Unit II

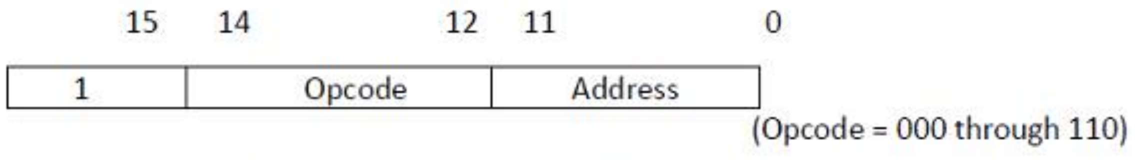
What are Computer Instructions?

A computer has programs stored in its RAM in the form of 1s and 0s that are interpreted by the CPU as instructions. One word of RAM includes one instruction in the machine language.

These instructions are loaded to the CPU one at a time, where it receives decoded and implemented. A basic computer has three instruction code formats such as the memory reference instruction, the register reference instruction, and the input-output instruction format.

Memory Reference Instruction

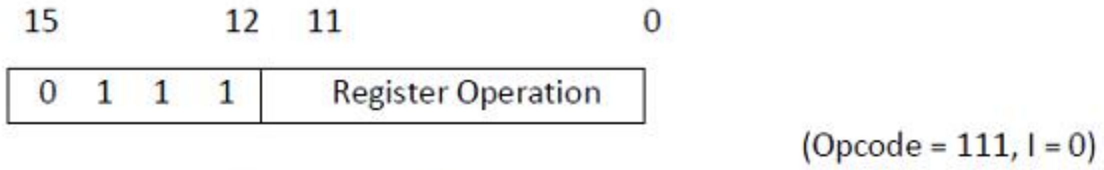
A memory-reference instruction uses 12 bits to specify an address and one bit to determine the addressing mode I. I is the same as 0 for direct address and to 1 for indirect address.



(a) Memory Reference Instruction

Register Reference Instruction

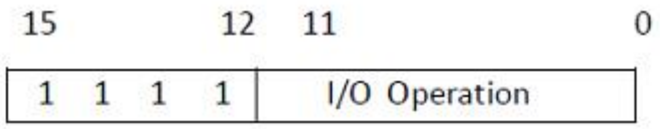
The register reference instructions are identified by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction. It determines an operation on or a test of the AC register. An operand from memory is not required because the additional 12 bits are used to determine the operation or test to be implemented.



(b) Register Reference Instruction

Input-Output Instruction

An input-output instruction does not require a reference to memory and is identified by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits can determine the type of input-output operation or test implemented.



(Opcode = 111, I = 1)

(c) Input-Output Instruction

The type of instruction is identified by the computer control from the four bits in positions 12 through 15 of the instruction. If the three opcode bits in positions 12 through 14 are not similar to 111, the instruction is a memory-reference type and the bit in position 15 is taken as the addressing mode I.

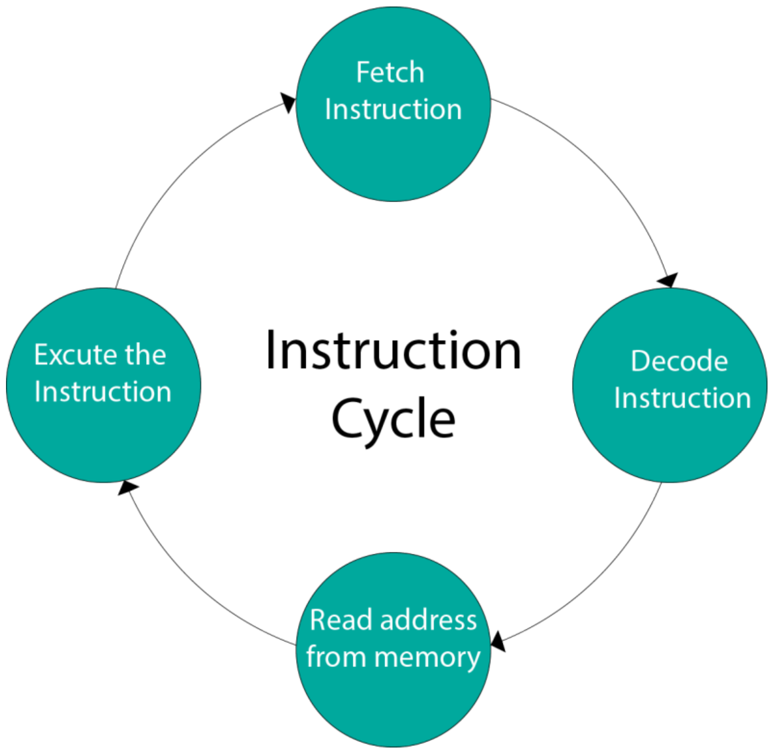
If the 3-bit opcode is similar to 111, the control then examines the bit in position 15. If this bit is 0, the instruction is a register-reference type. If the bit is 1, the instruction is an input-output type.

Instruction Cycle

A program residing in the memory unit of a computer consists of a sequence of instructions. These instructions are executed by the processor by going through a cycle for each instruction.

In a basic computer, each instruction cycle consists of the following phases:

1. Fetch instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory.
4. Execute the instruction.



What are Instruction Formats?

