UNIT-1

Topics : **Basic Structure of Computer** :Computer types, Functional units, basic operational concepts, Bus structures, Software, Performance, Multiprocessors and Multicomputer.
Machine Instructions and Programs: Numbers, Arithmetic Operations and programs, Instructions and Instruction Sequencing, Addressing modes, Basic Input/output operations, stacks and queues , Subroutines , Additional Instructions.

PART-1: **Basic Structure of Computer**

1. COMPUTER :

A computer is a machine that can be programmed to manipulate symbols. Its principal characteristics are:

- It responds to a specific set of instructions in a well-defined manner.
- It can execute a prerecorded list of instructions (a program).
- It can quickly store and retrieve large amounts of data.

Therefore computers can perform complex and repetitive procedures quickly, precisely and reliably.

COMPUTER TYPES:
Computers can be generally classified by size and power as follows:

1. **Supercomputer and Mainframe**
2. **Minicomputer**
3. **Workstation**
4. **Personal computer**
5. **Personal Computer Types**

SUPERCOMPUTER AND MAINFRAME:

- Supercomputer is a broad term for one of the fastest computers currently available. An extremely fast computer that can perform hundreds of millions of instructions per second.
- Supercomputers are very expensive and are employed for specialized applications that require immense amounts of mathematical calculations.
- Mainframe was a term originally referring to the cabinet containing the central processor unit or "main frame" of a room-filling Stone Age batch machine. After the emergence of smaller "minicomputer" designs in the early 1970s, the traditional big iron machines were described as "mainframe computers" and eventually just as mainframes.
- Nowadays a Mainframe is a very large and expensive computer capable of supporting hundreds, or even thousands, of users simultaneously. A powerful multi-user computer capable of supporting many hundreds or thousands of users simultaneously.
- In some ways, mainframes are more powerful than supercomputers because they support more simultaneous programs. But supercomputers can execute a single program faster than a mainframe.

MINI COMPUTER:

- It is a midsize computer.
- In the past decade, the distinction between large minicomputers and small mainframes has blurred, however, as has the distinction between small minicomputers and workstations.
- A minicomputer is a multiprocessing system capable of supporting from up to 200 users simultaneously.

WORKSTATION:

- It is a type of computer used for engineering applications (CAD/CAM), desktop publishing, software development, and other types of applications that require a moderate amount of computing power and relatively high quality graphics capabilities.
- Workstations generally come with a large, high-resolution graphics screen, at large amount of RAM, built-in network support, and a graphical user interface. Most workstations also have a mass storage device such as a disk drive, but a special type of workstation, called a diskless workstation, comes without a disk drive.
- The most common operating systems for workstations are UNIX and Windows NT. Like personal computers, most workstations are single-user computers. However, workstations are typically linked together to form a local-area network, although they can also be used as stand-alone systems.

PERSONAL COMPUTER:

- Personal computers first appeared in the late 1970s. One of the first and most popular personal computers was the Apple II, introduced in 1977 by Apple Computer.
- It can be defined as a small, relatively inexpensive computer designed for an individual user.
- In price, personal computers range anywhere from a few hundred pounds to over five thousand pounds.
- All are based on the microprocessor technology that enables manufacturers to put an entire CPU on one chip. Businesses use personal computers for word processing, accounting, desktop publishing, and for running spreadsheet and database management applications. At home, the most popular use for personal computers is for playing games and recently for surfing the Internet.
- The principal characteristics of personal computers are that they are single-user systems and are based on microprocessors.
- Personal computers are designed as single-user systems, it is common to link them together to form a network.

PERSONAL COMPUTER TYPES

Tower model- The term refers to a computer in which the power supply, motherboard, and mass storage devices are stacked on top of each other in a cabinet.

Desktop model- Desktop models designed to be very small are sometimes referred to as **slimline models**.

Notebook computer- An extremely lightweight personal computer. Notebook computers come with battery packs that enable you to run them without plugging them in.

Laptop computer- A small, portable computer -- small enough that it can sit on your lap. Nowadays, laptop computers are more frequently called notebook computers.

Subnotebook computer- A portable computer that is slightly lighter and smaller than a full-sized notebook computer. Typically, subnotebook computers have a smaller keyboard and screen, but are otherwise equivalent to notebook computers

Hand-held computer- Hand-held computers are also called PDAs, palmtops and pocket computers.

Palmtop- Palmtops that use a pen rather than a keyboard for input are often called hand-held computers or PDAs.

PDA- A typical PDA can function as a cellular phone, fax sender, and personal organizer. Unlike portable computers, most PDAs are pen-based, using a stylus rather than a keyboard for input.

2. FUNCTIONAL UNITS:

A computer consists of five functionally independent main parts input, memory, arithmetic logic unit (ALU), output and control unit.

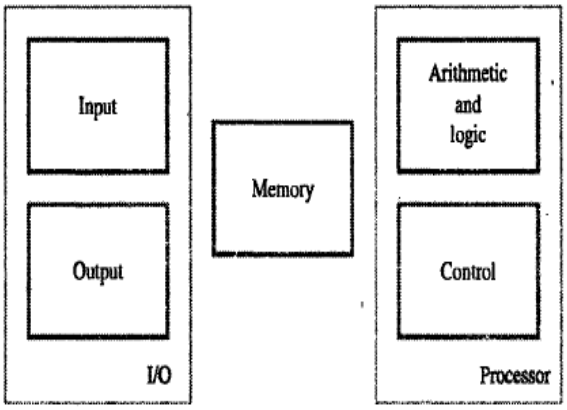


Fig: Basic Functional Units of a Computer

Input device accepts the coded information as source program i.e., high level language.This is either stored in the memory or immediately used by the processor to perform the desired operations. The program stored in the memory determines the processing steps. Basically the computer converts the source program to an object program i.e., into machine language (usingcompiler).

Finally the results are sent to the outside world through output device. All of the sections are coordinated by the control unit.

Input Unit:-

The high level language program, coded data is fed to a computer through input devices. Keyboard is a most common type. Whenever a key is pressed, one corresponding word or number is translated into its equivalent binary code over a cable & fed either to memory or processor. Microphone, Joystick, trackball, mouse, scanner are other input devices.

Memory Unit:-

Its function is to store programs and data. It is basically of two types:

1. **Primary memory**
2. **Secondary memory**

(i) Primary memory:- Is the one exclusively associated with the processor and operates at the electronics speeds. Programs must be stored in this memory while theyare being executed. The memory contains a large number of semiconductors storage cells, each capable of storing one bit of information. These are processed in a group of fixed size calledword.

To provide easy access to a word in memory, a distinct address is associated with eachword location. **Addresses are** numbers that identify memory location.

Number of bits in each word is called word length of the computer. Programs must reside in the memory during execution. Instructions and data can be written into the memory or read out under the control of processor.

Memory in which any location can be reached in a short and fixed amount of time afterspecifying its addresses is called Random-Access Memory (RAM). The time required to accessword in called memory access time.

Caches are the small fast RAM units, which are coupled with the processor and are often contained on the same IC chip to achieve high performance. Although primary storageis essential it tends to be expensive.

(ii) Secondary memory: -Is used where large amounts of data & programs have to be stored,particularly information that is accessed infrequently.

Examples:- Magnetic disks & tapes, optical disks(ie…CD-ROM's),flash drives etc.,

Arithmetic logic unit (ALU):-

Most of the computer operations are executed in ALU of the processor like addition, subtraction, division, multiplication, etc. the operands are brought into the ALU from memory and stored in high speed storage elements called registers. Then according to the instructions the operation is performed in the required sequence.

The control and the ALU are many times faster than other devices connected to a computer system. This enables a single processor to control a number of external devices suchas keyboards, displays, magnetic and optical disks, sensors and other mechanical controllers.

Output unit:-
This is the counter part of input unit. Its basic function is to send the processed resultsto the outside world.

Examples:- Printer, speakers, monitors are called as I/O units as they provide both an input and output functionality.

Control Unit:-
It effectively is the nerve center that sends signals to the run its and senses their states. The actual timing signals that govern the transfer of data between input unit, processor, memoryand output unit are generated by the Control unit.

3. BASIC OPERATIONAL CONCEPTS:

To perform a given task an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, whichexecutes the specified operations. Data to be used as operands are also stored in the memory.

Example:-Add LOCA,R_0

This instruction adds the operand at memory location LOCA, to operand in register $R_0$and places the sum into register.

This instruction requires the performance of several steps,

1. First the instruction is fetched from the memory into the processor.
2. The operand at LOCA is fetched and added to the contents of R_0.
3. Finally the resulting sum is stored in the register R_0.

The preceding add instruction combines a memory access operation with an ALU operations. In some other type of

computers, these two types of operations are performed byseparate instructions for performance reasons.

Load LOCA, R1

Add R1, R0

Transfers between the memory and the processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals. The data are then transferred to or from the memory.

The fig. show memory & the process or can be connected. In addition to the ALU & the controlcircuit, the processor contains a number of registers used for several different purposes.

Instruction Register (IR):- Holds the instruction that is currently being executed. Its output is available for the control circuits which generates the timing signals that control the various processing elements in one execution of instruction.

Program Counter (PC):-

This is another specialized register that keeps track of execution of a program. Itcontains the memory address of the next instruction to be fetched and executed Besides IR and PC, there are n-general purpose registers R0 through R_{n-1}.

The other two registers which facilitate communication with memory are:-

1. **MAR:-(Memory Address Register):-**It holds the address of the location to beaccessed.
2. **MDR:-(Memory Data Register):-**It contains the data to be written into or read out ofthe address location.

Operating steps are:

1. Programs reside in the memory & usually get there through the Input unit.
2. Execution of the program starts when the PC is set to point at the first instruction ofthe program.
3. Contents of PC are transferred to MAR and a Read Control Signal is sent to thememory.
4. After the time required to access the memory elapses, the address word is readoutOf the memory and loaded into the MDR.
5. Now contents of MDR are transferred to the IR & now the instruction is ready to bedecoded and executed.
6. If the instruction involves an operation by the ALU, it is necessary to obtain therequired operands.
7. An operand in the memory is fetched by sending its address to MAR & Initiating aread cycle.
8. When the operand has been read from the memory to the MDR, it is transferredFrom MDR to the ALU.
9. After one or two such repeated cycles, the ALU can perform the desired operation.
10. If the result of this operation is to be stored in the memory, the result is sent toMDR.
11. Address of location where the result is stored is sent to MAR & a write cycle isinitiated.

