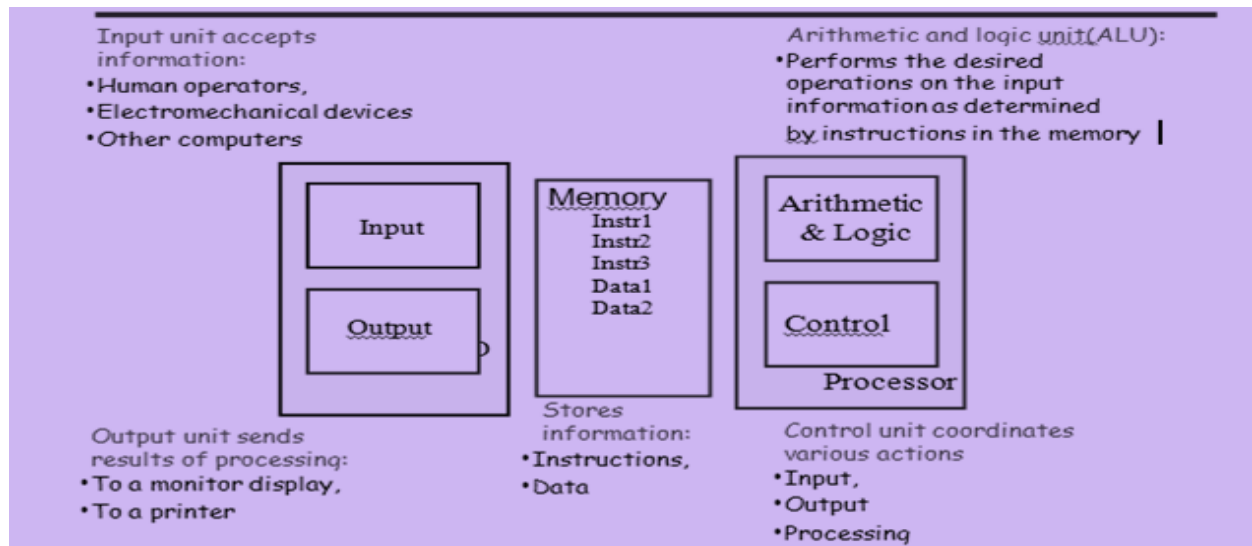


FUNCTIONAL UNITS OF COMPUTER

A computer consists of five functionally independent main parts: input, memory, arithmetic and logic, output, and control units.



The input unit accepts coded information from human operators using devices such as keyboards, or from other computers over digital communication lines. The information received is stored in the computer's memory, either for later use or to be processed immediately by the arithmetic and logic unit. The processing steps are specified by a program that is also stored in the memory. Finally, the results are sent back to the outside world through the output unit. All of these actions are coordinated by the control unit. An interconnection network provides the means for the functional units to exchange information and coordinate their actions. The arithmetic and logic circuits, in conjunction with the main control circuits, is the processor. Input and output equipment is often collectively referred to as the input-output (I/O) unit.

A program is a list of instructions which performs a task. Programs are stored in the memory. The processor fetches the program instructions from the memory, one after another, and performs the desired operations. The computer is controlled by the stored program, except for possible external interruption by an operator or by I/O devices connected to it. Data are numbers and characters that are used as operands by the instructions. Data are also stored in the memory. The instructions and data handled by a computer must be encoded in a suitable format. Each instruction, number, or character is encoded as a string of binary digits called bits, each having one of two possible values, 0 or 1, represented by the two stable states.

Input Unit

Computers accept coded information through input units. The most common input device is the keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted to the processor.

Many other kinds of input devices for human-computer interaction are available, including the touchpad, mouse, joystick, and trackball. These are often used as graphic input devices in conjunction with displays.

Microphones can be used to capture audio input which is then sampled and converted into digital codes for storage and processing.

Similarly, cameras can be used to capture video input.

Digital communication facilities, such as the Internet, can also provide input to a computer from other computers and database servers.

Memory Unit

The function of the memory unit is to store programs and data. There are two classes of storage, called primary and secondary.

Primary Memory

Primary memory, also called main memory, is a fast memory that operates at electronic speeds. Programs must be stored in this memory while they are being executed. The memory consists of a large number of semiconductor storage cells, each capable of storing one bit of information. These cells are rarely read or written individually.

Instead, they are handled in groups of fixed size called words. The memory is organized so that one word can be stored or retrieved in one basic operation. The number of bits in each word is referred to as the word length of the computer, typically 16, 32, or 64 bits.

To provide easy access to any word in the memory, a distinct address is associated with each word location. Addresses are consecutive numbers, starting from 0, that identify successive locations.

Instructions and data can be written into or read from the memory under the control of the processor. A memory in which any location can be accessed in a short and fixed amount of time after specifying its address is called a random-access memory (RAM). The time required to access one word is called the memory access time. This time is independent of the location of

the word being accessed. It typically ranges from a few nanoseconds (ns) to about 100 ns for current RAM units

Cache Memory

As an adjunct to the main memory, a smaller, faster RAM unit, called a cache, is used to hold sections of a program that are currently being executed, along with any associated data. The cache is tightly coupled with the processor and is usually contained on the same integrated-circuit chip. The purpose of the cache is to facilitate high instruction execution rates.

At the start of program execution, the cache is empty. As execution proceeds, instructions are fetched into the processor chip, and a copy of each is placed in the cache. When the execution of an instruction requires data, located in the main memory, the data are fetched and copies are also placed in the cache. If these instructions are available in the cache, they can be fetched quickly during the period of repeated use.

Secondary Storage

Although primary memory is essential, it tends to be expensive and does not retain information when power is turned off. Thus additional, less expensive, permanent secondary storage is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently. Access times for secondary storage are longer than for primary memory. The devices available are including magnetic disks, optical disks (DVD and CD), and flash memory devices.

Arithmetic and Logic Unit

Most computer operations are executed in the arithmetic and logic unit (ALU) of the processor. Any arithmetic or logic operation, such as addition, subtraction, multiplication division, or comparison of numbers, is initiated by bringing the required operands into the processor, where the operation is performed by the ALU.

