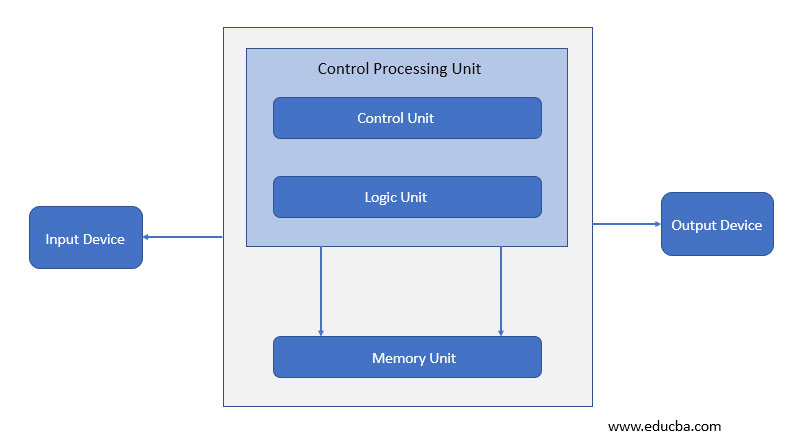
**Computer architecture**

A computer system is basically a machine that simplifies complicated tasks. It should maximize performance and reduce costs as well as power consumption.The different components in the Computer System Architecture are Input Unit, Output Unit, Storage Unit, Arithmetic Logic Unit, Control Unit etc.

A diagram that shows the flow of data between these units is as follows −



The input data travels from input unit to ALU. Similarly, the computed data travels from ALU to output unit. The data constantly moves from storage unit to ALU and back again. This is because stored data is computed on before being stored again. The control unit controls all the other units as well as their data.

Details about all the computer units are −

* **Input Unit**

The input unit provides data to the computer system from the outside. So, basically it links the external environment with the computer. It takes data from the input devices, converts it into machine language and then loads it into the computer system. Keyboard, mouse etc. are the most commonly used input devices.

* **Output Unit**

The output unit provides the results of computer process to the users i.e it links the computer with the external environment. Most of the output data is the form of audio or video. The different output devices are monitors, printers, speakers, headphones etc.

* **Storage Unit**

Storage unit contains many computer components that are used to store data. It is traditionally divided into primary storage and secondary storage.Primary storage is also known as the main memory and is the memory directly accessible by the CPU. Secondary or external storage is not directly accessible by the CPU. The data from secondary storage needs to be brought into the primary storage before the CPU can use it. Secondary storage contains a large amount of data permanently.

* **Arithmetic Logic Unit**

All the calculations related to the computer system are performed by the arithmetic logic unit. It can perform operations like addition, subtraction, multiplication, division etc. The control unit transfers data from storage unit to arithmetic logic unit when calculations need to be performed. The arithmetic logic unit and the control unit together form the central processing unit.

* **Control Unit**

This unit controls all the other units of the computer system and so is known as its central nervous system. It transfers data throughout the computer as required including from storage unit to central processing unit and vice versa. The control unit also dictates how the memory, input output devices, arithmetic logic unit etc. should behave.

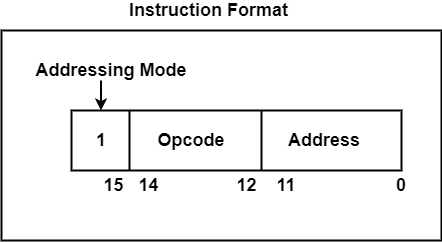
**Instruction codes**

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

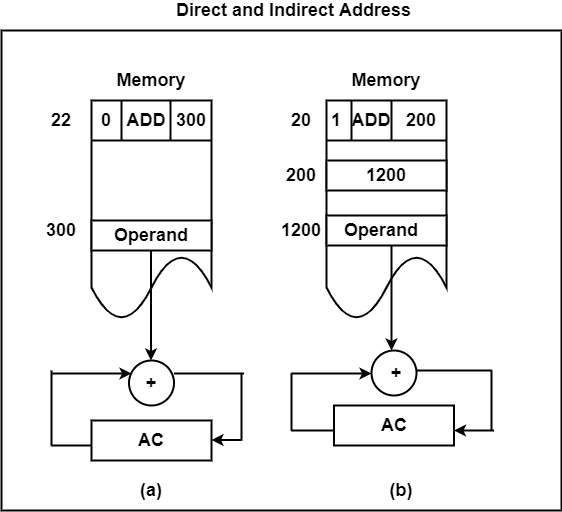
An opcode is a collection of bits that represents the basic operations including add, subtract, multiply, complement, and shift. The total number of operations provided through the computer determines the number of bits needed for the opcode. The minimum bits accessible to the opcode should be n for 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.



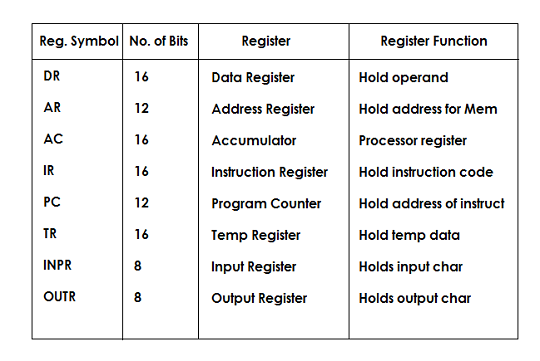
### Computer Registers

Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. The registers used by the CPU are often termed as Processor registers.

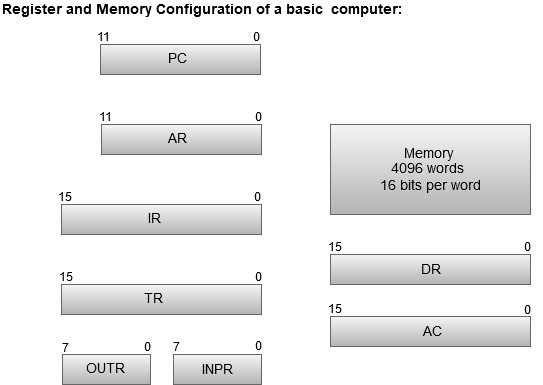
A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).

The computer needs processor registers for manipulating data and a register for holding a memory address. The register holding the memory location is used to calculate the address of the next instruction after the execution of the current instruction is completed.