The contents of PC are incremented so that PC points to the next instruction that is to be executed. Normal execution of a program may be preempted (temporarily interrupted) if some devices require urgent servicing, to do this one device raises an Interrupt signal.

An interrupt is quest signal from an I/O device for service by the processor. The processor provides the requested service by executing an appropriate interrupt service routine.

The Diversion may change the internal stage of the processor. Its state must be saved in the memory location before interruption. When the interrupt-routine service is completed the state of the processor is restored so that the interrupted program may continue.

4 BUS STRUCTURES:

Single bus structure

In computer architecture, a bus is a subsystem that transfers data between components inside a

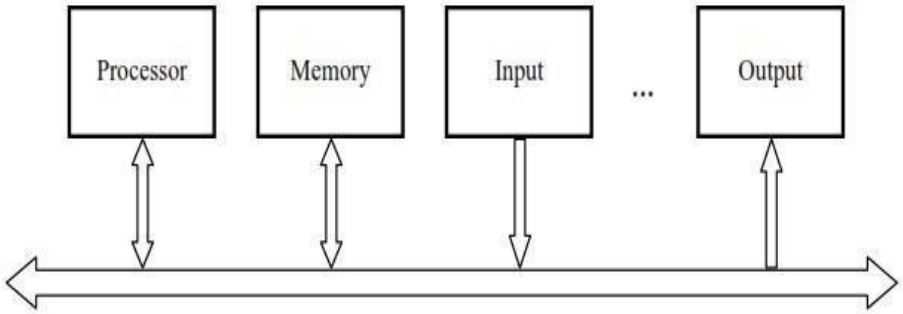


Figure 1.3.1 Single bus structure

computer, or between computers. Early computer buses were literally parallel electrical wires with multiple connections, but Modern computer buses can use both parallel and bit serial connections.

To achieve a reasonable speed of operation, a computer must be organized so that all its units can handle one full word of data at a given time.

When a word of data is transferred between units, all its bits are transferred in parallel, that is, the bits are transferred simultaneously over many wires, or lines, one bit per line.

A group of lines that serves as a connecting path for several devices is called a **bus.** In addition to the lines that carry the data, the bus must have lines for address and control purposes.

The simplest way to interconnect functional units is to use a single bus, All units are connected to this bus. Because the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time. Bus control lines are used to arbitrate multiple requests for use of the bus.

The main virtue of the single-bus structure is it's low cost and is flexibility for attaching peripheral" devices. Systems that contain multiple buses achieve more concurrency in operations by allowing two or more transfers to be carried out at the same time. This leads to better performance but at an increased cost.

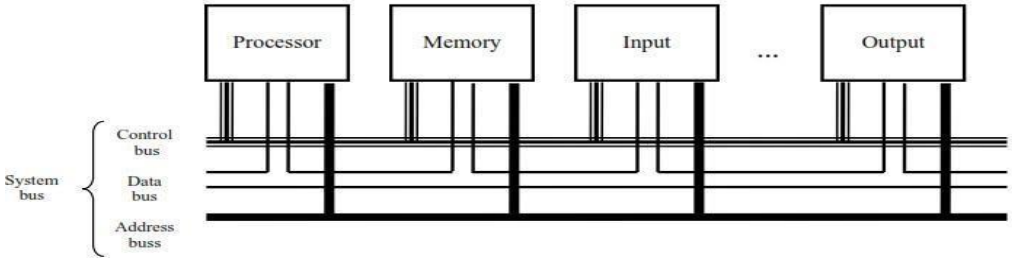


Figure 1.3.2 Bus interconnection scheme

The devices connected to a bus vary widely in their speed of operation. Some electromechanical devices, such as keyboards and printers are relatively slow. Other devices like magnetic or optical disks, are considerably faster.

Memory and processor units operate at electronic speeds, making them the fastest parts of a computer. Because all these devices must communicate with each other over a bus, an efficient transfer mechanism that is not constrained by the slow devices and that can be used to smooth out the differences in timing among processors, memories, and external devices is necessary.

A common approach is to include buffer registers with the devices to hold the information during transfers.

To illustrate this technique, consider the transfer of an encoded character from a processor to a character printer.

The processor sends the character over the bus to the printer buffer. Since the buffer is an electronic register, this transfer requires relatively little time.

Once the buffer is loaded, the printer can start printing without further intervention by the processor.

The bus and the processor are no longer needed and can be released for other activity. The printer continues printing the character in its buffer and is not available for further transfers until this process is completed.

Thus, buffer registers smooth out timing differences among processors, memories, and I/O devices. They prevent a high-speed processor from being locked to a slow I/O device during a sequence of data transfers.

This allows the processor to switch rapidly from one device to another, interweaving its processing activity with data transfers involving several I/O devices.

5 : SOFTWARE:

System software is a collection of programs that are executed as needed to perform functions such as

1.Receiving and interpreting user commands.
2.Entering and editing applications programs and storing them as files in secondary storage devices.
3.Managing the storage and retrieval of files in secondary storage devices.
4.Running standard application programs such as word processors , spreadsheets or games with data supplied by the user.
5. Controlling I/O units to receive input information and produce out put results.
6.Translating programs from source form prepared by the user into object form consisting of machine instructions.
7.Linking and running user-written application programs with existing standard library routines such as computation packages.

System software is responsible for the coordination of all activities in a computing system.

- **Application programs** are usually written in a high-level programming language, such as C,C++,Java, or Fortran , in which the programmer specifies mathematical or text processing operations.
- These operations are described in a format that is independent of the particular computer used to execute the program.
- A system software program called a **compiler** translates the high-level program language into a suitable machine language program containing instruction such as the **Add** and **Load** instructions.
- Another important system program that all programmers use is a **text editor.** It is used for entering and editing application programs.
- The user of this program interactively executes commands that allow statements of a source program entered at a keyboard to be accumulated in a file.
- A File is simply a sequence of alphanumeric characters or binary data is stored in memory or in secondary storage. A file can be referred to by a name chosen by the user.
- A key system software component called the operating system(os). This is a large program or actually a collection of routines i.e., used to control the sharing of and interaction among various computer units as they execute application programs.

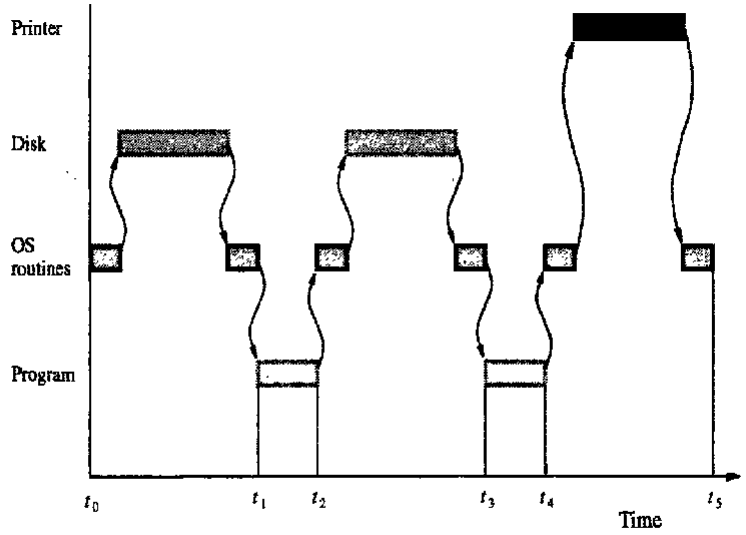


Figure 1.4 User program and OS routine sharing of the processor.

- Let us consider a system with one processor, one disk and one printer.
- Here the operating system manages the execution of more than one application program at the same time.
- Assume that the application program has been compiled from a high-level language form into a machine language form and stored on the disk.
- The first step is to transfer this file into the memory. After transfer is complete, execution of the program is started.
- The program's task involves reading a data file from the disk into the memory, performing some computation on the data and printing the results.
- When the execution of the program reaches the point where the data file is needed, the program requests the operating system to transfer the data file from the disk of the memory.
- The os performs this task and passes execution control back to the application program, which then proceeds to perform the required computation.
- When the computation is completed and the results are ready to be printed, the application program again sends a request to the operating system. An OS routine is then executed to cause the printer to print the results.

  - A convenient way to illustrate the sharing of the processor execution time by a time-line diagram, during the time period t0 to t1.
  - The transfer is completed, and the passes execution control to the application program. A similar pattern occurs from period t2 to t3 and t4 to t5.
  - At t5, the operating system may load and execute another application program.
  - The operating system manages the concurrent execution of several application programs to make the best possible use of computer resources.this pattern of concurrent execution is called **multiprogramming or multitasking.**

6. PERFORMANCE :

- **The most important measure of the performance of a computer is how quickly it can execute programs.**
- **The speed with which a computer executes program is affected by the design of its hardware and its machine language instructions.**

Computer performance is the amount of work accomplished by a computer system. The word performance in computer performance means "How well is the computer doing the work it is supposed to do?". It basically depends on response time, throughput and execution time of a computer system.
Response time is the time from start to completion of a task.

This also includes:
1. Operating system overhead.
2. Waiting for I/O and other processes

3. Accessing disk and memory
4. Time spent executing on the CPU or execution time.

Throughput is the total amount of work done in a given time.

CPU execution time is the total time a CPU spends computing on a given task. It also excludes time for I/O or running other programs. This is also referred to as simply CPU time.

Performance is determined by execution time as performance is inversely proportional to execution time.

- The total time required to execute the program in above figure t5-t0.This **elapsed time** is a measure of the performance of the entire computer system. It is effected by the speed of the processor, the disk and the printer.
- From figure The sum of these periods as the **processor time** needed to execute the program.
- The elapsed time for the execution of a program depends on all units in a computer system, the processor time depends on the hardware involved in the execution of the individual machine instructions.
- This hardware comprises the processor and the memory,which are usually connected by a bus as shown in below figure.
- At the start of execution, all program instructions and the required data are stored in the main memory. As execution proceeds, instructions are fetched one by one over the bus into the processor , and a copy is placed in the cache. When the execution of an instruction calls for data located in the main memory, the data are fetched and a copy is placed in the cache.
- The processor and a relatively small cache memory can be fabricated on a single integrated circuit chip. The internal speed of performing the basic steps of instruction processing on such ships is very high and is considerably faster than the speed at which instructions and data can be fetched form the main memory.