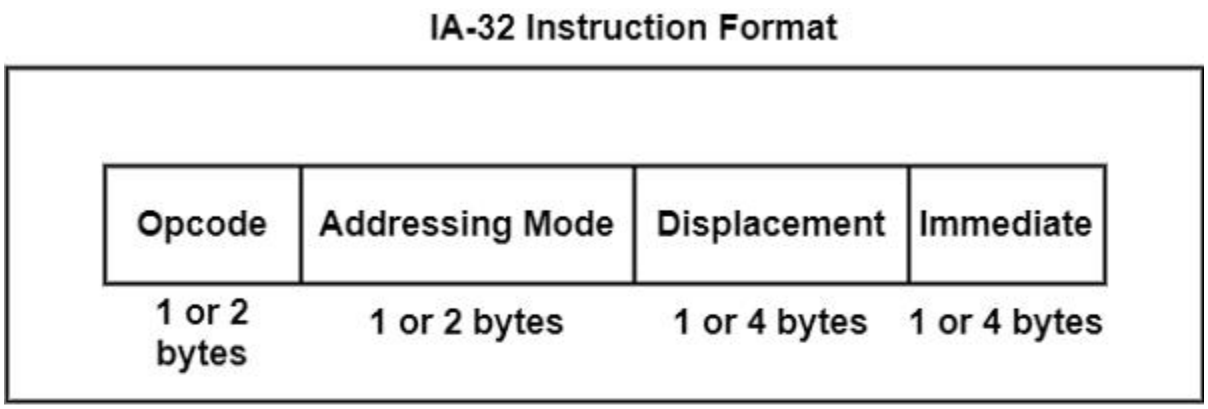
Instruction includes a set of operation codes and operands that manage with the operation codes. Instruction format supports the design of bits in an instruction. It contains fields including opcode, operands, and addressing mode.

The instruction length is generally preserved in multiples of the character length, which is 8 bits. When the instruction length is permanent, several bits are assigned to opcode, operands, and addressing modes.

The function of allocating bits in the instruction can be interpreted by considering the following elements −

- Number of addressing modes
- Number of operands
- Number of CPU registers
- Number of register sets
- Number of address lines

The figure displayed the general IA-32 (Intel Architecture- 32 bits) instruction format. IA-32 is the instruction format that can Intel’s most outstanding microprocessors. This instruction format includes four fields, such as opcode field, addressing mode field, displacement field, and immediate field.

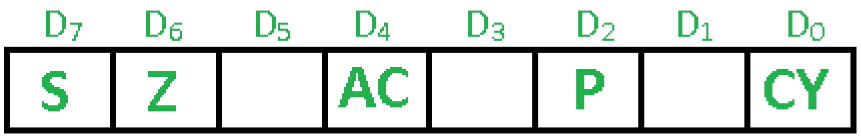


The opcode field has 1 or 2 bytes. The addressing mode field also includes 1 or 2 bytes. In the addressing mode field, an instruction needs only one byte if it uses only one register to generate the effective address of an operand.

The field that directly follows the addressing mode field is the displacement field. If an effective address for a memory operand is computed using the displacement value, then it uses either one or four bytes to encode. If an operand is an immediate value, then it is located in the immediate field and it appears either one or four bytes.

Program Status Word (PSW)

The PSW (often called the flag register) is a very important register. The **Flag register** is a Special Purpose Register. Depending upon the value of the result after any arithmetic and logical operation, the flag bits become set (1) or reset (0). In 8085 microprocessor, the flag register consists of 8 bits and only 5 of them are useful. The Flag register is a 5-bit register in the 8085 microprocessor that contains information about the status of the arithmetic and logic operations performed by the processor. The bits in the Flag register are used to indicate whether the result of an operation is zero, positive, negative, or if there was a carry or borrow during the operation. The 5 flags are:



Sign Flag (S) – After any operation if the MSB (B(7)) of the result is 1, it indicates the number is negative and the sign flag becomes set, i.e. 1. If the MSB is 0, it indicates the number is positive and the sign flag becomes reset i.e. 0. from 00H to 7F, sign flag is 0 from 80H to FF, sign flag is 1 1- MSB is 1 (negative) 0- MSB is 0 (positive)

Zero Flag (Z) – After any arithmetical or logical operation if the result is 0 (00)H, the zero flag becomes set i.e. 1, otherwise it becomes reset i.e. 0. 00H zero flags is 1. from 01H to FFH zero flag is 0 1- zero-result 0- non-zero result