When operands are brought into the processor, they are stored in high-speed storage elements called registers. Each register can store one word of data. Access times to registers are even shorter than access times to the cache unit on the processor chip.

Output Unit

Output unit function is to send processed results to the outside world. A familiar example of such a device is a printer. Most printers employ either photocopying techniques, as in laser printers, or ink jet streams. Such printers may generate output at speeds of 20 or more pages per minute. However, printers are mechanical devices, and as such are quite slow compared to the electronic speed of a processor.

Some units, such as graphic displays, provide both an output function, showing text and graphics, and an input function, through touchscreen capability. The dual role of such units is the reason for using the single name input/output (I/O) unit in many cases.

Control Unit

The memory, arithmetic and logic, and I/O units store and process information and perform input and output operations. The operation of these units must be coordinated in some way. This is the responsibility of the control unit. The control unit is effectively the nerve center that sends control signals to other units and senses their states.

I/O transfers, consisting of input and output operations, are controlled by program instructions that identify the devices involved and the information to be transferred.

Control circuits are responsible for generating the timing signals that govern the transfers. They determine when a given action is to take place. Data transfers between the processor and the memory are also managed by the control unit through timing signals. A large set of control lines (wires) carries the signals used for timing and synchronization of events in all units.

The operation of a computer can be summarized as follows:

- The computer accepts information in the form of programs and data through an input unit and stores it in the memory.
- Information stored in the memory is fetched under program control into an arithmetic and logic unit, where it is processed.
- Processed information leaves the computer through an output unit.
- All activities in the computer are directed by the control unit.

BASIC OPERATIONAL CONCEPTS The activity in a computer is governed by instructions. To perform a given task, an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be used as instruction operands are also stored in the memory. A typical instruction might be Load LOC,R2

This instruction reads the contents of a memory location whose address is represented symbolically by the label LOC and loads them into processor register R2. The original contents of location LOC are preserved, whereas those of register R2 are overwritten.

Execution of this instruction requires several steps. First, the instruction is fetched from the memory into the processor. Next, the operation to be performed is determined by the control unit. The operand at LOC is then fetched from the memory into the processor. Finally, the operand is stored in register R2. After operands have been loaded from memory into processor registers, arithmetic or logic operations can be performed on them. For example, the instruction

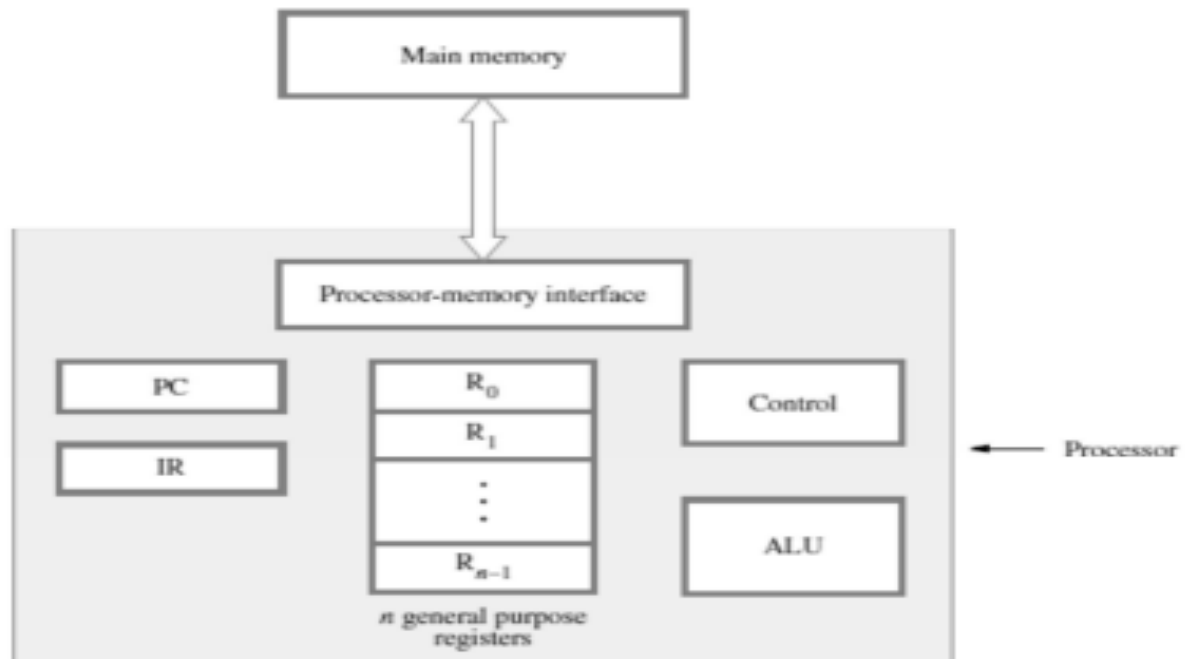
Add R2, R3,R4

adds the contents of registers R2 and R3, then places their sum into register R4. The operands in R2 and R3 are not altered, but the previous value in R4 is overwritten by the sum. After completing the desired operations, the results are in processor registers. They can be transferred to the memory using instructions such as

Store R4, LOC

This instruction copies the operand in register R4 to memory location LOC. The original contents of location LOC are overwritten, but those of R4 are preserved.

For **Load and Store instructions**, transfers between the memory and the processor are initiated by sending the address of the desired memory location to the memory unit and asserting the appropriate control signals. The data are then transferred to or from the memory. Figure shows how the memory and the processor can be connected. It also shows some components of the processor that have not been discussed yet. The interconnections between these components are not shown explicitly since we will only discuss their functional characteristics.