PROCESSOR CLOCK:

- Processor circuits are controlled by a timing signal called a clock. The clock defines regular time intervals , called clock cycles.
- To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps, such that each step can be completed in one clock cycle.
- The length P of one cycle is an important parameter that effects processor performance.
- Clock rate R=1/P
- The term million is denoted by the prefix Mega (M), Billion is denoted by the prefix Giga(G).
- Hence, 500 Million cycles per second is abbreviated to 500 Megahertz (MHz), and 1250 million cycles per second is abbreviated to 1.25 Gigahertz (GHz).

BASIC PERFORMANCE EQUATION:

$$T=NxS/R$$

T=Processor time .
N=the actual number of instruction executions.
S=Average number of basic steps needed to execute one machine instruction.
R=no of Cycles per second.

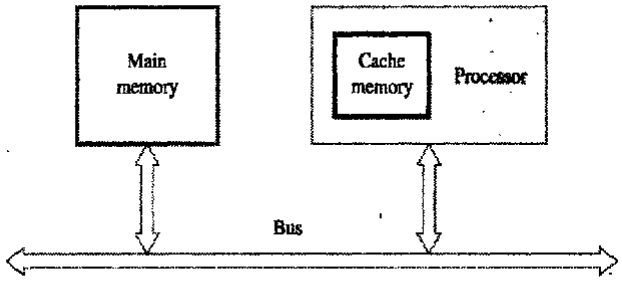


Figure 1.5 The processor cache.

7. MultiProcessor and Multicomputer:

Multiprocessor:

A Multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM. The main objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.

There are two types of multiprocessors, one is called shared memory multiprocessor and another is distributed memory.

multiprocessor. In shared memory multiprocessors, all the CPUs shares the common memory but in a distributed memory multiprocessor, every CPU has its own private memory.

Applications of Multiprocessor –

1. As a uniprocessor, such as single instruction, single data stream (SISD).
2. As a multiprocessor, such as single instruction, multiple data stream (SIMD), which is usually used for vector processing.
3. Multiple series of instructions in a single perspective, such as multiple instruction, single data stream (MISD), which is used for describing hyper-threading or pipelined processors.
4. Inside a single system for executing multiple, individual series of instructions in multiple perspectives, such as multiple instruction, multiple data stream (MIMD).

Benefits of using a Multiprocessor –

- Enhanced performance.
- Multiple applications.
- Multi-tasking inside an application.
- High throughput and responsiveness.
- Hardware sharing among CPUs.

2. Multicomputer:

A multicomputer system is a computer system with multiple processors that are connected together to solve a problem. Each processor has its own memory and it is accessible by that particular processor and those processors can communicate with each other via an interconnection network.

As the multicomputer is capable of messages passing between the processors, it is possible to divide the task between the processors to complete the task. Hence, a multicomputer can be used for distributed computing. It is cost effective and easier to build a multicomputer than a multiprocessor.

Difference between multiprocessor and Multicomputer:

1. Multiprocessor is a system with two or more central processing units (CPUs) that is capable of performing multiple tasks where as a multicomputer is a system with multiple processors that are attached via an interconnection network to perform a computation task.
2. A multiprocessor system is a single computer that operates with multiple CPUs where as a multicomputer system is a cluster of computers that operate as a singular computer.
3. Construction of multicomputer is easier and cost effective than a multiprocessor.
4. In multiprocessor system, program tends to be easier where as in multicomputer system, program tends to be more difficult.
5. Multiprocessor supports parallel computing, Multicomputer supports distributed computing.

PART-2: MACHINE INSTRUCTIONS AND PROGRAMS

1. NUMBERS:

A number system of base, or radix, r is a system that uses distinct symbols for r digits. Numbers are represented by a string of digit symbols.

To determine the quantity that the number represents, it is necessary to multiply each digit by an integer power of r and then form the sum of all weighted digits.

For example, the decimal number system in everyday use employs the radix 10 system. The 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The string of digits 724.5 is interpreted to represent the quantity $7*102+2*101+4*100+5*10-1$ i. e, 7 hundreds, plus 2 tens, plus 4 units, plus 5 tenths. Every decimal number can be similarly interpreted to find the quantity it represents. The binary number system uses the radix 2. The two digit symbols used are 0 and 1.The string of digits 101101 is interpreted to represent the quantity $1*25+0*24+1+23+1*22+0*21+1*20=45$

To distinguish between different radix numbers, the digits will be enclosed in parentheses and the radix of the number inserted as a subscript.

For example, to show the equality between decimal and binary forty-five we will write $(101101)_2 = (45)_{10}$.

Besides the decimal and binary number systems, the octal (radix 8) and hexadecimal (radix 16) are important in digital computer work.

The eight symbols of the octal system are 0, 1, 2, 3, 4, 5, 6, and 7. The 16 symbols of the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, 0, E, and F.

The last six symbols are, unfortunately, identical to the letters of the alphabet and can cause confusion at times.

However, this is the convention that has been adopted. When used to represent hexadecimal digits, the symbols A, B, C, D, E, F correspond to the decimal numbers 10, 11, 12, 13, 14, 15, respectively.

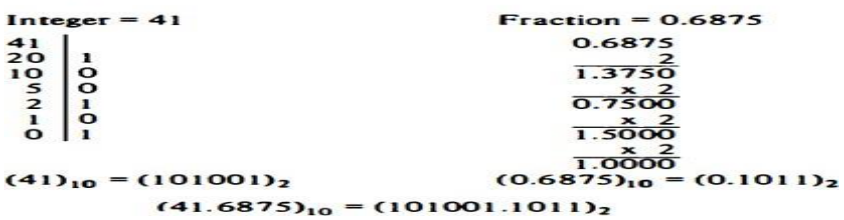
A number in radix r can be converted to the familiar decimal system by forming the sum of the weighted digits. For example, octal 736.4 is converted to decimal as follows: $(736.4)8=7*82+3*81+6*80+4*8-1=7*64+3*8+6*1+4/8=(478.5)10$

The equivalent decimal number of hexadecimal F3 is obtained from the following calculation: $(F3)16=F*16+3=15*16+3=(243)10$

Conversion from decimal to its equivalent representation in the radix r system is carried out by separating the number into its integer and fraction parts and converting each part separately. The conversion of a decimal integer into a base r representation is done by successive divisions by r and accumulation of the remainders.

The conversion of a decimal fraction to radix r representation is accomplished by successive multiplications by r and accumulation of the integer digits so obtained. Figure below demonstrates these procedures.

Figure Conversion of decimal 41.6875 into binary.



- **The conversion of decimal 41.6875** into binary is done by first separating the number into its integer part 41 and fraction part .6875.

The integer part is converted by dividing 41 by r = 2 to give an integer quotient of 20 and a remainder of 1 The quotient is again divided by 2 to give a new quotient and remainder.

This process is repeated until the integer quotient becomes 0. The coefficients of the binary number are obtained from the remainders with the first remainder giving the low-order bit of the converted binary number.

- **The fraction part** is converted by multiplying it by r = 2 to give an integer and a fraction. The new fraction (without the integer) is multiplied again by 2 to give a new integer and a new fraction.

This process is repeated until the fraction part becomes zero or until the number of digits obtained gives the required accuracy.

The coefficients of the binary fraction are obtained from the integer digits with the first integer computed being the digit to be placed next to the binary point.

Finally, the two parts are combined to give the total required conversion.

OCTAL AND HEXADECIMAL NUMBERS

The conversion from and to binary, octal, and hexadecimal representation plays an important part in digital computers.

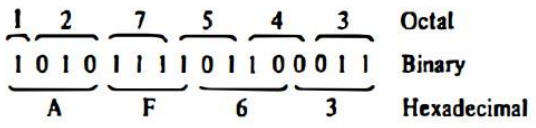
- **Since $2^3 = 8$ and $2^4 = 16$**, each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits.

The conversion from binary to octal is easily accomplished by partitioning the binary number into groups of three bits each. The corresponding octal digit is then assigned to each group of bits and the string of digits so obtained gives the octal equivalent of the binary number.

- **Starting from the low-order bit**, we partition the register into groups of three bits each (the sixteenth bit remains in a group by itself).

Each group of three bits is assigned its octal equivalent and placed on top of the register. The string of octal digits so obtained represents the octal equivalent of the binary number.

Conversion from binary to hexadecimal is similar except that the bits are divided into groups of four.



- **Figure** Binary, octal, and hexadecimal conversion.

TABLE Binary-Coded Octal Numbers

Octal number	Binary-coded octal	Decimal equivalent	
0	000	0	↑
1	001	1	
2	010	2	Code
3	011	3	for one
4	100	4	octal
5	101	5	digit
6	110	6	
7	111	7	↓
10	001 000	8	
11	001 001	9	
12	001 010	10	
24	010 100	20	
62	110 010	50	
143	001 100 011	99	
370	011 111 000	248	

TABLI Binary-Coded Hexadecimal Numbers

Hexadecimal number	Binary-coded hexadecimal	Decimal equivalent	
0	0000	0	↑
1	0001	1	
2	0010	2	
3	0011	3	
4	0100	4	
5	0101	5	
6	0110	6	Code
7	0111	7	for one
8	1000	8	hexadecimal
9	1001	9	digit
A	1010	10	
B	1011	11	
C	1100	12	
D	1101	13	
E	1110	14	
F	1111	15	↓
14	0001 0100	20	
32	0011 0010	50	
63	0110 0011	99	
F8	1111 1000	248	

The registers in a digital computer contain many bits. Specifying the content of registers by their binary values will require a long string of binary digits.

It is more convenient to specify content of registers by their octal or hexadecimal equivalent.

The number of digits is reduced by one-third in the octal designation and by one-fourth in the hexadecimal designation. For example, the binary number 1111 1111 1111 has 12 digits.

- **It can be expressed** in octals as 7777 (four digits) or in hexadecimal as FFF (three digits).

- **Computer manuals invariably** choose either the octal or the hexadecimal designation for specifying contents of registers.

DECIMAL REPRESENTATION

- **The binary number system** is the most natural system for a computer, but people are accustomed to the decimal system.

One way to solve this conflict is to convert all input decimal numbers into binary numbers, let the computer perform all arithmetic operations in binary and then convert the binary results back to decimal for the human user to understand.

However, it is also possible for the computer to perform arithmetic operations directly with decimal numbers provided they are placed in registers in a coded form.

