**BASIC STRUCTURE OF COMPUTERS**

UNIT-I

**Basic Structure of Computer:** Computer Types, Functional Units, Basic operational Concepts, Bus Structure, Software, Performance, Multiprocessors and Multicomputer.

**Machine Instructions and Programs:** Numbers, Arithmetic Operations and Characters, Memory Operations, Instructions and Instruction Sequencing, Addressing Modes, Basic Input/output Operations, Stacks and Queues, Subroutines, Additional Instructions.

**Computer:**

Computer is an electronic device that performs a set of instructions to store the data, retrieve data and process data. In this computer to calculate mathematical and logical operations are at the speed billions of seconds faster than human beings.

The term ‘computer’ is a Latin word, this means to calculate or programmable machine.

**Computer Organization:**

* Computer Architecture and organization is the study of internal working, structuring and implementation of a computer system.
* Computer architecture is concerned with the structure and behavior of the computer
* Computer organization is the high level aspects of a design such as
  + Memory system
  + Bus structure
  + Design of the CPU
* It refers to the operational units and their inter connections that realized the architectural specifications.

**COMPUTER TYPES**

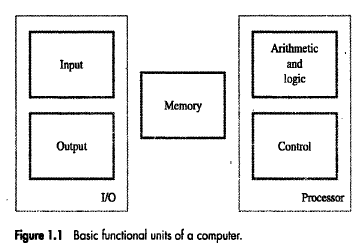
A computer can be defined as a fast electronic calculating machine that accepts the (data) digitized input information process it as per the list of internally stored instructions and produces the resulting information. List of instructions are called programs & internal storage is called computer memory.

The different types of computers are exist difference widely in size, cost and computational power.

1. **Micro Computers:** Micro computers are smaller computer. They contain only one CPU. One feature of a Micro computer is that the CPU is usually a single integrated circuit called a microprocessor. Micro computer is the integration of microprocessor and supporting memory and I/O device. The word length depends on 8 bits to 32 bits.
2. **Mini Computers:** Mini computers are the up version of the micro computers with the moderate speed and storage capacity. These are designed to smaller data words.
3. **Personal computers: -** This is the most common type found in homes, schools, Business offices etc., It is the most common type of desk top computers with processing and storage units along with various input and output devices.
4. **Portable Note book computers: -** These are compact and portable versions of PC with all of these components packed into a single unit size of briefcase.
5. **Work stations: -** These have high resolution input/output (I/O) graphics capability, but with same dimensions as that of desktop computer. These are used in engineering applications of interactive design work.
6. **Servers:** These are large storage unit and faster communication link. The computer major role in Internet communication.
7. **Super computers: -** These are used for large scale numerical calculations required in the applications like weather forecasting, robotic, aircraft design etc.,

**FUNCTIONAL UNIT**

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



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 one source program to an object program. i.e. into machine language. Finally the results are sent to the outside world through output device. All of these actions are coordinated by the control unit.

**Input unit: -**

The source program/high level languages program/coded information/simply 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. Examples of Joysticks, trackballs, mouse, scanners etc are other input devices.

**Memory unit: -**

It is function into store programs and data. It is basically to two types

**1. Primary memory**

2. **Secondary memory**

**1. Primary memory: -** Is the one exclusively associated with the processor and operates at the electronics speeds programs must be stored in this memory while they are being executed. The memory contains a large number of semiconductors storage cells. Each cell capable storing one bit of information. These are processed in a group of fixed site called word.

To provide easy access to a word in memory, a distinct address is associated with each word 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 after specifying its address is called random-access memory (RAM).

The time required to access one word in called memory access time. Memory which is only readable by the user and contents of which can’t be altered is called read only memory (ROM) it contains operating system.

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 storage is essential it tends to be expensive.

**2 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), floppies etc.,

**Control unit:-**

It effectively is the nerve center that sends signals to other units and senses their states. The actual timing signals that govern the transfer of data between input unit, processor, memory and output unit are generated by the control unit.

**Arithmetic logic unit (ALU):-**

Most of the computer operators 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 register. 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 such as key boards, displays, magnetic and optical disks, sensors and other mechanical controllers.

**Output unit:-**

These actually are the counterparts of input unit. Its basic function is to send the processed results to the outside world.

**Examples:-** Printer, speakers, monitor etc.

**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, which executes the specified operations. Data to be stored are also stored in the memory.

**Examples: -** Add LOCA, R0

This instruction adds the operand at memory location LOCA, to operand in register R0 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. Fetch the operand at location LOCA from main memory into the processor.

3. Add the memory operand contents of LOCA to the contents of register R0.

4. Finally the resulting sum is stored in the register R0.

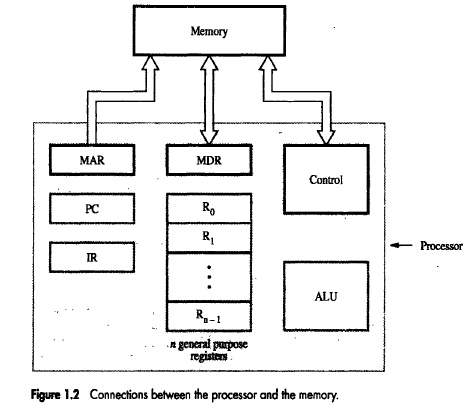
The preceding add instruction combines a memory access operation with an ALU Operations. In some other type of computers, these two types of operations are performed by separate instructions for performance reasons.

Load LOCA, R1

Add R1, R0

The following are the steps to execute the instructions.

1. Fetch the instruction from main memory into the processor
2. Fetch the operand at location LOCA from main memory into the register R1.
3. Add the contents of register R1 and content of register R0
4. Store the result sum into R0.



The figure shows how memory & the processor can be connected. In addition to the ALU and the control circuitry, the processor contains a number of registers used for several different purposes.

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

**The program counter PC:-** This is another specialized register that keeps track of execution of a program. It contains the memory address of the next instruction to be fetched and executed. Besides IR and PC, there are n-general purpose registers R0 through Rn-1.