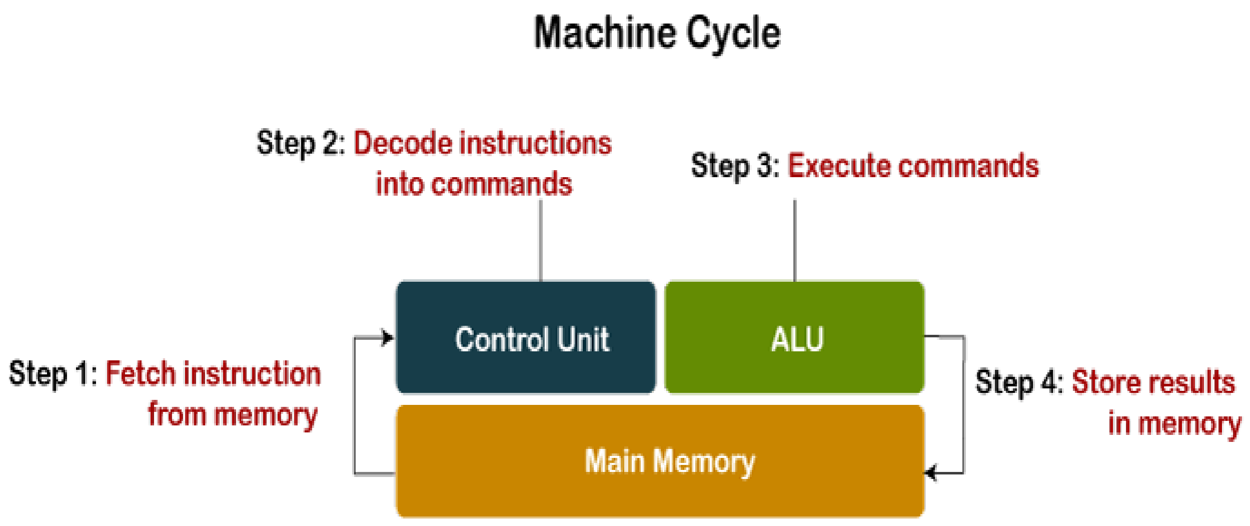
Auxiliary Carry Flag (AC) – This flag is used in the BCD number system(0-9). If after any arithmetic or logical operation D(3) generates any carry and passes it on to D(4) this flag becomes set i.e. 1, otherwise, it becomes reset i.e. 0. This is the only flag register that is not accessible by the programmer 1-carry out from bit 3 on addition or borrows into bit 3 on subtraction 0-otherwise

Parity Flag (P) – If after any arithmetic or logical operation the result has even parity, an even number of 1 bit, the parity register becomes set i.e. 1, otherwise it becomes reset i.e. 0. 1-accumulator has an even number of 1 bits 0-accumulator has odd parity.

Carry Flag (CY) – Carry is generated when performing n bit operations and the result is more than n bits, then this flag becomes set i.e. 1, otherwise, it becomes reset i.e. 0. During subtraction (A-B), if A>B it becomes reset, and if (A<B) it becomes set. Carry flag is also called the borrow flag. 1-carry out from MSB bit on addition or borrow into MSB bit on subtraction 0-no carry out or borrow into MSB bit.

ALU (Arithmetic Logic Unit)

In the computer system, ALU is a main component of the central processing unit, which stands for arithmetic logic unit and performs arithmetic and logic operations. It is also known as an integer unit (IU) that is an integrated circuit within a CPU or GPU, which is the last component to perform calculations in the processor. It has the ability to perform all processes related to arithmetic and logic operations such as addition, subtraction, and shifting operations, including Boolean comparisons (XOR, OR, AND, and NOT operations). Also, binary numbers can accomplish mathematical and bitwise operations. The arithmetic logic unit is split into AU (arithmetic unit) and LU (logic unit). The operands and code used by the ALU tell it which operations have to perform according to input data. When the ALU completes the processing of input, the information is sent to the computer's memory.


Machine Cycle
Step 2: Decode instructions into commands
Step 3: Execute commands
Step 1: Fetch instruction from memory
Control Unit
ALU
Step 4: Store results in memory
Main Memory

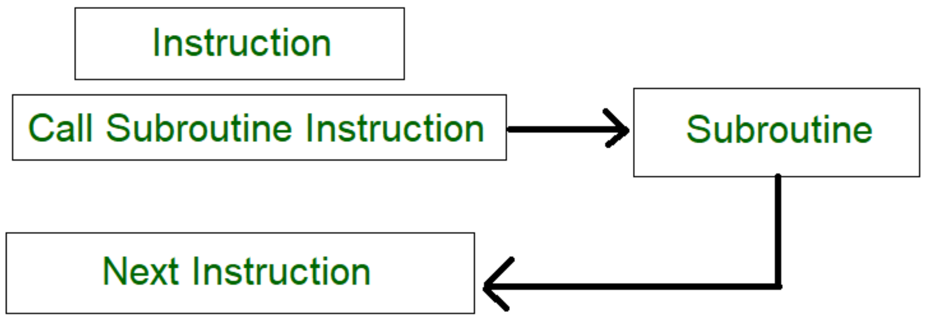
Except performing calculations related to addition and subtraction, ALUs handle the multiplication of two integers as they are designed to execute integer calculations; hence, its result is also an integer. However, division operations commonly may not be performed by ALU as division operations may produce a result in a floating-point number. Instead, the floating-point unit (FPU) usually handles the division operations; other non-integer calculations can also be performed by FPU.

Additionally, engineers can design the ALU to perform any type of operation. However, ALU becomes more costly as the operations become more complex because ALU destroys more heat and takes up more space in the CPU. This is the reason to make powerful ALU by engineers, which provides the surety that the CPU is fast and powerful as well.

The calculations needed by the CPU are handled by the arithmetic logic unit (ALU); most of the operations among them are logical in nature. If the CPU is made more powerful, which is made on the basis of the ALU is designed. Then it creates more heat and takes more power or energy. Therefore, it must be moderation between how complex and powerful ALU is and not be more costly. This is the main reason the faster CPUs are more costly; hence, they take much power and destroy more heat. Arithmetic and logic operations are the main operations that are performed by the ALU; it also performs bit-shifting operations.

Subroutine

A **set of instructions** that are used repeatedly in a program can be referred to as a Subroutine. Only one copy of this Instruction is stored in the memory. When a Subroutine is required it can be called many times during the Execution of a particular program. A call Subroutine Instruction calls the Subroutine. Care Should be taken while returning a Subroutine as a Subroutine can be called from a different place from the memory.
The content of the PC must be Saved by the call Subroutine Instruction to make a correct return to the calling program.