- **Decimal numbers enter the computer** usually as binary-coded alphanumeric characters. These codes, introduced later, may contain from six to eight bits for each decimal digit. When decimal numbers are used for internal arithmetic computations, they are converted to a binary code with four bits per digit.

- **A binary code is a group of n bits** that assume up to 2^n distinct combinations of 1's and 0's with each combination representing one element of the set that is being coded.

  A binary code will have some unassigned bit combinations if the number of elements in the set is not a multiple power of 2. The 10 decimal digits form such a set. A binary code that distinguishes among 10 elements must contain at least four bits, but six combinations will remain unassigned.

  Numerous different codes can be obtained by arranging four bits in 10 distinct combinations. The bit assignment most commonly used for the decimal digits is the straight binary assignment listed in the first 10 entries of Table below.

- **This particular code** is called binary-coded decimal and is commonly referred to by its abbreviation BCD.

  It is very important to understand the difference between the conversion of decimal numbers into binary and the binary coding of decimal numbers.

  For example, when converted to a binary number, the decimal number 99 is represented by the string of bits 1100011, but when represented in BCD, it becomes 1001 1001 .

- **The only difference** between a decimal number represented by the familiar digit symbols 0, 1, 2, ... , 9 and the BCD symbols 0001, 0010, ... , 1001 is in the symbols used to represent the digits-the number itself is exactly the same.

TABLE Binary-Coded Decimal (BCD) Numbers

Decimal number	Binary-coded decimal (BCD) number	
0	0000	
1	0001	
2	0010	
3	0011	Code for one decimal digit
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	1001	
10	0001 0000	
20	0010 0000	
50	0101 0000	
99	1001 1001	
248	0010 0100 1000	

ALPHANUMERIC REPRESENTATION

Many applications of digital computers require the handling of data that consist not only of numbers, but also of the letters of the alphabet and certain special characters.

- **An alphanumeric character set** is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet and a number of special characters, such as $, + , and =.

- **Such a set contains between 32 and 64 elements** (if only uppercase letters are included) or between 64 and 128 (if both uppercase and lowercase letters are included).

- **In the first case**, the binary code will require six bits and in the second case, seven bits. The standard alphanumeric binary code is the ASCII (American Standard Code for Information Interchange), which uses seven bits to code 128 characters.

- **The binary code** for the uppercase letters, the decimal digits, and a few special characters is listed in Table below.

  Note that the decimal digits in ASCII can be converted to BCD by removing the three high-order bits, 011.

  Binary codes play an important part in digital computer operations. The codes must be in binary because registers can only hold binary information. One must realize that binary codes merely change the symbols, not the meaning of the discrete elements they represent.

- **The operations** specified for digital computers must take into consideration the meaning of the bits stored in registers so that operations are performed on operands of the same type.

TABLE American Standard Code for Information Interchange (ASCII)

Character	Binary code	Character	Binary code
A	100 0001	0	011 0000
B	100 0010	1	011 0001
C	100 0011	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011		
L	100 1100		
M	100 1101	space	010 0000
N	100 1110	.	010 1110
O	100 1111	(	010 1000
P	101 0000	+	010 1011
Q	101 0001	$	010 0100
R	101 0010	*	010 1010
S	101 0011	)	010 1001
T	101 0100	-	010 1101
U	101 0101	/	010 1111
V	101 0110	,	010 1100
W	101 0111	=	011 1101
X	101 1000		
Y	101 1001		
Z	101 1010		

2. ARITHMETIC OPERATIONS AND PROGRAMS:

The ALU is the core of the computer - it performs arithmetic operations on data that not only realize the goals of various applications (e.g., scientific and engineering programs), but also manipulate addresses (e.g., pointer arithmetic).

NUMBER REPRESENTATION

We obviously need to represent both positive and negative numbers. Three systems are used for representing such numbers :

- Sign-and-magnitude
- 1's-complement
- 2's-complement

In all three systems, the leftmost bit is 0 for positive numbers and 1 for negative numbers. Fig 2.1 illustrates all

three representations using 4-bit numbers.

Positive values have identical representations in al systems, but negative values have different representations.

In the sign-and-magnitude systems, negative values are represented by changing the most significant bit (b3 in figure 2.1) from 0 to 1 in the B vector of the corresponding positive value.

B	Values represented		
b3b2b1b0	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	-0	-0	-0
1 0 0 1	-1	-1	-1
1 0 1 0	-2	-2	-2
1 0 1 1	-3	-3	-3
1 1 0 0	-4	-4	-4
1 1 0 1	-5	-5	-5

Hence, the 2's complement of a number is obtained by adding 1 to the 1's complement of that number.

Addition of Positive numbers:-

Consider adding two 1-bit numbers. The results are shown in figure 2.2. Note that the sum of 1 and 1 requires the 2-bit vector 10 to represent the value 2. We say that the sum is 0 and

the carry-out is 1. In order to add multiple-bit numbers, we use a method analogous to that used for manual computation with decimal numbers.

We add bit pairs starting from the low-order (right) and of the bit vectors, propagating carries toward the high- order (left) end.

0	1	0	1
+ 0	+ 0	+ 1	+ 1
0	1	1	1 0

Carry-out

Figure 2.2 Addition of 1-bit numbers.

Addition and Substraction of signed numbers:

There are three systems for representing positive and negative numbers or signed numbers.
The sign-magnitude system is the simplest representation, but it is also the awkward for addition and substraction operations.

1. 1's –complement method
2. 2's complement methos for performing addition and substraction operations.

-

- **Sign Magnitude**

Sign magnitude is a very simple representation of negative numbers. In sign magnitude the first bit is dedicated to represent the sign and hence it is called sign bit.
Sign bit '1' represents negative sign.
Sign bit '0' represents positive sign.
In sign magnitude representation of a n – bit number, the first bit will represent sign and rest n-1 bits represent magnitude of number.

For example,

- +25 = 011001

Where 11001 = 25
And 0 for '+'

- -25 = 111001

Where 11001 = 25
And 1 for '-'.

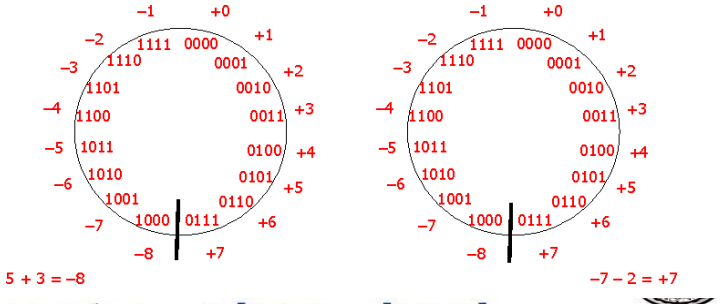
Range of number represented by sign magnitude method = $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$ (for n bit number)

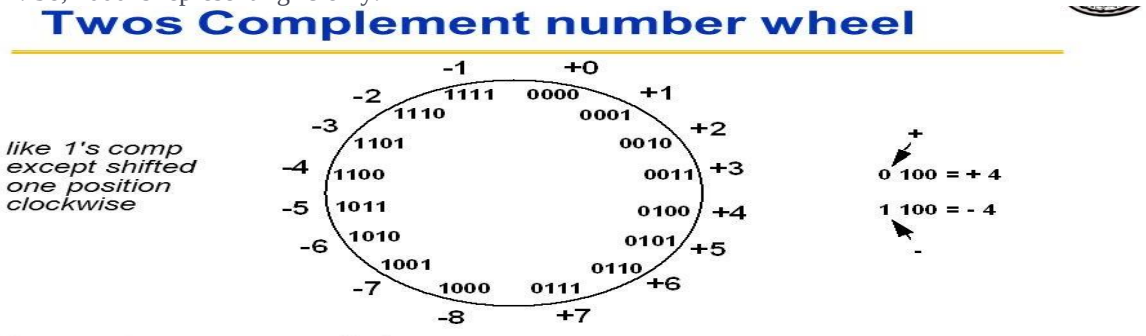
But there is one problem in sign magnitude and that is we have two representations of 0

+0 = 000000
– 0 = 100000

- **2's complement method**

To represent a negative number in this form, first we need to take the 1's complement of the number represented in simple positive binary form and then add 1 to it.
For example:
(-8)10 = (1000)2
1's complement of 1000 = 0111
Adding 1 to it, 0111 + 1 = 1000
So, (-8)10 = (1000)2
Please don't get confused with (8)10 =1000 and (-8)10=1000 as with 4 bits, we can't represent a positive number more than 7. So, 1000 is representing -8 only.





Range of number represented by 2's complement = $(-2^{n-1}$ to $2^{n-1}-1)$

->The rules governing the addition and substraction of n-bit signed numbers using the 2's complement representation system.
1. To add two numbers , add their n-bit representations, ignoring the carry-out signal from the most significant bit (MSB) position.The sum will be the algebraically correct value in the 2's complement representation as long as the answer is in the range 2n-1 through +2n-1 -1.

2. To substract two numbers, X and Y i.e., to perform X-Y , form the 2's complement of Y and then add it to X, as in rule 1 . Again the result will be the algebraically correct value in the 2's complement representation system if the answer is in the range – 2n-1 through +2n-1-1.

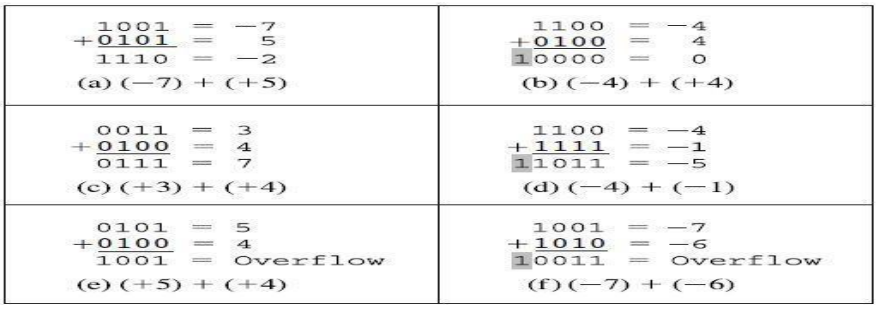


Figure 9.3 Addition of Numbers in Twos Complement Representation

10001 − 11101 = 101110:

	1				1
	1	0	0	0	1
−	1	1	1	0	1
1	0	1	1	1	0

1110 − 1111 = 11101:

1	1	1		
	1	1	1	0
−	1	1	1	1
1	1	1	0	1

3. INSTRUCTIONS AND INSTRUCTION SEQUENCING

A computer must have instructions capable of performing four types of operations.