Following is the list of some of the most common registers used in a basic computer:



Skip Ad



* The Memory unit has a capacity of 4096 words, and each word contains 16 bits.
* The Data Register (DR) contains 16 bits which hold the operand read from the memory location.
* The Memory Address Register (MAR) contains 12 bits which hold the address for the memory location.
* The Program Counter (PC) also contains 12 bits which hold the address of the next instruction to be read from memory after the current instruction is executed.
* The Accumulator (AC) register is a general purpose processing register.
* The instruction read from memory is placed in the Instruction register (IR).
* The Temporary Register (TR) is used for holding the temporary data during the processing.
* The Input Registers (IR) holds the input characters given by the user.
* The Output Registers (OR) holds the output after processing the input data.

**Computer Instructions**

Computer instructions are a set of machine language instructions that a particular processor understands and executes. A computer performs tasks on the basis of the instruction provided.

An instruction comprises of groups called fields. These fields include:

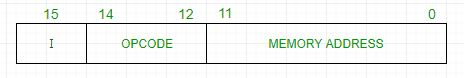
* The Operation code (Opcode) field which specifies the operation to be performed.
* The Address field which contains the location of the operand, i.e., register or memory location.
* The Mode field which specifies how the operand will be located.

IMG_256

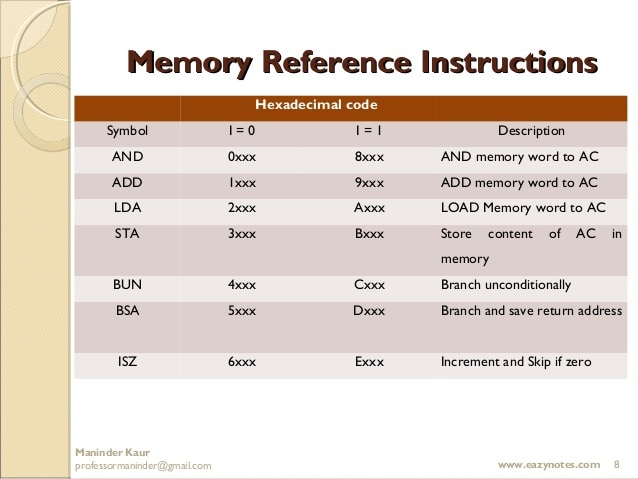
A basic computer has three instruction code formats which are:

1. Memory - reference instruction
2. Register - reference instruction
3. Input-Output instruction

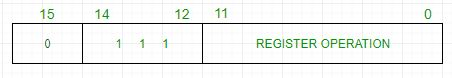
**Memory - reference instruction**



In Memory-reference instruction, 12 bits of memory is used to specify an address and one bit to specify the addressing mode 'I'.

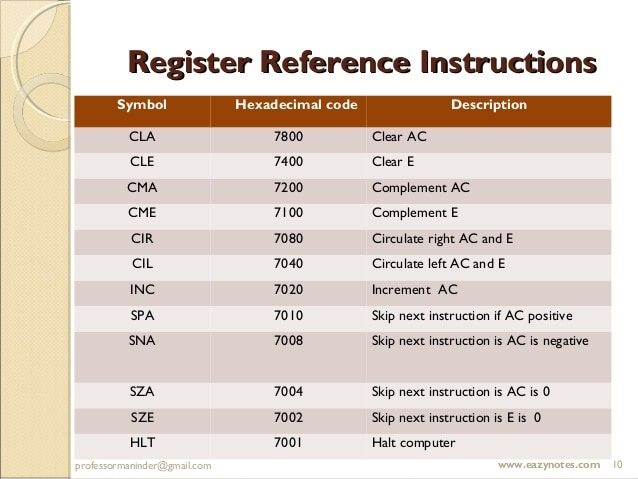


**Register - reference instruction**

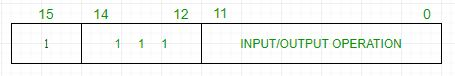


The Register-reference instructions are represented by the Opcode 111 with a 0 in the leftmost bit (bit 15) of the instruction.

A Register-reference instruction specifies an operation on or a test of the AC (Accumulator) register.



## Input-Output instruction



Just like the Register-reference instruction, an Input-Output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of the input-output operation or test performed.

### IMG_256

**Timing and control**

The timing for all registers in the basic computer is controlled by a master clock generator

The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit

The clock pulses do not change the state of a register unless the register is enabled by a control signal (i.e., Load)

The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and micro operations for the accumulator

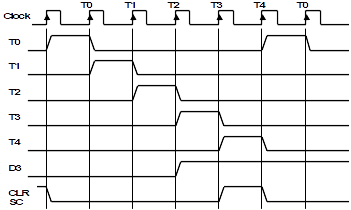
There are two major types of control organization:

**Hardwired control**

In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.

**Micro programmed control**

In the microprogrammed organization, the control information is stored in a control memory (if the design is modified, the micro program in control memory has to be updated)

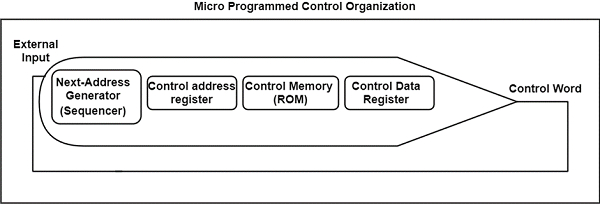


A control unit whose binary control values are saved as words in memory is called a microprogrammed control unit.

A controller results in the instructions to be implemented by constructing a definite collection of signals at each system clock beat. Each of these output signals generates one micro-operation including register transfer. Thus, the sets of control signals are generated definite micro-operations that can be saved in the memory.

Each bit that forms the microinstruction is linked to one control signal. When the bit is set, the control signal is active. When it is cleared the control signal turns inactive. These microinstructions in a sequence can be saved in the internal ’control’ memory. The control unit of a microprogram-controlled computer is a computer inside a computer.

The following image shows the block diagram of a Microprogrammed Control organization.

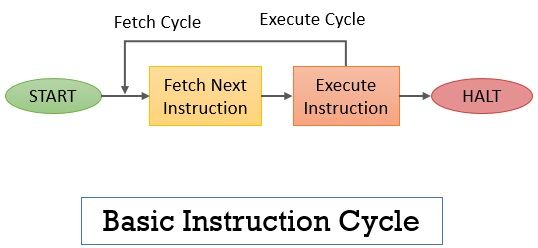


**Instruction cycle**

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.



## Fetch Cycle

The address instruction to be implemented is held at the program counter. The processor fetches the instruction from the memory that is pointed by the PC.