The subroutine linkage method is a way in which computers call and return the Subroutine. The simplest way of Subroutine linkage is saving the return address in a specific location, such as a register which can be called a link register called Subroutine.

Advantages of Subroutines

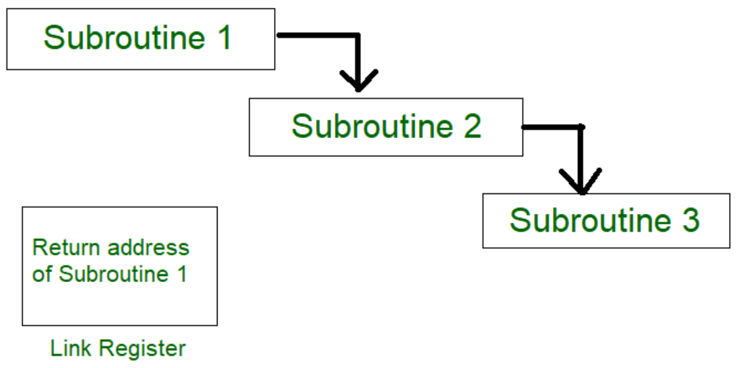
- **Code reuse:** Subroutines can be reused in multiple parts of a program, which can save time and reduce the amount of code that needs to be written.
- **Modularity:** Subroutines help to break complex programs into smaller, more manageable parts, making them easier to understand, maintain, and modify.
- **Encapsulation:** Subroutines provide a way to encapsulate functionality, hiding the implementation details from other parts of the program.

Disadvantages of Subroutines

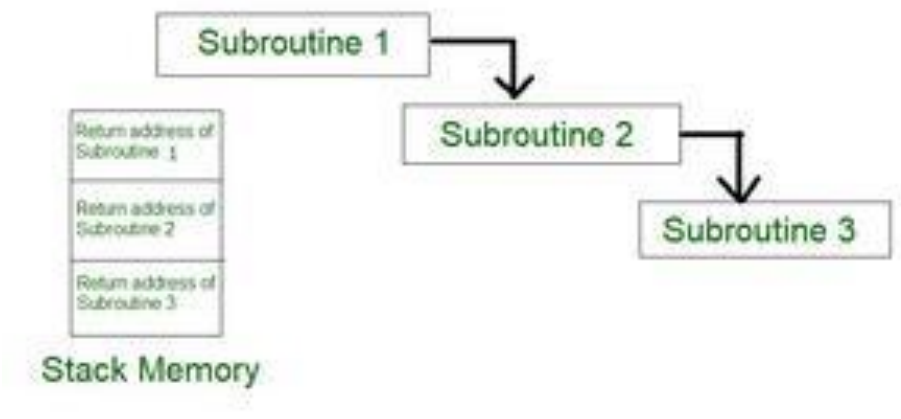
- **Overhead:** Calling a subroutine can incur some overhead, such as the time and memory required to push and pop data on the stack.
- **Complexity:** Subroutine nesting can make programs more complex and difficult to understand, particularly if the nesting is deep or the control flow is complicated.
- **Side Effects:** Subroutines can have unintended side effects, such as modifying global variables or changing the state of the program, which can make debugging and testing more difficult.

What is Subroutine Nesting?

Subroutine nesting is a common Programming practice In which one Subroutine calls another Subroutine.



From the above figure, assume that when Subroutine 1 calls Subroutine 2 the return address of Subroutine 2 should be saved somewhere. So if the link register stores the return address of Subroutine 1 this will be (destroyed/overwritten) by the return address of Subroutine 2. As the last Subroutine called is the first one to be returned (Last in first out format). So stack data structure is the most efficient way to store the return addresses of the Subroutines.


Subroutine 1
Subroutine 2
Subroutine 3
Return address of Subroutine 1
Return address of Subroutine 2
Return address of Subroutine 3
Stack Memory

Interrupt

Interrupt is the mechanism by which modules like I/O or memory may interrupt the normal processing by CPU. It may be either clicking a mouse, dragging a cursor, printing a document etc. the case where interrupt is getting generated. **Why we require Interrupt?** External devices are comparatively slower than CPU. So if there is no interrupt CPU would waste a lot of time waiting for external devices to match its speed with that of CPU. This decreases the efficiency of CPU. Hence, interrupt is required to eliminate these limitations.

With Interrupt:

1. Suppose CPU instructs printer to print a certain document.
2. While printer does its task, CPU engaged in executing other tasks.
3. When printer is done with its given work, it tells CPU that it has done with its work. (The word 'tells' here is interrupt which sends one message that printer has done its work successfully.).