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

**1. MAR – (Memory Address Register):-** It holds the address of the location to be accessed.

**2. MDR – (Memory Data Register):-** It contains the data to be written into or read out of the address location.

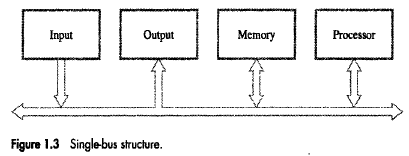
**Operating steps are**

1. Programs reside in the memory & usually get these through the I/P unit.
2. Execution of the program starts when the PC is set to point at the first instruction of the program.
3. Contents of PC are transferred to MAR and a Read Control Signal is sent to the memory.
4. After the time required to access the memory elapses, the address word is read out of the memory and loaded into the MDR.
5. Now contents of MDR are transferred to the IR & now the instruction is ready to be decoded and executed.
6. If the instruction involves an operation by the ALU, it is necessary to obtain the required operands.
7. An operand in the memory is fetched by sending its address to MAR & Initiating a read cycle.
8. When the operand has been read from the memory to the MDR, it is transferred from 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 to MDR.
11. Address of location where the result is stored is sent to MAR & a write cycle is initiated.
12. The contents of PC are incremented so that PC points to the next instruction that is to be executed.

**BUS STRUCTURES**

The Bus structure is simplest and most common way of interconnecting various parts of the computer. 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. A group of lines that serve as a connecting port 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 purpose. Simplest way to interconnect is to use the single bus as shown



Since 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 one bus. Single bus structure is

Low cost

Very flexible for attaching peripheral devices

Multiple bus structure certainly increases the performance but also increases the cost significantly. All the interconnected devices are not of same speed & time leads to a bit of a problem. This is solved by using cache registers (ie buffer registers). These buffers are electronic registers of small capacity when compared to the main memory but of comparable speed.

The instructions from the processor at once are loaded into these buffers and then the complete transfer of data at a fast rate will take place.

**SOFTWARE**

A set of instructions to the process of creating and running is called program. The collections of programs are called software. The software is classified into two types are System software and Application software.

**System Software:**

System software consists of several programs. It is directly responsible for controlling, integrating and managing the individual hardware components of a computer system. The system software is a collection of programs that are executed as needed to perform functions such as

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

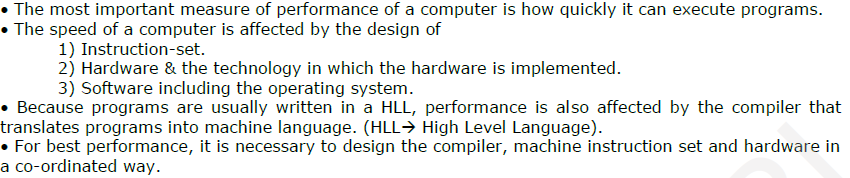
The Language translators are to convert instruction from high level language to machine level language. There are three types of translators, i.e Assembler, Interpreter, Compiler.

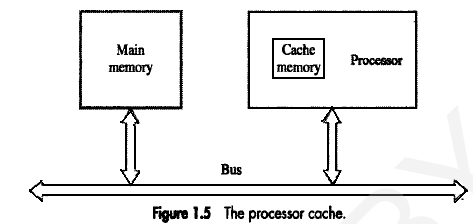
* **Assembler:** The assembler is a program. The main operation of assembler is to convert assembly language to machine language.
* **Interpreter:** Interpreter is a program that is converting into high level language to Intermediate code. Intermediate code executed by the processor. Interpreter executes each and every line individually. It does not provide .exe files.
* **Compiler:** Compiler is a program, these compilers to compile the whole program at a time. The programming code will be translating high level language to machine level language code, which is called object code. The object code can be executed directly on the machine.

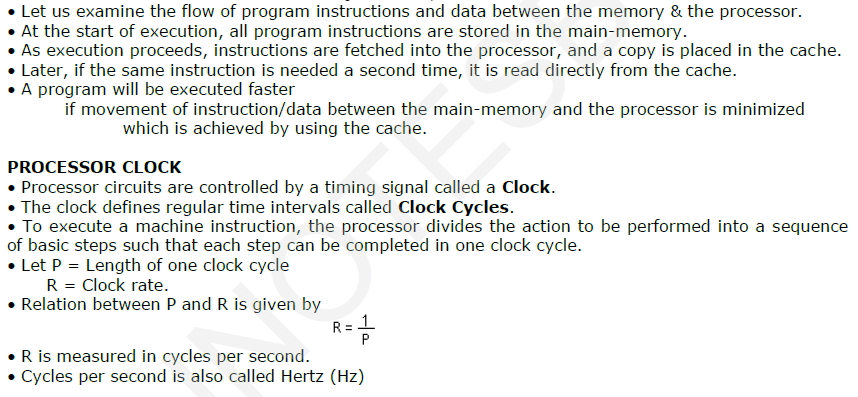
The System software supported useful for two programs are linker and loader.

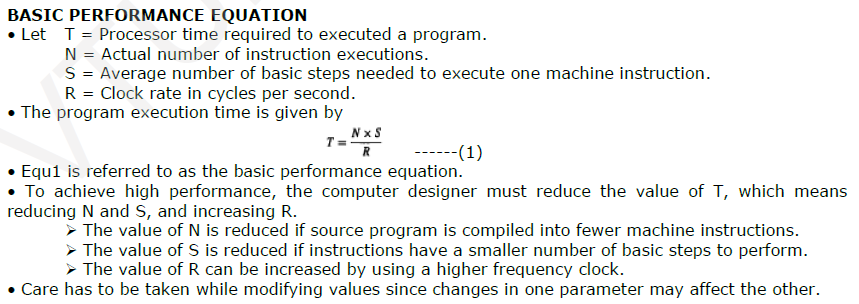
* **Linker:** Generally, software comprises millions of lines of programming statements or code. It is usually much more efficient to divide this code into logical groups and store in different independent modules. Each module can be individual test and debugged. When all the modules work they are linked together to form a large executable program. This single program is performed by linking together the several object modules and libraries by a system program called linker.
* **Loader:** The loader is a part of the operating system that brings an executable file residing on disk into memory and executes.

