Chapter-1- **Introduction to Computers and Number Systems**

1.1. **Concepts of Computer Organization and Architecture**

Computer Organization refers to the operational units and their interconnections that realize the architectural specifications. The organization of computer systems refers to how the components and connections can be made. There can be many possibilities for organization to realize similar architecture. For instance, Intel x86(80286, 80386, 80486), Pentium, and Pentium Pro all have the same architecture with different organizations. And also IBM PowerPC (601, 603, etc.) have the same architecture, but different organizations.

Examples of organization concerns are things that are transparent to the programmer:

- control signals
- interfaces between computer and peripherals
- the memory technology being used

Computer Architecture refers to those attributes of a system that have a direct impact on the logical execution of a program. It is really concerned with interfacing issues between hardware and software.

Architecture concerns include:

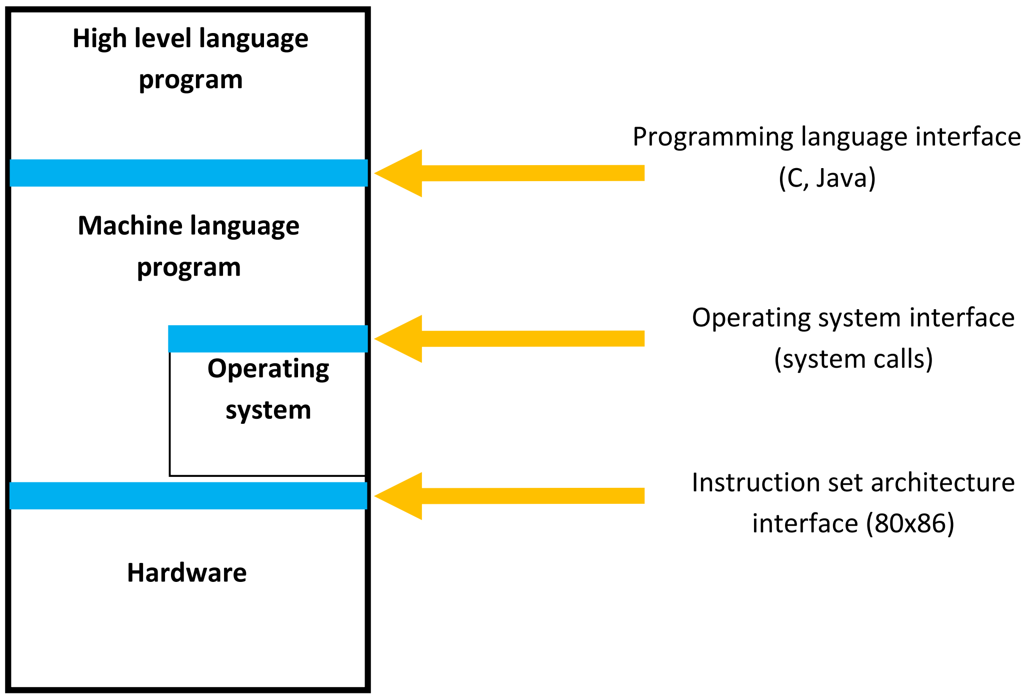
- the instruction set, i.e. which instructions does the machine understand
- how do the instructions need to be formatted into bits, i.e. the number of bits used to represent various data types
- how big is the memory
- how many registers are available: for instance x86 have 4 or 8 general purpose registers.
- I/O mechanisms
- memory addressing techniques, i.e. how to access memory

As an example, the fact that a multiplication instruction is available in a computer system is an architecture issue; whereas, how that multiplication is implemented is a computer organization issue.

Based on how complex the number of instructions a machine supports, computers systems can be categorized as RISC (reduced instruction set computer) and CISC (complex instruction set computers).

CISC have larger and more complex instructions set; whereas, RISC have smaller and simpler instruction set.

The following diagram illustrates more on how an architectural difference among machines can be created:


High level language program
Machine language program
Operating system
Hardware
Programming language interface (C, Java)
Operating system interface (system calls)
Instruction set architecture interface (80x86)

Instruction set architecture (ISA), or simply architecture, of a computer refers to the abstraction of the hardware/ software interface.

The instruction set architecture includes anything programmers need to know to make a binary machine language program work correctly, including instructions, I/O devices, and so on. Typically, the operating system will encapsulate the details of doing I/O, allocating memory, and other low-level system functions so that application programmers do not need to worry about such details.

1.2. Evolution of Computers

Although computer Professionals do not agree on exact dates or specifics, computer developments are often categorized by generations. A completely new kind of machine was developed in each generation. The major characteristics that distinguish these generations are:

- Dominant type of electronic circuit elements used.
- Major secondary storage media used
- Computer language used.
- Types or characteristic of operating system used.
- Memory access time (time to store or retrieve data from memory).

The improvements made in each generation over the previous one resulted in faster, smaller, more reliable, and cheaper computers.

First Generation Computers (1945~56GC)

- Used vacuum tubes as components for the electronic circuit –resulted in very large sized machine.
- Punched cads were the main source of inputs, and magnetic drums were used for internal storage.
- Operate in a speed of milliseconds (thousand of a second) and could handle more than 10,000 additions each second.
- Most applications were scientific calculations.

Second Generation Computers (1957~62GC)

- Characterized by transistors-solid state device made from silicon which is smaller, cheaper, faster, dissipate less energy and more reliable than vacuum tube but work in the same way with the vacuum tube, invented by Bell Labs.
- Magnetic tapes (similar with home tape cassette), used for main storage,
- Operate in microseconds (millionths of a second) with more than 200,000 additions possible each second.

- Business applications become more commonplace, with large data files stored on magnetic tape and disk.
- High-level languages COBOL and FORTRAN were introduced during this period. Batch operating systems are used that permitted rapid processing of magnetic tape files.

Third Generation Computers (1963~72GC)

- Characterized by solid-state logic and integrated circuit (IC), which consists of thousands of small circuits (such as transistors), etched on a silicon chip.
- Computer storage switched from magnetic cores to integrated circuit boards that provide modularity (expandable storage) and compatibility (interchangeable equipment)
- New input/output methods such as optical scanning and plotters.
- Software become more important with sophisticated operating systems, improved programming languages,

Fourth Generation Computers (1973 GC ~Now)

- Greatly expanded storage capabilities and improved circuitry.
- Has large-scale integrated circuit (LSI), Very large scale integration (VLSI), and ultra large scale integration (ULSI), which has several hundred thousand transistors placed on one tiny silicon chip.
- Computer memory operates at speeds of nano-seconds (billionths of a second) with large computers capable of adding 15million numbers per second.
- Hence, this generation mainly focused on hardware technologies.

Fifth Generation Computers (Future)

Major advances in hardware component technology signaled each of the above four generations. It is believed that the fifth generation will arrive when breakthroughs in software make it possible to produce truly intelligent machines.

An architecture, which makes use of the changes in technology and allows a simple and natural methodology for solving problems, is being sought. These computers will also be able to interact with them in natural languages such as English. Japans are working intensively on the project for developing the fifth generation.

Fifth generation computers are characterized by advancements in software developments.

They are expected to communicate with human being in natural (human) language. And they are also expected to reason like intelligent human being. These goals can be achieved by developing sophisticated software.

1.3. **Evolution Of The Intel X86 Architecture**

Intel x86 architecture has evolved over the years. From a 29, 000 transistors microprocessor to 820 million transistors, the organization and technology has changed dramatically.

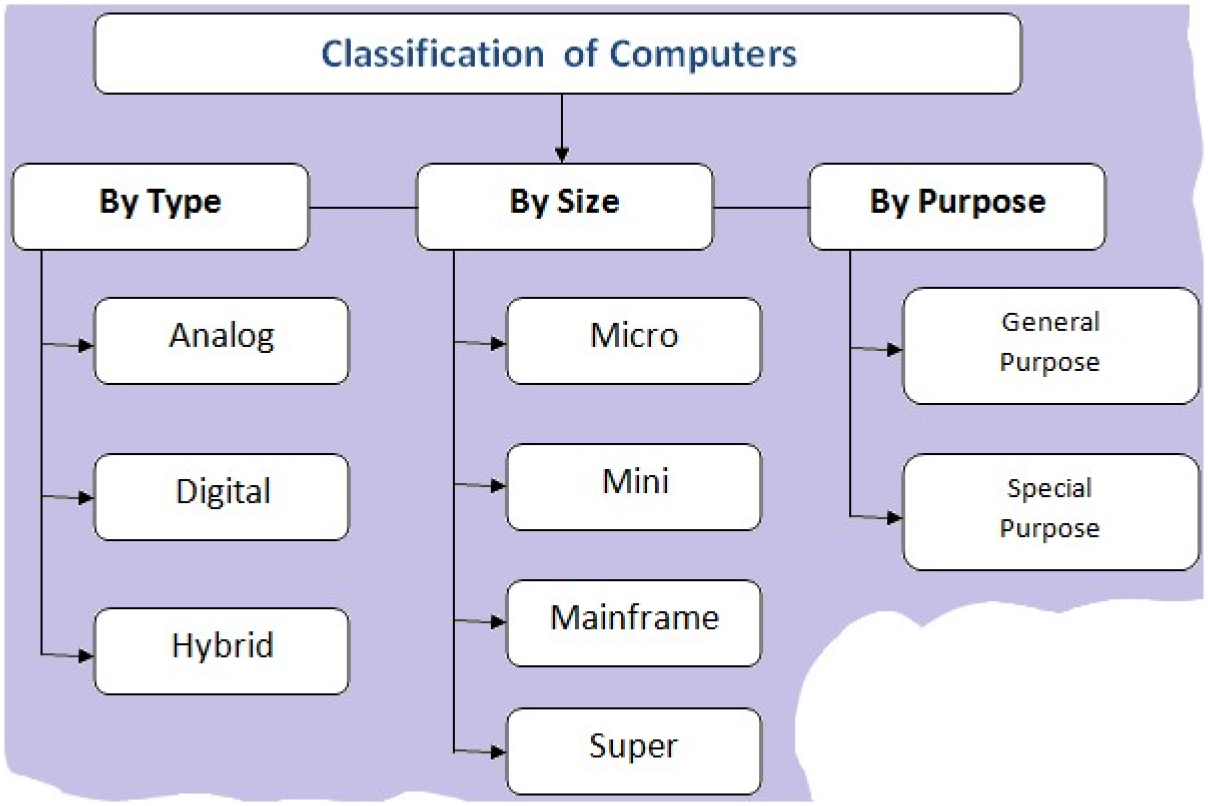
Some of the highlights of evolution of x86 architecture are:

- 8080 – It was the world's first general purpose microprocessor. It was an 8-bit machine, with an 8-bit data path to memory. It was used in the first personal computer.
- 8086 – It was a 16-bit machine and was far more powerful than previous one. It had a wider data path of 16-bits and larger registers along with an instruction cache or queue that pre-fetches a few instructions before they are executed. It is the first appearance of 8086 architecture. It has a real mode and an addressable memory of 1 MB.
- 80286 – It has an addressable memory of 16 MB instead of just 1 MB and contains two modes-real mode and first generation 16-bit protected mode. It has a data transfer width of 16-bits and programming model of 16-bits (16-bits general purpose registers and 16-bit addressing).
- 80386 – It was Intel's first 32-bit machine. Due to its 32-bit architecture it was able to compete against the complexity and power of microcomputers and mainframes introduced just a few years earlier. It was the first processor to support multitasking and contained the 32-bit protected mode. It also implemented the concept of paging (permitted 32-bit virtual memory address to be translated into 32-bit physical memory address). It has an addressable physical memory of 4 GB and data transfer width of 32-bits.

- 80486 – It introduced the concept of cache technology and instruction pipelining. It contained write protect feature and offered a built in math co-processor that offloaded complex math operations from the main CPU.
- Pentium – The use of superscalar techniques was introduced as multiple instructions started executing in parallel. The page size extension (PSE) feature was added as a minor enhancement in paging.
- Pentium Pro – It used register renaming, branch prediction, data flow analysis, speculative execution and more pipeline stages. Advanced optimization techniques in microcode were also added along with level 2 cache. It implemented the second generation address translation in which 32-bit virtual address is translated into 36-bit physical memory address.
- Pentium II – It was able to process video, audio and graphics data efficiently by incorporating Intel MMX technology (multimedia data set).
- Pentium III – It supports 3D graphics software. It has a maximum CPU clock rate of 1.4 GHz and contained 70 new instructions.
- Pentium 4 – It implements third generation address translation that translates 48-bit virtual memory address to 48-bit physical memory address. It contains other floating point enhancements for multimedia.
- Core – It is the first Intel microprocessor with dual core that is the implementation of 2 processors on a single chip. There is an addition of Visualizing Technology.
- Core 2 – It extends the architecture to 64-bits and core 2 Quad provides four processors on a single chip. The register set as well as addressing modes are of 64-bits.

1.4. **Current Classification of Computers**

Computers can be classified based on their size, purpose, or type. The following picture summarizes this:



A microcomputer is a small, relatively inexpensive computer with a microprocessor as its central processing unit (CPU). It includes a microprocessor, memory and minimal input/output (I/O) circuitry mounted on a single printed circuit board (PCB).

Microcomputers are sometimes called personal computers. These include the following list of computers:

- Desktop computers
- Laptops and notebook computers
- Tablet computers
- smart-phones etc

A minicomputer, or mini, is a class of smaller general purpose computers that developed in the mid-1960s. The class formed a distinct group with its own software architectures and operating systems. Minis were designed for control, instrumentation, human interaction, and communication switching as distinct from calculation and record keeping.