Advantages:

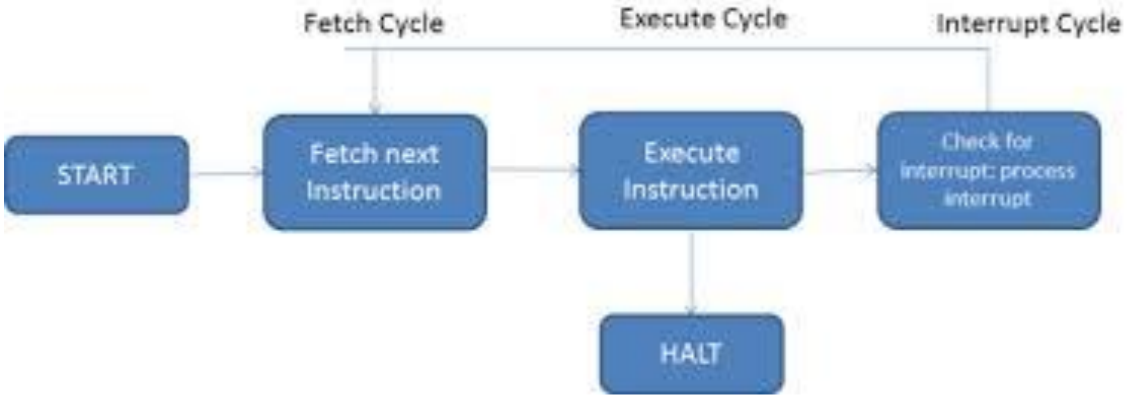
- It increases the efficiency of CPU.
- It decreases the waiting time of CPU.
- Stops the wastage of instruction cycle.
- Enables multitasking by allowing the CPU to quickly switch between different processes.
- Simplifies input/output (I/O) operations by allowing devices to communicate directly with the CPU.

Disadvantages:

- CPU has to do a lot of work to handle interrupts, resume its previous execution of programs (in short, overhead required to handle the interrupt request.).
- Overhead required to handle the interrupt request can reduce the efficiency of the system.
- Interrupt storms can occur when there is high levels of interrupt activity.
- Priority inversion can occur when a low-priority task holds a resource needed by a higher-priority task.

Interrupt Cycle:

An instruction cycle (sometimes called fetch-and-execute cycle, fetch-decode-execute cycle, or FDX) is the basic operation cycle of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction requires, and carries out those actions. This cycle is repeated continuously by the central processing unit (CPU), from boot up to when the computer is shut down.



Block diagram of Interrupt Cycle

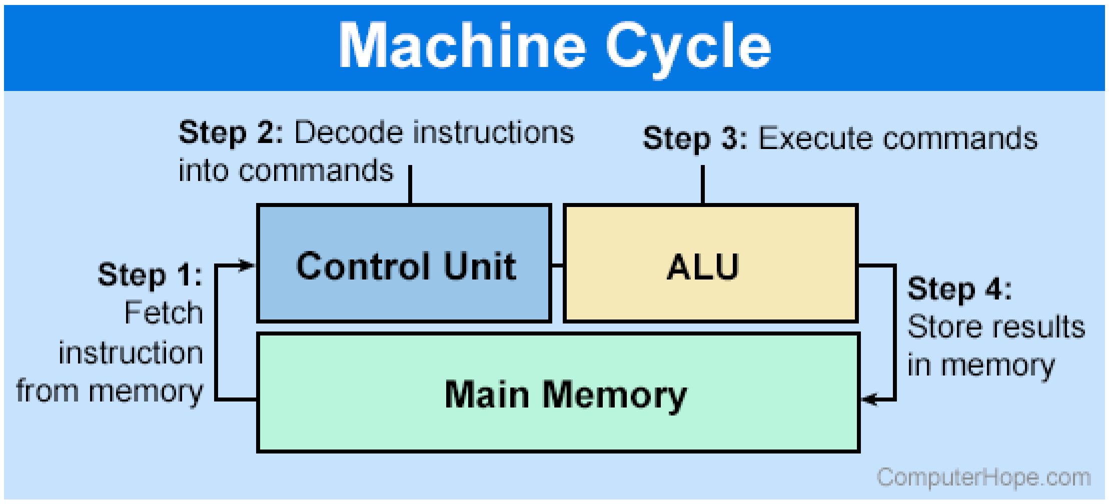
- After the execute cycle is completed, a test is made to determine if an interrupt was enabled (e.g. so that another process can access the CPU)
- If not, instruction cycle returns to the fetch cycle
- If so, the interrupt cycle might perform the following tasks: (simplified...)
- move the current value of PC into MBR
- move the PC-save-address into MBR
- move the interrupt-routine-address into PC
- move the contents of the address in MBR into indicated memory cell
- continue the instruction cycle within the interrupt routine
- after the interrupt routine finishes, the PC-save-address is used to reset the value of PC and program execution can continue

Execution cycle/CPU cycle

A CPU cycle, also known as a machine cycle, is the basic operation performed by a computer's central processing unit (CPU). It consists of fetching an instruction from memory, decoding it, executing it, and then storing the results. This process is repeated for each instruction in a computer program. As for why a computer can't continuously execute instructions, there are a few reasons.

First, the CPU needs time to fetch and decode instructions, perform calculations, and store results.

Additionally, other components of the computer, such as memory and input/output devices, also require access to the CPU. This sharing of resources means that the CPU cannot continuously execute instructions without interruption.