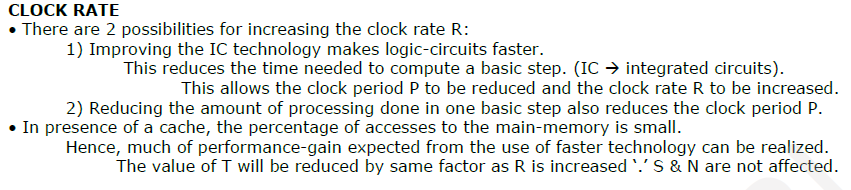
**PERFORMANCE**

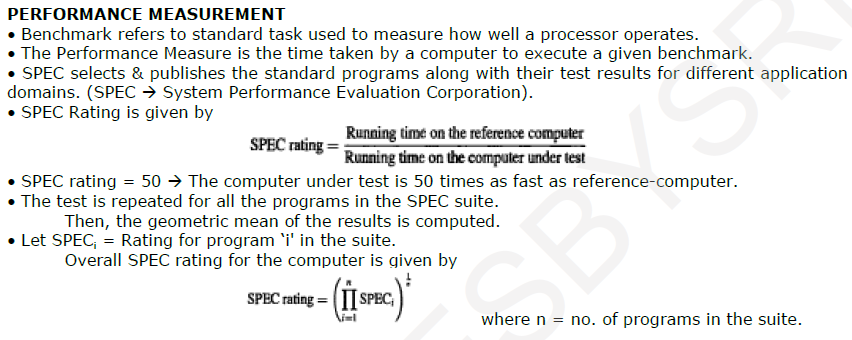


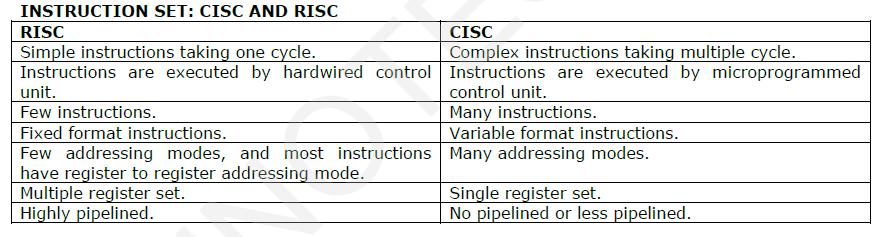












**MULTIPROCESSOR & MULTICOMPUTERS:-**

* Large computers that contain a number of processor units are called multiprocessor system.
* These systems either execute a number of different application tasks in parallel or execute subtasks of a single large task in parallel.
* All processors usually have access to all memory locations in such system & hence they are called shared memory multiprocessor systems.
* The high performance of these systems comes with much increased complexity and cost.
* In contrast to multiprocessor systems, it is also possible to use an interconnected group of complete computers to achieve high total computational power. These computers normally have access to their own memory units when the tasks they are executing need to communicate data they do so by exchanging messages over a communication network. This properly distinguishes them from shared memory multiprocessors, leading to name message-passing multi computer.

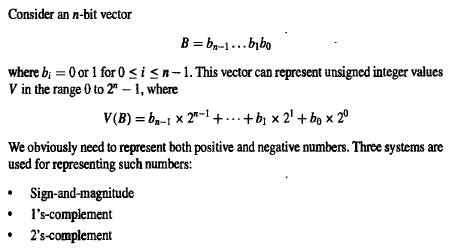
**CHAPTER-2**

**MACHINE INSTRUCTION AND PROGRAMS**

**NUMBERS, ARTHMETIC OPERATIONS AND CHARACTERS**

Computers are built using logic circuits that operate on information represented by two valued electrical signals. We label the two values as 0 and 1, and we define the amount of information represented by such a signal as a bit of information, where bit stands for binary digit. The most natural way to represent a number in a computer system is by a string of bits called a binary number. A text character can also be represented by a string of bits called a character code.

**Number Representation:**



In all three systems, the leftmost bit is 0 for positive numbers and 1 for negative numbers in all three representations using 4-bit numbers. Positive values have identical representations in all systems, but negative values have different representations.

**Sign-and-magnitude:**

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

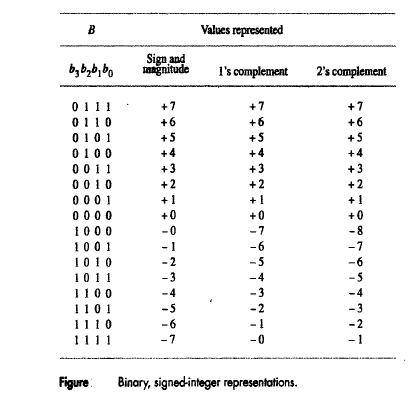
For example, +5 is represented by 0101, and -5 is represented by 1101.

**1’s Complement:**

1’s complement representation; negative values are obtained by complementing each bit of the corresponding positive number. Thus, the representation for -3 is obtained by complementing each bit in the vector 0011 to yield 1100. Clearly, the same operation, bit complementing, is done in converting a negative number to the corresponding positive value. Converting either way is referred to as forming the 1’s-complement of a given number.

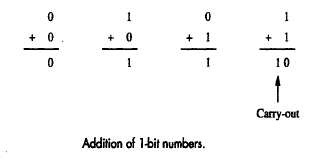
**2’s Complement:**

In the 2’s-complement system, forming the 2’s-complement of a number is done by subtracting that number from 2n. Hence, the 2’s complement of a number is obtained by adding 1 to the 1’s complement of that number.



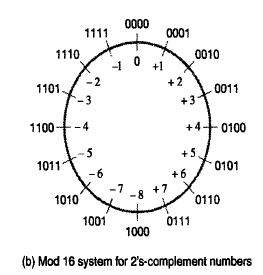
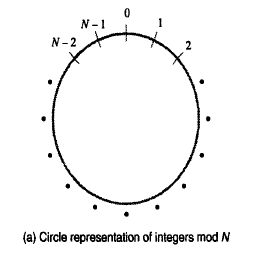
**Addition of Positive numbers:-**

Consider adding two 1-bit numbers. 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.