A mainframe computer is a computer used primarily by large organizations for critical applications like bulk data processing for tasks such as censuses, industry and consumer statistics, enterprise resource planning, and large-scale transaction processing. A mainframe computer is large but not as large as a supercomputer and has more processing power than some other classes of computers, such as minicomputers, servers, workstations, and personal computers. Mainframe computers are often used as servers.

A supercomputer is focused on performing tasks involving intense numerical calculations such as weather forecasting and complex scientific computations. A supercomputer is a computer that is at the front-line of current processing capacity, particularly speed of calculation.

Analog computer is used to measure the physical quantities, temperature, pressure weight and height and gives the result in the form of digits.

Analog computers are used by civil engineers, mechanical engineers and electrical engineers.

Digital computers are used to calculate digits. Today digital computers are being used in everywhere. Smart Phone and Laptop are example of digital computer.

Hybrid computer combine both analog and digital computer properties. Hybrid computer is used in medical field. Smart watch is an example of Hybrid Computer.

Computers can also be classified as special purpose and general purpose.

Special purpose computers are made to successfully meet the requirement of particular task or application. They incorporate the instruction needed into the design of internal storage. So they can perform the given task on simple commands.

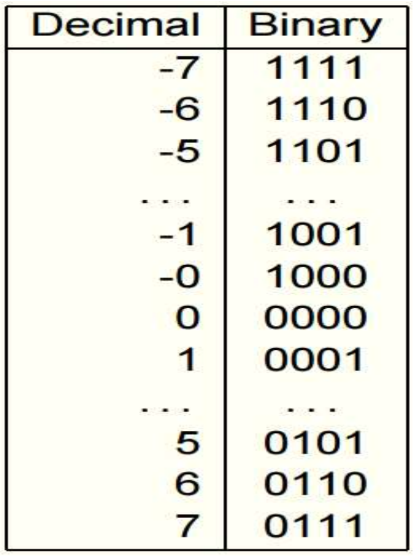
Automatic Teller Machine (ATM), Washing machines, Surveillance equipment, Traffic-control computers, and Oil-exploration systems are some examples of special purpose computers.

General purpose computers are made to process several types of tasks. All microcomputers are examples of general purpose computers.

1.5. **Number Systems**

Numbers are kept in computer hardware as a series of high and low electronic signals, and so they are considered base 2 numbers. (Just as base 10 numbers are called decimal numbers, base 2 numbers are called binary numbers.) A single digit of a binary number is thus the "atom" of computing, since all information is composed of binary digits or bits. This fundamental building block can be one of two values, which can be thought of as several alternatives: high or low, on or off, true or false, or 1 or 0.

Hardware can be designed to add, subtract, multiply, and divide these binary bit patterns. If the number that is the proper result of such operations cannot be represented by these rightmost hardware bits, overflow is said to have occurred. It's up to the programming language, the operating system, and the program to determine what to do if overflow occurs. Computer programs calculate both positive and negative numbers, so we need a representation that distinguishes the positive from the negative. The most obvious solution is to add a separate sign, which conveniently can be represented in a single bit; the name for this representation is sign and magnitude. Following is an example:



Decimal	Binary
-7	1111
-6	1110
-5	1101
. . .	. . .
-1	1001
-0	1000
0	0000
1	0001
. . .	. . .
5	0101
6	0110
7	0111

Computer use several types of number systems. These include decimal, binary, octal, and hexadecimal number systems.

Decimal Number System:

This system is a base ten number system. It uses the digits from 0 to 9. Every decimal number is formed using these digits.

Binary Number System:

This system is a base two number system. It uses the digits 0 and 1. Every binary number is formed using these two digits.

Octal Number System:

This system is a base eight number system. It uses the digits 0 to 7. Every octal number is formed using these eight digits.

Hexadecimal Number System:

This system is a base sixteen number system. It uses the digits 0 to 9 and A to F. Every hexadecimal number is formed using these sixteen digits.

1.6. Converting Between Binary and Decimal Numbers

It is a simple matter to convert a number from binary notation to decimal notation.

All that is required is to multiply each binary digit by the appropriate power of 2 and add the results; by the same analogy, you can convert octal/hexadecimal to decimal numbers by making the power of 8/16, respectively, and adding the results.

To convert from decimal to binary, octal, or hexadecimal, recursively divide the decimal number by 2/8/16, and put the quotients from right to left in each step.

1.7. Integer Number Representation

For purposes of computer storage and processing we do not have the benefit of minus signs and periods. Only binary digits (0 and 1) may be used to represent numbers. If we are limited to nonnegative integers, the representation is straightforward.

There are several techniques to represent numbers including sign magnitude, two’s complement, floating-point etc.

Sign-Magnitude Representation

There are several alternative conventions used to represent negative as well as positive integers, all of which involve treating the most significant (leftmost) bit in the word as a sign bit. If the sign bit is 0, the number is positive; if the sign bit is 1, the number is negative. The simplest form of representation that employs a sign bit is the sign-magnitude representation. In an n-bit word, the rightmost bits hold the magnitude of the integer.

There are several drawbacks to sign-magnitude representation. One is that addition and subtraction require a consideration of both the signs of the numbers and their relative magnitudes to carry out the required operation.

Another drawback is that there are two representations of 0: **`+0 = (00000000)`** and **`-0 = (10000000)`**.

Because of these drawbacks, sign-magnitude representation is rarely used in implementing the integer portion of the ALU. Instead, the most common scheme is twos complement representation.

Two’s Complement Representation

Two’s complement is used to represent binary integers. A positive integer is represented as in sign magnitude. A negative number is represented by taking the Boolean complement of each bit of the corresponding positive number, then adding 1 to the resulting bit pattern viewed as an unsigned integer.

Two’s complement representation uses the most significant bit as a sign bit, making it easy to test whether an integer is positive or negative. It differs from the use of the sign-magnitude representation in the way that the other bits are interpreted.

1.8. Floating Point Numbers

Going beyond signed and unsigned integers, programming languages support numbers with fractions, which are called real in mathematics. Here are some examples of real numbers:

3.14159265

2.71828

0.000000001

$3.15576_{ten} \times 10^9$

$1.0_{ten} \times 10^{-9}$

Computer arithmetic that supports such numbers is called floating point because it represents numbers in which the binary point is not fixed, as it is for integers.

A designer of a floating-point representation must find a compromise between the size of the fraction and the size of the exponent, because a fixed word size means you must take a bit from one to add a bit to the other. This tradeoff is between precision and range: increasing the size of the fraction enhances the precision of the fraction, while increasing the size of the exponent increases the range of numbers that can be represented.

Fraction is the value, generally between 0 and 1, placed in the fraction field and exponent is the value that is placed in the exponent field.

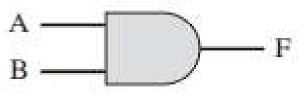
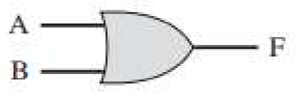
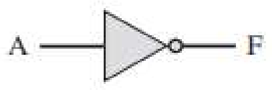
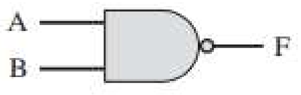
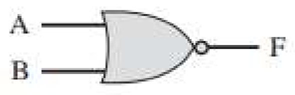
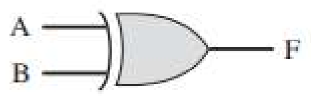
Chapter-2- **Introduction to Digital Logic Circuits**

2. 1. **Logic Gates, Symbols, and Truth Tables**

Every digital electronic component is made of circuit elements known as logic gates. The three basic logic gates are AND gate, OR gate, and NOT gate.

Additionally, using those gates other types of gates can also be constructed. These are exclusive OR gate (XOR), NOT AND (NAND), and NOT OR gate (NOR).

Each gate has one or two inputs and one output. However, all of the gates except NOT gate can have more than two inputs. Thus, (X+Y+Z) can be implemented with a single OR gate with three inputs.

Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \bullet B$ or $F = AB$	A B | F; 0 0 | 0; 0 1 | 0; 1 0 | 0; 1 1 | 1
OR		$F = A + B$	A B | F; 0 0 | 0; 0 1 | 1; 1 0 | 1; 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F; 0 | 1; 1 | 0
NAND		$F = \overline{AB}$	A B | F; 0 0 | 1; 0 1 | 1; 1 0 | 1; 1 1 | 0
NOR		$F = \overline{A + B}$	A B | F; 0 0 | 1; 0 1 | 0; 1 0 | 0; 1 1 | 0
XOR		$F = A \oplus B$	A B | F; 0 0 | 0; 0 1 | 1; 1 0 | 1; 1 1 | 0

The above is a symbolic representation, an algebraic function, and truth table of each of the logic type:

The XOR gate performs the following logic:

$$x \oplus y = \overline{x}y + x\overline{y}$$

2. 2. **Boolean Algebra**

There are different techniques to show the function of some combined digital circuits, blocks. These include truth tables and equations.

Truth tables show the values of the outputs for each possible set of input values.

For a logic block with n inputs, there are $\mathbf{2^n}$ entries in the truth table, since there are that many possible combinations of input values. Each entry specifies the value of the outputs for that particular input combination.

Another approach is to express the logic function with logic equations. This is done with the use of Boolean algebra.

The digital circuitry in digital computers and other digital systems is designed, and its behavior is analyzed, with the use of a mathematical discipline known as Boolean algebra.

Boolean algebra turns out to be a convenient tool in two areas:

- Analysis: It is an economical way of describing the function of digital circuitry.
- Design: Given a desired function, Boolean algebra can be applied to develop a simplified implementation of that function.

As with any algebra, Boolean algebra makes use of variables and operations. In this case, the variables and operations are logical variables and operations. Thus, a variable may take on the value 1 (TRUE) or 0 (FALSE). The basic logical operations are AND, OR, and NOT, which are symbolically represented by dot (.), plus sign (+), and over-bar(complement) (-), respectively.

- The OR operator is written as +, as in A + B. The result of an OR operator is 1 if either of the variables is 1. The OR operation is also called a logical sum, since its result is 1 if either operand is 1.
- The AND operator is written as **.** (*dot*) , as in A . B. The result of an AND operator is 1 only if both inputs are 1. The AND operator is also called logical product, since its result is 1 only if both operands are 1.
- The unary operator NOT is written as A'. The result of a NOT operator is 1 only if the input is 0. Applying the operator NOT to a logical value results in an inversion or negation of the value (i.e., if the input is 0 the output is 1, and vice versa).

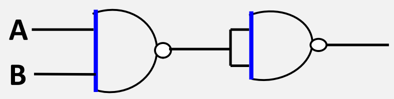
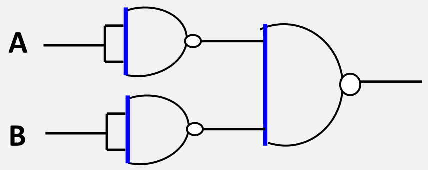
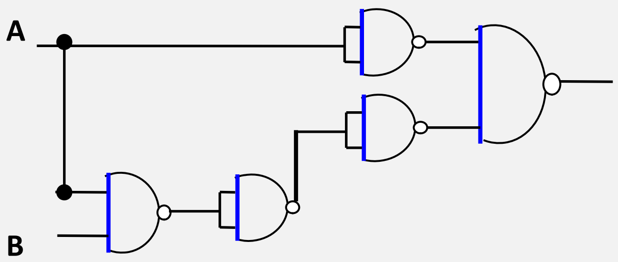
There are several laws of Boolean algebra that are helpful in manipulating logic equations.

Each of the laws stated below can be proven through the use of truth tables. While some of them are the same as the laws of ordinary algebra, some of them are not. In particular, one should note De Morgan's laws, the first distribution law, the idem-potency laws, the first domination law and the absorption laws.

1. A'' = A	Double Complement
2. (A+B)' = A'B' (AB) ' = A'+B'	De Morgan
3. A+B = B+A AB = BA	Commutation
4. (A+B) +C = A+(B+C) (AB) C = A(BC)	Association
5. A+(BC) = (A+B)(A+C) A(B+C) = (AB) + (AC)	Distribution
6. A+A = A AA = A	Idem-potency
7. A+0 = A A1 = A	Identity
8. A+A' = 1 AA' = 0	Inverse
9. A+1= 1 A0= 0	Domination
10. A+ (AB)= A A(A+B) = A	Absorption

For example, in Boolean Algebra, the first distribution law indicates that 1 + (1)(1) is equivalent to (1 + 1)(1 + 1) = (1)(1) = 1, while in ordinary algebra, 1+(2)(3) = 7 is not equivalent to (1+2)(1+3) = (3)(4) = 12.

Exercise 1. Redraw the following logic circuits after simplifying them using Boolean rule.

Original Circuit Diagram	Simplified Form
	
	
	

Exercise 2. Simplify the following Boolean expressions using Boolean rule.

$A+\bar{A}B$

$(\overline{ABC}) + (\bar{A}+\bar{C})B$

$(ABC + (\bar{A}+\bar{B}+\bar{C}))B$

$A+\bar{A}B+\bar{A}\bar{B}C$

$A\bar{B}+\bar{A}$

$\overline{A\ (A+B) + AB}$

$(A\ C) + (\overline{\bar{A}+\bar{C}})B$

$(\overline{AB+ ABC})\bar{A}+\bar{B}$

Exercise 3. According to the following truth table that shows input/output combination of different logic circuits, first try to find the Boolean expression for each output (circuit 1, circuit 2 …) and then draw the logic diagram.