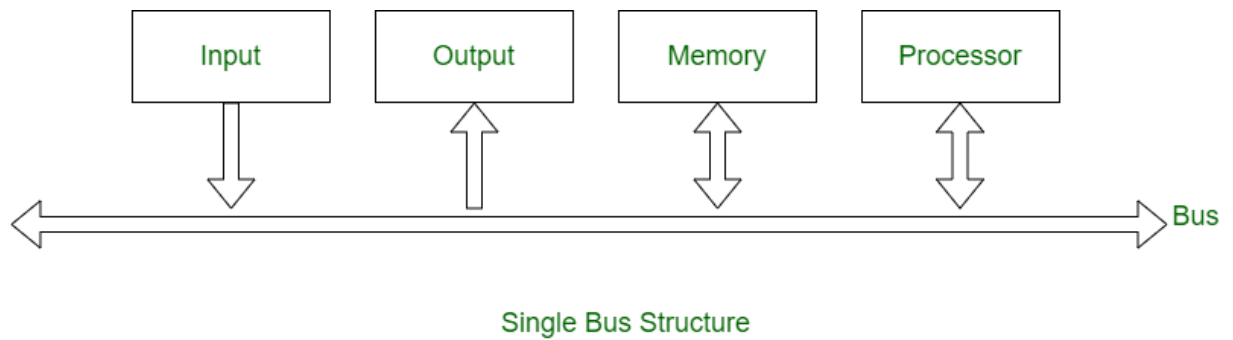


In addition to the ALU and the control circuitry, the processor contains a number of registers used for several different purposes. **The instruction register (IR) holds the instruction that is currently being executed.** Its output is available to the control circuits, which generate the timing signals that control the various processing elements involved in executing the instruction. The program counter (PC) is another specialized register. **It contains the memory address of the next instruction to be fetched and executed.** During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed. **It is customary to say that the PC points to the next instruction that is to be fetched from the memory.** In addition to the IR and PC, Figure shows general-purpose registers R_0 through R_{n-1} , often called processor registers. They serve a variety of functions, including holding operands that have been loaded from the memory for processing. **The processor-memory interface is a circuit which manages the transfer of data between the main memory and the processor. If a word is to be read from the memory, the interface sends the address of that word to the memory along with a Read control signal.** The interface waits for the word to be retrieved, then transfers it to the appropriate processor register. **If a word is to be written into memory, the interface transfers both the address and the word to the memory along with a Write control signal. A program must be in the main memory in order for it to be executed.** It is often transferred there from secondary storage through the input unit. Execution of the program begins when the PC is set to point to the first instruction of the program. The contents of the PC are transferred to the memory along with a Read control signal. When the addressed word (in this case, the first instruction of the program) has been

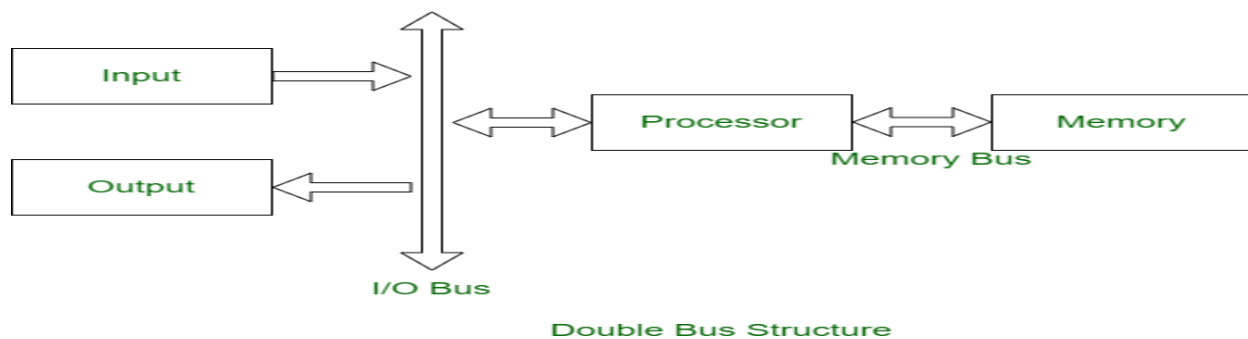
fetched from the memory it is loaded into register IR. At this point, the instruction is ready to be interpreted and executed. Instructions such as Load, Store, and Add perform data transfer and arithmetic operations. If an operand that resides in the memory is required for an instruction, it is fetched by sending its address to the memory and initiating a Read operation. When the operand has been fetched from the memory, it is transferred to a processor register. After operands have been fetched in this way, the ALU can perform a desired arithmetic operation, such as Add, on the values in processor registers. The result is sent to a processor register. If the result is to be written into the memory with a Store instruction, it is transferred from the processor register to the memory, along with the address of the location where the result is to be stored, then a Write operation is initiated. At some point during the execution of each instruction, the contents of the PC are incremented so that the PC points to the next instruction to be executed. Thus, as soon as the execution of the current instruction is completed, the processor is ready to fetch a new instruction. In addition to transferring data between the memory and the processor, the computer accepts data from input devices and sends data to output devices. Thus, some machine instructions are provided for the purpose of handling I/O transfers. Normal execution of a program may be preempted if some device requires urgent service. For example, a monitoring device in a computer-controlled industrial process may detect a dangerous condition. In order to respond immediately, execution of the current program must be suspended. To cause this, the device raises an interrupt signal, which is a request for service by the processor. The processor provides the requested service by executing a program called an interrupt-service routine. Because such diversions may alter the internal state of the processor, its state must be saved in the memory before servicing the interrupt request. Normally, the information that is saved includes the contents of the PC, the contents of the general-purpose registers, and some control information. When the interrupt-service routine is completed, the state of the processor is restored from the memory so that the interrupted program may continue.

SINGLE BUS STRUCTURES

1. **Single Bus Structure:** In a single bus structure, one common bus is used to communicate between peripherals and microprocessors. It has disadvantages due to the use of one common bus.



- 2. Double Bus Structure:** In a double bus structure, one bus is used to fetch instructions while other is used to fetch data, required for execution. It is to overcome the bottleneck of a single bus structure.




For a computer to achieve its operation, the functional units need to communicate with each other.

- In order to communicate, they need to be connected. Memory Input Output Processor
- Functional units may be connected by a group of parallel wires.
- The group of parallel wires is called a bus.
- Each wire in a bus can transfer one bit of information.
- The number of parallel wires in a bus is equal to the word length of a computer.

Bus Structures: Speed Issue

Different devices have different transfer/operate speed.

If the speed of bus is bounded by the slowest device connected to it, the efficiency will be very low.  **How to solve this?**

A common approach – use buffer registers.