- Data transfers between the memory and the processor registers.
- Arithmetic and logic operations on data.
- Program sequencing and control.
- I/O transfers.

(REGISTER TRANSFER NOTATION):-

Transfer of information from one location in the computer to another. Possible locations that may be involved in such transfers are memory locations that may be involved in such transfers are memory locations, processor registers, or registers in the I/O sub system.

Most of the time, we identify a location by a symbolic name standing for its hardware binary address.

Example, names for the addresses of memory locations may be LOC, PLACE, A, VAR2; processor registers names may be R0, R5; and I/O register names may be DATAIN, OUTSTATUS, and so on.

The contents of a location are denoted by placing square brackets around the name ofthe location. Thus, the expression:

R1 [LOC]

Means that the contents of memory location LOC are transferred into processor registerR1.

As another example, consider the operation that adds the contents of registers R1 and R2, and then places their sum into register R3. This action is indicated as:

R3 [R1] + [R2]

This type of notation is known as Register Transfer Notation (RTN). Note that the right-hand side of an RTN expressional ways denotes a value, and the left-hand side is the name of a location where the value is to be places, over writing the old contents of that location.

(ASSEMBLY LANGUAGE NOTATION):-

Another type of notation to represent machine instructions and programs. For this, we use an assembly language format. For example, an instruction that causes the transfer describedabove, from memory

location LOC to processor register R1, is specified by the statement.

Move LOC, R1

The contents of LOC are unchanged by the execution of this instruction, but the oldcontents of register R1 are over written.

These example of adding two numbers contained in processor registers R1 and R2and placing their sum in R3 can be specified by the assembly language statement.

Add R1, R2, R3

(INSTRUCTION EXECUTION AND STRAIGHT-LINES EQUENCING):-

In the preceding discussion of instruction formats, we used to taskC □ [A] + [B] for illustration.

The following fig. shows a possible program segment for this task as it appears in the memory of a computer.

We have assumed that the computer allows one memory operand per instruction and hasa number of processor registers.

The three instructions of the program are in successive word locations, starting at location i.e. since each instruction is 4 bytes long, the second and third instructions start at addresses i+4 and i+8.

Let us consider how this program is executed. The processor contains a registercalled the program counter (PC), which holds the address of the instruction to be executed next. To begin executing a program, the address of its first instruction (I in our example) must be placed into the PC.

Then, the processor control circuits use the information in the PC to fetch and executeinstructions, one at a time, in the order of increasing addresses.

This is called straight-line sequencing. During the execution of each instruction, thePC is incremented by 4 to point to then extinstruction.

Thus, after the Move instruction at location i+8 is executed, the PC contains the valuei+12, which is the address of the first instruction of then ext program segment.

Executing a given instruction is a two-phase procedure. In the first phase, calledinstruction fetch, the instruction is fetched from the memory location whose address is in the PC. This instruction is placed in the instruction register (IR) in the processor.

The instruction in IR is examined to determine which operation is to be performed. The specified operation is then performed by the processor.

This often involves fetching operands from the memory or from processor registers,performing an arithmetic or logic operation, and storing the result in the destination location.

(BRANCHING)**:-**

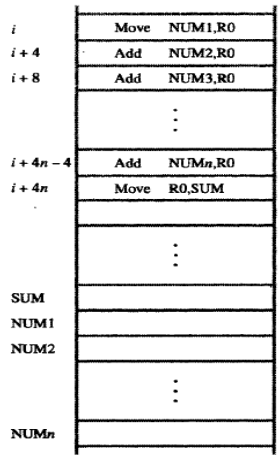


Fig (a): A straight line program for adding n numbers

Consider the task of adding a list of n numbers. The program is shown in fig(a). Instead of usinga long list of add instructions, it is possible to place a single add instruction in a program loop, as shown in fig (b).

The loop is a straight-line sequence of instructions executed as many times as needed.

It starts at location LOOP and ends at the instruction Branch>0.

During each pass through this loop, the address of then ext list entry is determined, and that entryis fetched and added to R0.

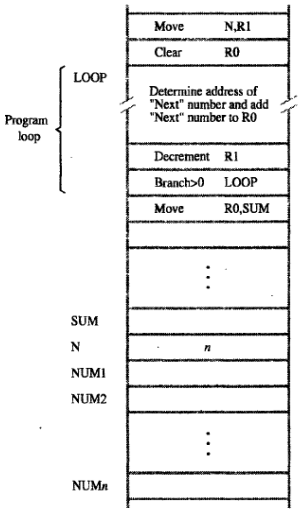


Fig (b) Using a loop to add n numbers

Assume that the number of entries in the list, n is stored in memory location N, as shown.

Register R1 is used as a counter to determine the number of times the loop is executed.Hence, the contents of location N are loaded intoregisterR1atthebeginning of the program. Then, with in the body of the loop, the Instruction.

Decrement R1 Reduces the contents of R1 by 1 each time through the loop. This type of instruction loads a new value into the program counter.

As a result, the processor fetches and executes the instruction at this new address, called thebranch target, instead of the instruction at the location that follows the branch instruction in sequential address order.A condition a branch instruction causes a branch only if a specified condition is satisfied. If thecondition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed.

Branch > 0 LOOP

(branch if greater that 0) is a conditional branch instruction that causes a branch to locationLOOP if the result of the immediately preceding instruction, which is the decremented value in register R1, is greater than zero. This means that the loop is repeated, as long as there are entriesin the list that are yet to be added to R0.

At the end of the pass through the loop, the Decrement instruction produces a value of zero, and hence, branching does not occur.

(CONDITION CODES):-

The processor keeps track of information about the results of various operations for use bysubsequent conditional branch instructions.

This is accomplished by recording the required information in individual bits, oftencalled condition code flags.

These lags are usually grouped together in a special processor register called thecondition code register or status register.

Individual condition code flags are set to 1 or cleared to 0, depending on the outcomeof the operation performed.

Four commonly used flags are:

N(negative) Set to1; if the result is negative; otherwise, cleared to 0.

Z(zero) Set to 1; if the result is 0; otherwise, cleared to 0.

V(overflow) Set to1; if arithmetic overflow occurs; otherwise, cleared to 0.

C(carry) Set to1; if a carry-out results from the operation; otherwise, cleared to 0.

The instruction Branch>0, discussed in the previous section, is an example of a branchinstruction that tests one or more of the condition flags.

It causes a branch if the value tested is neither negative nor equal to zero. That is the branch is taken if neither N nor Z is1.The conditions are given as logic expressions involving the condition code flags.

In some computers, the condition code flags are affected automatically by instructions that perform arithmetic or logic operations. However, this is not always the case. A number of computers have two versions of an Add instruction.

4.ADDRESSING MODES:

The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

1. Register addressing mode - The operand is the contents of a processor register; the name (address) of the register is given in the instruction.

Example: **MOVE R1,R2**

This instruction copies the contents of register R2 to R1.

2. Absolute addressing mode - The operand is in a memory location; the address of this location is given explicitly in the instruction. (In some assembly languages, this mode is called Direct.

Example: **MOVE LOC,R2**

This instruction copies the contents of memory location of LOC to register R2.

3. Immediate addressing mode - The operand is given explicitly in the instruction. Example: **MOVE #200 , R0**

The above statement places the value 200 in the register R0. A common convention is to use the sharp sign (#) in front of the value to indicate that this value is to be used as an immediate operand.

4. Indirect addressing mode - The effective address of the operand is the contents of a register or memory location whose address appears in the instruction.

Example **Add (R2),R0**

Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2. The initialization section of the program loads the counter value n from memory location N into Rl and uses the Immediate addressing mode to place the address value NUM 1, which is the address of the first number in the list, into R2.

5. Index mode - The effective address of the operand is generated by adding a constant value to the contents of a register. The register used may be either a special register provided for this purpose, or, more commonly; it may be anyone of a set of general-purpose registers in the processor. In either case, it is referred to as an index register. We indicate the Index mode symbolically as **X(Ri).** Where X denotes the constant value contained in the instruction and Ri is the name of the register involved. The effective address of the operand is given by

EA = X + [Ri]. The contents of the index register are not changed in the process of generating the effective address.

Relative Addressing

We have defined the Index mode using general-purpose processor registers. A useful version of this mode is obtained if the program counter, PC, is used instead of a general purpose register. Then, X(PC) can be used to address a memory location that is X bytes away from the location presently pointed to by the program counter. Since the addressed location is identified "relative" to the program counter, which always identifies the current execution point in a program, the name Relative mode is associated with this type of addressing.

6.Relative mode - The effective address is determined by the Index mode using the program counter in place of the general-purpose register Ri.

This mode can be used to access data operands. But, its most common use is to specify the target address in branch instructions. An instruction such as **Branch>O LOOP** causes program

execution to go to the branch target location identified by the name LOOP if the branch condition is satisfied. This location can be computed by specifying it as an offset from the current value of the program counter. Since the branch target may be either before or after the branch instruction, the offset is given as a signed number.