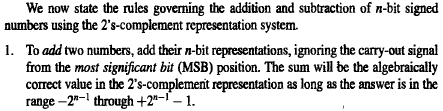


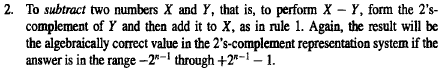
**Addition and Subtraction of Signed Numbers:**

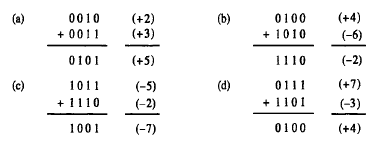
* The sign-and-magnitude system is the simplest representation, but it is also the most awkward for addition and subtraction operations.
* The 2’s complement system is the most efficient method for performing addition and subtraction operations.
* 2’s complement arithmetic considers addition modulo N (written as mod N).
* A graphical device for the description of addition mod N of positive integers is a circle with the N values 0 through N – 1.
* For example the case N = 16, the operation (7+4) mod 16 yields the value 11. To perform this operation graphically, locate 7 on the circle and then move 4 units in the clockwise direction to arrive at the answer 11.
* Now consider a different interpretation of the mod 16 circle. Let the values 0 through 15 be represented by the 4-bit binary vectors 0000, 0001, ….. ,1111, according to the binary number system.
* Then reinterpret these binary vectors to represent the signed numbers from -8 through +7 in the 2’s complement method as shown in figure.

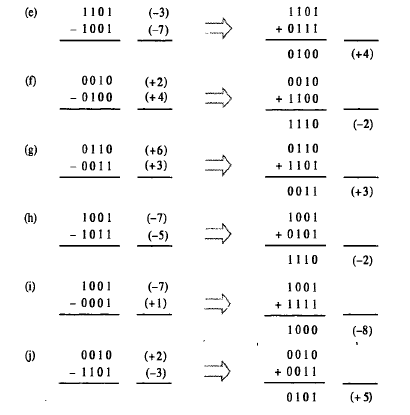


**Figure: Modular number systems and the 2’s complement system**

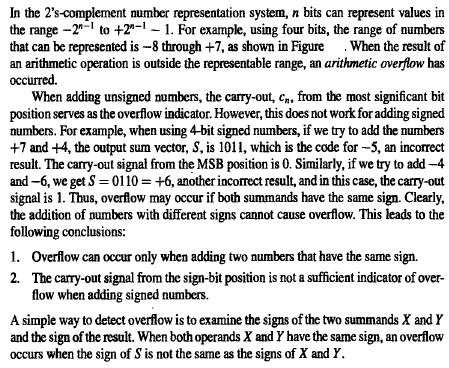
****

****

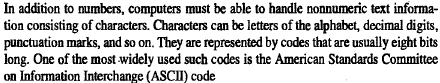
****

****

**Overflow in Integer Arithmetic:**



**Characters:**

****

**Memory Operations:**

Two basic Operations involving in the Memory

1.Load(Read or Fetch)

2.Store(Write)

Load operation transfer a copy of the contents of a specific memory location to the processor

The Store Operation transfer an item of information from the processor to specific Memory Location

**INSTRUCTIONS AND INSTRUCTION SEQUENCING:**

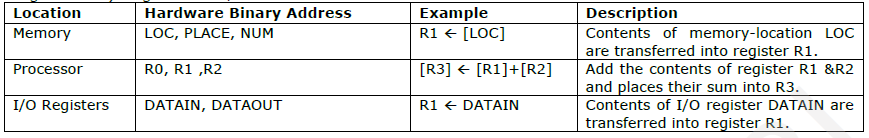
A computer program consists of a sequence of small steps. Such as adding two numbers, testing a particular condition, Reading a character from the keyboard or Sending a character to be displayed on a display screen.

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

* Data transfers between the memory and the processor registers(MOV, PUSH, POP)
* Arithmetic and logic operations on data (ADD, SUB, MUL, DIV, AND, OR, NOT)
* Program sequencing and control (LOOP, INT)
* I/O transfers (DATAIN, DATAOUT)

**Register Transfer Notation (RTN):**

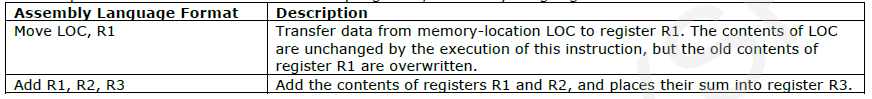
Transfer of information from one location in the computer to another. Possible locations that may be involved in such transfers are memory locations that may be involved in such transfers are memory locations, processor registers, or registers in the I/O subsystem. Most of the time, we identify a location by a symbolic name standing for its hardware binary address.



**Note:** The right hand side of an RTN expression always denotes a value and left hand side is the name of location where the value is to be placed overwriting the old contents of that location.

**Assembly Language Notation (ALN):**

Another type of notation to represent machine instructions and programs, Assembly language formate is used.



**Basic Instruction types:**

* In a high-level language program is a command to the computer to add the current values of the two variables called A and B, and to assign the sum to a third variable C. The above high-level language statement requires the action.

C 🡨 [A] + [B]

To carry out this action, the contents of memory locations A and B are fetched from the memory and transferred into the processor where their sum is computed. This result is then sent back to the memory and stored in location C.

* Furthermore, assume that this instruction contains the memory addresses of the three operands A, B, and C. This three-address instruction can be represented symbolically as

Add A, B, C

Operands A and B are called the source operands, C is called the destination operand, and Add is the operation to be performed on the operands. A general instruction of this type has the format.

Operation Source1, Source 2, Destination

* An alternative approach is to use a sequence of simpler instructions to perform the same task, with each instruction having only one or two operands. Suppose that two-address instructions of the form.

Operation Source, Destination

An Add instruction of this type is

Add A, B

Which perform the operation is

B 🡨 [A] + [B]

When the sum calculated, the result is sent to the memory and stored in location B. Replacing the original contents of this location. This means that operand B is both source and destination.

* The problem can be solved by using another two-address instruction that copies the contents of one memory location into another. Such an instruction is

Move B, C

Which perform the operations is

C 🡨 [B]

The operation C 🡨 [A] + [B] can now be performed by the two instruction sequence

Move B, C

Add A, C

* A processer register, usually called the accumulator, may be used for one address instruction.

Add A

That means add the contents of memory location A to contents of the accumulator register and place the sum back into the accumulator. Let us also introduce the one address instruction

Load A and

Store A

The Load instruction copies the contents of memory location A into the accumulator and store instruction copies the contents of the accumulator into memory location A. Using only one – address instructions the operation C 🡨 [A] + [B] can be performed by executing the sequence of instructions.

Load A

Add B

Store C

* Some early computers were designed around a single accumulator structure. Most modern computers have a number of general-purpose processor registers – typically 8 to 32, and even considerably more in some cases. Access to data in these registers is much faster than to data stored in memory locations because the registers are inside the processor

Let Ri represent a general-purpose register. The instructions

Load A, Ri

Store Ri, A and

Add A, Ri.