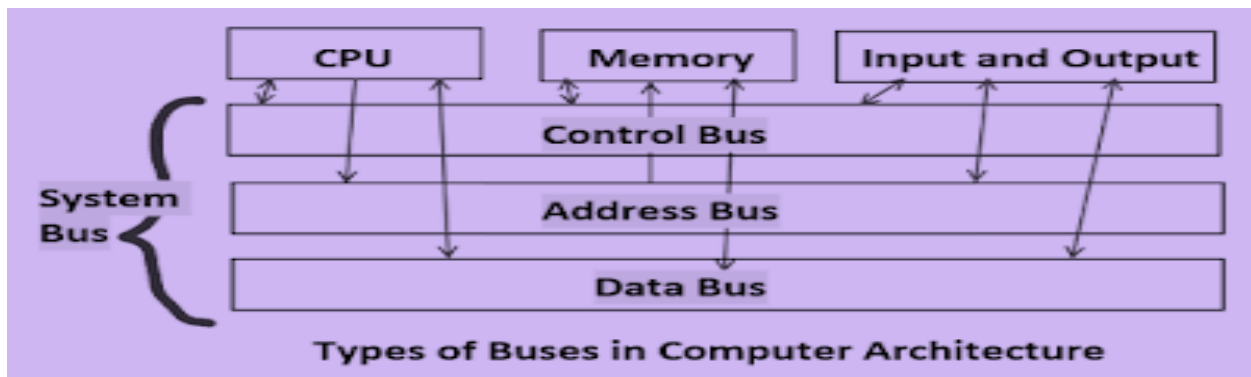
Differences between Single Bus and Double Bus Structure :

S. No.	Single Bus Structure	Double Bus Structure
1.	The same bus is shared by three units (Memory, Processor, and I/O units).	The two independent buses link various units together.
2.	One common bus is used for communication between peripherals and processors.	Two buses are used, one for communication from peripherals and the other for the processor.
3.	The same memory address space is utilized by I/O units.	Here, the I/O bus is used to connect I/O units and processor and other one, memory bus is used to connect memory and processor.
4.	Instructions and data both are transferred in same bus.	Instructions and data both are transferred in different buses.
5.	Its performance is low.	Its performance is high.
6.	The cost of a single bus structure is low.	The cost of a double bus structure is high.
7.	Number of cycles for execution is more.	Number of cycles for execution is less.
8.	Execution of the process is slow.	Execution of the process is fast.
9.	Number of registers associated are less.	Number of registers associated are more.
10.	At a time single operand can be read from the bus.	At a time two operands can be read.

S. No.	Single Bus Structure	Double Bus Structure
11.	Advantages- <ul style="list-style-type: none"> • Less expensive • Simplicity 	Advantages- <ul style="list-style-type: none"> • Better performance • Improves Efficiency

3 buses Structure: 3 Buses are used:

- Address bus
- Data bus
- Control bus



Address bus - carries memory addresses from the processor to other components such as primary storage and input/output devices. The address bus is **unidirectional**.

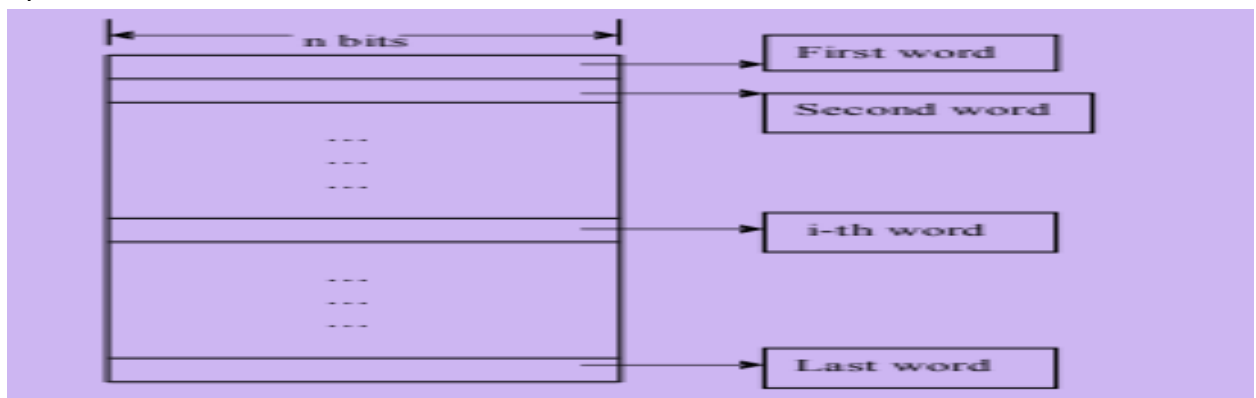
Data bus - carries the data between the processor and other components. The data bus is **bidirectional**.

Control bus - carries control signals from the processor to other components. The control bus also carries the clock's pulses. The control bus is unidirectional. Example memory read, memory write, IO read, IO write.

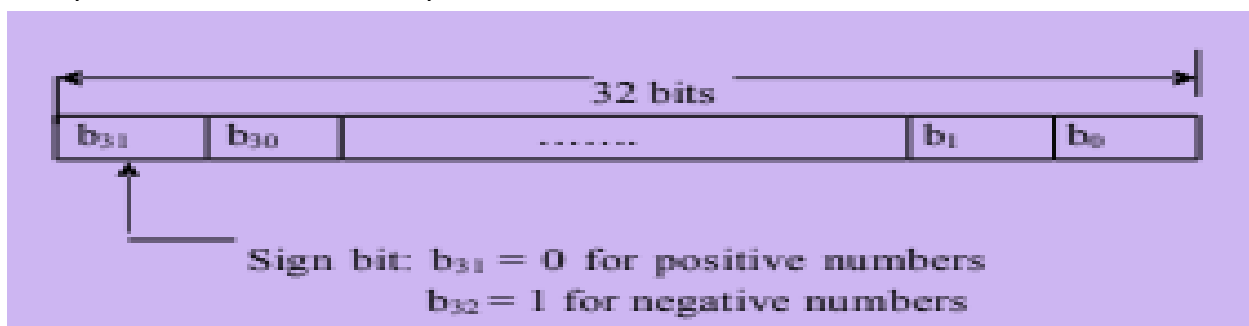
Memory Addresses and Operations

- The memory consists of many millions of storage cells, each of which can store a bit of information having the value 0 or 1.