7. Autoincrement mode - The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list. We denote the Autoincrement mode by putting the specified register in parentheses, to show that the contents of the register are used as the effective address, followed by a plus sign to indicate that these contents are to be incremented after the operand is accessed. Thus, the Autoincrement mode is written as **(Ri) +**

As a companion for the Autoincrement mode, another useful mode accesses the items of a list in the reverse order:

8. Autodecrement mode - The contents of a register specified in the instruction is first automatically decremented and is then used as the effective address of the operand. We denote the Autodecrement mode by putting the specified register in parentheses, preceded by a minus sign to indicate that the contents of the register are to be decremented before being used as the effective address. Thus, we write **-(Ri)**

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	Ri	EA = Ri
Absolute (Direct)	LOC	EA = LOC
Indirect	(Ri)	EA = [Ri]
	(LOC)	EA = [LOC]
Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri,Rj)	EA = [Ril + [Rj]
Base with index and offset	X(Ri,Rj)	EA = [Ri] + [Rj] + X
Relative	X(PC)	EA = [PC] +X
Autoincrement	(Ri)+	EA = [Ri]; Increment Ri
Autodecrement	-(Ri)	Decrement Ri; EA = [Ri]

EA = effective address Value = a signed number

Table 1.1 Generic addressing modes

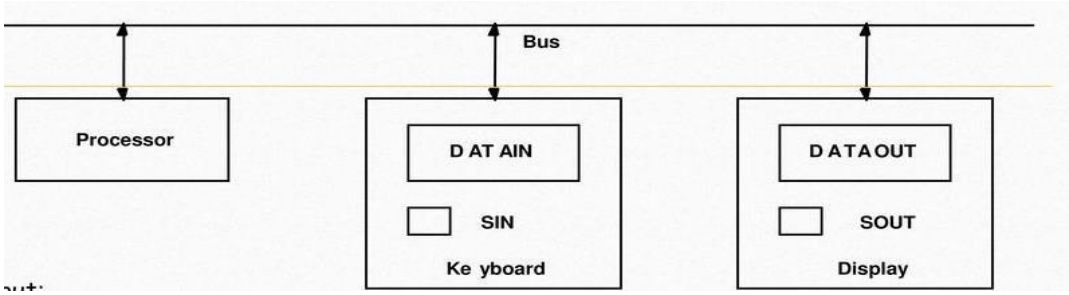
6. BASIC INPUT/OUTPUT OPERATIONS:

- Transfer information between the processor and the memory. Perform arithmetic and logic operations Program sequencing and flow control.
- Input/Output operations which transfer data from the processor or memory to and from the real world are essential.
- In general, the rate of transfer from any input device to the processor, or from the processor to any output device is likely to the slower than the speed of a processor.

The difference in speed makes it necessary to create mechanisms to synchronize the data transfer between them.

Let us consider a simple task of reading a character from a keyboard and displaying that character on a display screen.

- A simple way of performing the task is called **program-controlled I/O.**
- There are two separate blocks of instructions in the I/O program that perform this task.
- One block of instructions transfers the character into the processor.
- Another block of instructions causes the character to be displayed.



Input:

- When a key is struck on the keyboard, an 8-bit character code is stored in the buffer register DATAIN.
- A status control flag SIN is set to 1 to indicate that a valid character is in DATAIN.
- A program monitors SIN, and when SIN is set to 1, it reads the contents of DATAIN.

- When the character is transferred to the processor, SIN is automatically cleared.Initial state of SIN is 0.

Output:

- When SOUT is equal to 1, the display is ready to receive a character.
- A program monitors SOUT, and when SOUT is set to 1, the processor transfers a character code to the buffer DATAOUT.
- Transfer of a character code to DATAOUT clears SOUT to 0.
- Initial state of SOUT is 1.

Buffer registers DATAIN and DATAOUT, and status flags SIN and SOUT are part of a circuitry known as **device interface**.

To perform I/O transfers, we need machine instructions to:

1.Monitor the status of the status registers.

2. Transfer data among the I/O devices and the processor.

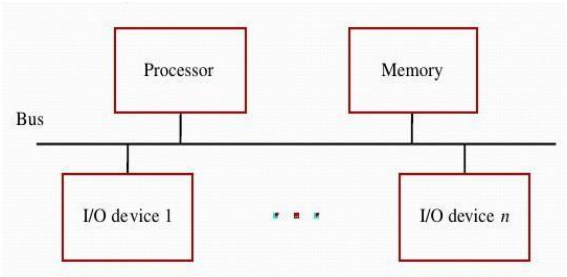
- ✓ Instructions have similar format to the instructions used for moving data between the the processor and the memory.

How does the processor address the I/O devices:

- ✓ Some memory address values may be used to refer to peripheral device buffer registers such as DATAIN and DATAOUT.

This arrangement is called **as Memory-Mapped I/O**.

- Multiple I/O devices may be connected to the processor and the memory via a bus.
- Bus consists of three sets of lines to carry address, data and control signals
- Each I/O device is assigned an unique address.
- To access an I/O device, the processor places the address on the address lines.
- The device recognizes the address, and responds to the control signals.



I/O devices and the memory may share the same address space:

- Memory-mapped I/O.
- Any machine instruction that can access memory can be used to transfer data to or from an I/O device.
- Simpler software.

I/O devices and the memory may have different address spaces:

- Special instructions to transfer data to and from I/O devices.
- I/O devices may have to deal with fewer address lines.
- I/O address lines need not be physically separate from memory address lines.
- In fact, address lines may be shared between I/O devices and memory, with a control signal to indicate whether it is a memory address or an I/O address.

Program-controlled I/O:Processor repeatedly monitors a status flag to achieve the necessary synchronization.Processor polls the I/O device.Two other mechanisms used for synchronizing data transfers between the processor and memory:

1.Interrupts.

2.Direct Memory Access

In program-controlled I/O, when the processor continuously monitors the status of the device, it does not perform any useful tasks.

An alternate approach would be for the I/O device to alert the processor when it becomes ready.Do so by sending a hardware signal called an interrupt to the processor.
At least one of the bus control lines, called an interrupt-request line is dedicated for this purpose.

Processor can perform other useful tasks while it is waiting for the device to be ready.

7.STACKS AND QUEUES:

Stack: A stack is a list of data elements, usually words, or bytes ,with the accessing restriction that elements can be added or removed at one end of the list only. This end is called the top of the stack and the other end is called the bottom. The structure is sometimes referred to as a **pushdown stack.**

- Last-in-First-out stack is the last data item placed on the stack is the first one removed when retrieval begins.
- A stack is a limited access data structure - elements can be added and removed from the stack only at the top.
- push adds an item to the top of the stack, pop removes the item from the top.
- In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack.
- A Processor register is used to keep track of the address of the element of the stack i.e., at the top at any given time. This register is called the stack pointer (SP).It could be one of the general-purpose registers or a register dedicated to this function.
- A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.
- If we assume a byte-addressable memory with a 32-bit word length, the push operation can be implemented as

```
Substract #4,SP

Move newitem,(SP)
```

Where the substract instruction substracts the source operand 4 from the destination operand contained in SP and places the result in SP.

```
Move (SP),ITEM

Add       #4,SP
```

Move NEWITEM,-(SP)

These two instructions move the top value from the stack into location ITEM and then increment the stack pointer by 4 so that it points to the new top element. If the processor has the Autoincrement and autodecrement addressing modes,then the push operation can be performed by the single instruction.

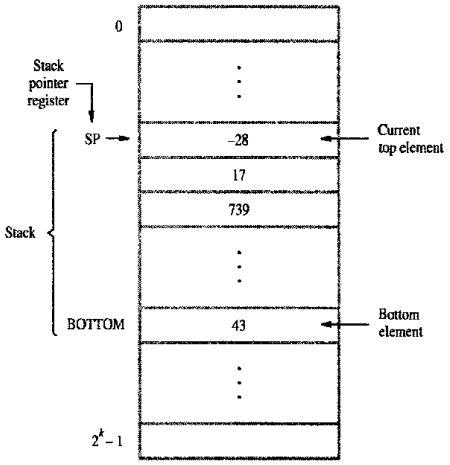


Figure 2.21 A stack of words in the memory.

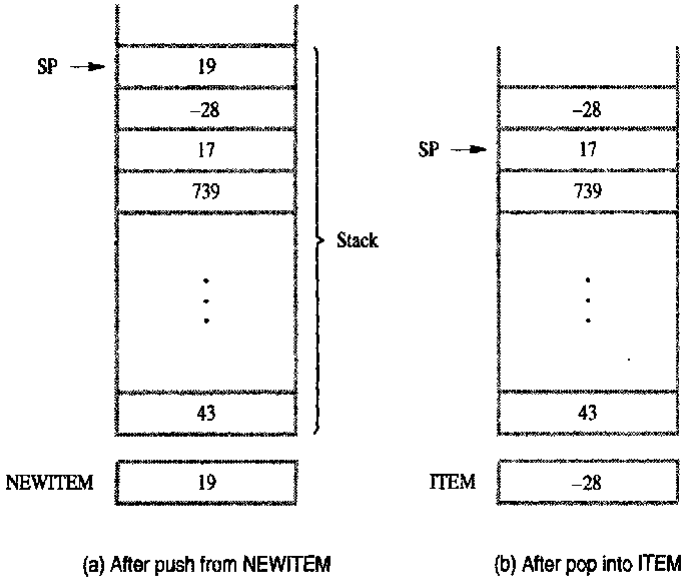
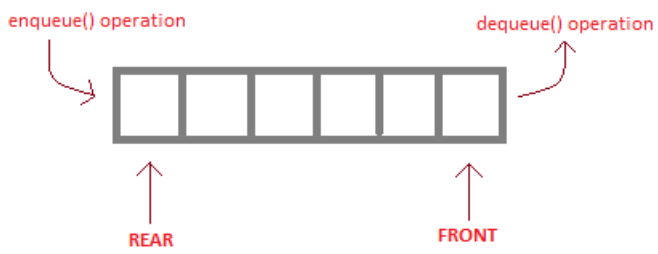


Figure 2.22 Effect of stack operations on the stack in Figure 2.21.

Queue:

- Another useful data structure i.e., similar to the stack is called a queue.
- Data stored and retrieved from a queue on a first-in-first-out basis.
- An excellent example of a queue is a line of students in the food court of the UC.
- New additions to a line made to the back of the queue, while removal (or serving) happens in the front. In the queue only two operations are allowed enqueue and dequeue.
- Enqueue means to insert an item into the back of the queue, dequeue means removing the front item.
- A queue would continuously move through the memory of a computer in the direction of higher addresses. One way to limit the queue to a fixed memory region is to use a circular buffer.
- The first entry in the queue entered into location BEGINNING, and the successive entries are appended to the queue by entering them at successively higher addresses, By the time the back of the queue reaches END.
- The picture demonstrates the FIFO access. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.



Queue: First In First Out (FIFO): The first object into a queue is the first object to leave the queue, used by a queue.

Stack: Last In First Out (LIFO): The last object into a stack is the first object to leave the stack, used by a stack OR

Stack: Last In First Out (FILO): The First object or item in a stack is the last object or item to leave the stack.

Queue: Last In First Out (LILO): The last object or item in a queue is the last object or item to leave the queue.