Are generalizations of the Load, Store, and Add instructions for the single-accumulator case, in which register Ri performs the function of the accumulator.

* When a processor has several general-purpose registers, many instructions involve only operands that are in the register. In fact, in many modern processors, computations can be performed directly only on data held in processor registers. Instructions such as

Add Ri, Rj

Or

Add Ri, Rj, Rk

In both of these instructions, the source operands are the contents of registers Ri and Rj. In the first instruction, Rj also serves as the destination register, whereas in the second instruction, a third register, Rk, is used as the destination.

* It is often necessary to transfer data between different locations. This is achieved with the instruction

Move Source, Destination

When data are moved to or from a processor register, the Move instruction can be used rather than the Load or Store instructions because the order of the source and destination operands determines which operation is intended. Thus,

Move A, Ri is the same as Load A, Ri

And

Move Ri, A is the same as Store Ri, A

* In processors where arithmetic operations are allowed only on operands that are processor registers, the C = A + B task can be performed by the instruction sequence

Move A, Ri

Move B, Rj

Add Ri, Rj

Move Rj, C

In processors where one operand may be in the memory but the other must be in register, an instruction sequence for the required task would be

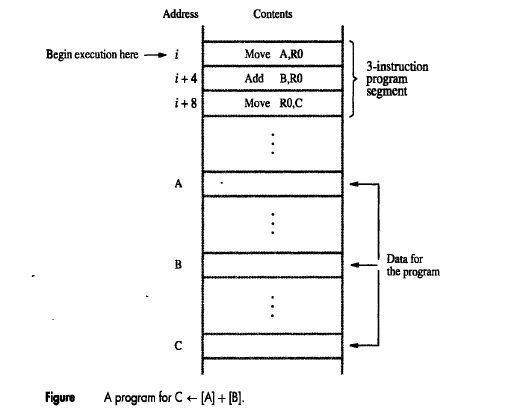
Move A, Ri

Add B, Ri

Move Ri, C

**Instruction Execution and Straight-Line Sequencing:**

In the preceding discussion of instruction formats, we used to task C 🡨 [A] + [B]. It 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 has a number of processor registers. The three instructions of the program are in successive word locations, starting at location i. Since each instruction is 4 bytes long, the second and third instructions start at addresses i + 4 and i + 8.



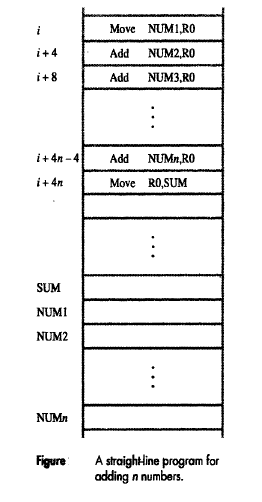
Let us consider how this program is executed. The processor contains a register called the program counter (PC), which holds the address of the instruction to be executed next. To begin executing a program, the address of its first instruction must be placed into the PC. Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called straight-line sequencing. During the execution of each instruction, the PC is incremented by 4 to point to the next instruction. Thus, after the Move instruction at location i + 8 is executed, the PC contains the value i + 12, which is the address of the first instruction of the next program segment.

Executing a given instruction is a two-phase procedure. In the first phase, called instruction 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.

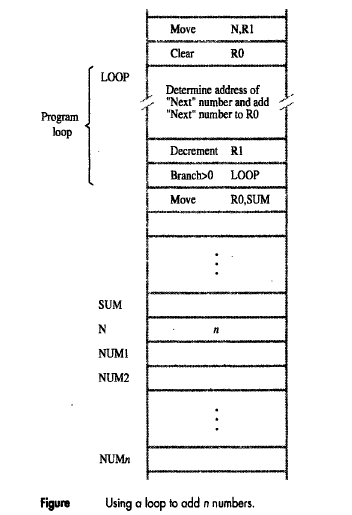
**Branching:**

Branching is a transfer of control from the current statement to another statement construct in the program unit.

* Consider the program adding a list of n numbers.
* The addresses of the memory locations containing the n numbers are symbolically given as NUM1, NUM2, …… NUMn and a separate Add instruction is used to add each number to the contents of register R0.
* After all the numbers have been added the result is placed in memory location SUM.



* Instead of using a long list of add instructions, it is possible to place a single add instruction in a program loop, as shown in figure.
* 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 the next list entry is determined, and that entry is fetched and added toR0.
* Number of entries in the list ‘n’ is stored in memory location N.
* Register R1 is used as a counter to determine the number of time the loop is executed.
* The Loop is a straight line sequence of instructions executed as many times needed. The loop starts at location loop and ends at the instruction Branch > 0.
* During each pass Address of the next list entry is determined, that entry is fetched and added to R0.
* The instruction decrement1 reduces the content of R1 by 1 each time through the loop.
* Then Branch Instruction loads a new value into the PC. As a result the processor fetches and executes the instruction at this new address called the Branch target.
* A conditional branch instruction causes a branch only if a specified condition is satisfied. If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed
* It moves the final result from R0 into memory location SUM.

**Condition Codes:**

The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions. This is accomplished by recording the required information in individual bits, often called condition code flags. These flags are usually grouped together in a special processor register called the condition code register or status register. Individual condition code flags are set to 1 or cleared to 0, depending on the outcome of the operation performed.

Four commonly used flags are

**N(negative):** Set to 1 if the result is negative; otherwise, cleared to 0

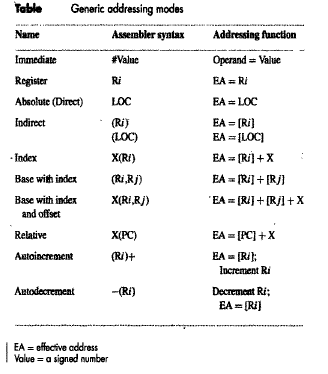
**Z(zero) :** Set to 1 if the result is 0; otherwise, cleared to 0

**V(overflow) :** Set ot1 if arithmetic overflow occurs; otherwise, cleared to 0

**C(carry) :** Set to 1 if a carry-out results from the operation; otherwise, cleared to 0

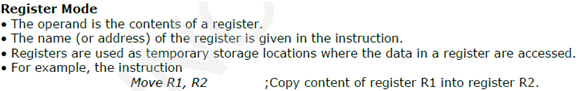
**ADDRESSING MODES:**

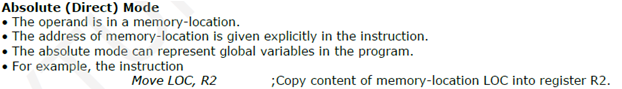
The different ways in which the location of an operand is specified in an instruction are referred to as Addressing modes.