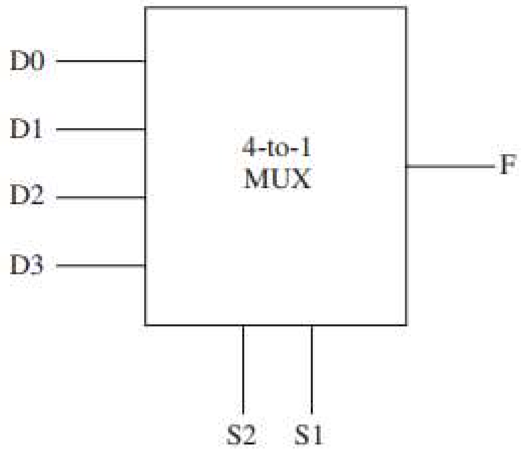
A	**B**	**C**	**Circuit 1**	**Circuit 2**	**Circuit 3**	**Circuit 4**
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	1	1
0	1	1	1	0	0	0
1	0	0	0	1	0	0
1	0	1	1	0	0	1
1	1	0	1	0	0	0
1	1	1	0	0	0	1

2. 3. Combinational Circuits

Logic blocks are categorized as one of two types, depending on whether they contain memory or not. Blocks without memory are called combinational circuit; the output of a combinational block depends only on the current input. Blocks with memory, which are known as sequential circuits, the outputs can depend on both the inputs and the value stored in memory, which is called the state of the logic block.

Multiplexers

The multiplexer connects multiple inputs to a single output. At any time, one of the inputs is selected to be passed to the output. A general block diagram representation is shown below.



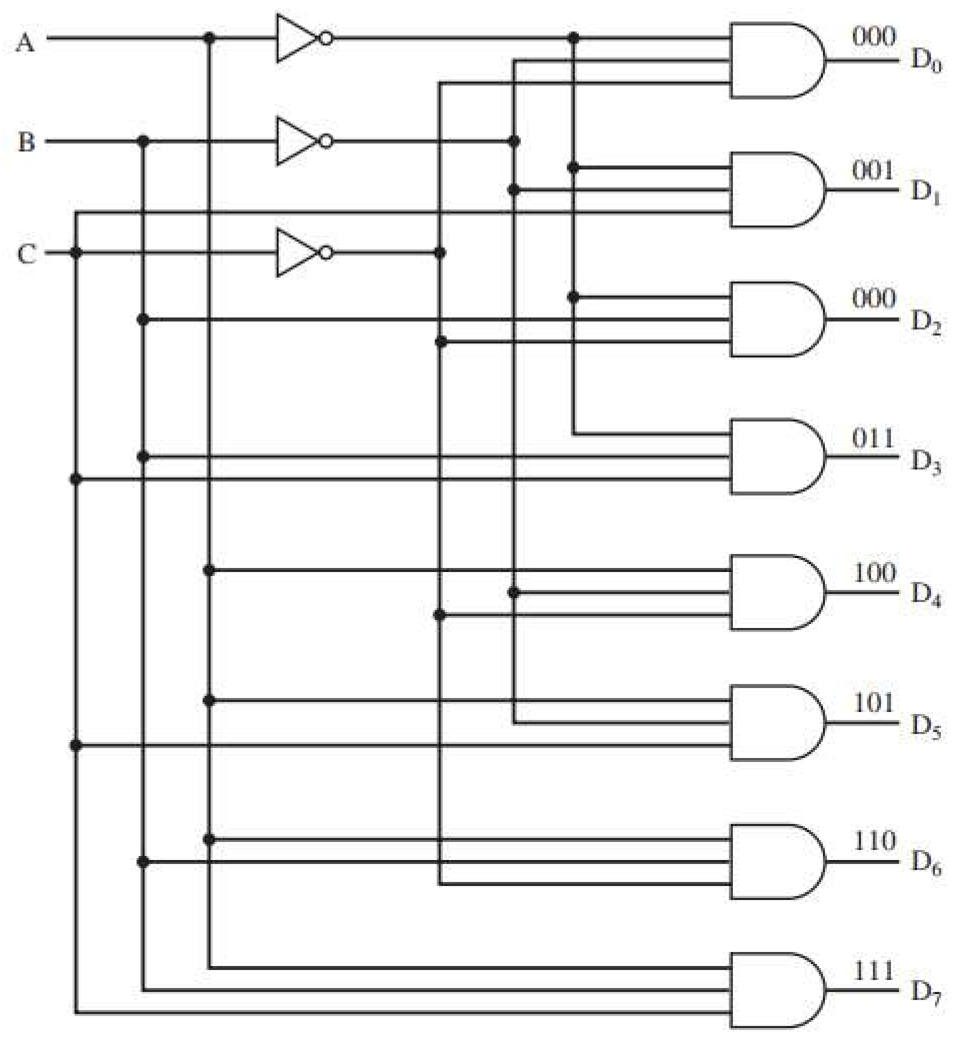
The above represents a 4-to-1 multiplexer. There are four input lines, labeled D0, D1, D2, and D3. One of these lines is selected to provide the output signal F. To select one of the four possible inputs, a 2-bit selection code is needed, and this is implemented as two select lines labeled S1 and S2.

Multiplexers are used in digital circuits to control signal and data routing. An example is the loading of the program counter (PC).

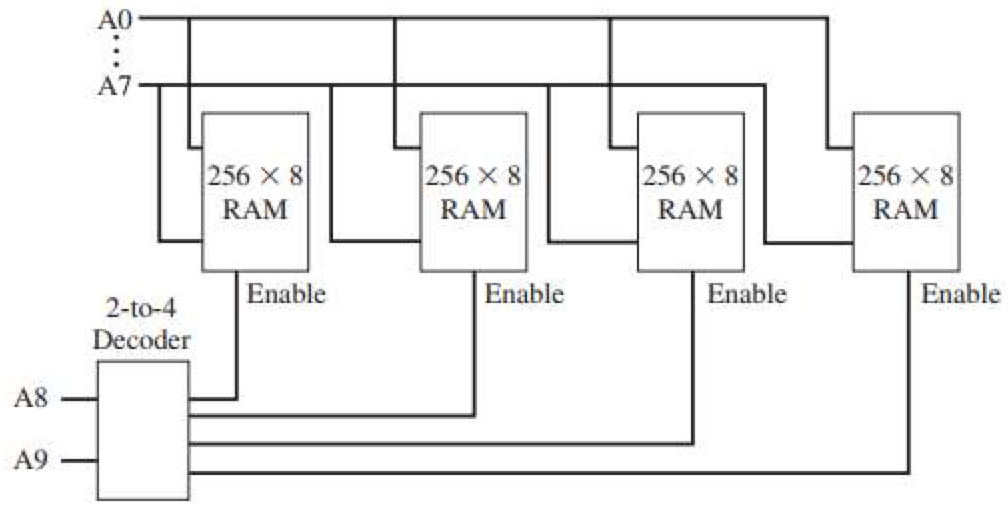
Decoders

A decoder is a combinational circuit with a number of output lines, only one of which is asserted at any time, dependent on the pattern of input lines. In general, a decoder has n inputs and 2^n outputs.

The following figure shows a decoder with three inputs and eight outputs.



And the following diagram illustrates a 2 to 4 address decoder.



Each chip requires 8 address lines, and these are supplied by the lower-order 8 bits of the address (A_0, A_1, A_2, ... A_7). The higher-order 2 bits of the 10-bit address (A_8, A_9) are used to select one of the four RAM chips. For this purpose, a 2-to-4 decoder is used whose output enables one of the four chips.

Read-Only Memory

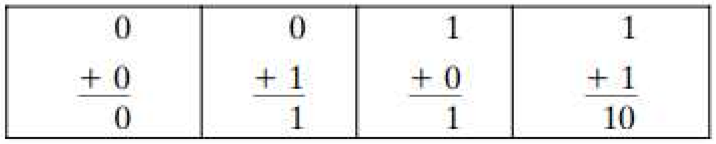
Combinational circuits are often referred to as "memoryless" circuits, because their output depends only on their current input and no history of prior inputs is retained.

However, there is one sort of memory that is implemented with combinational circuits, namely read-only memory (ROM).

ROM is a memory unit that performs only the read operation. This implies that the binary information stored in a ROM is permanent and was created during the fabrication process. Thus, a given input to the ROM (address lines) always produces the same output (data lines). Because the outputs are a function only of the present inputs, the ROM is in fact a combinational circuit.

Adders

Binary addition differs from Boolean algebra in that the result includes a carry term as in the following.



$\begin{array}{r}0\\+0\\\hline 0\end{array}$	$\begin{array}{r}0\\+1\\\hline 1\end{array}$	$\begin{array}{r}1\\+0\\\hline 1\end{array}$	$\begin{array}{r}1\\+1\\\hline 10\end{array}$

However, addition can still be dealt with in Boolean terms.

Adders are of two groups: half adder and full adder.

Half Adder:

A half-adder adds two bits and produces a sum and a carry output. Adders are important in computers and also in other types of digital systems in which numerical data are processed.

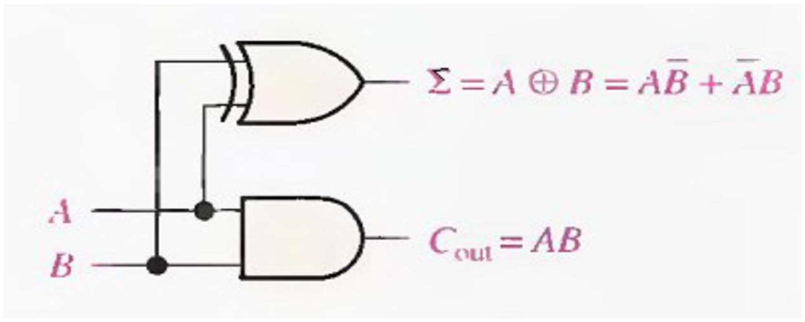
The following truth table summarizes the logic of half adder. A truth table is a canonical representation of a Boolean function.

A	**B**	**Sum (S)=A+B**	**Carry (C_o)**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the above truth table you can derive the logic expression for the two outputs, S and Co, as follows.

$$S=\overline{A}B+A\overline{B}$$
$$C_o=AB$$

And the logic diagram can be put as:

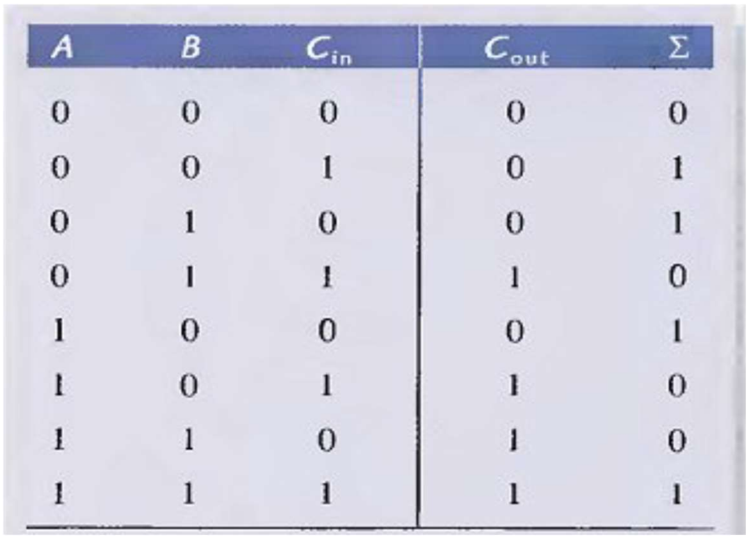


The Full Adder

The second category of adder is the full-adder. The full-adder accepts two input bits and an input carry and generates a sum output and an output carry.

The basic difference between a full-adder and a half-adder is that the full-adder accepts an input carry.

Following is a truth table for the Full adder:



A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Assignment: Using the above truth table, find the logic expressions for the two outputs (Carry and Sum) and then draw a single logic diagram for both.

2. 4. **Sequential Circuits**

Combinational circuits implement the essential functions of a digital computer. However, except for the special case of ROM, they provide no memory or state information, elements also essential to the operation of a digital computer. For the latter purposes, a more complex form of digital logic circuit is used: the sequential circuit. The current output of a sequential circuit depends not only on the current input, but also on the past history of inputs. Another and generally more useful way to view it is that the current output of a sequential circuit depends on the current input and the current state of that circuit.

Following are some examples of sequential circuits:

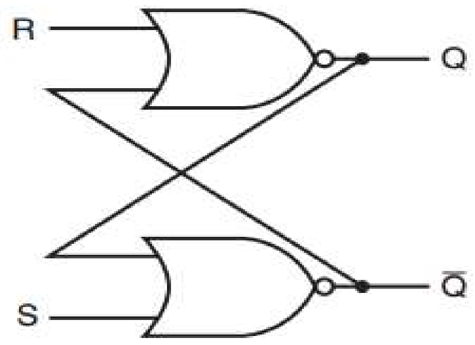
Flip-Flops

Flip-flops and latches are the simplest memory elements. All memory elements store state, and the output from any memory element depends both on the inputs and on the value that has been stored inside the memory element. In both flip-flops and latches, the output is equal to the value of the stored state inside the element.

Memory elements can be clocked or unclocked.

The simplest types of memory elements are un-clocked; i.e. they do not have any clock input. And an example can be SR-latches (set-reset latches) . These latches are made up of two NOR gates.