Machine Cycle
Step 2: Decode instructions into commands
Step 3: Execute commands
Step 1: Fetch instruction from memory
Control Unit
ALU
Step 4: Store results in memory
Main Memory
ComputerHope.com

Control Memory

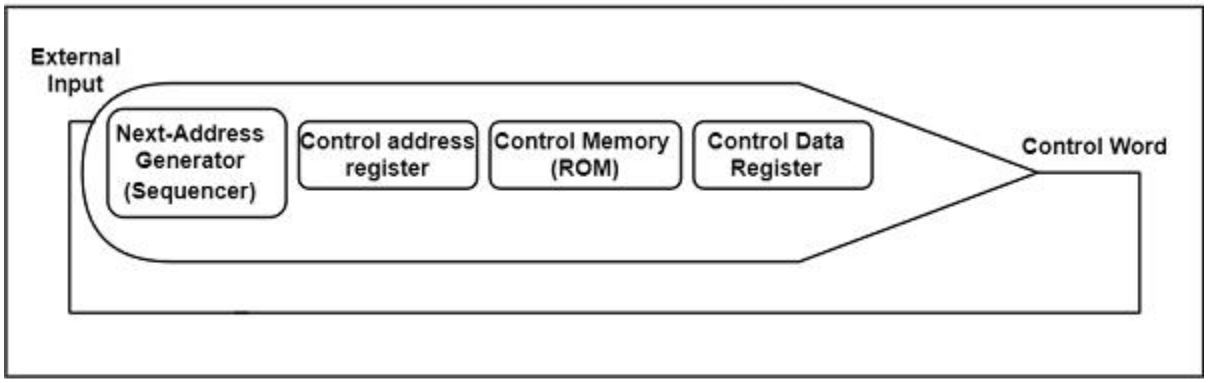
A control memory is a part of the control unit. Any computer that involves microprogrammed control consists of two memories. They are the main memory and the control memory. Programs are usually stored in the main memory by the users. Whenever the programs change, the data is also modified in the main memory. They consist of machine instructions and data.

The control memory consists of microprograms that are fixed and cannot be modified frequently. They contain microinstructions that specify the internal control signals required to execute register micro-operations.

The machine instructions generate a chain of microinstructions in the control memory. Their function is to generate micro-operations that can fetch instructions from the main memory, compute the effective address, execute the operation, and return control to fetch phase and continue the cycle.

The figure shows the general configuration of a microprogrammed control organization.

Micro Programmed Control Organization



Here, the control is presumed to be a Read-Only Memory (ROM), where all the control information is stored permanently. ROM provides the address of the microinstruction. The other register, that is, the control data register stores the microinstruction that is read from the memory. It consists of a control word that holds one or more micro-operations for the data processor.

The next address must be computed once this operation is completed. It is computed in the next address generator. Then, it is sent to the control address register to be read. The next address generator is also known as the microprogram sequencer. Based on the inputs to a sequencer, it determines the address of the next microinstruction. The microinstructions can be specified in several ways.

Design of Control Unit

A control unit drives the corresponding processing hardware by generating a set of signals that are in sync with the master clock. The two major operations performed by the control unit are instruction interpretation and instruction sequencing.

The main function of a control unit is to fetch the data from the main memory, determine the devices and the operations involved with it, and produce control signals to execute the operations.

Types of Control Unit

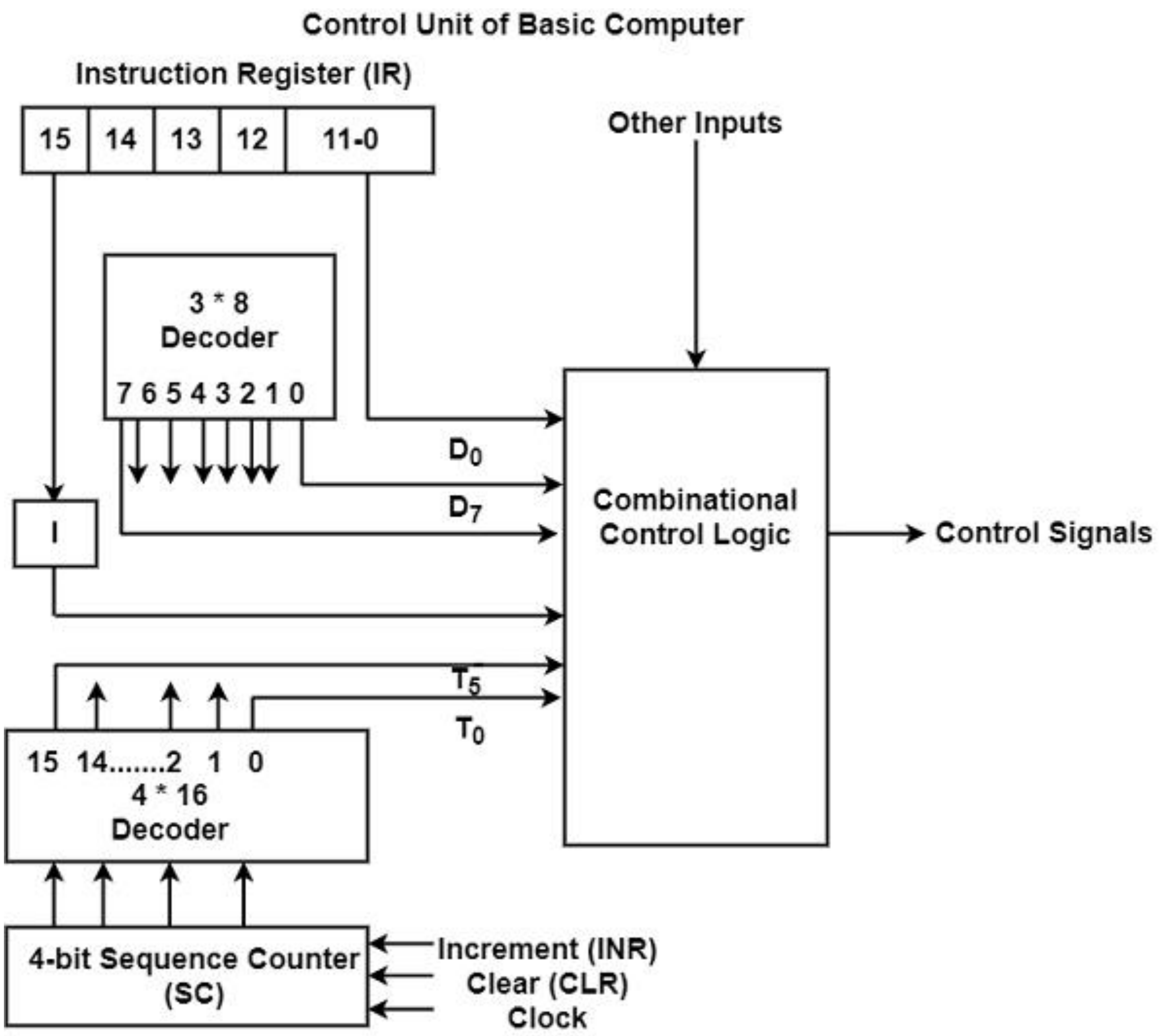
There are two types of control organization that are as follows −