8.SUBROUTINES

- In a given program, it is often necessary to perform a particular sub task many times on different data-values. Such as sub task is usually called as subroutine.
- For example, a subroutine may evaluate the sine function or sort a list of values into increasing or decreasing order.
- It is possible to include the block of instructions that constitute a subroutine at every placewhere it is needed in the program.
- However, to save space, only one copy of the instructions that constitute the subroutineis placed in the memory, and any program that requires the use of the subroutine simply branches to its starting location.
- When a program branches to a subroutine we say that it is calling the subroutine. The instruction that performs this branch operation is named a Call instruction.
- After a subroutine has been executed, the calling program must resume execution, continuing immediately after the instruction that called the subroutine.
- The subroutine is said to return to the program that called it by executing a Return instruction. The way in which a computer makes it possible to call and return from subroutines is referred to as its subroutine linkage method.
- The simplest subroutine linkage method is to save the return address in a specific location, which may be a register dedicated to this function.
- Such as register is called the link register. When the subroutine complete sits task, the Return instruction returns to the calling program by branching in directly through the link register.
- The Call instruction is just a special branch instruction that performs the following operations:
    - Branch to the target address specified by the instruction.
    - The Return instruction is a special branch instruction that performs the operation.
    - Branch to the address contained in the link register.

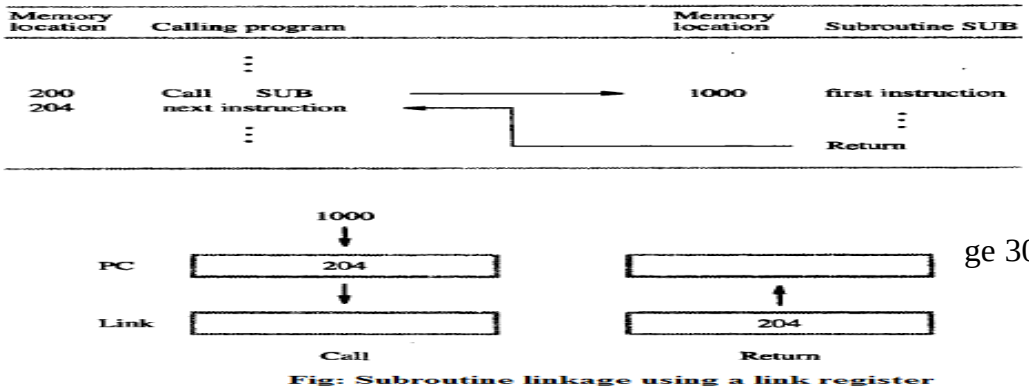


Fig: Subroutine linkage using a link register

SUBROUTINE NESTING AND THE PROCESSOR STACK:-

- A common programming practice, called subroutine nesting, is to have one subroutine call another. In this case, the return address of the second call is also stored in the link register, destroying its previous contents.
- Hence, it is essential to save the contents of the link register in some other location before calling another subroutine. Otherwise, the return address of the first subroutine will be lost.
- Subroutine nesting can be carried out to any depth. Eventually, the last subroutine called completes its computations and returns to the subroutine that called it. The return addressneeded for this first return is the last one generated in the nested call sequence. That is, return addresses are generated and used in a last-in-first-out order.
- This suggests that the return addresses associated with subroutine calls should be pushed ontoa stack. A particular register is designated as the stack pointer, SP, to be used in this operation. Thestack pointer points to a stack called the processor stack.
- The Call instruction pushes the contents of the PC onto the processor stack and loads the subroutine address into the PC.
- The Return instruction pops the return address from the processor stack into the PC.

PARAMETER PASSING :

When calling a subroutine , a program must provide to the subroutine the parameters i.e., the operands or their addresses , to be used in the computation.This exchange of information between a calling program and a subroutine is referred to as parameter passing.

- Parameter passing may be accomplished in several ways. The parameters may be placed in registers or in memory locations, where they can be accessed by the subroutine.
- Passing parameters through processor registers in straightforward and efficient for adding a list of numbers can be implemented as a subroutine, with the parameter passed through registers.
- The purpose of the subroutines is to add a list of numbers. Instead of passing the actual list entries , the calling program passes the address of the first number in the list. This technique is called passing by reference.
- The second parameter is passed by value, The actual number of entries , n is passed to the subroutine.

Calling program

```
        Move      N,R1        R1 serves as a counter.
        Move      #NUM1,R2    R2 points to the list.
        Call      LISTADD     Call subroutine.
        Move      R0,SUM      Save result.
        .
        .
        .
```

Subroutine

```
LISTADD Clear     R0          Initialize sum to 0.
LOOP    Add       (R2)+,R0    Add entry from list.
        Decrement R1
        Branch>0  LOOP
        Return                Return to calling program.
```

Figure 2.25 Program of Figure 2.16 written as a subroutine; parameters passed through registers.

THE STACK FRAME:

During the execution of the subroutine, six locations at the top of the stack contain entries that are needed by the subroutine.

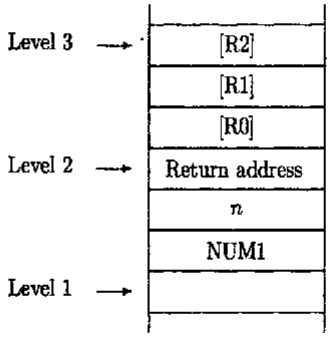
These locations constitute a private work space for the subroutine, created at the time subroutine is entered and freed up when the subroutine returns control to the calling program. Such space is called a stack frame.

Assume top of stack is at level 1 below.

```
            Move        #NUM1,-(SP)   Push parameters onto stack.
            Move        N,-(SP)
            Call        LISTADD       Call subroutine
                                        (top of stack at level 2).
            Move        4(SP),SUM     Save result.
            Add         #8,SP         Restore top of stack
                                        (top of stack at level 1).
            :

LISTADD     MoveMultiple R0-R2,-(SP)  Save registers
                                        (top of stack at level 3).
            Move        16(SP),R1     Initialize counter to n.
            Move        20(SP),R2     Initialize pointer to the list.
            Clear       R0            Initialize sum to 0.
LOOP        Add         (R2)+,R0      Add entry from list.
            Decrement   R1
            Branch>0    LOOP
            Move        R0,20(SP)     Put result on the stack.
            MoveMultiple (SP)+,R0-R2  Restore registers.
            Return                    Return to calling program.
```

(a) Calling program and subroutine



(b) Top of stack at various times

Figure 2.26 Program of Figure 2.16 written as a subroutine; parameters passed on the stack.

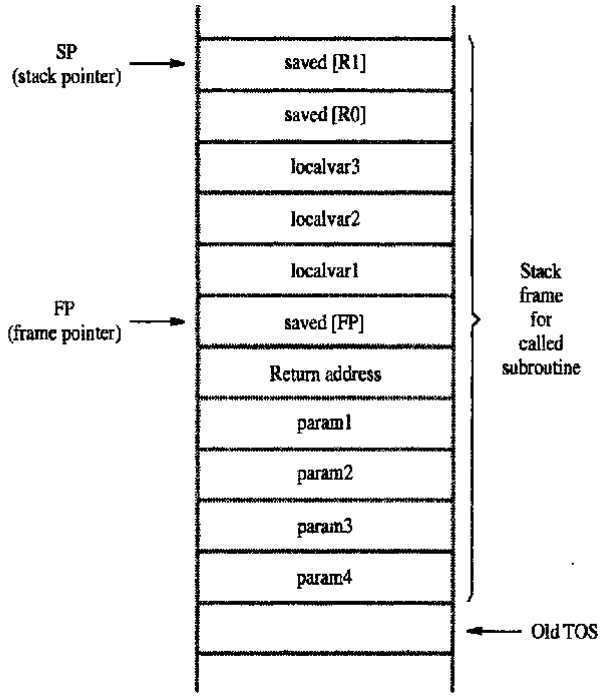


Figure 2.27 A subroutine stack frame example.

Fig 2.27 shows an example of a commonly used layout for information in a stack frame. In addition to the stack pointer SP, It is useful to have another pointer register, called the frame pointer (FP), for convenient access to the parameters passed to the subroutine and to the local memory variables used by the subroutine.

STACK FRAMES FOR NESTED SUBROUTINES:

The stack is the proper data structure for holding return addresses when subroutines are nested. It should be clear that the complete stack frames for nested subroutines build up on the processor stack as they are called.

- Example of a main program calling a first subroutine SUB1, which then calls a second subroutine SUB2, is shown in figure.
- The stack frames corresponding to these two nested subroutines are shown in fig.

Memory location		Instructions		Comments
Main program				
		⋮		
2000		Move	PARAM2,−(SP)	Place parameters on stack.
2004		Move	PARAM1,−(SP)	
2008		Call	SUB1	
2012		Move	(SP),RESULT	Store result.
2016		Add	#8,SP	Restore stack level.
2020		next instruction		
		⋮		
First subroutine				
2100	SUB1	Move	FP,−(SP)	Save frame pointer register.
2104		Move	SP,FP	Load the frame pointer.
2108		MoveMultiple	R0−R3,−(SP)	Save registers.
2112		Move	8(FP),R0	Get first parameter.
		Move	12(FP),R1	Get second parameter.
		⋮		
		Move	PARAM3,−(SP)	Place a parameter on stack.
2160		Call	SUB2	
2164		Move	(SP)+,R2	Pop SUB2 result into R2.
		⋮		
		Move	R3,8(FP)	Place answer on stack.
		MoveMultiple	(SP)+,R0−R3	Restore registers.
		Move	(SP)+,FP	Restore frame pointer register.
		Return		Return to Main program.
Second subroutine				
3000	SUB2	Move	FP,−(SP)	Save frame pointer register.
		Move	SP,FP	Load the frame pointer.
		MoveMultiple	R0−R1,−(SP)	Save registers R0 and R1.
		Move	8(FP),R0	Get the parameter.
		⋮		
		Move	R1,8(FP)	Place SUB2 result on stack.
		MoveMultiple	(SP)+,R0−R1	Restore registers R0 and R1.
		Move	(SP)+,FP	Restore frame pointer register.
		Return		Return to Subroutine 1.

Figure 2.28 Nested subroutines.

9. ADDITIONAL INSTRUCTIONS:

We have additional following instructions : Move, Load , Store, clear,add, substract, Increment, decrement, branch, compre , call,testbit and return. These 13 instructions along with the addressing modes have allowed us to write routines to machine instruction sequencing, including branching and the subroutine structure.

Small set of instructions has a number of redundencies. The Load and Store instructions can be replaced by Move and the increment and decrement instructions can be replaced by add and substract respectively.

Clear can be replaced by a Move instruction containing an immediate operand of Zero.