Following is a diagram of an SR-latch.



The truth table for SR-latch is as follows:

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	–

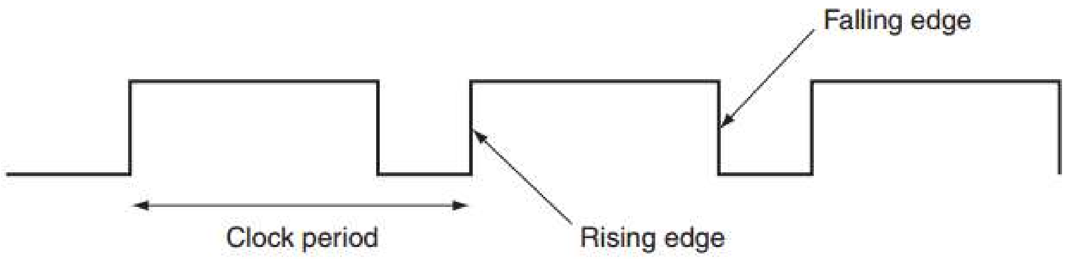
The outputs Q and Q' represent the value of the stored state and its complement. When neither S nor R are asserted (means both are zero or false), the cross-coupled NOR gates act as inverters and store the previous values of Q and Q', i.e. no change is made.

This cross-coupled structure is the basis for more complex memory elements that allow us to store data signals. These elements contain additional gates used to store signal values and to cause the state to be updated only in conjunction with a clock.

Most latches and flip-flops are clocked, which means that they have a clock input and the change of state is triggered by that clock. The difference between a flip-flop and a latch is the point at which the clock causes the state to actually change. In a clocked latch, the state is changed whenever the appropriate inputs change and the clock is asserted, whereas in a flip-flop, the state is changed only on a clock edge.

Clocks are needed in sequential logic to decide when an element that contains state should be updated. A clock is simply a free-running signal with a fixed cycle time; the clock frequency is simply the inverse of the cycle time.

The following figures illustrates a clock.

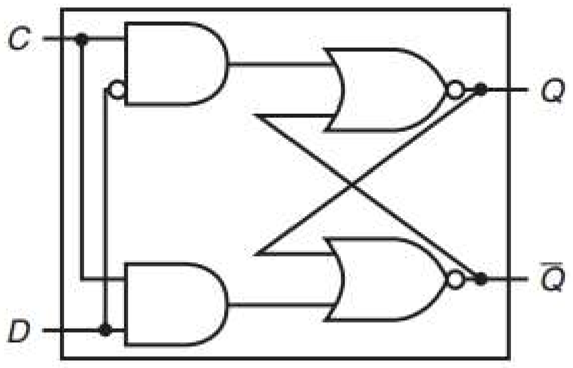


The clock cycle time or clock period is divided into two portions: when the clock is high and when the clock is low. The clock signal oscilates between the high and low values. The clock period is the time for one full cycle. In an edge-triggered design, either the rising or falling edge of the clock is active and causes state to be changed. Clocked flip-flops are example of circuits which are edge-triggered.

Flip-flops are made up of clocked latches. For computer applications, the function of both flip-flops and latches is to store a signal. A type of sequential circuit known as D latch or D flip-flop stores the value of its data input signal in the internal memory. Although there are many other types of latch and flip-flop, the D type is the only basic building block that is needed. A D-latch has two inputs and two outputs. The inputs are the data value to be stored (called D) and a clock signal (called C) that indicates when the latch should read the value on the D input and store it. The outputs are simply the value of the internal state (Q) and its complement (Q'). When the clock input C is asserted, the latch is said to be open, and the value of the output (Q) becomes the value of the input D. When the clock input C is deasserted, the latch is said to be closed, and the value of the output (Q) is whatever value was stored the last time the latch was open.

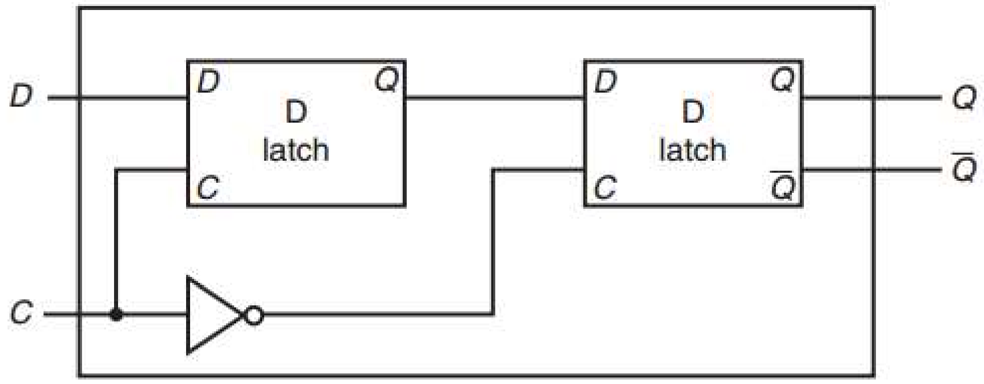
Following are figures that show the truth table and the structure of D latch which is the cross coupled latch with two additional gates.

D	Q_{n+1}
0	0
1	1



We use flip-flops as the basic building block, rather than latches. Flip-flops are not transparent, i.e. their outputs change only on the clock edge. A flip-flop can be built so that it triggers on either the rising (positive) or falling (negative) clock edge.

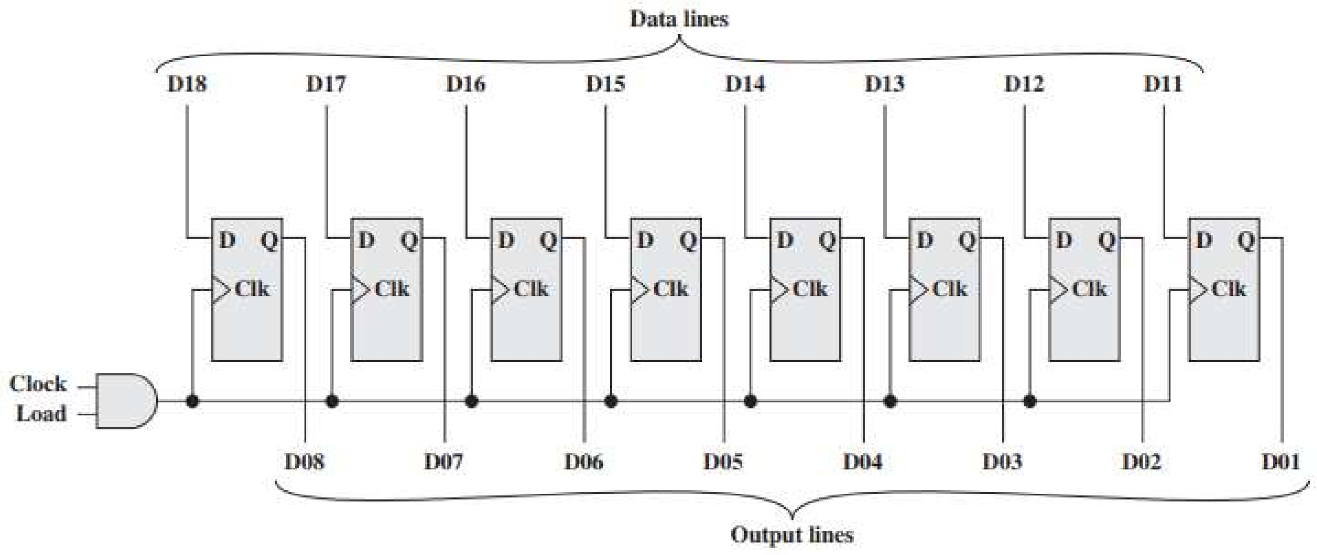
The figure below shows how a falling-edge D flip-flop is constructed from a pair of D latches. In a D flip-flop, the output is stored when the clock edge occurs.



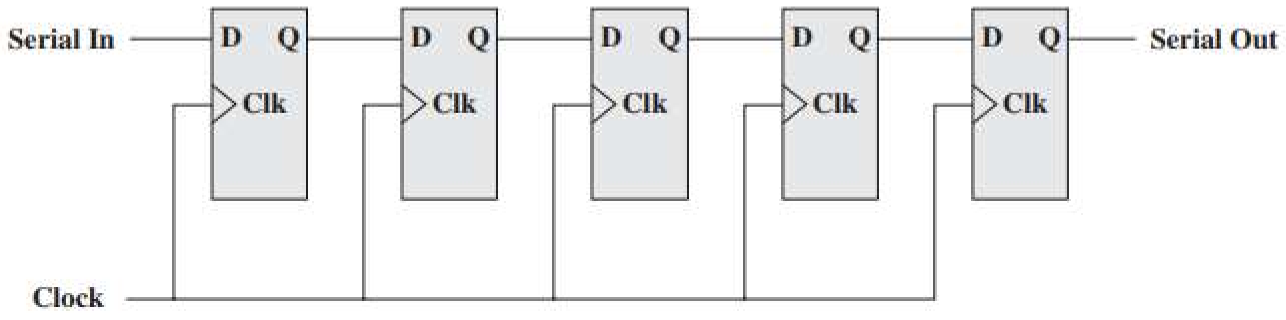
Registers
A register is a digital circuit used within the CPU to store one or more bits of data. Two basic types of registers are commonly used: *parallel registers* and *shift registers*.

A parallel register consists of a set of 1-bit memories that can be read or written simultaneously. It is used to store data.

The figure below is an 8-bit register that illustrates the operation of a parallel register using D flip-flops. A control signal, labeled load, controls writing into the register from signal lines, D11 through D18. These lines might be the output of multiplexers, so that data from a variety of sources can be loaded into the register.



A shift register accepts and/or transfers information serially. The figure below shows a 5-bit shift register constructed from clocked D flip-flops. Data are input only to the leftmost flip-flop. With each clock pulse, data are shifted to the right one position, and the rightmost bit is transferred out. Shift registers can be used to interface to serial I/O devices. In addition, they can be used within the ALU to perform logical shift and rotate functions.



Counters

Another useful category of sequential circuit is the counter. A counter is a register whose value is easily incremented by 1 modulo the capacity of the register; that is, after the maximum value is achieved the next increment sets the counter value to 0. Thus, a register made up of n flip-flops can count up to 2^n-1. An example of a counter in the CPU is the program counter.

Chapter-3- **Micro-Operation Basics and the CPU**

3. 1. **Micro-Operations**

The operation of a computer, in executing a program, consists of a sequence of instruction cycles, with one machine instruction per cycle. Of course, this sequence of instruction cycles is not necessarily the same as the written sequence of instructions that make up the program, because of the existence of branching instructions.

Each instruction cycle is made up of a number of smaller units that include fetch, decode, and execute sub-cycles. The execution of each sub-cycle involves one or more shorter operations, that is, micro-operations.

A micro-operation is an elementary operation that the CPU performs at a single clock pulse. Micro-operations are the functional, or atomic, operations of a processor.

All micro-operations fall into one of the following categories:

- Transfer data from one register to another
- Transfer data from a register to an external interface (e.g., system bus)
- Transfer data from an external interface to a register
- Perform an arithmetic or logic operation, using registers for input and output

3. 2. **Central Processing Unit (CPU)**

Microprocessor, simply the CPU, is the hardware component that performs the overall activities of the computer system. Any operation that is performed inside of the CPU is called micro-operation.

The basic functional elements of the processor are the following:

- ALU
- Registers
- Internal data paths
- External data paths
- Control unit

The ALU is the functional essence of the computer. Registers are used to store data internal to the processor. Some registers contain status information needed to manage instruction sequencing. Others contain data that go to or come from the ALU, memory, and I/O modules. Internal data paths are used to move data between registers and between register and ALU. External data paths link registers to memory and I/O modules, often by means of a system bus. The control unit causes operations to happen within the processor.

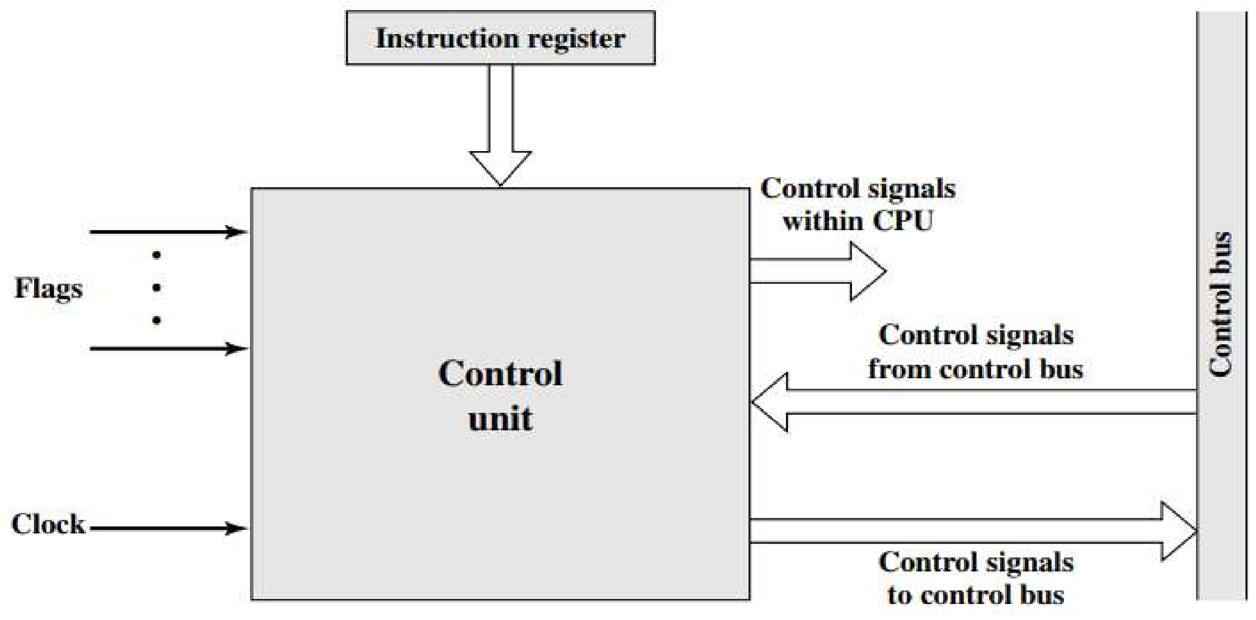
The control unit performs two basic tasks:

- Sequencing: The control unit causes the processor to step through a series of micro-operations in the proper sequence, based on the program being executed
- Execution: The control unit causes each micro-operation to be performed

The key to how the control unit operates is the use of control signals.