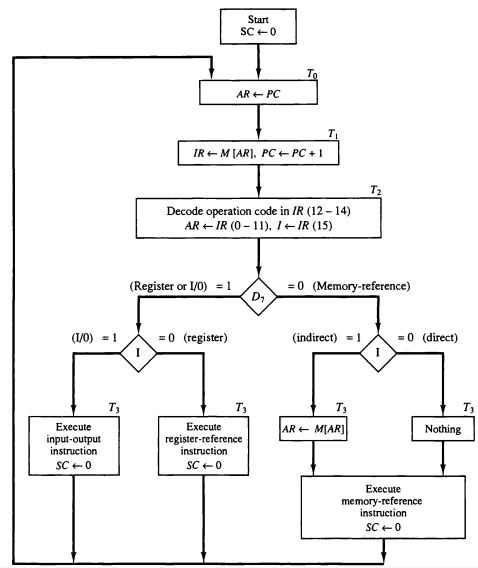
Next, the PC is incremented to display the address of the next instruction. This instruction is loaded onto the instruction register. The processor reads the instruction and executes the important procedures.

## Execute Cycle

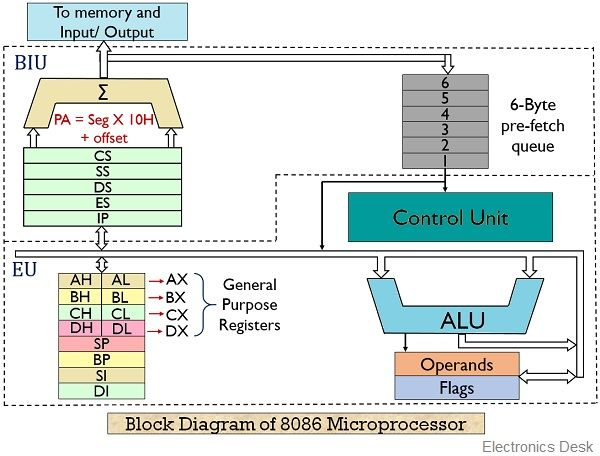
The data transfers for implementation takes place in two methods are as follows −

* **Processor-memory** − The data sent from the processor to memory or from memory to processor.
* **Processor-Input/Output** − The data can be transferred to or from a peripheral device by the transfer between a processor and an I/O device.

In the execute cycle, the processor implements the important operations on the information, and consistently the control calls for the modification in the sequence of data implementation. These two methods associate and complete the execute cycle.



**8086 architecture**

****

## Bus Interface Unit (BIU)

The segment registers, instruction pointer and 6-byte instruction queue are associated with the bus interface unit (BIU).

The BIU:

* Handles transfer of data and addresses,
* Fetches instruction codes, stores fetched instruction codes in first-in-first-out register set called a **queue**,
* Reads data from memory and I/O devices,
* Writes data to memory and I/O devices,
* It relocates addresses of operands since it gets un-relocated operand addresses from EU. The EU tells the BIU from where to fetch instructions or where to read data.

It has the following functional parts:

* **Instruction Queue:** When EU executes instructions, the BIU gets 6-bytes of the next instruction and stores them in the instruction queue and this process is known as instruction pre fetch. This process increases the speed of the processor.
* **Segment Registers:** A segment register contains the addresses of instructions and data in memory which are used by the processor to access memory locations. It points to the starting address of a memory segment currently being used.  
  There are 4 segment registers in 8086 as given below:
* **Code Segment Register (CS):** Code segment of the memory holds instruction codes of a program.
* **Data Segment Register (DS):** The data, variables and constants given in the program are held in the data segment of the memory.
* **Stack Segment Register (SS):** Stack segment holds addresses and data of subroutines. It also holds the contents of registers and memory locations given in PUSH instruction.
* **Extra Segment Register (ES):** Extra segment holds the destination addresses of some data of certain string instructions.
* **Instruction Pointer (IP):** The instruction pointer in the 8086 microprocessor acts as a program counter. It indicates to the address of the next instruction to be executed.

Execution Unit (EU)

* The **EU** receives opcode of an instruction from the queue, decodes it and then executes it. While Execution, unit decodes or executes an instruction, then the BIU fetches instruction codes from the memory and stores them in the queue.
* The BIU and EU operate in parallel independently. This makes processing faster.
* General purpose registers, stack pointer, base pointer and index registers, ALU, flag registers (FLAGS), instruction decoder and timing and control unit constitute execution unit (EU).

**General Purpose Registers:** There are four 16-bit general purpose registers: AX (Accumulator Register), BX (Base Register), CX (Counter) and DX. Each of these 16-bit registers are further subdivided into 8-bit registers as shown below: The use of general-purpose registers is to store temporary data. While the instructions are executed in the control unit, they may work on some numeric value or some operands. These need to be stored somewhere so that the processor can operate on them easily. So, these registers are used in these cases. There are 4 general-purpose registers of 16-bit length each. Each of them is further divided into two subparts of 8-bit length each: one high, which stores the higher-order bits and another low which stores the lower order bits.

1. AX = [AH:AL]
2. BX = [BH:BL]
3. CX = [CH:CL]
4. DX = [DH:DL]

| **16-bit registers** | **8-bit high-order registers** | **8-bit low-order registers** |
| --- | --- | --- |
| AX | AH | AL |
| BX | BH | BL |
| CX | CH | CL |
| DX | DH | DL |

* **Index Register:** The following four registers are in the group of pointer and index registers:
  + Stack Pointer (SP)
  + Base Pointer (BP)
  + Source Index (SI)
  + Destination Index (DI)

Top of Form

The pointers will always store some address or memory location. In **8086 Microprocessor**, they usually store the offset through which the actual address is calculated.

1. **Base Pointer (BP):**  
   The Base pointer stores the base address of the memory. Also, it acts as an offset for Stack Segment (SS).
2. **Stack Pointer (SP):**  
   The Stack Pointer Points at the current top value of the Stack. Like the BP, it also acts as an offset to the Stack Segment (SS).  
   The indexes are used with the extra segment and they usually are used for copying the contents of a particular block of memory to a new location.
3. **Source Index (SI):**  
   It stores the offset address of the source.
4. **Destination Index (DI):**  
   It stores the offset address of the Destination.

**ALU:** It handles all arithmetic and logical operations. Such as addition, subtraction, multiplication, division, AND, OR, NOT operations.