LOGIC INSTRUCTIONS:

The processor instruction set provides the instructions AND, OR, XOR, TEST, and NOT Boolean logic, which tests, sets, and clears the bits according to the need of the program.

The following instructions are going to perform **logical operations** between two fullword operands, a byte in storage and an immediate byte, or between two fields in storage.

A logical operation works on a bitwise level. Starting on the left, a bit from each operand has the logical operation performed and results in a boolean (true/false) value. This process proceeds until all of the bits have had the logical operation performed.

The possible logical operations are AND, OR, and EXCLUSIVE OR. Since these instructions work on a bit-wise level, a value of 0 is considered false and a value of 1 is considered true.

The AND operation is used to set bits to 0

AND a bit with 1, it stays the same. AND a bit with 0, it becomes 0

RX Format: label N R,D(X,B)

the OR operation is used to set bits to 1

OR a bit with 0, it stays the same. OR a bit with 1, it becomes 1

RX Format: label O R,D(X,B)

The XOR operation is used to set bits to the opposite value.

XOR a bit with 0, it stays the same. XOR a bit with 1, it becomes the opposite value

RX Format: label X R,D(X,B)

SHIFT AND ROTATE INSTRUCTIONS:

Shift and rotate instructions move bit strings (or operand treated as a bit string).

Shift instructions move a bit string (or operand treated as a bit string) to the **right** or **left**, with excess bits discarded (although one or more bits might be preserved in flags). In **arithmetic shift left** or **logical shift left** zeros are shifted into the low-order bit. In **arithmetic shift right** the sign bit (most significant bit) is shifted into the high-order bit. In **logical shift right** zeros are shifted into the high-order bit.

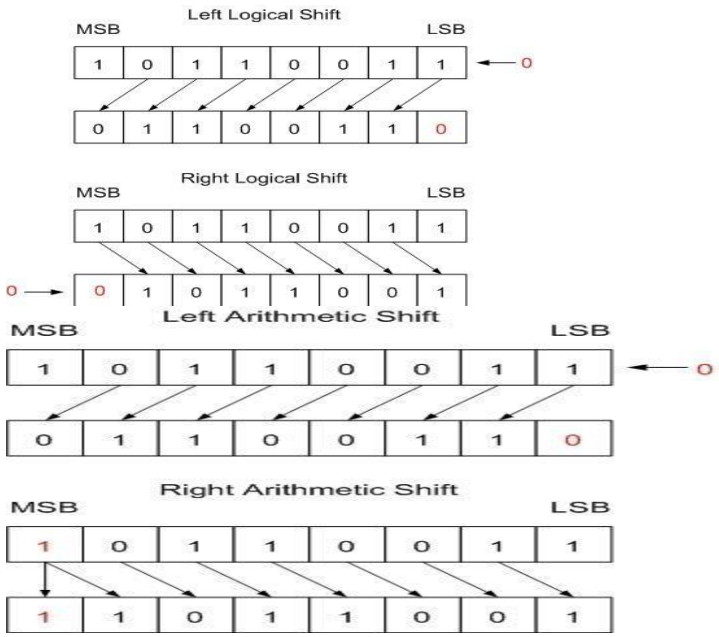
Rotate instructions are similar to shift instructions, except that rotate instructions are circular, with the bits shifted out one end returning on the other end. Rotates can be to the left or right. Rotates can also employ an extend bit for multi-precision rotates.

Logical Shifts:

Two logical shift instructions are needed. One for shifting left and another for shifting right. These instructions shift an operand over a number of the position specified in a count operand contained in the instructions.

The general instruction format is LShiftL count.dst

The count operand may be given as an immediate operand , or it may be contained as a processor register. To complete the description of the left shift operation. We need to specify the bit values brought into the vacated positions at the right end of the destination.



Arithmetic Shift:

- A *Left Arithmetic Shift* of one position moves each bit to the left by one. The vacant least significant bit (LSB) is filled with zero and the most significant bit (MSB) is discarded. It is identical to Left Logical Shift.
- A *Right Arithmetic Shift* of one position moves each bit to the right by one. The least significant bit is discarded and the vacant MSB is filled with the value of the previous (now shifted one position to the right) MSB.

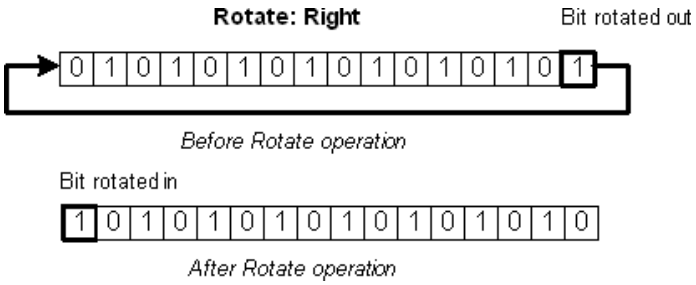
Arithmetic shifts are suitable for signed integers (i.e. integers that can be both positive and negative) that uses two's complement representation for negative numbers.

Arithmetic left shift is identical to logical left shift and can be used in the same way to multiply, both positive and negative values, by two.

With arithmetic right shift new bits get the same value as the sign bit (the leftmost bit). This ensures that the sign (+/−) remains the same before and after. One step with arithmetic right shift is almost the same as integer division by two. The difference is that the result is always rounded down (towards minus infinity) instead of towards zero.

Circular shift

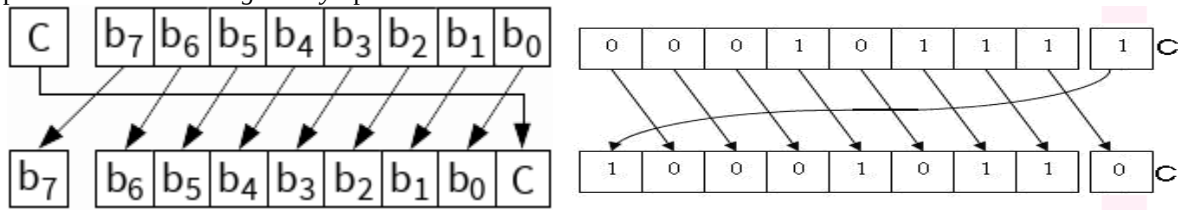
Circular shifts, also called *rotations*, use the bit that got shifted out at one end and inserts it back as the new bit value at the other end. Circular shifts are often used for cryptographic applications and are suitable when it is desirable to not lose any bit values.



Rotate through carry

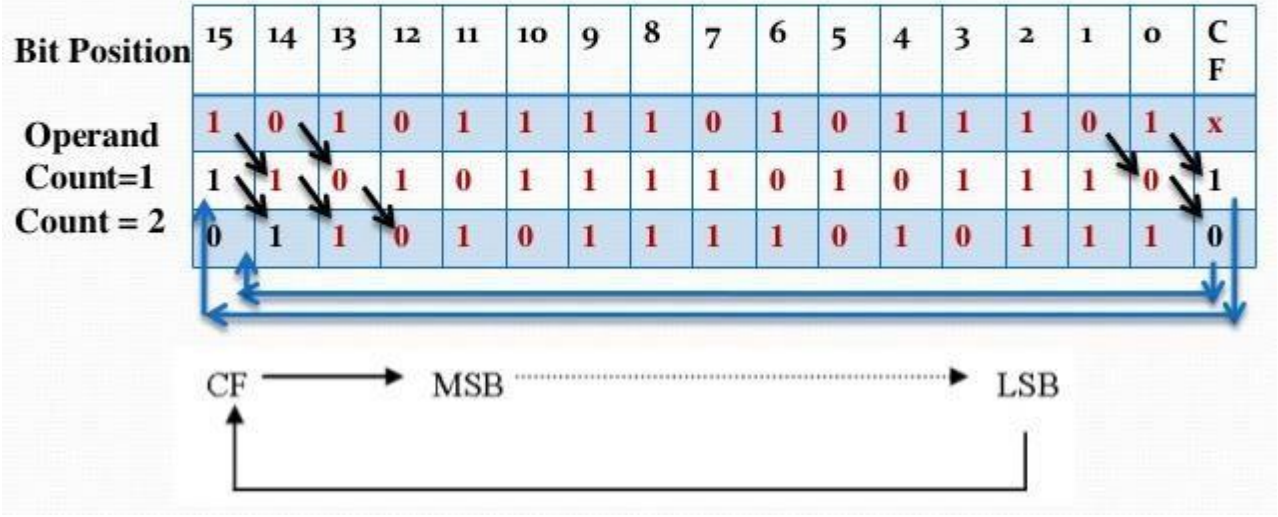
The value of the last bit that got shifted out is normally stored in a carry flag. A special type of circular shift, called *rotate through carry*, uses the old value of this flag for the bit that is shifted in.

Rotate through carry can be used to shift larger values than the computer can normally handle. For example, if a computer can only perform shifts on 32 bits at a time, but we want to perform an arithmetic right shift on a 64-bit value, we can do the calculations in two steps. First we perform an arithmetic right shift on the half containing the most significant bits. The bit that got shifted out will be stored in the carry flag. To finish the calculation we then perform a *rotate through carry* operation on the second half.



3. Rotate Right without carry ROR :

1. **Perform bit-wise right shifts on the operand either by one or count specified**
2. **LSB is shifted to CF & simultaneously shifted to MSB.**



Multiplication by left shift:

The result of a Left Shift operation is a multiplication by 2^n, where n is the number of shifted bit positions.

Example:

Let's take the decimal number 2 represented as 4 bit binary number 0010. By shifting in to the left with one position we get 0100 which is 4 in decimal representation. If we shift it once more we get binary value 1000 which is 8 in decimal representation.

For unsigned representation, when the first "1" is shifted out of the left edge, the operation has overflowed. The result of the multiplication is larger than the largest possible.

Shifting left on signed values also works, but overflow occurs when the most significant bit changes values (from 0 to 1, or 1 to 0).

Division by right shift:

The result of a Right Shift operation is a division by 2^n, where n is the number of shifted bit positions.

Example:

If we have the binary number 01110101 (117 decimal) and we perform **arithmetic right shift** by 1 bit we get the binary number 00111010 (58 decimal). So we have divided the original number by 2.

If we have the binary number 1010 (-6 decimal) and we perform **arithmetic right shift** by 1 bit we get the binary number 1101 (-3 decimal). So we have divided the original negative number by 2.