For the control unit to perform its function, it must have inputs that allow it to determine the state of the system and outputs that allow it to control the behavior of the system. These are the external specifications of the control unit. Internally, the control unit must have the logic required to perform its sequencing and execution functions.

Following is block diagram of the control unit.


Instruction register
Control signals within CPU
Flags
Control unit
Control signals from control bus
Control bus
Clock
Control signals to control bus

The control signals are of three types:

- Those that activate an ALU function
- Those that activate a data path, and
- Those that are signals on the external system bus or other external interface.

All of these signals are ultimately applied directly as binary inputs to individual logic gates.

3. 3. **General Register Organization Of 8086 CPU**

The 8086 has a powerful set of registers. It includes general purpose registers, segment registers, pointers and index registers and flag register. All the registers of 8086 are 16-bit registers.

General Data Registers

The registers AX, BX, CX and DX are the general purpose 16-bit registers. AX is used as 16-bit accumulator, with the lower 8-bits of AX designated as AL and higher 8-bits as AH. AL can be used as an 8-bit accumulator for 8-bit operations. This is the most important general purpose register having multiple functions.

The register BX is used as offset storage for forming physical addresses in case of certain addressing modes.

The register CX is also used as a default counter in case of string and loop instructions.

DX register is a general purpose register which may be used as an implicit operand or destination in case of a few instructions.

Segment Registers

Unlike 8085, the 8086 addresses a segmented memory. The complete 1 megabyte memory is divided into 16 logical segments. Each segment thus contains 64 Kbytes of memory.

There are four segment registers such as:

- Code Segment Register (CS),
- Data Segment Register (DS),

- Extra Segment Register (ES) and
- Stack Segment Register (SS)

Generally segment register is used to store the upper 16-bits of the starting address of a particular segment. The contents of the segment register are called segment base address. The code segment register is used for addressing a memory location in the code segment of the memory, where the executable program is stored. Similarly, the data segment register points to the data segment of the memory, where the data is stored. The extra segment also refers to a segment which essentially is another data segment of the memory. Thus the extra segment also contains data.

The stack segment register is used for addressing stack segment of memory. The stack segment is that segment of memory which is used to store stack data. The CPU uses the stack for temporarily storing important data, e.g. the contents of the CPU registers which will be required at a later stage

While addressing any location in the memory bank, the physical address is calculated from two parts, the first is segment address and the second is offset. The segment registers contain 16-bit segment base addresses, related to different segments. Any of the pointers and index registers or BX may contain the offset of the location to be addressed. Thus the CS, DS, SS and ES segment registers respectively contain the segment addresses for the code, data, stack and extra segments of memory.

Pointers and Index Registers

The pointers contain offset within the particular segments. The pointers IP, BP and SP usually contain offsets within the code, data and stack segments respectively. The index registers are used as general purpose registers as well as for offset storage in case of indexed, based indexed and relative based indexed addressing modes. The register SI is generally used to store the offset of source data in data segment while the register DI is used to store the offset of destination in data or extra segment. The index registers are particularly useful for string manipulations.

Flag Register

The 8086 has a 16-bit flag register which is divided into two parts such as

- conditional code or status flags and
- machine control flags

The flag register of 8086 the condition code flag register is the lower byte of the 16-bit flag register along with the overflow flag. These flag registers of 8086 reflects the results of the operations performed by ALU. The control flag register is the higher byte of the flag register of 8086. It contains three flags such as direction flag (D), interrupt flag (I) and trap flag (T). The description of each flag bit is as follows:

- S-Sign Flag: This flag is set, when the result of any computation is negative. For signed computations, the sign flag equals the MSB of the result.
- Z-Zero Flag: This flag is set, if the result of the computation or comparison performed by the previous instruction/instructions is zero.
- P-Parity Flag: This flag is set to 1, if the lower byte of the result contains even number of 1s.
- C-Carry Flag: This flag is set, when there is a carry out of MSB in case of addition or borrow in case of subtraction.
- T-Trap Flag: If this flag is set, the processor enters the single step execution mode. In other words, a trap interrupt is generated after execution of each instruction. The processor executes the current instruction and the control is transferred to the Trap interrupt service routine.
- I-interrupt Flag: If this flag is set, the mask-able interrupts are recognized by the CPU, otherwise, they are ignored.
- D-Direction Flag: This is used by string manipulation instructions. If this flag bit is. '0', the string is processed beginning from the lowest address to the highest address, i.e. auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e. auto decrementing mode.
- AC-Auxiliary Carry Flag: This is set, if there is a carry from the lowest nibble, i.e. bit three, during addition or borrow for the lowest nibble, i.e. bit three, during subtraction.
- O-Overflow Flag: This flag is set, if an overflow occurs, i.e. if the result of a signed operation is large enough to be accommodated in a destination

register. For example, in case of the addition of two signed numbers, if the result overflows into the sign bit, i.e. the result is of more than 7-bits in size in case of 8-bit signed operations and more than15-bits in size in case of 16-bit signed operations, and then the overflow flag will be set.

3. 4. **Instruction Format**

An instruction format defines the layout of the bits of an instruction, in terms of its constituent fields. An instruction format must include an opcode and, implicitly or explicitly, zero or more operands. Each explicit operand is referenced using one of the addressing modes which are discussed in the coming section.

The format must, implicitly or explicitly, indicate the addressing mode for each operand. For most instruction sets, more than one instruction format is used.

Three-Address Instructions

In these types of instructions, all operand addresses are explicitly defined. Here the instructions format has three different address fields specifying memory or processor register operand.

Example:

ADD R1, C, B ---> i.e. the operands at memory addresses C and B are added and the resultant sum is stored at processor register R1.

Advantage:

- It results in short programs when evaluating arithmetic expressions.
- Less execution time.

Disadvantage:

- Too many bits are required for binary-coded instructions to specify three addresses
- More complex decoding and processing circuits are needed.

Two Address instructions

Here the instruction format has two different address fields, each specifying either a memory or a processor register operand.

Example:

ADD X, Y ---> adds the content of X and Y and stores the sum in memory address X.

Advantage:

- Less execution time compare to one-address instructions.
- Number of instructions needed to evaluate an arithmetic expression is lesser compared to three address instructions.

Disadvantage:

- It results in comparatively longer programs than three-address instructions when evaluating arithmetic expression.
- Comparatively more execution time than three-address instruction.

One Address Instructions

Such instruction format has a single explicit address field and uses an implied accumulator (AC) register for all data manipulation.

Example

ADD X ---> i.e. the operand at memory location X is added to the accumulator content and the result is stored in the accumulator itself.

Advantage:

- Match less number of bits is required to specify the single operand address.
- Less complicated decoding is needed.

Disadvantage:

- Number of instructions needed to evaluate an arithmetic expression is much more compared to two and three address instructions.
- More execution time needed compare to two and three address instructions.
- Each instruction can perform limited range of functions each instruction can perform.

Zero Address Instructions

Such instructions do not contain any explicit addresses (except for PUSH and POP instructions).

Example

ADD---> Here the top two operands in the stack are removed from the stack and added, then the result is again placed at the top of the stack

Advantage:

- Do not Contained any explicit address. So, instructions are simple.

Disadvantage:

- To evaluate arithmetic expression, it first must be converted to post-fix (Reverse Polish Notation, RPN) notation.

RISC-Address Instructions

RISC is an abbreviation of Reduced Instruction Set Computer. RISC processor has instructions that are simple and have simple addressing modes. A RISC style instruction engages one word in memory. Execution of a RISC instruction is faster and takes one clock cycle per instruction.

3. 5. **Addressing Modes of the 8086 Microprocessor**

The 80x86 processors let you access memory in many different ways. The 80x86 memory addressing modes provide flexible access to memory, allowing you to easily access variables, arrays, records, pointers, and other complex data types.

The five common memory addressing modes are:

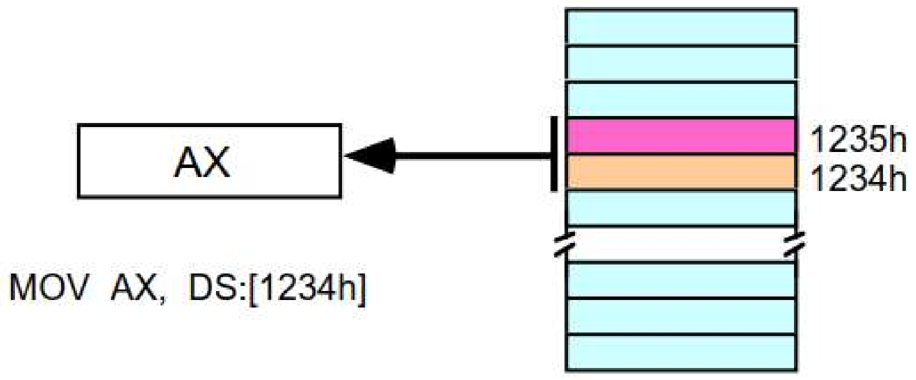
- displacement-only,
- base,
- displacement plus base,
- base plus indexed, and
- Displacement plus base plus indexed

The Displacement Only Addressing Mode:

The most common addressing mode, and the one that's easiest to understand, is the displacement-only (or direct) addressing mode. The displacement-only addressing mode consists of a 16 bit constant that specifies the address of the target location. The instruction **MOV al,ds:[8088h]** loads the **al** register with a copy of the byte at memory location 8088h. Likewise, the instruction **MOV ds:[1234h],dl** stores the value in the **dl** register to memory location **1234h**.

Instead of writing address **ds:[8088h]** and **ds:[1234h]** you can use simple variable names like x and y.

The following diagram shows how to access a word:



The Register Indirect Addressing Modes

There are four forms of this addressing mode on the 8086, best demonstrated by the following instructions:

```
mov      al, [bx]
mov      al, [bp]
mov      al, [si]
mov      al, [di]
```

These four addressing modes reference the byte at the offset found in the **bx**, **bp**, **si**, or **di** register, respectively. The [**bx**], [**si**], and [**di**] modes use the **ds** segment by default. The [**bp**] addressing mode uses the stack segment (ss) by default.

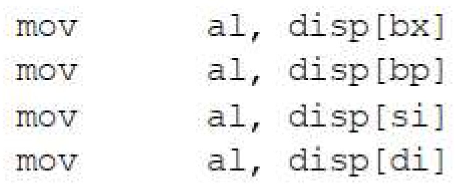
You can use the segment override prefix symbols if you wish to access data in different segments. The following instructions demonstrate the use of these overrides:

```
mov      al, cs:[bx]
mov      al, ds:[bp]
mov      al, ss:[si]
mov      al, es:[di]
```

Intel refers to [bx] and [bp] as base addressing modes and **bx** and **bp** as base registers (in fact, **bp** stands for base pointer). Intel refers to the [si] and [di] addressing modes as indexed addressing modes (**si** stands for source index, **di** stands for destination index).

Indexed Addressing Modes

The indexed addressing modes use the following syntax:

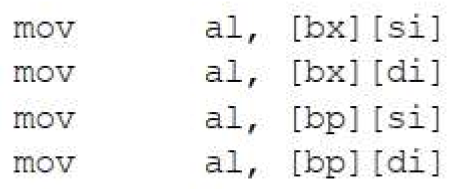


```
mov     al, disp[bx]
mov     al, disp[bp]
mov     al, disp[si]
mov     al, disp[di]
```

If **bx** contains 1000h, then the instruction **MOV cl,20h[bx]** will load **cl** from memory location **ds:1020h**. Likewise, if **bp** contains 2020h, then the instruction **MOV dh,1000h[bp]** will load dh from location **ss:3020**.

Based Indexed Addressing Modes

The based indexed addressing modes are simply combinations of the register indirect addressing modes. These addressing modes form the offset by adding together a base register (**bx** or **bp**) and an index register (**si** or **di**). The allowable forms for these addressing modes are:



```
mov     al, [bx][si]
mov     al, [bx][di]
mov     al, [bp][si]
mov     al, [bp][di]
```

Suppose that **bx** contains 1000h and **si** contains 880h. Then the instruction,

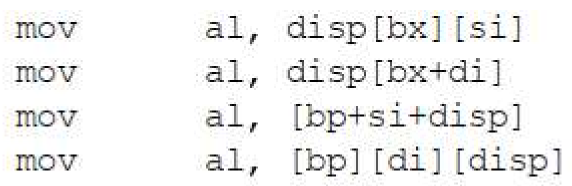
```
MOV AL, [bx],[si]
```

would load **AL** from location **DS:1880h**.