* **Flag Register:** It is a 16?bit register which exactly behaves like a flip-flop, means it changes states according to the result stored in the accumulator. It has 9 flags and they are divided into 2 groups i.e. conditional and control flags.
  + **Conditional Flags:** This flag represents the result of the last arithmetic or logical instruction executed. Conditional flags are:
    - Carry Flag
    - Auxiliary Flag
    - Parity Flag
    - Zero Flag
    - Sign Flag
    - Overflow Flag
  + **Control Flags:** It controls the operations of the execution unit. Control flags are:
    - Trap Flag
    - Interrupt Flag
    - Direction Flag

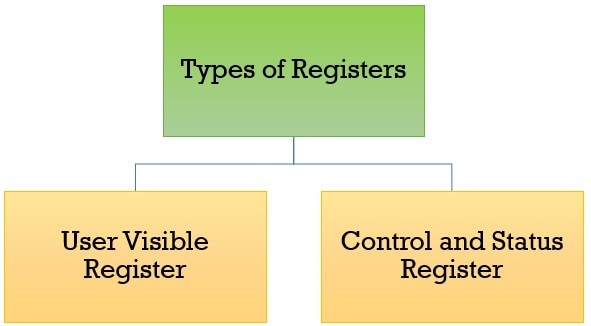
## Register organization

**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.

## Types of Registers

As we have discussed above, registers can be organized into two main categories i.e. the **User-Visible Registers** and the **Control and Status Registers**. Although we can’t separate the registers in the processors clearly among these two categories.



This is because in some processors, a register may be user-visible and in some, the same may not be user-visible. But for our rest of discussion regarding register organization, we will consider these two categories of register.

1. [**User Visible Registers**](https://binaryterms.com/register-organization.html#UserVisibleRegisters)

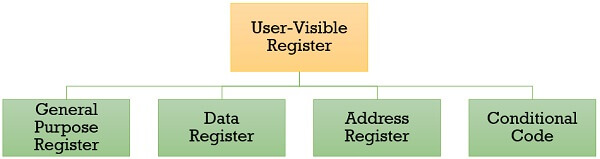
* [General Purpose Register](https://binaryterms.com/register-organization.html#GeneralPurposeRegister)
* [Data Register](https://binaryterms.com/register-organization.html#DataRegister)
* [Address Register](https://binaryterms.com/register-organization.html#AddressRegister)
* [Condition Codes](https://binaryterms.com/register-organization.html#ConditionCodes)

1. [**Control and Status Registers**](https://binaryterms.com/register-organization.html#ControlandStatusRegisters)

* [Program Counter](https://binaryterms.com/register-organization.html#ProgramCounter)
* [Instruction Register](https://binaryterms.com/register-organization.html#InstructionRegister)
* [Memory Address Register](https://binaryterms.com/register-organization.html#MemoryAddressRegister)
* [Memory Buffer Register](https://binaryterms.com/register-organization.html#MemoryBufferRegister)

### 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** or 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**.

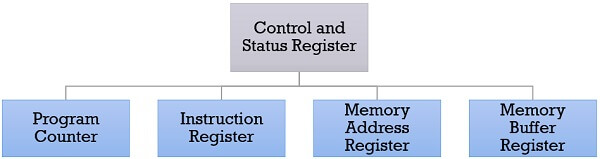
#### 4. Condition Code

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.

So, these are the types of registers a processor has. The processor designer organizes the registers according to the requirement of the processor.

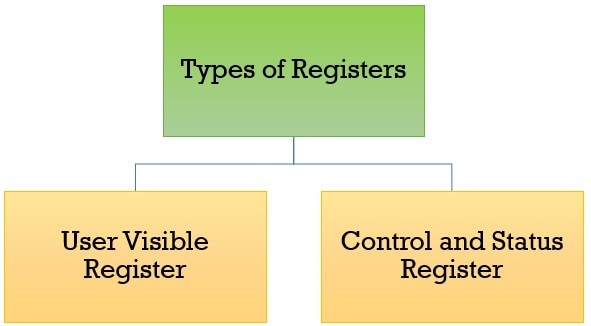
## Register organization

**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.

## Types of Registers

As we have discussed above, registers can be organized into two main categories i.e. the **User-Visible Registers** and the **Control and Status Registers**. Although we can’t separate the registers in the processors clearly among these two categories.



This is because in some processors, a register may be user-visible and in some, the same may not be user-visible. But for our rest of discussion regarding register organization, we will consider these two categories of register.

1. [**User Visible Registers**](https://binaryterms.com/register-organization.html#UserVisibleRegisters)

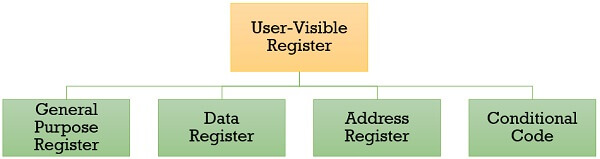
* [General Purpose Register](https://binaryterms.com/register-organization.html#GeneralPurposeRegister)
* [Data Register](https://binaryterms.com/register-organization.html#DataRegister)
* [Address Register](https://binaryterms.com/register-organization.html#AddressRegister)
* [Condition Codes](https://binaryterms.com/register-organization.html#ConditionCodes)

1. [**Control and Status Registers**](https://binaryterms.com/register-organization.html#ControlandStatusRegisters)

* [Program Counter](https://binaryterms.com/register-organization.html#ProgramCounter)
* [Instruction Register](https://binaryterms.com/register-organization.html#InstructionRegister)
* [Memory Address Register](https://binaryterms.com/register-organization.html#MemoryAddressRegister)
* [Memory Buffer Register](https://binaryterms.com/register-organization.html#MemoryBufferRegister)

### 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** or 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**.

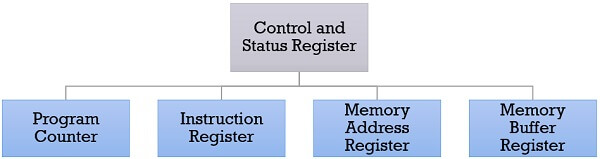
#### 4. Condition Code

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.