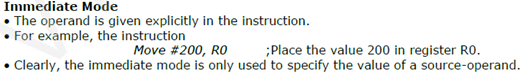


**Implementation of variable and constants:**

* Variables and constants are the simplest data types and are found in almost every computer program.
* In Assembly language a variable is represented by allocating a register or a memory location to hold its value.
* The value can be changed as needed using appropriate instructions.
* There are 2 accessing modes to access the variables.
  + - Register mode
    - Absolute mode
* Next, the Address and data constants can be represented in assembly languge using the Immediate mode.







**Indirection and Pointers:**

Instruction does not give the operand or its address explicitly. The instruction provides information from which the new address of the operand can be determined. This address is called Effective Address (EA) of the operand.

**Indirect mode:**

* The EA of the operand is the contents of a register or memory location.
* The register or memory location that contains the address of an operand is called a pointer.
* We denotes Indirection by
  + - Name of the register
    - New address given the instruction.

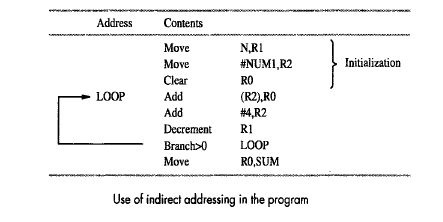
Ex: Add (R1), R0

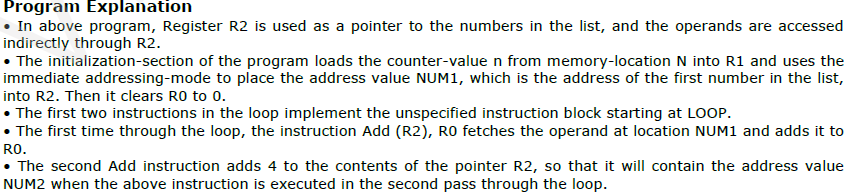
The operand is in memory. Register R1 gives the effective address (B) of the operand. The data is read from location B and added to contents of register R0.



* To execute the Add instruction in figure (a), the processor uses the value which is in Register R1, as the EA of the operand. It request a read operation from the memory to read the contents of location B. The value read is the desired operand, which the processor adds to the contents of register R0.
* Indirect addressing through a memory location is also possible as shown in figure (b), the processor first reads the contents of memory location A, the requests a second read operation using the value B as an address to obtain the operand.

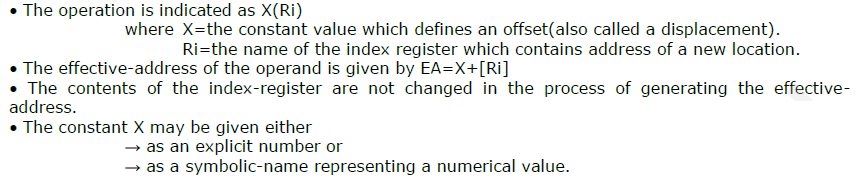
Let us now return to the program for adding a list of numbers. Indirect addressing can be used to access successive numbers in the list.

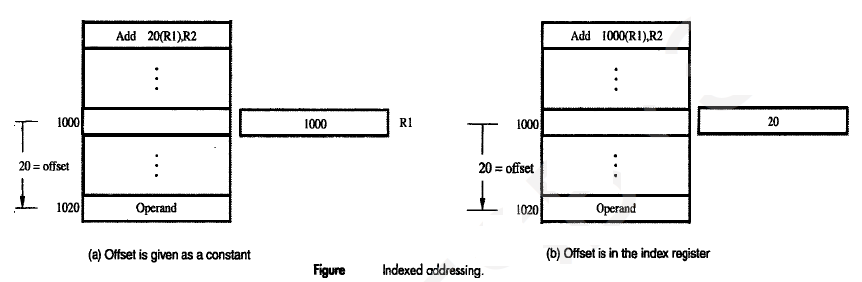




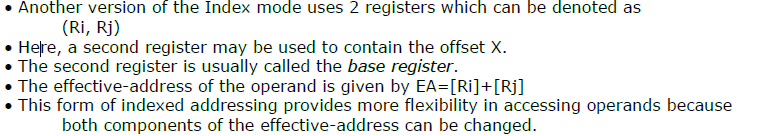
**Indexing and Arrays:**

A different kind of flexibility for accessing operands is useful in Lists and Arrays.

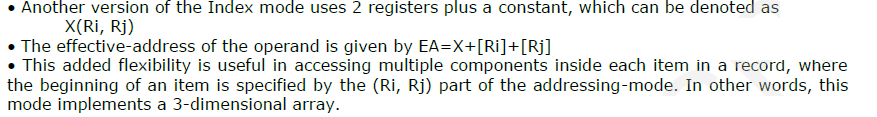
**Index mode**:



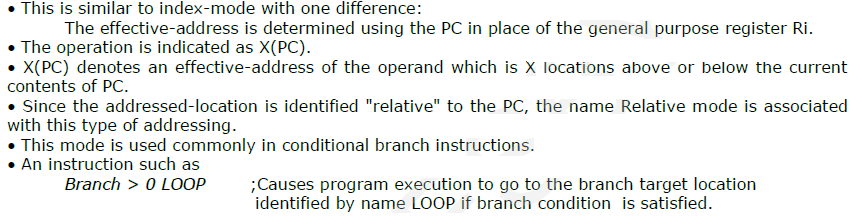
**Base with Index mode:**



**Base with Index and Offset :**



**Relative Addressing mode:**



**Additional Modes:**

The five basic addressing modes found in most computers. Although these modes success for general computation, many computers provide additional modes intended to certain programming tasks. The two modes described next are useful for accessing data items in successive locations in the memory.