Hardwired Control

In the hardwired organization, the control logic is executed with gates, flip-flops, decoders, and other digital circuits. It can be optimized to make a quick mode of operation. In the micro-programmed organization, the control data is saved in the control memory.

The control memory is programmed to start the needed sequence of micro-operations. A hardwired control requires changes in the wiring among the various elements if the design has to be modified or changed.

The block diagram of the control unit is displayed in the figure. It includes two decoders, a sequence counter, and several control logic gates.



Some instruction that is read from the memory is placed in the Instruction Register (IR). Therefore, the IR is divided into three elements such as I bit, opcode, and bits from 0 through 11. The opcodes are decoded with a 3 * 8 decoder whose outputs are indicated by symbols D_0 through D_7.

The binary value of the respective opcode is the subscripted number in the symbol. The symbol I which is the 15th bit of the instruction is transferred to a flip flop. The control logic gates have the bits that are used from 0 through 11.

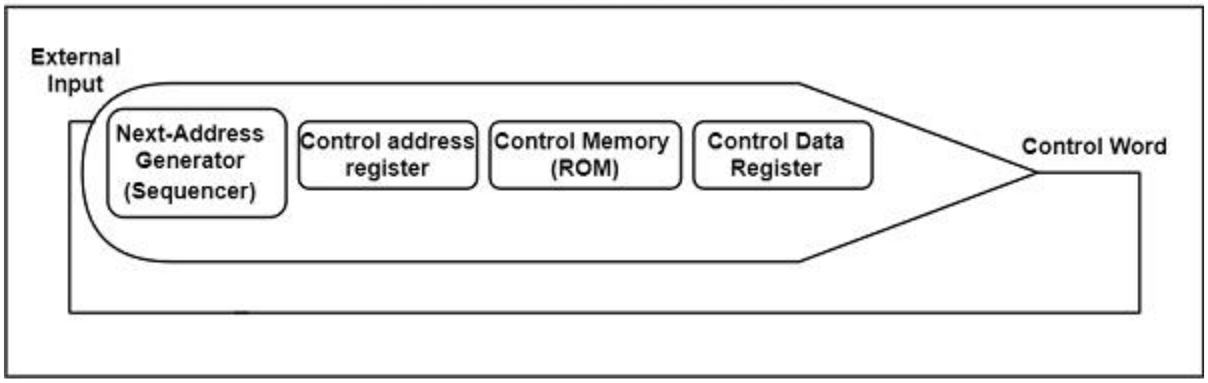
The sequence counter is 4-bit counts in binary from 0 through 15. It can be incremented or cleared synchronously. The timing signals from T_0 through T_{15} are the decoded outputs of the decoder.

Microprogrammed Control

The microprogrammed control stores its control data in the control memory. It can start the important set of micro-operations; the control memory is programmed. The changes and modifications in a micro-programmed control can be completed by upgrading the microprogram in the control memory.

The figure displays the general configuration of a microprogrammed control organization.

Micro Programmed Control Organization



The control is pretended to be a Read-Only Memory (ROM), where all the control data is saved permanently. ROM supports the address of the microinstruction. The other register is the control data register that stores the microinstruction that is read from the memory. It includes a control word that holds one or more microoperations for the data processor.

The next address should be evaluated during this operation is done. It is evaluated in the next address generator. Therefore, it is transferred to the control address register to be read. The next address generator is referred to as the microprogram sequencer. It depends on the inputs to a sequencer, it decides the address of the next microinstruction. The microinstructions can be determined in different approaches.

RISC and CISC

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.

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.

Characteristics of CISC

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

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

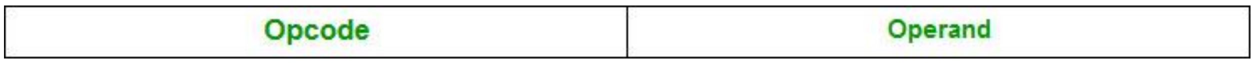
- **Slower execution:** CISC processors take longer to execute instructions because they have more complex instructions and need more time to decode them.
- **More complex design:** CISC processors have more complex instruction sets, which makes them more difficult to design and manufacture.
- **Higher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

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.
Addressing modes for 8086 instructions are divided into two categories:
1) Addressing modes for data

2) Addressing modes for branch

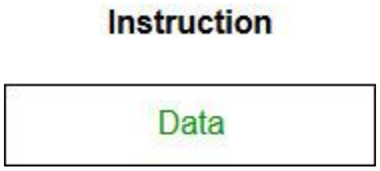
The 8086 memory addressing modes provide flexible access to memory, allowing you to easily access variables, arrays, records, pointers, and other complex data types. The key to good assembly language programming is the proper use of memory addressing modes.