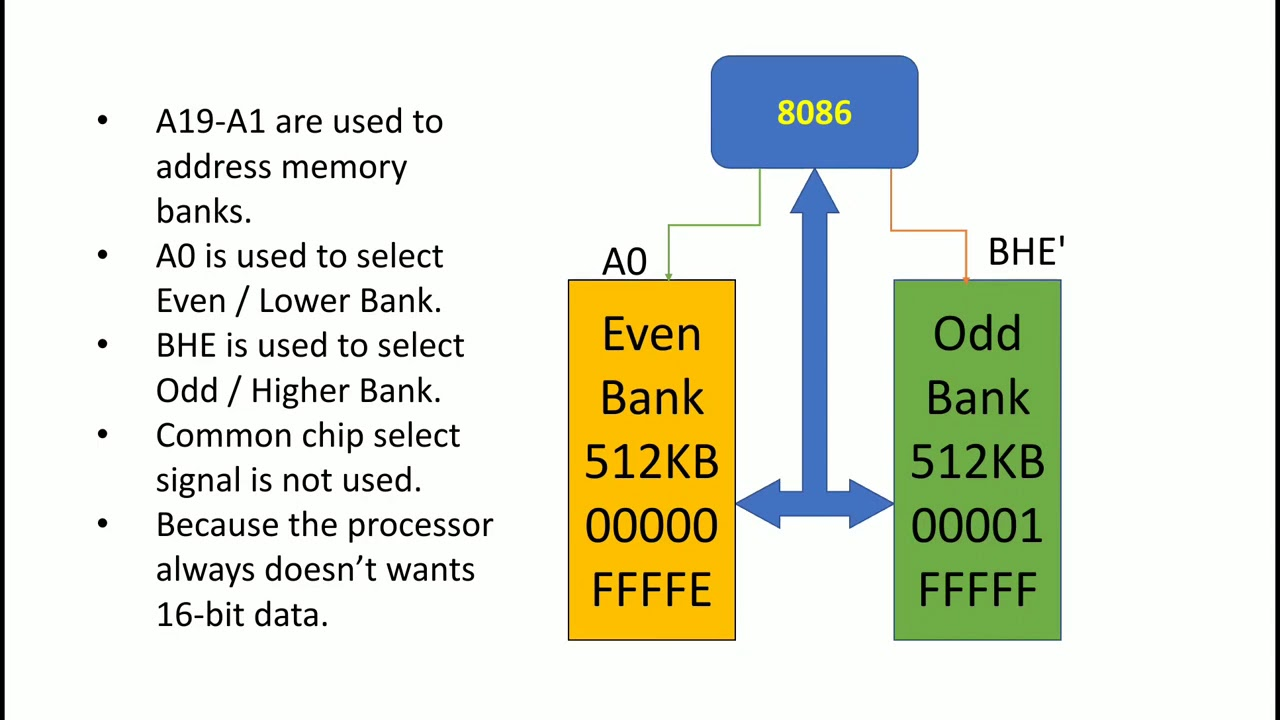
So, these are the types of registers a processor has. The processor designer organizes the registers according to the requirement of the processor.

**Physical memory organization**

The 8086 processor provides a 16 bit data bus. So It is capable of transferring 16 bits in one cycle but each memory location is only of a byte(8 bits), therefore we need two cycles to access 16 bits(8 bit each) from two different memory locations. The solution to this problem is Memory Banking.Through Memory banking our goal is to access two consecutive memory locations in one cycle(transfer 16 bits).

The memory chip is equally divided into two parts(banks). One of the banks contain even addresses called **Even bank** and the other contain odd addresses called **Odd bank**.Even bank always gives lower byte So Even bank is also called **Lower bank**(LB) and Odd bank is also called a **Higher bank**(HB).

This banking scheme allows to access two aligned memory locations from both banks simultaneously and process 16 bit data transfer.Memory banking doesn’t make it compulsory to transfer 16 bits, it facilitates the 16 bit data transfer.  
The choice between 8 bit and 16 bit transfer depends on the instructions given by the programmer

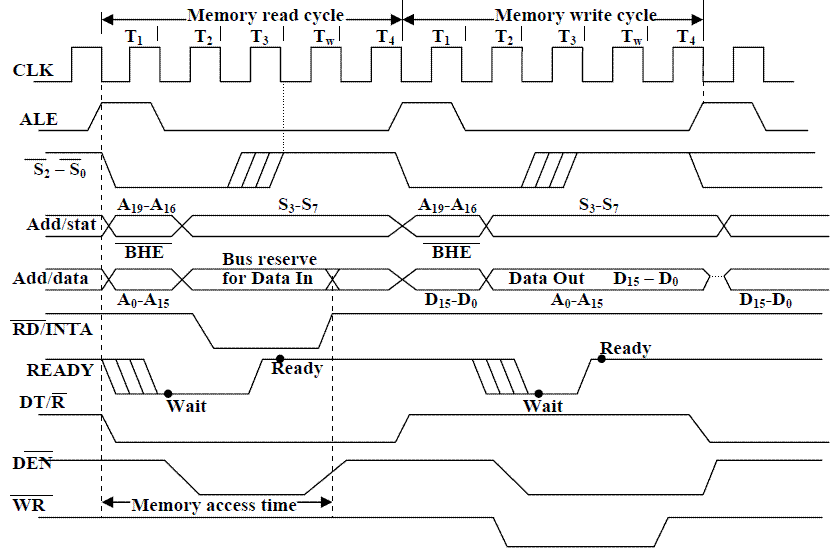


The least Significant bit of address (A0 is not used for byte selection)it is reserved for bank selection. Therefore A0=0 will select Even bank. The BHE signal is used for selection of odd bank.the processor will used combination of this two signals decide type of data transfer.

| BHE | A0 | types of Transfer |
| --- | --- | --- |
| 0 | 0 | 16 bit data transfer from both HB and LB |
| 0 | 1 | 8 bit data transfer from HB |
| 1 | 0 | 8 bit data transfer from LB |
| 1 | 1 | None(Idle) |

### General bus operation of 8086

* The 8086 has a combined address and data bus commonly referred as a time multiplexed address and data bus.
* The main reason behind multiplexing address and data over the same pins is the maximum utilization of processor pins and it facilitates the use of 40 pin standard DIP package.
* The bus can be demultiplexed using a few latches and transceiver, when ever required.
* Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T1, T2, T3, T4. The address is transmitted by the processor during T1. It is present on the bus only for one cycle.
* The negative edge of this ALE pulse is used to separate the address and the data or status information. In maximum mode, the status lines S0, S1 and S2 are used to indicate the type of operation.
* Status bits S3 to S7 are multiplexed with higher order address bits and the BHE signal. Address is valid during T1 while status bits S3 to S7 are valid during T2 through T4.



**Maximum mode**

* In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
* In this mode, the processor derives the status signal S2, S1, S0. Another chip called bus controller derives the control signal using this status information .
* In the maximum mode, there may be more than one microprocessor in the system configuration.

**Minimum mode**

* In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.
* In this mode, all the control signals are given out by the microprocessor chip itself.
* There is a single microprocessor in the minimum mode system.