**Auto Increment mode:**

Effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.**(R1)+**

**Example:**

**Add R1, (R2)+**

R1 = R1 +M[R2]

R2 = R2 + d

Useful for stepping through arrays in a loop. R2 – start of array d*–* size of an element

**Auto Decrement Mode:**

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

**Example:**

**Add R1, -(R2)**

**R2 = R2 - d**

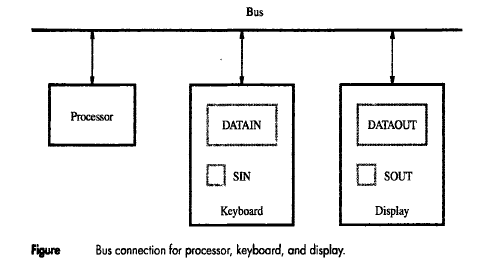
R1 = R1 +M[R2]

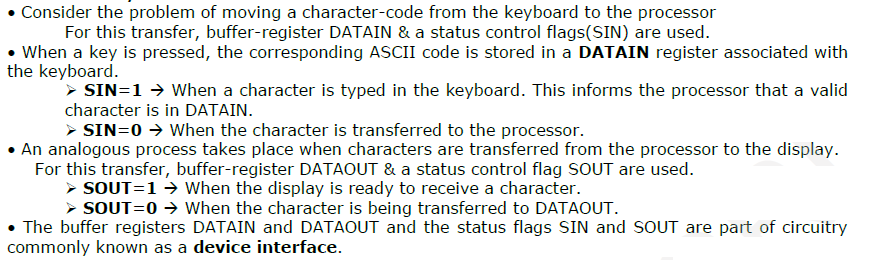
Auto decrement mode is same as auto increment mode. Both can also be used to implement a stack as push and pop. Auto increment and Auto decrement modes are useful for implementing “Last-In-First-Out” data structures.

**BASIC INPUT / OUTPUT OPERATIONS:**

* Input / Output operations are essential a task that reads in character input from a keyboard and produces character output on a display screen.
* A simple way of performing such I/O tasks is to use a method known as program – controlled I / O.
* The rate of data transfer from the keyboard to a computer is limited by the typing speed of the user, which is to exceed too few characters per second.
* The rate of output transfer from the computer to the display is much higher.
* It is determined by the rate at which character can be transmitted over the link b/w the computer and the display device, several thousand character per second.
* This is still much slower than the speed of processor that can execute many millions of instructions per second.
* The difference in speed b/w the processor and I/O device creates for mechanisms to synchronize the transfer of data between them.
* A solution in this problem, input is sent from the keyboard in a similar way the processor waits for a signal indicating that a character key has been struck and that its code is available in some buffer register. Then the processor proceeds to read that code.
* On output, the processor sends the first character and then waits for a signal from the display that the character has been received. It then sends the second character and so on.

The keyboard and the display are separate device as shown in figure, the action of striking a key on the keyboard does not automatically cause the corresponding character to be displayed on the screen. One block of instructions in the I/O program transfers the character into the processor, and another associated block of instructions causes the character to be displayed.

**



For example, the processor can monitor the keyboard status flag SIN and transfer a character from DATAIN to register R1 by the following sequence of operations

READWAIT Branch to READWAIT if SIN = 0

Input from DATAIN to R1

The Branch operation is usually implemented by two machine instructions. The first instruction tests the status flag and the second performs the branch. The main idea is that the processor monitors the status flag by executing a short wait loop and proceeds to transfer the input data when SIN is set to 1 as a result of a key being struck. The input operation resets SIN to 0. An analogous sequence of operations is used for transferring output to the display.

WRITEWAIT Branch to WRITEWAIT if SOUT = 0

Output from R1 to DATAOUT

Until now, the addresses issued by the processor to access instructions and operands always refer to memory locations. Many computers use an arrangement called memory – mapped I/O in which some memory address values are used to refer to peripheral device buffer registers, such as DATAIN and DATOUT.

For example, the contents of the keyboard character buffer DATAIN can be transferred to register R1 in the processor by the instruction.

MoveByte DATAIN, R1

Similarly, the contents of register R1 can be transferred to DATAOUT by the instruction

MoveByte R1, DATAOUT

The status flags SIN and SOUT are automatically cleared when the buffer registers DATAIN and DATAOUT are referenced respectively. The MoveByte operation code signifies that the operand size is a byte, to distinguish it from the operation code Move that has been used for word operands. It is possible to deal with the status flags SIN and SOUT in the same way, by assigning them distinct addresses. However, it is more common to include SIN and SOUT in device status registers, one for each of the two devices. Let us assume that bit b3 in registers INSTATUS and OUTSTATUS corresponds to SIN and SOUT respectively. The read operation just described may now be implemented by the machine instruction sequence

READWAIT Testbit #3, INSTATUS

Branch = 0 READWAIT

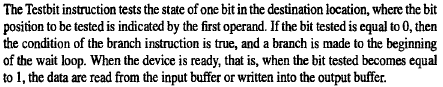
MoveByte DATAIN, R1

The write operation may be implemented as

WRITEWAIT Testbit #3, OUTSTATUS

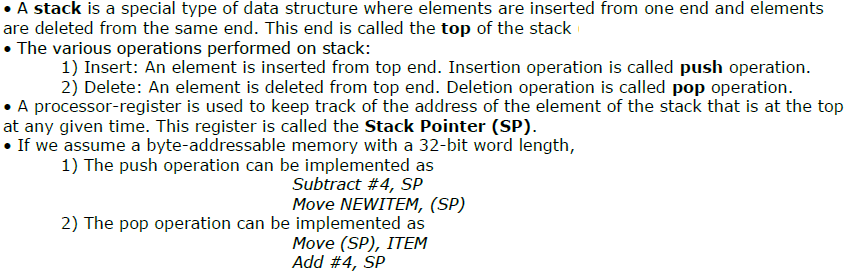
Branch = 0 WRITEWAIT

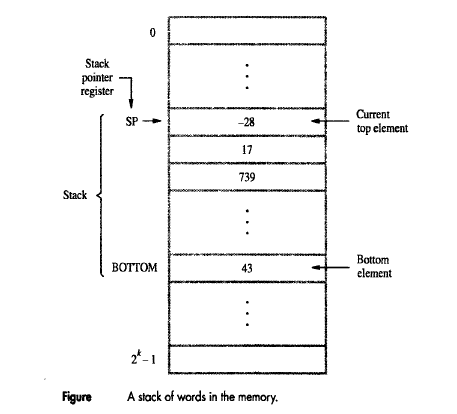
MoveByte R1, DATAOUT



**STACKS AND QUEUES:**

A computer program often needs to perform a particular subtask using the familiar subroutine structure. In order to organize the control and information linkage between the main program and the subroutine, a data structure called a stack is used.