Based Indexed Plus Displacement Addressing Mode

These addressing modes are a slight modification of the base/indexed addressing modes with the addition of an eight bit or sixteen bit constant.

Following are the allowable forms:



```
mov     al, disp[bx][si]
mov     al, disp[bx+di]
mov     al, [bp+si+disp]
mov     al, [bp][di][disp]
```

3. 6. **Categories Of Computer Instructions**

80x86 instructions can be divided into eight different classes:

- Data movement instruction (MOV, PUSH, POP…)
- Conversion (cbw, cwd…)
- Arithmetic instructions (add, inc, sub, dec, neg, mul, div …)
- Logical, shift, rotate, and bit instructions (and, or, xor, not, shl, shr…)
- I/O instructions (IN, OUT)
- String instructions (movs…)
- Program flow control instructions (jmp, call, ret…)
- Miscellaneous instruction

Data Movement/Transfer Instructions

The data movement instructions copy values from one location to another. These instructions include the following:

- The MOV Instruction:

The **mov** instruction takes several different forms among which the following are some:

```
MOV reg, reg  ;copies from general register to another general register
MOV mem, reg  ;copies from general register to memory
MOV reg, mem  ;copies from memory to general register
MOV mem, immediate_data  ;copies literal data to memory
MOV reg, immediate_data  ;copies literal data to general register
```

There is no memory to memory move operation. There is no form of the MOV instruction that allows you to encode two memory addresses into the same instruction. Also, you cannot move immediate data into a segment register.

The operands to the MOV instruction may be bytes, words, or double words. Both operands must be the same size, otherwise, it is error. If the operand to copy is an immediate data, the CPU extends to the size of the destination operand (unless it is too big to fit in the destination operand, which is an error).

Note that, the MOV instructions do not affect any flags. In particular, the 80x86 preserves the flag values across the execution of a MOV instruction.

- The XCHG instruction:

This instruction swaps two values:

Syntax: XCHG operand1, operand2 ; exchanges/swaps the value of operand1 and operand2

Both operands must be of the same size.

- The LDS and LES Instructions:

These instructions let you load a 16 bit general purpose register (such as DS and ES).

Example:

LDS reg, mem ; loads the 16 bit data in memory to register

The load instructions do not affect the flag registers.

- The PUSH and POP Instructions

The 80x86 push and pop instructions manipulate data on the 80x86's hardware stack. They have various formats.

The push instructions move data onto the 80x86 hardware stack and the pop instructions move data from the stack to memory or to a register.

Arithmetic Instructions

Most machines provide the basic arithmetic operations of add, subtract, multiply, and divide. These are invariably provided for signed integer (fixed-point) numbers. Often they are also provided for floating-point numbers.

Examples: `ADD, SUB, MUL (unsigned number), IMUL (signed number), DIV, IDIV`

Logical Instructions

Most machines also provide a variety of operations for manipulating individual bits of a word or other addressable units. They are based upon Boolean operations.

Example can be: NOT, AND, OR, XOR

In addition to bitwise logical operations, most machines provide a variety of shifting and rotating functions.

With a logical shift, the bits of a word are shifted left or right. On one end, the bit shifted out is lost. On the other end, a 0 is shifted in.

Conversion

Conversion instructions are those that change the format or operate on the format of data. An example is converting from decimal to binary.

Program Control

Branch, skip, and procedure call are examples of this category instruction.

- **Branch Instructions:**

A branch instruction, also called a jump instruction, has as one of its operands the address of the next instruction to be executed. Most often, the instruction is a conditional branch instruction. That is, the branch is made (update program counter to equal address specified in operand) only if a certain condition is met. Otherwise, the next instruction in sequence is executed (increment program counter as usual). A branch instruction in which the branch is always taken is an unconditional branch.

Example of conditional branch:

BRP X ………… Branch to location X if result is positive

BRN X ………… Branch to location X if result is negative

BRZ X ………… Branch to location X if result is zero

BRO X ………… Branch to location X if overflow occurs.

In all of these cases, the result referred to is the result of the most recent operation that set the condition code.

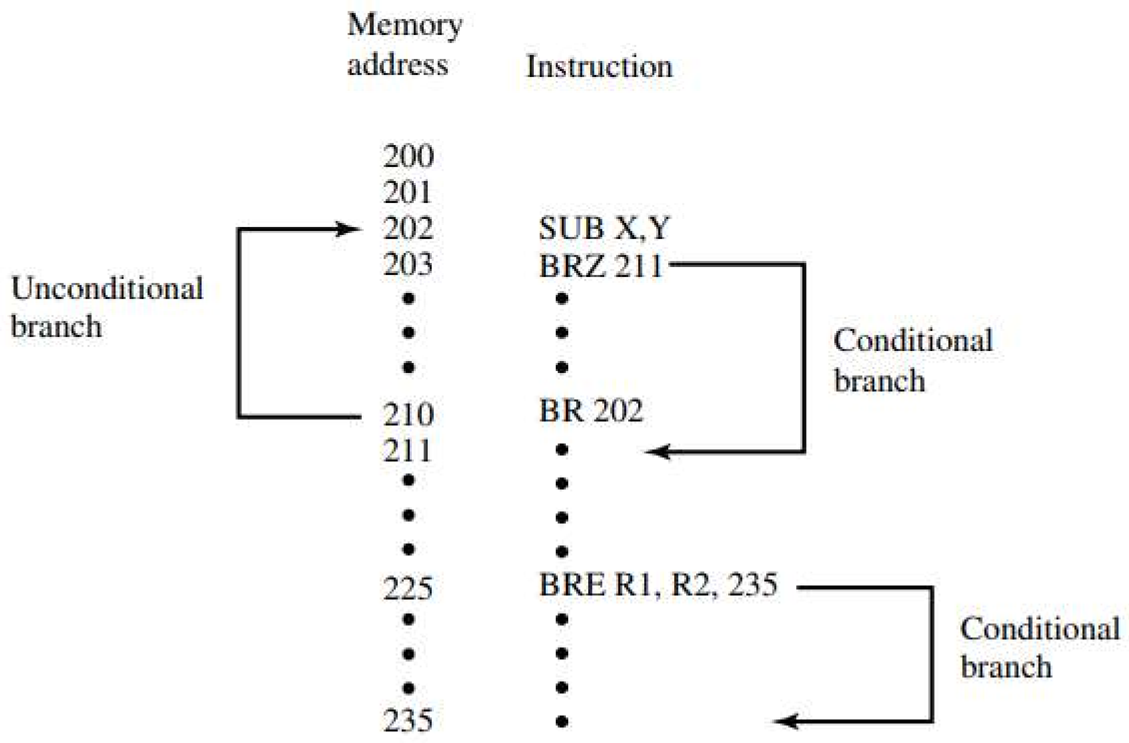
Another approach that can be used with a three-address instruction format is to perform a comparison and specify a branch in the same instruction.

For example,

BRE R1, R2, X ………… Branch to X if content of R1 equals content of R2

A branch can be either forward (an instruction with a higher address) or backward (lower address). The example below shows how an unconditional and a conditional branch can be used to create a repeating loop of instructions.

The instructions in locations 202 through 210 will be executed repeatedly until the result of subtracting Y from X is 0.



- **Skip Instruction**

Another form of transfer-of-control instruction is the skip instruction. The skip instruction includes an implied address. Typically, the skip implies that one instruction be skipped; thus, the implied address equals the address of the next instruction plus one instruction length.

Because the skip instruction does not require a destination address field, it is free to do other things. A typical example is the increment-and-skip-if-zero (ISZ) instruction.

Consider the following program fragment:

```
301
•
•
•
309   ISZ   R1
310   BR    301
311
```

In this fragment, the two transfer-of-control instructions are used to implement an iterative loop. R1 is set with the negative of the number of iterations to be performed. At the end of the loop, R1 is incremented. If it is not 0 the program branches back to the beginning of the loop. Otherwise, the branch is skipped, and the program continues with the next instruction after the end of the loop.

- **Procedure Call Instruction**

A procedure is a named group of instructions and we use that name when we wish to execute those instructions, instead of having to write the instructions again. The group of instructions are referred to as subprogram, also procedure.

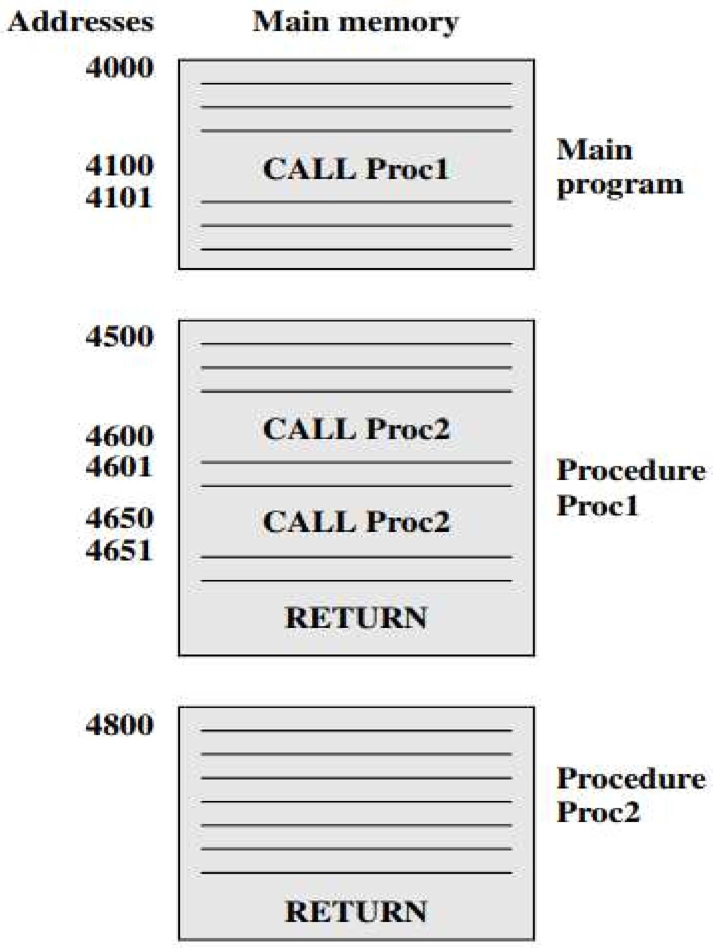
Whenever you want the subprogram to execute, you write the name of the subprogram, and this action is referred as subprogram/procedure calling.

The two principal reasons for the use of procedures are economy and modularity. A procedure allows the same piece of code to be used many times. This is important for economy in programming effort and for making the most efficient use of storage space in the system (the program must be stored).

Procedures also allow large programming tasks to be subdivided into smaller units. This use of modularity greatly eases the programming task.

The procedure mechanism involves two basic instructions: a CALL instruction that branches from the present location to the procedure, and a RETURN instruction that returns from the procedure to the place from which it was called. Both of these are forms of branching instructions.

Following is a figure that illustrates how the CALL and RETURN instructions can be used.



In this example, there is a main program starting at location 4000.This program includes a call to procedure PROC1, starting at location 4500. When this call instruction is encountered, the processor suspends execution of the main program and begins execution of PROC1 by fetching the next instruction from location 4500. Within PROC1, there are two calls to PROC2 at location 4800. In each case, the execution of PROC1 is suspended and PROC2 is executed. The RETURN statement causes the processor to go back to the calling program and continue execution at the instruction after the corresponding CALL instruction.

3. 7. **Characteristics of CISC and RISC Computers**

The architecture of the Central Processing Unit (CPU) operates the capacity to function from "Instruction Set Architecture" to where it was designed. The architectural design of the CPU is of two types:

- Reduced instruction set computing (RISC) and
- Complex instruction set computing (CISC)

A complex instruction set computer (CISC) is a computer where single instructions can perform numerous low-level operations like a load from memory, an arithmetic operation, and a memory store or are accomplished by multi-step processes or addressing modes in single instructions, as its name proposes "Complex Instruction Set".

A reduced instruction set computer (RISC) is a computer that only uses simple commands that can be divided into several instructions which achieve low-level operation within a single CLOCK cycle, as its name proposes "Reduced Instruction Set.

In RISC, every instruction is expected to attain very small jobs. In this machine, the instruction sets are modest and simple, which help in comprising more complex commands. Each instruction is about a similar length; these are wound together to get compound tasks done in a single operation. Most commands are completed in one machine cycle.

The hardware part of the Intel is named as Complex Instruction Set Computer (CISC), and Apple hardware is Reduced Instruction Set Computer (RISC).

Chapter-4- **Memory Organization**

4. 1. **Memory Hierarchy**

Memory Hierarchy refers to the structure that uses multiple levels of memories; as the distance from the processor increases, the size of the memories and the access time both increase.

A memory hierarchy can consist of multiple levels, but data is copied between only two adjacent levels at a time. Within each level, the unit of information that is present or not is called a block or a line. Usually we transfer an entire block when we copy something between levels. The upper level—the one closer to the processor—is smaller and faster than the lower level, since the upper level uses technology that is more expensive.

If the data requested by the processor appears in some block in the upper level, this is called a hit. If the data is not found in the upper level, the request is called a miss. The lower level in the hierarchy is then accessed to retrieve the block containing the requested data.