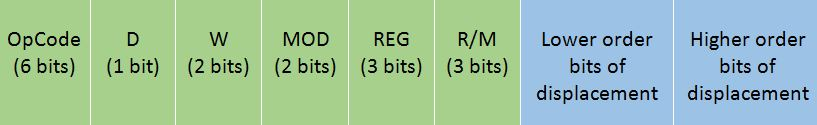
# Instruction Format in 8086 Microprocessor

In this article, we are going to study the **instruction format that is followed in the 8086 microprocessor**.  
Submitted by [Monika Sharma](https://www.includehelp.com/Members/Monika-Sharma.aspx), on July 22, 2019

For every **instruction that is executed in the 8086 microprocessor**, an **instruction format** is available that is the binary representation of that instruction.

This instruction format can be coded from 1 to 6 bytes depending upon the addressing modes used for instructions.

The general Instruction format that most of the **instructions of the 8086 microprocessor** follow is:



* The Opcode stands for Operation Code. Every Instruction has a unique 6-bit opcode. For example, the opcode for **MOV** is 100010.
* **D** stands for direction  
  If **D=0**, then the direction is from the register  
  If **D=1**, then the direction is to the register
* **W** stands for word  
  If **W=0**, then only a byte is being transferred, i.e. 8 bits  
  If **W=1**, them a whole word is being transferred, i.e. 16 bits
* The **MOD** and **R/M** together is calculated based upon the addressing mode and register being used in it. This is calculated as follows:

| **R/M** | **0 0 (Memory Mode with no displacement)** | **0 1 (Memory mode with 8 bit displacement)** | **1 0 (Memory Mode with 16 bit displacement)** | **1 1 (Register Mode)** |
| --- | --- | --- | --- | --- |
| 000 | [BX] + [SI] | [BX] + [SI] + d8 | [BX] + [SI] + d16 | AL AX |
| 001 | [BX] + [DI] | [BX] + [DI] + d8 | [BX] + [DI] + d16 | CL CX |
| 010 | [BP] + [SI] | [BP] + [SI] + d8 | [BP] + [SI] + d16 | DL DX |
| 011 | [BP] + [DI] | [BP] + [DI] + d8 | [BP] + [DI] + d16 | BL BX |
| 100 | [SI] | [SI] + d8 | [SI] + d16 | AH SP |
| 101 | [DI] | [DI] + d8 | [DI] + d16 | CH BP |
| 110 | d16 (direct) | [BP] + d8 | [BP] + d16 | DH SI |
| 111 | [BX] | [BX] + d8 | [BX] + d16 | BH DI |

* REG stands for register selected. It is a 3-bit code which is calculated as follows:

| **REG Code** | **Register Selected** |
| --- | --- |
| 0 0 0 | AL AX |
| 0 0 1 | CL CX |
| 0 1 0 | DL DX |
| 0 1 1 | BL BX |
| 1 0 0 | AH SP |
| 1 0 1 | CH BP |
| 1 1 0 | DH SI |
| 1 1 1 | BH DI |

* The low order displacement and high order displacement are optional and the instruction format contains them only if there exists any displacement in the instruction. If the displacement is of 8 bits, then only the cell of low order displacement infilled and if the displacement is of 16 bits, then both the cells od low order and high order are filled, with the exact bits that the displacement number represents.

Addressing modes of 8086

The way for which an operand is specified for an instruction in the accumulator, in a general purpose register or in memory location, is called **addressing mode**.

The 8086 microprocessors have 8 addressing modes. Two addressing modes have been provided for instructions which operate on register or immediate data.

**These two addressing modes are:**

**Register Addressing:** In register addressing, the operand is placed in one of the 16-bit or 8-bit general purpose registers.

**Example**

* MOV AX, CX
* ADD AL, BL
* ADD CX, DX

**Immediate Addressing:** In immediate addressing, the operand is specified in the instruction itself.

**Example**

* MOV AL, 35H
* MOV BX, 0301H
* MOV [0401], 3598H
* ADD AX, 4836H

The remaining 6 addressing modes specify the location of an operand which is placed in a memory.

**These 6 addressing modes are:**

**Direct Addressing:** In direct addressing mode, the operand?s offset is given in the instruction as an 8-bit or 16-bit displacement element.

**Example**

* ADD AL, [0301]

The instruction adds the content of the offset address 0301 to AL. the operand is placed at the given offset (0301) within the data segment DS.

**Register Indirect Addressing:** The operand's offset is placed in any one of the registers BX, BP, SI or DI as specified in the instruction.

**Example**

* MOV AX, [BX]

It moves the contents of memory locations addressed by the register BX to the register AX.

**Based Addressing:** The operand's offset is the sum of an 8-bit or 16-bit displacement and the contents of the base register BX or BP. BX is used as base register for data segment, and the BP is used as a base register for stack segment.

**Effective address (Offset)** = [BX + 8-bit or 16-bit displacement].

**Example**

* MOV AL, [BX+05]; an example of 8-bit displacement.
* MOV AL, [BX + 1346H]; example of 16-bit displacement.

**Indexed Addressing:** The offset of an operand is the sum of the content of an index register SI or DI and an 8-bit or 16-bit displacement.

Offset (Effective Address) = [SI or DI + 8-bit or 16-bit displacement]

**Example**

* MOV AX, [SI + 05]; 8-bit displacement.
* MOV AX, [SI + 1528H]; 16-bit displacement.

**Based Indexed Addressing:** The offset of operand is the sum of the content of a base register BX or BP and an index register SI or DI.

Effective Address (Offset) = [BX or BP] + [SI or DI]

Here, BX is used for a base register for data segment, and BP is used as a base register for stack segment.

**Example**

* ADD AX, [BX + SI]
* MOV CX, [BX + SI]

**Based Indexed with Displacement:** In this mode of addressing, the operand's offset is given by:

**Effective Address (Offset)** = [BX or BP] + [SI or DI] + 8-bit or 16-bit displacement

**Example**

* MOV AX, [BX + SI + 05]; 8-bit displacement
* MOV AX, [BX + SI + 1235H]; 16-bit displacement
* Instructions are classified on the basis of functions they perform. They are categorized into the following main types:

**Instruction set**

## Data Transfer instruction

* All the instructions which perform data movement come under this category. The source data may be a register, memory location, port etc. the destination may be a register, memory location or port. The following instructions come under this category:

| **Instruction** | **Description** |
| --- | --- |
| MOV | Moves data from register to register, register to memory, memory to register, memory to accumulator, accumulator to memory, etc. |
| LDS | Loads a word from the specified memory locations into specified register. It also loads a word from the next two memory locations into DS register. |
| LES | Loads a word from the specified memory locations into the specified register. It also loads a word from next two memory locations into ES register. |
| LEA | Loads offset address into the specified register. |
| LAHF | Loads low order 8-bits of the flag register into AH register. |
| SAHF | Stores the content of AH register into low order bits of the flags register. |
| XLAT/XLATB | Reads a byte from the lookup table. |
| XCHG | Exchanges the contents of the 16-bit or 8-bit specified register with the contents of AX register, specified register or memory locations. |
| PUSH | Pushes (sends, writes or moves) the content of a specified register or memory location(s) onto the top of the stack. |
| POP | Pops (reads) two bytes from the top of the stack and keeps them in a specified register, or memory location(s). |
| POPF | Pops (reads) two bytes from the top of the stack and keeps them in the flag register. |
| IN | Transfers data from a port to the accumulator or AX, DX or AL register. |
| OUT | Transfers data from accumulator or AL or AX register to an I/O port identified by the second byte of the instruction. |

## Arithmetic Instructions

* Instructions of this group perform addition, subtraction, multiplication, division, increment, decrement, comparison, ASCII and decimal adjustment etc.

| **instruction** | **Description** |
| --- | --- |
| ADD | Adds data to the accumulator i.e. AL or AX register or memory locations. |
| ADC | Adds specified operands and the carry status (i.e. carry of the previous stage). |
| SUB | Subtract immediate data from accumulator, memory or register. |
| SBB | Subtract immediate data with borrow from accumulator, memory or register. |
| MUL | Unsigned 8-bit or 16-bit multiplication. |
| IMUL | Signed 8-bit or 16-bit multiplication. |
| DIV | Unsigned 8-bit or 16-bit division. |
| IDIV | Signed 8-bit or 16-bit division. |
| INC | Increment Register or memory by 1. |
| DEC | Decrement register or memory by 1. |
| DAA | **Decimal Adjust after BCD Addition:** When two BCD numbers are added, the DAA is used after ADD or ADC instruction to get correct answer in BCD. |
| DAS | **Decimal Adjust after BCD Subtraction:** When two BCD numbers are added, the DAS is used after SUB or SBB instruction to get correct answer in BCD. |
| AAA | **ASCII Adjust for Addition:** When ASCII codes of two decimal digits are added, the AAA is used after addition to get correct answer in unpacked BCD. |
| AAD | **Adjust AX Register for Division:** It converts two unpacked BCD digits in AX to the equivalent binary number. This adjustment is done before dividing two unpacked BCD digits in AX by an unpacked BCD byte. |
| AAM | **Adjust result of BCD Multiplication:** This instruction is used after the multiplication of two unpacked BCD. |
| AAS | **ASCII Adjust for Subtraction:** This instruction is used to get the correct result in unpacked BCD after the subtraction of the ASCII code of a number from ASCII code another number. |
| CBW | Convert signed Byte to signed Word. |
| CWD | Convert signed Word to signed Doubleword. |
| NEG | Obtains 2's complement (i.e. negative) of the content of an 8-bit or 16-bit specified register or memory location(s). |
| CMP | Compare Immediate data, register or memory with accumulator, register or memory location(s). |

## Logical Instructions

Instruction of this group perform logical AND, OR, XOR, NOT and TEST operations. **The following instructions come under this category:**

| **Instruction** | **Description** |
| --- | --- |
| AND | Performs bit by bit logical AND operation of two operands and places the result in the specified destination. |
| OR | Performs bit by bit logical OR operation of two operands and places the result in the specified destination. |
| XOR | Performs bit by bit logical XOR operation of two operands and places the result in the specified destination. |
| NOT | Takes one's complement of the content of a specified register or memory location(s). |
| TEST | Perform logical AND operation of a specified operand with another specified operand. |

## Rotate Instructions

## The following instructions come under this category:

| **Instruction** | **Description** |
| --- | --- |
| RCL | Rotate all bits of the operand left by specified number of bits through carry flag. |
| RCR | Rotate all bits of the operand right by specified number of bits through carry flag. |
| ROL | Rotate all bits of the operand left by specified number of bits. |
| ROR | Rotate all bits of the operand right by specified number of bits. |

## Shift Instructions

**The following instructions come under this category:**

| **Instruction** | **Description** |
| --- | --- |
| SAL or SHL | Shifts each bit of operand left by specified number of bits and put zero in LSB position. |
| SAR | Shift each bit of any operand right by specified number of bits. Copy old MSB into new MSB. |
| SHR | Shift each bit of operand right by specified number of bits and put zero in MSB position. |

## Branch Instructions

It is also called program execution transfer instruction. Instructions of this group transfer program execution from the normal sequence of instructions to the specified destination or target. The following instructions come under this category:

| **Instruction** | **Description** |
| --- | --- |
| JA or JNBE | Jump if above, not below, or equal i.e. when CF and ZF = 0 |
| JAE/JNB/JNC | Jump if above, not below, equal or no carry i.e. when CF = 0 |
| JB/JNAE/JC | Jump if below, not above, equal or carry i.e. when CF = 0 |
| JBE/JNA | Jump if below, not above, or equal i.e. when CF and ZF = 1 |
| JCXZ | Jump if CX register = 0 |
| JE/JZ | Jump if zero or equal i.e. when ZF = 1 |
| JG/JNLE | Jump if greater, not less or equal i.e. when ZF = 0 and CF = OF |
| JGE/JNL | Jump if greater, not less or equal i.e. when SF = OF |
| JL/JNGE | Jump if less, not greater than or equal i.e. when SF ≠ OF |
| JLE/JNG | Jump if less, equal or not greater i.e. when ZF = 1 and SF ≠ OF |
| JMP | Causes the program execution to jump unconditionally to the memory address or label given in the instruction. |
| CALL | Calls a procedure whose address is given in the instruction and saves their return address to the stack. |
| RET | Returns program execution from a procedure (subroutine) to the next instruction or main program. |
| IRET | Returns program execution from an interrupt service procedure (subroutine) to the main program. |
| INT | Used to generate software interrupt at the desired point in a program. |
| INTO | Software interrupts to indicate overflow after arithmetic operation. |
| LOOP | Jump to defined label until CX = 0. |
| LOOPZ/LOOPE | Decrement CX register and jump if CX ≠ 0 and ZF = 1. |
| LOOPNZ/LOOPNE | Decrement CX register and jump if CX ≠ 0 and ZF = 0. |