- ▶ Data is accessed in n bit unit called “word” .
- ▶ K Bit address creates address space of 2^K from 0 through $2^K - 1$
- ▶ Example :A 32-bit address creates an address space of 2^{32} or 4G (4 giga) locations.
- ▶ It is impractical to assign addresses to each bit location of memory so we can use concept of byte addressability.
- ▶ **Byte Addressability** : assign address to each byte of word .
- ▶ successive words are located at addresses 0,4,8,...., if each word consisting of four bytes.



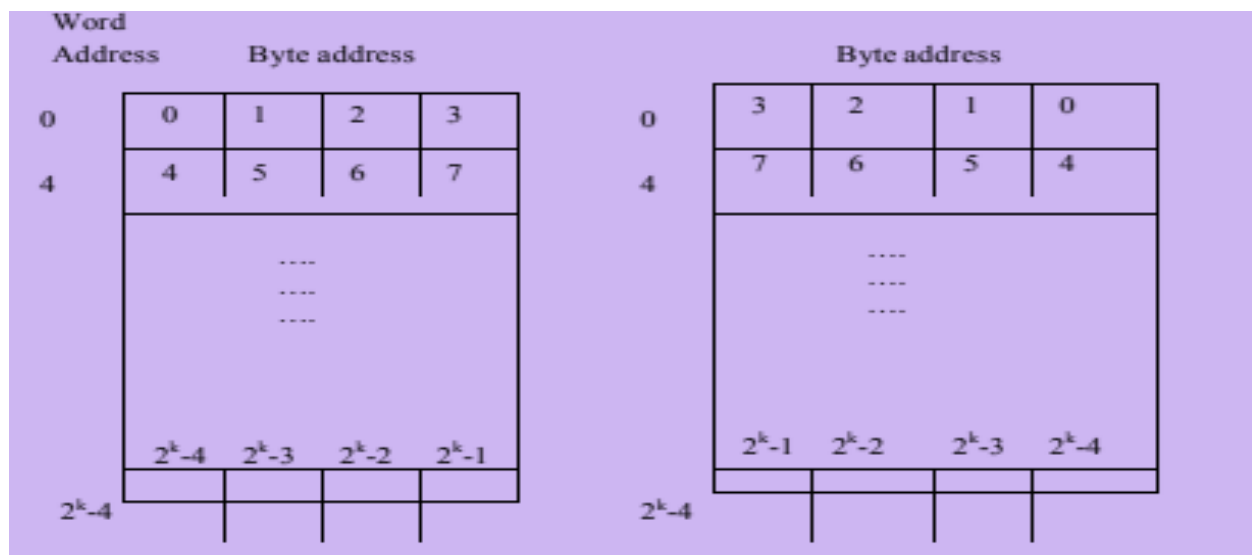
Ways to store word in memory.



8 bits	8 bits	8 bits	8 bits
ASCII	ASCII	ASCII	ASCII
Character	Character	character	character

BIG-ENDIAN AND LITTLE-ENDIAN ASSIGNMENTS:-

- 2 ways that byte addresses can be assigned across words:
- Big-Endian: Lower byte addresses are used for the more significant bytes (the leftmost bytes) of the word.
- Little-Endian: Higher byte addresses are used for the more significant bytes (the leftmost bytes) of the word.



Word Alignment: Words are said to be aligned if they begin at a byte address that is a multiple of the number of bytes present in the word.

Example 4 byte word = 0, 4, 8, 12,

2 byte word = 0, 2, 4, 6,

A number can be accessed through its word address.

Character by their byte address.

String can be accessed by byte address of starting character. End of string can be recognized by either by End of string character or by checking length of string.

Memory Operations: To execute an instruction, the processor control circuits must cause the word (or words) containing the instruction to be transferred from the memory to the processor.

▶ **Load** (or Read or Fetch):

▶ **Store** (or Write):

Assembly Language

- ▶ Machine understands language of 0 and 1.
- ▶ We can't write code in 0 and 1 form rather we use normal words like move ,add, increment etc.
- ▶ When we write programs for a specific computer, the normal words need to be replaced by acronyms called mnemonics.
 - E.g., MOV, ADD, INC
 - A complete set of symbolic names and rules for their use constitute a programming language, referred to as the **assembly language**.
 - Assembler:
 - Assembly language code -> machine language Code
 - Each mnemonic represents the binary pattern, or OP code for the operation performed by the instruction.
 - Assembler converts mnemonic to the binary op code.
 - Addressing mode :
 - ADD #5,R3 Or ADDI 5,(# or I with ADD to denote Immediate addressing mode)
 - MOVE #5,(R2) or MOVEI 5,(R2) (Indirect Addressing Mode)

Assembly Language Directives: The statements which provide additional information to the assembler to translate source program into an object program are called assembler directives or commands.

- Assembly language allows programmer to specify other information necessary to translate the source program into an object program.

- How to assign numerical values to the names.
- Where to place instructions in the memory.
- Where to place data operands in the memory.

NUMBER NOTATION:

- Most assemblers allow numerical values to be specified in different ways, using conventions that are defined by the assembly language syntax.
- Example: Consider decimal number 93
- **Decimal Number Representation**
- `ADD #93, R1`
- **Binary Number Representation**
- It identified by a prefix symbol such as a percent sign
- `ADD %#01011101, R1`
- **Hexadecimal Representation**
- In assembly language, a hex representation is often identified by a dollar sign prefix. Example
- `ADD #$5D, R1`

INSTRUCTION AND INSTRUCTION SEQUENCING:

- ▶ A computer have instructions to perform four types of operations.
 - Data transfers between the memory and the processor registers
 - Arithmetic and logic operations on data
 - Program sequencing and control
 - I/O transfers

REGISTER TRANSFER NOTATION:- Identify location by a symbolic name standing for its binary address.

- ▶ **Example** LOC,R0,DATAIN etc.

- The contents of a location are denoted by placing square brackets around the name of the location.

$R1 \leftarrow [LOC]$

Assembly Language Notation: This Notation is use to represent machine language program. Assembly language code is symbolic machine code directly communicate with computer resources like registers.

