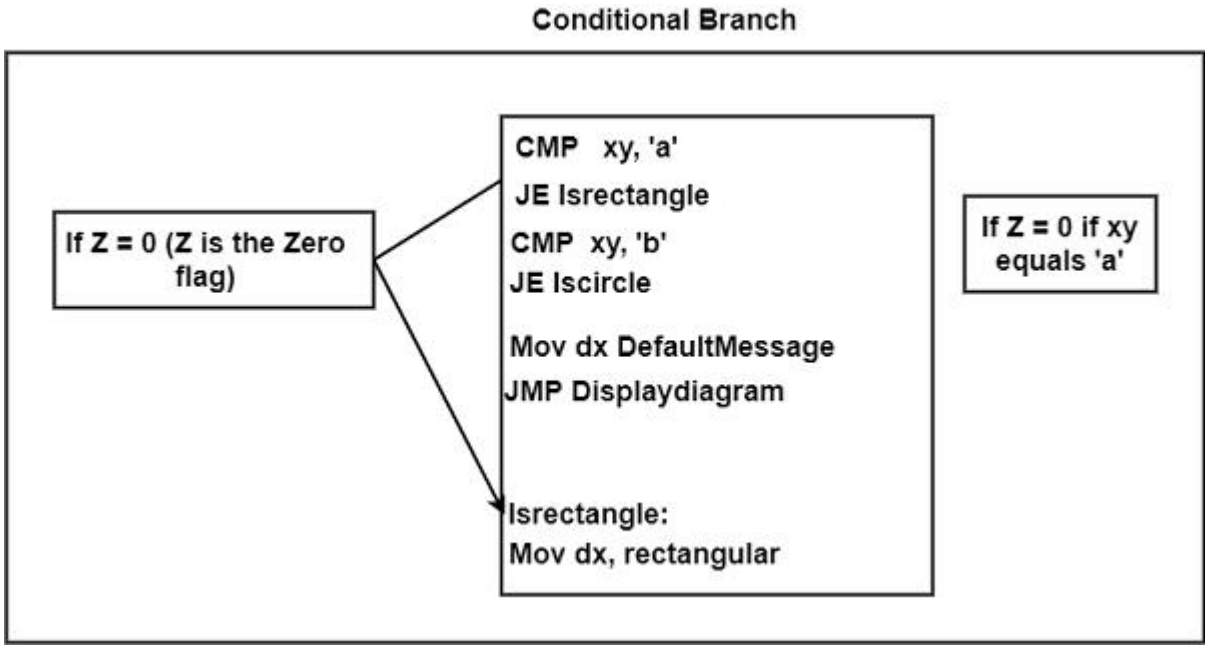
Name	Mnemonics
Branch	BR
Jump	JMP
Skip	SKP
Call	Call
Return	RET

Name	Mnemonics
Compare (by Subtraction)	CMP
Test (by ANDing)	TST

The branch is a one-address instruction. It is represented as BR ADR, where ADR is a mnemonic for an address. The branch instruction transfers the value of ADR into the program counter. The branch and jump instructions are interchangeably used to mean the same. However, sometimes they denote different addressing modes.

The conditional branch instructions such as ‘branch if positive’, or ‘branch if zero’ specifies the condition to transfer the flow of execution. When the condition is met, the branch address is loaded in the program counter.

The figure depicts the conditional branch instructions.



The compare instruction performs an arithmetic subtraction. Here, the result of the operation is not saved; instead, the status bit conditions are set. The test instruction performs the logical AND operation on two operands and updates the status bits.

RISC

RISC is the way to make hardware simpler whereas CISC is the single instruction that handles multiple work. In this article, we are going to discuss RISC and CISC in detail as well as the Difference between RISC and CISC, Let's proceed with RISC first.

Reduced Instruction Set Architecture (RISC)

The main idea behind this is to simplify hardware 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.

Characteristics of RISC

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

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.