Here, CF = Carry Flag

ZF = Zero Flag  
OF = Overflow Flag SF = Sign Flag  
CX = Register

## Flag Manipulation and Processor Control Instructions.

## The following instructions come under this category:

| **Instruction** | **Description** |
| --- | --- |
| CLC | **Clear Carry Flag:** This instruction resets the carry flag CF to 0. |
| CLD | **Clear Direction Flag:** This instruction resets the direction flag DF to 0. |
| CLI | **Clear Interrupt Flag:** This instruction resets the interrupt flag IF to 0. |
| CMC | This instruction take complement of carry flag CF. |
| STC | Set carry flag CF to 1. |
| STD | Set direction flag to 1. |
| STI | Set interrupt flag IF to 1. |
| HLT | Halt processing. It stops program execution. |
| NOP | Performs no operation. |
| ESC | **Escape:** makes bus free for external master like a coprocessor or peripheral device. |
| WAIT | When WAIT instruction is executed, the processor enters an idle state in which the processor does no processing. |
| LOCK | It is a prefix instruction. It makes the LOCK pin low till the execution of the next instruction. |

## String Instructions

* String is series of bytes or series of words stored in sequential memory locations. The 8086 provides some instructions which handle string operations such as string movement, comparison, scan, load and store.

| **Instruction** | **Description** |
| --- | --- |
| MOVS/MOVSB/MOVSW | Moves 8-bit or 16-bit data from the memory location(s) addressed by SI register to the memory location addressed by DI register. |
| CMPS/CMPSB/CMPSW | Compares the content of memory location addressed by DI register with the content of memory location addressed by SI register. |
| SCAS/SCASB/SCASW | Compares the content of accumulator with the content of memory location addressed by DI register in the extra segment ES. |
| LODS/LODSB/LODSW | Loads 8-bit or 16-bit data from memory location addressed by SI register into AL or AX register. |
| STOS/STOSB/STOSW | Stores 8-bit or 16-bit data from AL or AX register in the memory location addressed by DI register. |
| REP | Repeats the given instruction until CX ≠ 0 |
| REPE/ REPZ | Repeats the given instruction till CX ≠ 0 and ZF = 1 |
| REPNE/REPNZ | Repeats the given instruction till CX ≠ 0 and ZF = 0 |

## **Assembler directives**

**Introduction:**

Assembler directives are the directions to the assembler   which indicate how an operand or section of the program is to be processed. These are also called pseudo operations which are not executable by the microprocessor. The following section explains the basic assembler directives for 8086.

**ASSEMBLER DIRECTIVES:**

The various directives are explained below.

**1. ASSUME**: The ASSUME directive is used to inform the assembler the name of the logical segment it should use for a specified segment.

Ex:  ASSUME   DS: DATA tells the assembler that for any program instruction which refers to the data segment ,it should use the logical segment called DATA.

**2.DB  -**Define byte. It is used to declare a byte variable or set aside one or more storage locations of type byte in memory.

For example, CURRENT\_VALUE DB 36H tells the assembler to reserve   1 byte of memory for a variable named CURRENT\_ VALUE and to put the value   36 H in that memory location when the program is loaded into RAM .

**3. DW -Define word.**It tells the assembler to define a variable of type word or to reserve storage locations of type word in memory.

**4**. **DD(define double word**) :This directive is used to declare a variable of type double word or restore memory locations which can be accessed as type double word.

**5.DQ (define quadword) :**This directive is used to tell the assembler to declare a variable  4 words in length  or to reserve 4 words of storage in memory .

**6.DT (define ten bytes):**It is used to inform the assembler to define a variable which is **10** bytes in length or to reserve 10 bytes of storage  in memory.

**7.** **EQU –Equate**It is used to give a name   to some value or symbol**.**Every time the assembler finds the given name in the program, it will replace the name with the value or symbol we have equated with that name

**8.ORG**   -**Originate**  : The ORG statement changes the starting offset address of the data.

It allows to set the location counter to a desired value at any point in the program.For example the statement ORG   3000H  tells the assembler to set the location counter to 3000H.

**9** **.PROC**- Procedure: It is used to identify the start of a procedure.  Or  subroutine.

**10. END**- End program .This directive indicates the assembler that this is the end of the program module.The assembler ignores any statements after an END directive.

**11**. **ENDP**-   End procedure: It indicates the end of the procedure (subroutine) to the assembler.

**12.ENDS**-End Segment: This directive is used with the name of the segment to indicate the end of that logical segment.

Ex: CODE  SEGMENT : Start of logical segment containing code

       CODE ENDS           : End of the segment named CODE.

**Assembly Language Programming**

Assembly language is a low-level programming language for a computer or other programmable device specific to a particular computer architecture in contrast to most high-level programming languages, which are generally portable across multiple systems. Assembly language is converted into executable machine code by a utility program referred to as an assembler like NASM, MASM, etc.

Advantages:

An understanding of assembly language provides knowledge of:

*  Interface of programs with OS, processor and BIOS;
*  Representation of data in memory and other external devices;
*  How processor accesses and executes instruction;
*  How instructions accesses and process data;
*  How a program access 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.

**Introduction to MASM**

MASM: (Microsoft assembler)

MASM COMMANDS:

Structure of Program:

.model tiny/small/medium/large

.Stack &lt;some number&gt;

.data

; Initialize data

; which is used in program.

.code

; Program logic goes here.

;

end

**AIM: 1. Write assembly language programs to evaluate the expressions:**

i) a = b + c – d \* e

ii) z = x \* y + w – v +u / k

a. Considering 8-bit, 16 bit and 32 bit binary numbers as b, c, d, e.

b. Considering 2 digit, 4 digit and 8 digit BCD numbers.

Take the input in consecutive memory locations and also Display the results by using “int xx” of

8086. Validate program for the boundary conditions.

EQUIPMENT REQUIRED: Personal Computer, O.S: WIN-XP

SOFTWARE: Macro assembler (ESA/MASM) Version: 5.0

PROGRAM:

i) a=b+c-d\*e

assume cs:code, ds:data

data segment

b db 02

c db 04

d db 01

e db 03

a db 01 dup()

data ends

code segment

start:mov ax,data

mov ds,ax

mov al,b

mov bl,c

add al,bl

mov cl,d

sub al,cl

mov dl,e

mul dl

mov a,al

int 03

code ends

end start

**RESULT**:AX=000F

ii) Z=X\*Y+W-V+U/K

assume cs:code,ds:data

data segment

x db 02

y db 04

w db 09

v db 03

u dw 0006

k db 03

z db 01 dup()

data ends

code segment

start:mov ax,data

mov ds,ax

mov al,x

mov bl,y

mul bl

mov cl,w

add al,cl

mov dl,v

sub al,dl

mov cl,al

mov ax,u

mov bl,k

div bl

add cl,al

mov z,cl

int 03

code ends

end start

**RESULT**:CL=0010