Assembly Language Notation Format

[label] opcode <operands> OR <addresses> ;comments

Represent machine instructions and programs.

Move LOC, R1 $R1 \leftarrow [LOC]$

Add R1, R2, R3 $R3 \leftarrow [R1] + [R2]$

Examples of different types of instructions in assembly language notation.

- Data transfers between processor and memory.

Move A, B (B = A).

Move A, R1 (R1 = A).

- Arithmetic and logic operation:

Add A, B, C (C = A + B)

- Sequencing:

Jump Label (Jump to the subroutine which starts at
Label).

- Input/output data transfer:

Input PORT, R5 (Read from i/o port "PORT" to register R5).

INSTRUCTION TYPES:

Instructions can be classified based on the number of operand addresses they include.

- ▶ **3-address instructions** are almost always instructions that implement binary operations.

Format : operation S1,S2,D

Add A, B, C

- ▶ **2-address instructions** one operand serves as a source and destination: Format:
operation S1,D

Add A, B

- ▶ **1-address instructions** require only one operand.

Load A

load content of accumulator to Location A

- ▶ **0-address instructions** do not operate on operands.

PUSH A

INSTRUCTION EXECUTION AND STRAIGHT LINE SEQUENCING:

$C \leftarrow [A] + [B]$

Instruction Execution phases:

- ▶ **Instruction fetch**

Fetch the instruction from memory location whose address is present in PC , Place Instruction in IR

- ▶ **Instruction Execute**

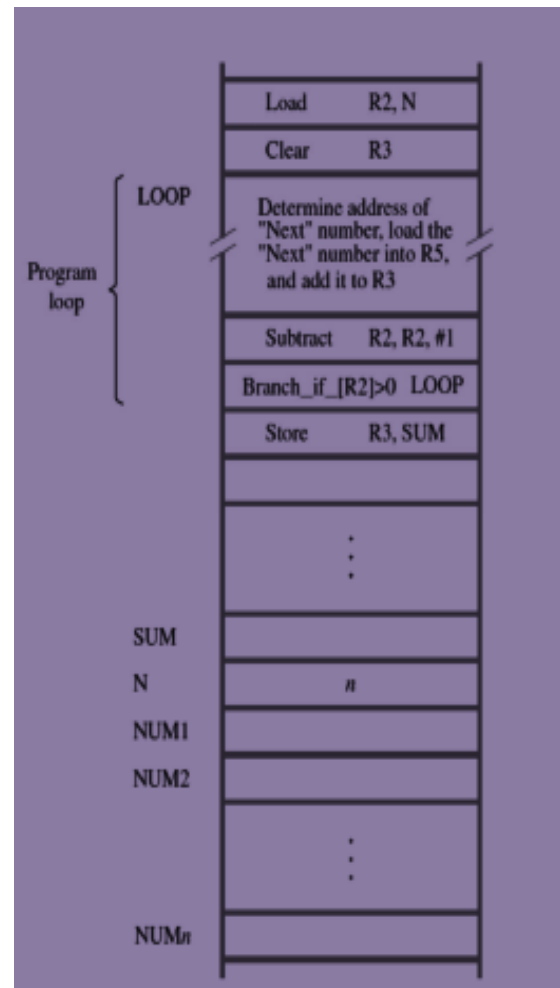
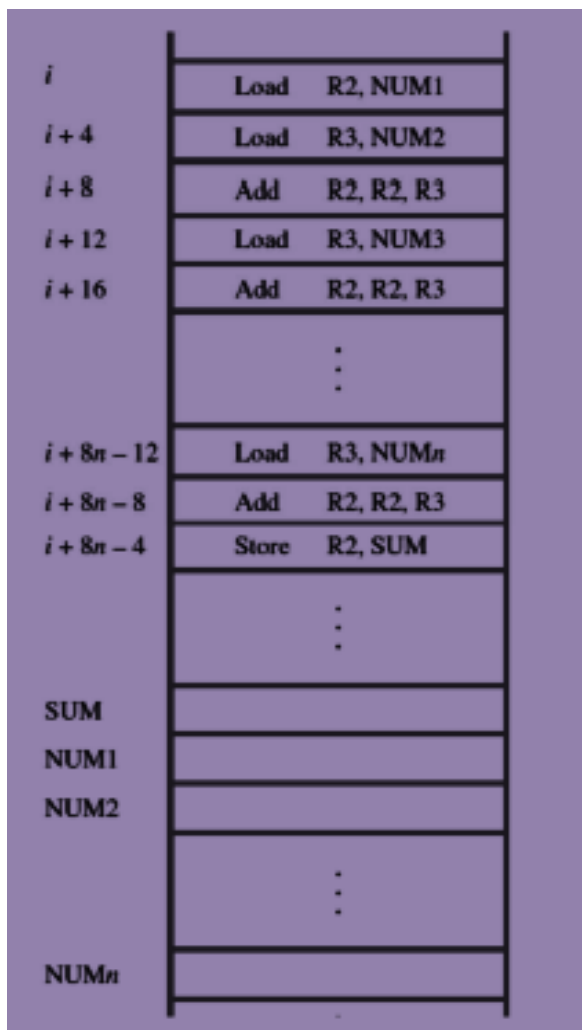
Examine IR ,decode opcode and CPU execute operation.(fetch operand, perform operations and store result).

Branching :

- ▶ **Task :** To add n numbers

- ▶ **Solution 1:**One add instruction to add each number. And finally store sum in memory.

- Solution 2: Put add instruction in a loop and put loop exit condition as last statement of loop.



Condition Codes: The processor keeps track of information about the result of various operations. This information is stored in individual bits called condition code flags.