The hit rate, or hit ratio, is the fraction of memory accesses found in the upper level; it is often used as a measure of the performance of the memory hierarchy. The miss rate (which is 1−hit rate) is the fraction of memory accesses not found in the upper level.

Since performance is the major reason for having a memory hierarchy, the time to service hits and misses is important. Hit time is the time to access the upper level of the memory hierarchy, which includes the time needed to determine whether the access is a hit or a miss, the time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, and then pass the block to the requestor.

4. 2. **Memory Technology**

There are four primary technologies used today in memory hierarchies. These are:

- Dynamic RAM (DRAM)
- Static RAM (SRAM)

- Flash Memory and
- Magnetic Disks

Main memory is implemented from DRAM (dynamic random access memory), while levels closer to the processor (caches) use SRAM (static random access memory).

DRAM is less costly per bit than SRAM, although it is substantially slower. The price difference arises because DRAM uses significantly less area per bit of memory, and DRAMs thus have larger capacity for the same amount of silicon; the speed difference arises from several factors.

The third technology is flash memory. This nonvolatile memory is the secondary memory in Personal Mobile Devices.

The fourth technology, used to implement the largest and lowest level in the hierarchy in servers, is magnetic disk. The access time and price per bit vary widely among these technologies.

SRAM Technology:

SRAMs are simply integrated circuits that are memory arrays with (usually) a single access port that can provide either a read or a write. SRAMs have a fixed access time to any datum, though the read and write access times may differ. SRAMs don't need to refresh and so the access time is very close to the cycle time. SRAMs typically use six to eight transistors per bit to prevent the information from being disturbed when read. SRAM needs only minimal power to retain the charge in standby mode.

DRAM Technology:

In a SRAM, as long as power is applied, the value can be kept indefinitely. In a dynamic RAM (DRAM), the value kept in a cell is stored as a charge in a capacitor. A single transistor is then used to access this stored charge, either to read the value or to overwrite the charge stored there. Because DRAMs use only a single transistor per bit of storage, they are much denser and cheaper per bit than SRAM. As DRAMs store the charge on a capacitor, it cannot be kept indefinitely and must periodically be refreshed. That is why this memory structure is called dynamic, as opposed to the static storage in an SRAM cell.

Disk Memory:

These are normally called storage devices. They store data permanently. They can be hard disks, CDs, Flash disks etc.

4. 3. Cache Memory Organization

It is obvious that cache memories generally increase performance of the system. But the problem is how do the system knows if a data item is in the cache? Moreover, if it is, how do the system find it? The answers are related. If each word can go in exactly one place in the cache, then it is straightforward to find the word if it is in the cache. The simplest way to assign a location in the cache for each word in memory is to assign the cache location based on the address of the word in memory. This cache structure is called direct mapped, since each memory location is mapped directly to exactly one location in the cache. The typical mapping between addresses and cache locations for a direct mapped cache is usually simple.

For example, almost all direct-mapped caches use this mapping to find a block:

```
    (Block address) modulo (Number of blocks in the cache)
```

Another type of memory mapping technique is fully associative mapping. It is a cache structure in which a block can be placed in any location in the cache.

To find a given block in a fully associative cache, all the entries in the cache must be searched because a block can be placed in any one. To make the search practical, it is done in parallel with a comparator associated with each cache entry. These comparators significantly increase the hardware cost, effectively making fully associative placement practical only for caches with small numbers of blocks.

There is also a third type of mapping technique. It is called a set-associative mapping. In a set-associative cache, there are a fixed number of locations where each block can be placed. A set-associative cache with n locations for a block is called an n-way set-associative cache. An n-way set-associative cache consists of a number of sets, each of which consists of n blocks. Each block in the memory maps to a unique set in the cache given by the index field, and a block can be placed in any element of that set. Thus, a set-associative placement combines direct-mapped placement and fully associative placement: a block is directly mapped into a set, and then all the blocks in the set are searched for a match.

In a set-associative cache, the set containing a memory block is given by:

```
(Block number) modulo (Number of sets in the cache)
```

Since the block may be placed in any element of the set, all the tags of all the elements of the set must be searched. In a fully associative cache, the block can go anywhere, and all tags of all the blocks in the cache must be searched.

The choice among direct-mapped, set-associative, or fully associative mapping in any memory hierarchy will depend on the cost of a miss versus the cost of implementing associativity, both in time and in extra hardware.

4. 4. External Memory

There are various types of external memories which are discussed below:

Magnetic Disk:

A magnetic hard disk consists of a collection of platters, which rotate on a spindle at 5400 to 15,000 revolutions per minute. The metal platters are covered with magnetic recording material on both sides, similar to the material found on a cassette or videotape. To read and write information on a hard disk, a movable arm containing a small electromagnetic coil called a read-write head is located just above each surface. The entire drive is permanently sealed to control the environment inside the drive, which, in turn, allows the disk heads to be much closer to the drive surface.

Each disk surface is divided into concentric circles, called tracks. There are typically tens of thousands of tracks per surface. Each track is in turn divided into sectors that contain the information; each track may have thousands of sectors. Sectors are typically 512 to 4096 bytes in size. The sequence recorded on the magnetic media is a sector number, a gap, the information for that sector including error correction code, a gap, the sector number of the next sector, and so on.

The disk heads for each surface are connected together and move in conjunction, so that every head is over the same track of every surface. The term cylinder is used to refer to all the tracks under the heads at a given point on all surfaces.

To access data, the operating system must direct the disk through a three-stage process. The first step is to position the head over the proper track. This operation is called a seek, and the time to move the head to the desired track is called the seek time. Disk manufacturers report minimum seek time, maximum seek time, and average seek time in their manuals. The first two are easy to measure, but the average is open to wide interpretation because it depends on the seek distance. The industry calculates average seek time as the sum of the time for all possible seeks divided by the number of possible seeks. Average seek times are usually advertised as 3 ms to 13 ms, but, depending on the application and scheduling of disk requests, the actual average seek time may be only 25% to 33% of the advertised number because of locality of disk references. This locality arises both because of successive accesses to the same file and because the operating system tries to schedule such accesses together. Once the head has reached the correct track, we must wait for the desired sector to rotate under the read/write head. This time is called the rotational latency or rotational delay. The average latency to the desired information is halfway around the disk. Disks rotate at 5400 RPM to 15,000 RPM. The average rotational latency at 5400 RPM is given as:

$$\text{Average rotational latency} = \frac{0.5 \text{ rotation}}{5400 \text{ RPM}} = \frac{0.5 \text{ rotation}}{5400 \text{ RPM}/\left(60 \dfrac{\text{seconds}}{\text{minute}}\right)}$$
$$= 0.0056 \text{ seconds} = 5.6 \text{ ms}$$

The last component of a disk access, transfer time, is the time to transfer a block of bits. The transfer time is a function of the sector size, the rotation speed, and the recording density of a track.

In summary, the two primary differences between magnetic disks and semiconductor memory technologies are that disks have a slower access time because they are mechanical devices—flash is 1000 times as fast and DRAM is 100,000 times as fast—yet they are cheaper per bit because they have very high storage capacity at a modest cost—disk is 10 to 100 time cheaper. Magnetic disks are nonvolatile like flash.

RAID Technology:

RAID is a set of physical disk drives viewed by the operating system as a single logical drive

Redundant array of independent disks is an organization of disks that uses an array of small and inexpensive disks so as to increase both performance and reliability.

The flaw in the argument was that disk arrays could make reliability much worse.

The unique contribution of the RAID proposal is to address effectively the need for redundancy. Although allowing multiple heads and actuators to operate simultaneously achieves higher I/O and transfer rates, the use of multiple devices increases the probability of failure.

One weakness of the RAID systems is repair. First, to avoid making the data unavailable during repair, the array must be designed to allow the failed disks to be replaced without having to turn off the system. RAIDs have enough redundancy to allow continuous operation, but hot-swapping disks place demands on the physical and electrical design of the array and the disk interfaces. Second, another failure could occur during repair, so the repair time affects the chances of losing data: the longer the repair time, the greater the chances of another failure that will lose data.

Optical Memory:

The Compact Disk, CD, is a non-erasable disk that can store more than 60 minutes of audio information on one side. The huge commercial success of the CD enabled the development of low-cost optical-disk storage technology that has revolutionized computer data storage. A variety of optical-disk systems have been introduced which include the following.

- **CD-ROM (**Compact Disk Read-Only Memory): A non-erasable disk used for storing computer data. The standard system uses 12-cm disks and can hold more than 650 Mbytes.

- **CD-R** (CD Recordable): Similar to a CD-ROM. The user can write to the disk only once.
- **CD-RW** (CD Rewritable): Similar to a CD-ROM. The user can erase and rewrite to the disk multiple times.
- **DVD** (Digital Versatile Disk): A technology for producing digitized, compressed representation of video information, as well as large volumes of other digital data. Both 8 and 12 cm diameters are used, with a double-sided capacity of up to 17 Gigabytes. The basic DVD is read-only (DVD-ROM).
- **DVD-R** (DVD Recordable): Similar to a DVD-ROM. The user can write to the disk only once. Only one-sided disks can be used.
- **DVD-RW** (DVD Rewritable): Similar to a DVD-ROM. The user can erase and rewrite to the disk multiple times. Only one-sided disks can be used.
- **Blu-Ray DVD**: High definition video disk. Provides considerably greater data storage density than DVD, using a 405-nm (blue-violet) laser. A single layer on a single side can store 25 Gigabytes.

Magnetic Tape:

Tape systems use the same reading and recording techniques as disk systems. The medium is flexible polyester tape coated with magnetizable material. The coating may consist of particles of pure metal in special binders or vapor-plated metal films.

A tape drive is a sequential-access device. If the tape head is positioned at record 1, then to read record N, it is necessary to read physical records 1 through N-1, one at a time. If the head is currently positioned beyond the desired record, it is necessary to rewind the tape certain distance and begin reading forward.

Magnetic tape was the first kind of secondary memory. It is still widely used as the lowest-cost, slowest-speed member of the memory hierarchy.

Chapter-5- **Input–Output Organization and Parallel Processing**

5.1. Computer Peripheral Devices

I/O operations are accomplished through a wide assortment of external devices that provide a means of exchanging data between the external environment and the computer. An external device attaches to the computer by a link to an I/O module. The link is used to exchange control, status, and data between the I/O module and the external device. An external device connected to an I/O module is often referred to as a peripheral device or, simply, a peripheral.

We can broadly classify external devices into three categories:

- Human readable: Suitable for communicating with the computer user
- Machine readable: Suitable for communicating with equipment
- Communication: Suitable for communicating with remote devices

Examples of human-readable devices are video display terminals (VDTs) and printers.

Examples of machine-readable devices are magnetic disk and tape systems and sensors and actuators, such as those used in a robotics application.

5.2. Modes Of Input-Output (I/O) Transfer

There are three basic techniques for performing I/O:

Programmed I/O:

With programmed I/O, data are exchanged between the processor and the I/O module. The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data. When the processor issues a command to the I/O module, it must wait until the I/O operation is complete. If the processor is faster than the I/O module, this is wasteful of processor time.

The processor issues an I/O command, on behalf of a process, to an I/O module; that process then busy waits for the operation to be completed before proceeding. The CPU controls the entire process hence wasting CPU time.

Following is a detailed sequence of operations in programmed I/O.

i) CPU requests I/O operation

ii) I/O module performs operation

iii) I/O module sets status bits

iv) CPU checks status bits periodically

v) I/O module does not inform CPU directly

vi) I/O module does not interrupt CPU

vii) CPU may wait or come back later

Interrupt-Driven I/O:

The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The processor, while waiting, must repeatedly interrogate the status of the I/O module. As a result, the level of the performance of the entire system is severely degraded.

An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work. The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor. The processor then executes the data transfer, as before, and then resumes its former processing.

When the processor issues an I/O command on behalf of a process, there are then two possibilities. If the I/O instruction from the process is non-blocking, then the processor continues to execute instructions from the process that issued the I/O command. If the I/O instruction is blocking, then the next instruction that the processor executes is from the OS, which will put the current process in a blocked state and schedule another process.

Let us consider how this works, first from the point of view of the I/O module. For input, the I/O module receives a READ command from the processor. The I/O module then proceeds to read data in from an associated peripheral. Once the data are in the module's data register, the module signals an interrupt to the processor over a control line. The module then waits until its data are requested by the processor. When the request is made, the module places its data on the data bus and is then ready for another I/O operation.

Following are basic operation sequences in interrupt driven I/O.

i) CPU issues READ command

ii) I/O module gets data from peripheral while CPU does other work

iii) I/O module interrupts CPU

iv) CPU requests data

v) I/O module transfers data

With both programmed and interrupt I/O, the processor is responsible for extracting data from main memory for output and storing data in main memory for input. The alternative is known as direct memory access (DMA). In this mode, the I/O module and main memory exchange data directly, without processor involvement.

Direct Memory Access (DMA):

Both interrupt driven and programmed I/O require active CPU intervention. The processor must transfer data, i.e. read from device or memory and write to memory or device. The transfer rate is limited and the CPU is tied up.

A DMA module controls the exchange of data between main memory and an I/O module. The processor sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred.