An assembly language program instruction consists of two parts



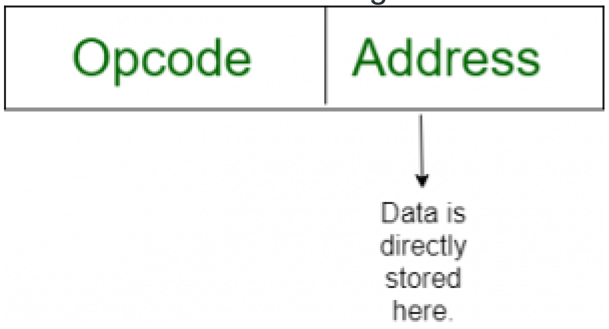
According to different ways of specifying an operand by 8086 microprocessor, different addressing modes are used by 8086.

Addressing modes used by 8086 microprocessor are discussed below:
Implied mode:: In implied addressing the operand is specified in the instruction itself. In this mode the data is 8 bits or 16 bits long and data is the part of instruction. Zero address instruction are designed with implied addressing mode.



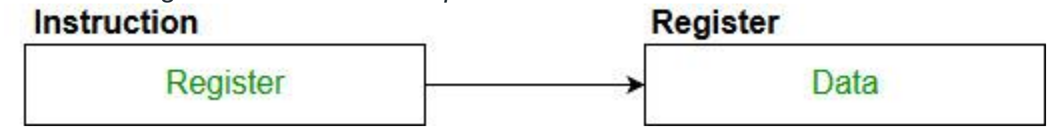
```
Example:  CLC (used to reset Carry flag to 0)
```

Immediate addressing mode (symbol #): In this mode data is present in address field of instruction .Designed like one address instruction format.



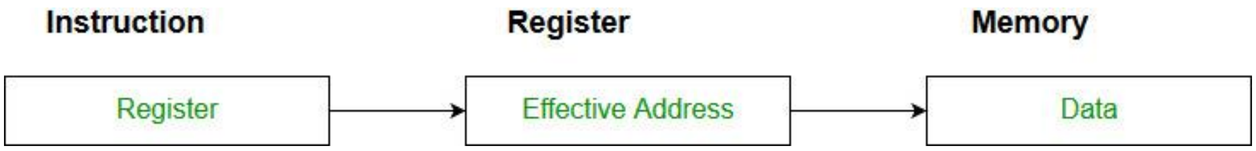
```
Example:  MOV AL, 35H (move the data 35H into AL register)
```

Register mode: In register addressing the operand is placed in one of 8 bit or 16 bit general purpose registers. The data is in the register that is specified by the instruction.
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: In this addressing the operand's offset is placed in any one of the registers BX,BP,SI,DI as specified in the instruction. The effective address of the data is in the base register or an index register that is specified by the instruction.
Here two register reference is required to access the data.



The 8086 CPUs let you access memory indirectly through a register using the register indirect addressing modes.

```
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)**: Effective address of the operand is the contents of a register specified in the instruction. Before accessing the operand, the contents of this register are automatically decremented to point to the previous consecutive memory location. –**(R1)**

Here one register reference, one memory reference and one ALU operation is required to access the data.

Example:

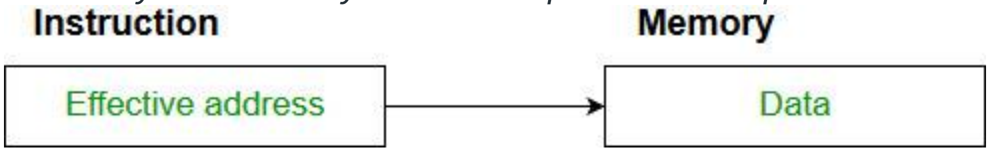
```
Add R1,-(R2)   //OR

R2 = R2-d
R1 = R1 + M[R2]
```

Auto decrement mode is same as auto increment mode. Both can also be used to implement a stack as push and pop . Auto increment and Auto decrement modes are useful for implementing "Last-In-First-Out" data structures.

- **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** :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 mode effective address is in the register, and corresponding register name will be maintained in the address field of an instruction.
     Here one register reference, one memory reference is required to access the data.
  2. Memory Indirect: In this mode effective address is in the memory, and corresponding memory address will be maintained in the address field of an instruction.
     Here two memory reference is required to access the data.

- **Indexed addressing mode**: The operand's offset is the sum of the content of an index register SI or DI and an 8 bit or 16 bit displacement.
  `Example: MOV AX, [SI +05]`

- **Based Indexed Addressing:** The operand's offset is sum of the content of a base register BX or BP and an index register SI or DI.
  `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. PC relative and based register both addressing modes are suitable for program relocation at runtime.
2. Based register addressing mode is best suitable to write position independent codes.

Advantages of Addressing Modes

1. To give programmers to facilities such as Pointers, counters for loop controls, indexing of data and program relocation.
2. To reduce the number bits in the addressing field of the Instruction.