- Flags are together stored in status register (also called condition code register).
- N(negative)
- Z (zero)
- V (overflow)
- C (carry)

EXAMPLE

► A: 1111 0000

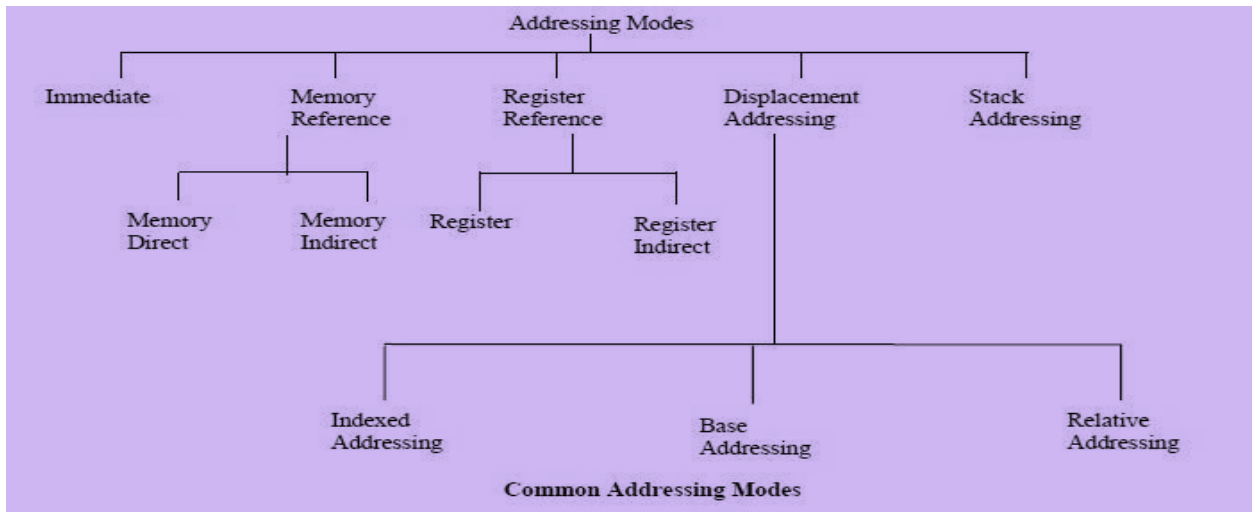
► B: 00010100

► Perform A-B

	1	1	1	1	0	0	0	0
	1	1	1	0	1	1	0	0
1	1	1	0	1	1	1	0	0

► Here C=1 V=0 Z=0 N=0

ADDRESSING MODES— The term addressing modes refers to the way in which the operand of an instruction is specified.



Implied mode:: In implied addressing the operand is specified in the instruction itself.

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

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

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

Register mode: The data is in the register that is specified by the instruction.

Register Indirect mode: The effective address of the data is in the base register or an index register that is specified by the instruction.

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)+**

Example:

Add R1, (R2)+ // OR

$R1 = R1 + M[R2]$

$R2 = R2 + d$

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

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)**

Example:

Add R1, -(R2) //OR

$R2 = R2 - d$

$R1 = R1 + M[R2]$

Direct Address mode: Effective address is equal to address part of instruction.

Indirect Address mode: In this mode the address field of the instruction gives the address where the effective address is stored in memory.

Relative address mode: Effective address is obtained by adding the content of program counter to address part of instruction to get effective address.

$X(pc) \rightarrow EA = [pc] + X$

Indexed address mode: Effective address is obtained by adding the content of index register to address part of instruction to get effective address.

Base Register Addressing mode:

$EA = \text{Content of Base register} + \text{address part of Instruction.}$

8086 Micro Processor Case Study

Enhanced version of 8085 Microprocessor .

16 bit microprocessor .

20 address lines

16 data lines .

Powerful instruction set

2 mode of operation: Maximum mode and Minimum mode

versions:

8086 → 5MHz

8086-2 → 8MHz

8086-1 → 10 MHz

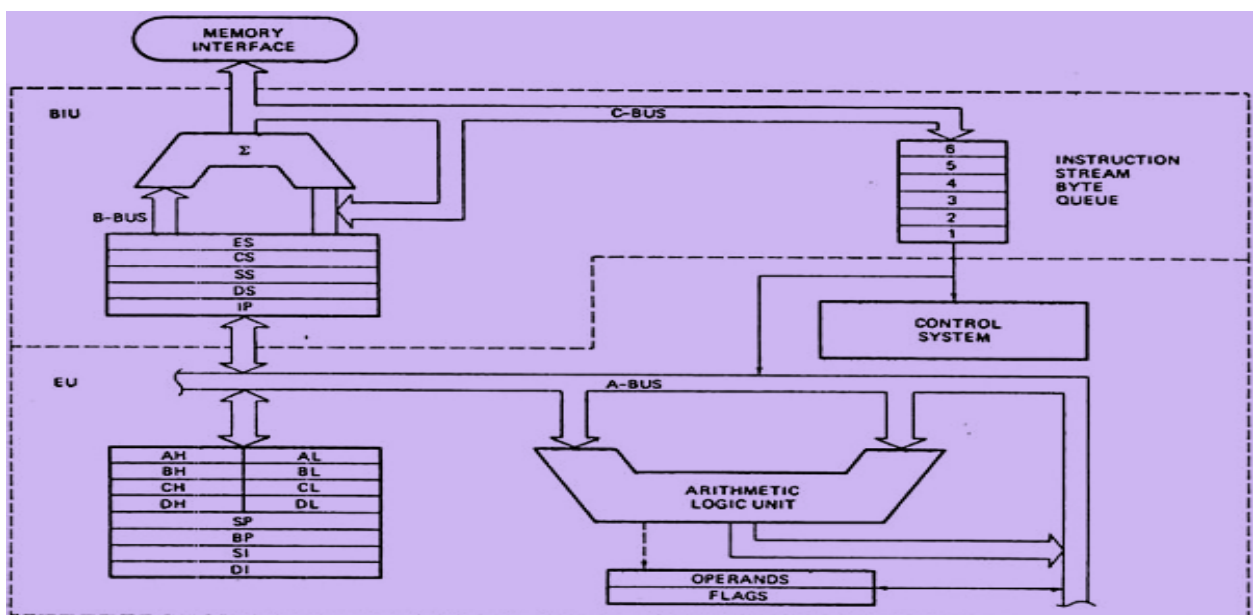
2 stage Pipeline 1. Instruction fetch and 2 Instruction Execute.

256 vectored interrupts.

29,000 transistors.

2 functional units are present:

- **1. EU(Execution Unit):** Execution unit gives instructions to BIU stating from where to fetch the data and then decode and execute those instructions.