5.3. Priority Interrupt

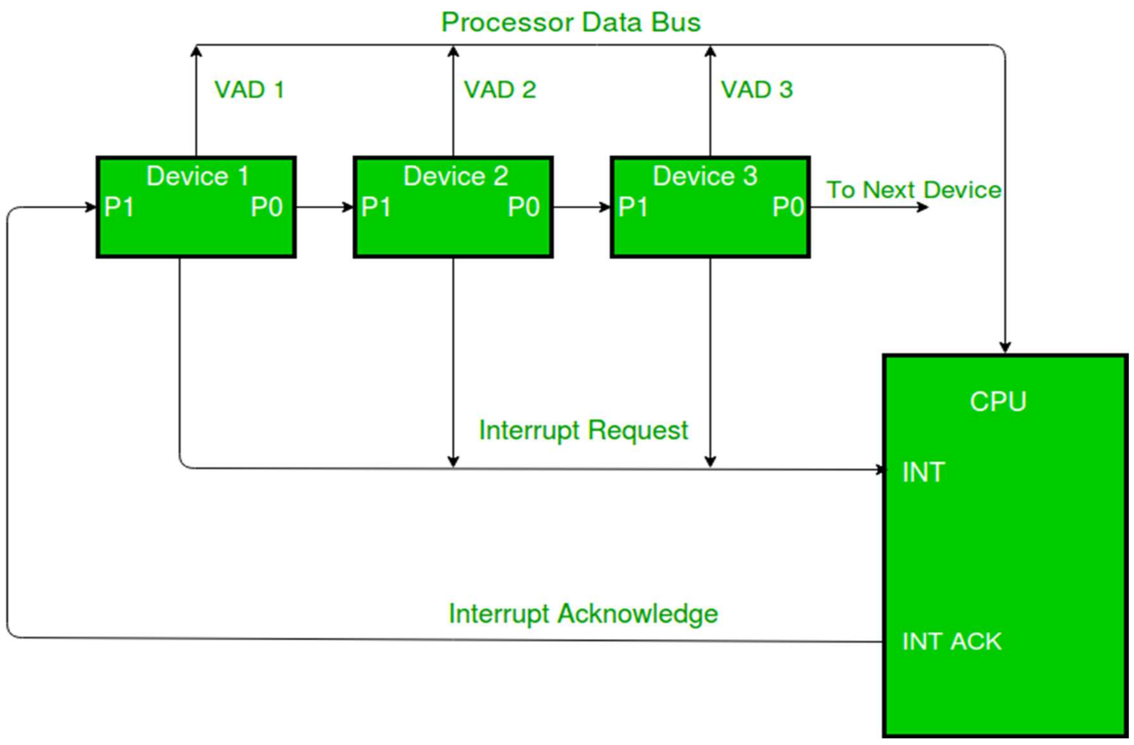
An interrupt in computer architecture is a signal that requests the processor to suspend its current execution and service the occurred interrupt. To service the interrupt the processor executes the corresponding interrupt service routine (ISR). After the execution of the interrupt service routine, the processor resumes the execution of the suspended program. Interrupts can be of two types of hardware interrupts and software interrupts.

A priority interrupt is a system which decides the priority at which various devices, which generates the interrupt signal at the same time, will be serviced by the CPU. When two or more devices interrupt the computer simultaneously, the computer services the device with the higher priority first.

Now, when the processor is executing an interrupt service routine the processor priority level is set to the priority of the device of which the interrupt processor is servicing. Thus the processor accepts only the interrupts from the device with the higher priority and ignores the interrupts from the device with the same or low priority. To set the priority level of the processor some bits of the processor's status register is used.

5.4. Daisy-Chaining Priority

The daisy-chaining method involves connecting all the devices that can request an interrupt in a serial manner. This configuration is governed by the priority of the devices. The device with the highest priority is placed first followed by the second highest priority device and so on. The figure below describes this method.



There is an interrupt request line which is common to all the devices and goes into the CPU.

- When no interrupts are pending, the line is in HIGH state. But if any of the devices raises an interrupt, it places the interrupt request line in the LOW state.
- The CPU acknowledges this interrupt request from the line and then enables the interrupt acknowledge line in response to the request.
- This signal is received at the PI(Priority in) input of device 1.
- If the device has not requested the interrupt, it passes this signal to the next device through its PO(priority out) output. (PI = 1 & PO = 1)
- However, if the device had requested the interrupt, (PI =1 & PO = 0)
    - The device consumes the acknowledge signal and block its further use by placing 0 at its PO (priority out) output.
    - The device then proceeds to place its interrupt vector address (VAD) into the data bus of CPU. VAD is the address of the service routine which services that device.
    - The device puts its interrupt request signal in HIGH state to indicate its interrupt has been taken care of.

If a device gets 0 at its PI input, it generates 0 at the PO output to tell other devices that acknowledge signal has been blocked. (PI = 0 & PO = 0).

Hence, the device having PI = 1 and PO = 0 is the highest priority device that is requesting an interrupt. Therefore, by daisy chain arrangement we have ensured that the highest priority interrupt gets serviced first and have established a hierarchy. The farther a device is from the first device, the lower its priority.

5.5. Parallel Processing

A traditional way to increase system performance is to use multiple processors that can execute in parallel to support a given workload. The two most common multiple-processor organizations are:

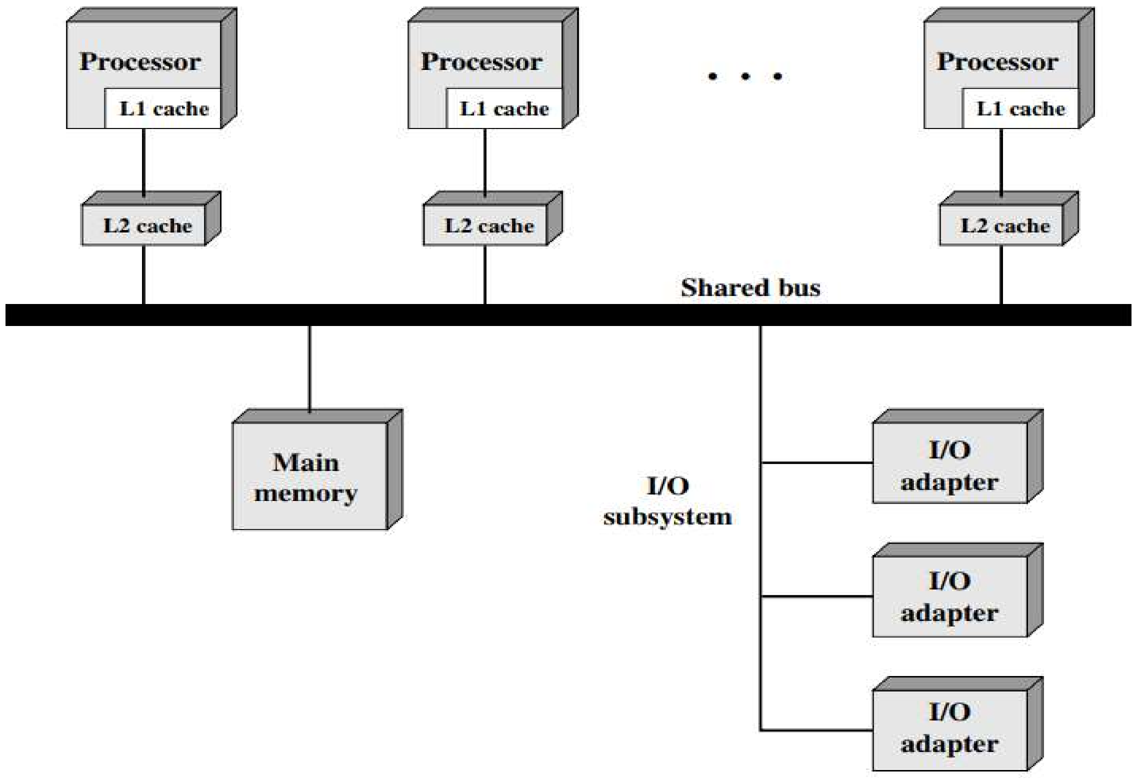
- symmetric multiprocessors (SMPs) (also called *shared memory processors)* and
- clusters

An SMP consists of multiple similar processors within the same computer interconnected by a bus or some sort of switching arrangement. The most critical problem to address in an SMP is that of cache coherence. Each processor has its own cache and so it is possible for a given line of data to be present in more than one cache. If such a line is altered in one cache, then both main memory and the other cache have an invalid version of that line. Cache coherence protocols are designed to cope with this problem.

The processors share the same main memory and I/O facilities and are interconnected by a bus or other internal connection scheme, such that memory access time is approximately the same for each processor.

When more than one processor are implemented on a single chip, the configuration is referred to as *chip multiprocessing*. A related design scheme is to replicate some of the components of a single processor so that the processor can execute multiple threads concurrently; this is known as a *multithreaded processor*.

following is a diagram that illustrates SMP.



A cluster is a group of interconnected, whole computers working together as a unified computing resource that can create the illusion of being one machine. The term whole computer means a system that can run on its own, apart from the cluster.

Clustering is an alternative to symmetric multiprocessing as an approach to providing high performance and high availability and is particularly attractive for server applications.

5.6. Multi-Core Computers

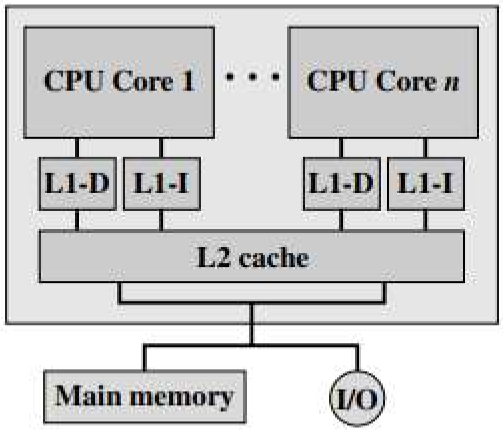
A multi-core computer, or chip multiprocessor, combines two or more processors, which are called cores, on a single computer chip. Typically, each core consists of all of the components of an independent processor, such as registers, ALU, pipeline hardware, and control unit, plus L1 instruction and data caches.

The multi-core architecture poses challenges to software developers to exploit the capability for multithreading across multiple cores. The main variables in a multi-core organization are the number of processors on the chip, the number of levels of cache memory, and the extent to which cache memory is shared.

Following are some examples of multi-core processors:

Intel Core Duo:

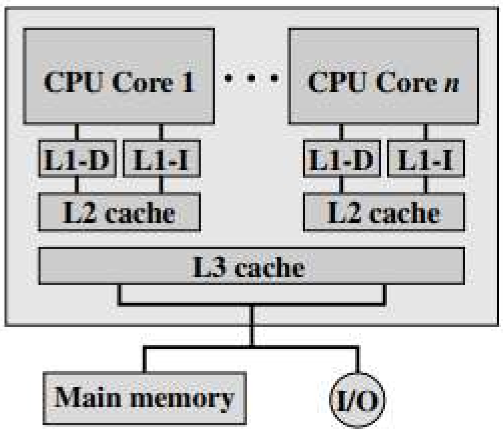
Intel core duo implements **two** x86 superscalar processors with a shared L2 cache as in the following diagram:



In this case, each core has a 32-KB instruction cache and a 32-KB data cache having L2 size of 2MB.

Intel Core i7

As the amount of cache memory available on the chip continues to grow, performance considerations dictate splitting off a separate, shared L3 cache, with dedicated L1 and L2 caches for each core processor and Intel core i7 is an example for this. Here is a diagram for Intel core i7.



Core i7 implements **four** x86 symmetric multithreaded processors, each with a dedicated L2 cache, and with a shared L3 cache.

The size of the L2 cache is 256KB each and that of the L3 cache is 8MB.

Chapter-6- **Machine And Assembly Languages Concepts**

6.1. **Assembly Language Basics**

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

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

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

Assembly language operation codes (opcodes) are easily remembered, for instance (MOV for move instructions, JMP for jump).

Almost every line of source coding in an assembly language source program translates directly into a machine instruction for a particular processor. Therefore, the assembly language programmer must be familiar with both the assembly language and the processor for which he is programming.

6.2. **What is Assembler?**

Assembler is a program that translates a symbolic version of instructions into the binary version.

For example, the programmer would write:

Add A, B

Then the assembler would translate this notation into:

1000110010100000

Assembly language requires the programmer to write one line for every instruction that the computer will follow, forcing the programmer to think like the computer.

The assembler turns the assembly language program into an object file, which is a combination of machine language instructions, data, and information needed to place instructions properly in memory. To produce the binary version of each instruction in the assembly language program, the assembler must determine the addresses corresponding to all labels. Assemblers keep track of labels used in branches and data transfer instructions in a symbol table. Such a table contains pairs of symbols and addresses.

6.3. **Directives**

Directives are commands that are part of the assembler syntax but are not related to the x86 processor instruction set. All assembler directives begin with a period (**.**).

Assembler directives enable you to do the following:

- Assemble code and data into specified sections.
- Reserve space in memory for uninitialized variables

Assembler directives do not represent instructions, and are not translated into machine code. And the directive must exist on a separate line from any other assembler directive or assembler instruction.

Examples:

.data ………………. reserves space for initialized variables

.bss ………………. reserves space for uninitialized variables in the .bss section. This directive is usually used to allocate space in RAM.

.text ………………. Assembles into the executable code

6.4. **Procedure and Functions**

Procedures or subroutines are very important in assembly language, as the assembly language programs tend to be large in size. Procedures are identified by a name. Following this name, the body of the procedure is described which performs a well-defined job. End of the procedure is indicated by a RETURN statement.

A function is a piece of code that is designed to perform a subtask in the program. Functions can have local variables, receive arguments, and pass a result back to the calling program.