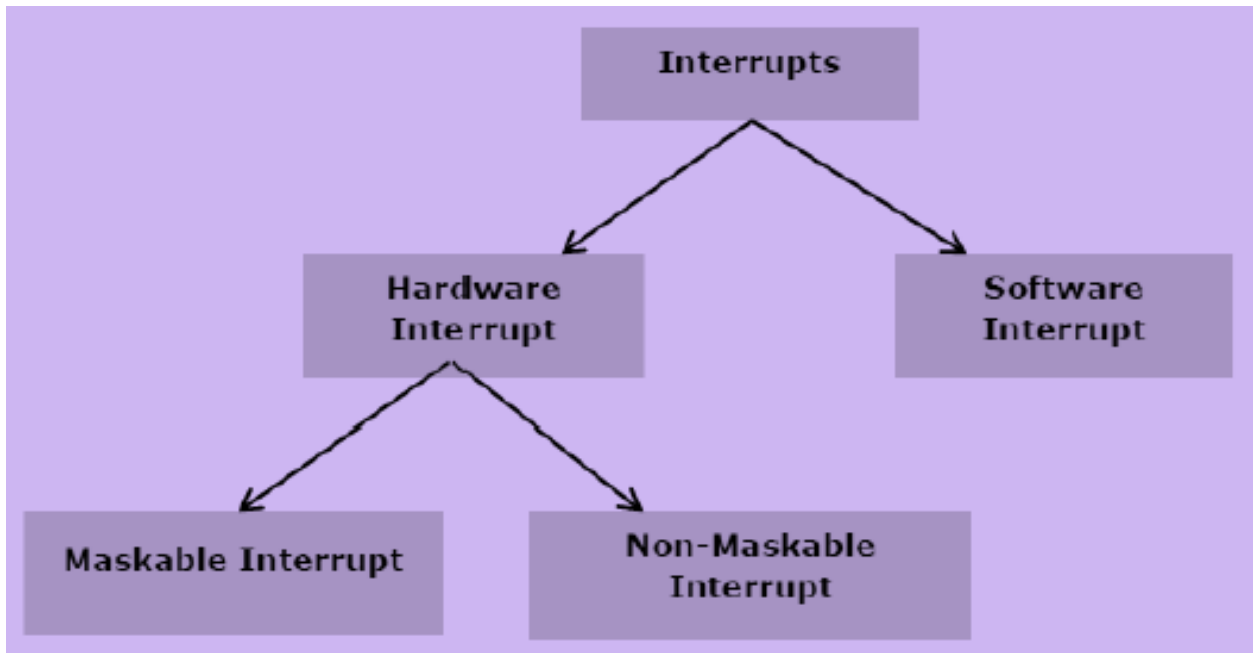
- **ALU:** It handles all arithmetic and logical operations, like +, -, ×, /, OR, AND, NOT operations.
- **Flag Register**

- ▶ 16-bit register. it changes its status according to the result stored in the accumulator.
- ▶ Conditional Flags: It represents the result of the last arithmetic or logical instruction executed. Ex carry flag, zero flag, sign flag etc
- ▶ Control Flags: Controls the operation of EU. Trap flag, Interrupt flag etc.
- ▶ General purpose register : There are 8 (8 bit) general purpose registers, i.e., AH, AL, BH, BL, CH, CL, DH, and DL.
- ▶ Stack Pointer register: It is a 16-bit register, which holds the address from the start of the segment to the memory location, where a word was most recently stored on the stack.

2.BIU (Bus Interface Unit):BIU takes care of all data and addresses transfers on the buses for the EU like sending addresses, fetching instructions from the memory, reading data from the ports and the memory as well as writing data to the ports and the memory.

- Instruction queue – BIU contains the instruction queue. BIU gets upto 6 bytes of next instructions and stores them in the instruction queue.
- Segment register – BIU has 4 segment buses, i.e. CS, DS, SS& ES. It holds the addresses of instructions and data in memory, which are used by the processor to access memory locations. It also contains 1 pointer register IP.
- Instruction pointer(IP): Holds the address of the next instruction to be executed by the EU.
- The 8086 microprocessor supports 8 types of instructions –
- Data Transfer Instructions :MOV, PUSH, POP, IN, OUT etc.
- Arithmetic Instructions :ADD, INC, SBB, MUL, DIV etc.
- Bit Manipulation Instructions :NOT, AND, OR, ROL, SHR, SHL etc.
- String Instructions :REP, OUTS, MOVS etc.
- Program Execution Transfer Instructions (Branch & Loop Instructions) :CALL, RET, JMP .
- Processor Control Instructions :STC, CLC, STD etc.
- Iteration Control Instructions :LOOP, JCXZ etc.
- Interrupt Instructions :INT, INTO, IRET etc.

Microprocessor 8086 Interrupts:



Hardware interrupts

- ▶ Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.
- ▶ The 8086 has two hardware pins: NMI and INTR.
- ▶ NMI is a non-maskable interrupt.
- ▶ INTR is a maskable interrupt having lower priority.
- ▶ One more pin, INTA, is called interrupt acknowledge.

Software Interrupts

- ▶ Some instructions are inserted at the desired position into the program to create interrupts.
- ▶ **INT-Interrupt Instruction with type number:**
- ▶ 2 byte instruction: First Byte provides opcode:
Second Byte provides Interrupt Number.

Type 0 interrupt is 000000H (divide by zero)

Type 1 interrupt is 000004H (so on)

8086 Addressing Modes:

- ▶ **Based indexed with displacement mode:** operand offset=base register content +index register content+8 Or 16 bit displacement.

MOV AX,[BX+DI+08]

- ▶ **Based addressing mode:** content offset =sum of base register and 8 or 16 bit displacement.

ADD CL,[BX+08]

- ▶ **Immediate addressing mode:** operand is a part of instruction.
- ▶ **Based-index addressing mode:** offset address of operand = base register + index register. MOV AL,[BP+SI]

- ▶ **Indexed mode:** operand offset address =index register content+displacement.

MOV AX, [SI+2000]

- ▶ **Direct addressing mode:** EA is present in instruction itself.
- ▶ **Register addressing mode:** Address field specifies register which stores content.
- ▶ **Register Indirect addressing mode:** register's content is the memory location where operand address is stored.