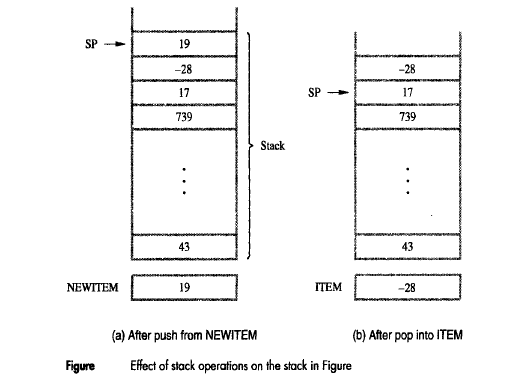


* These two instructions move the top value from the stack into location ITEM and then increment the stack pointer by 4 so that it points to the new top element.
* If the processor has the Autoincrement and Autodecrement addressing modes, then the push operation can be performed by the single instruction

Move NEWITEM, -(SP)

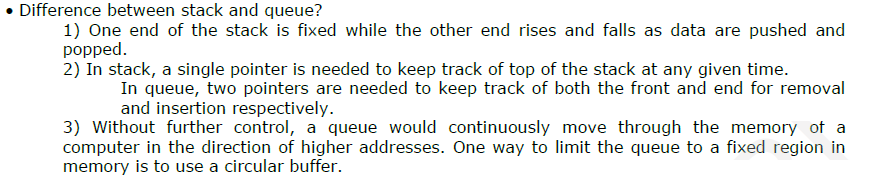
And the pop operation can be performed by

Move (SP)+, ITEM

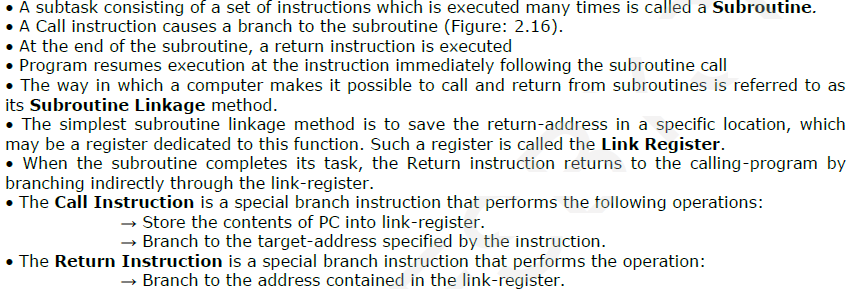


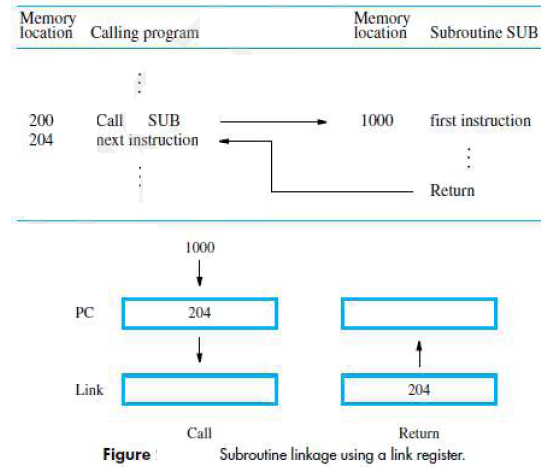
**Queues:**

Another useful data structure that is similar to the stack is called a queue. Data are stored in and retrieved from a queue on a first-in-first-out (FIFO) basis. Thus, if we assume that the queue grows in the direction of increasing addresses in the memory, which is a common practice, new data are added at the back (high-address end) and retrieved from the front (low-address end) of the queue.

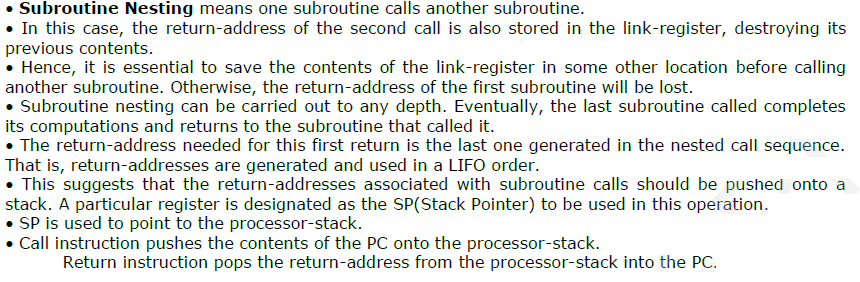


**SUBROUTINES:**

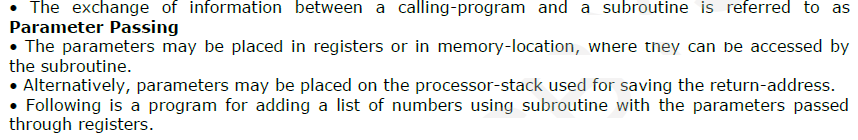


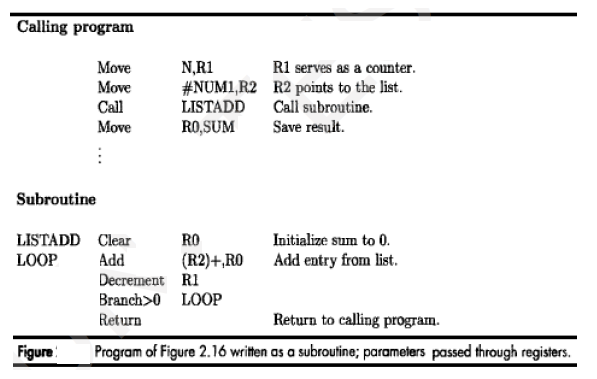


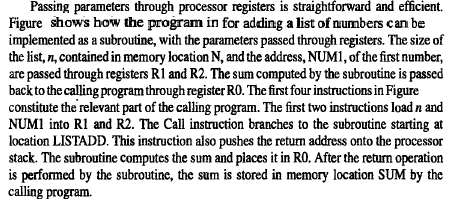
**Subroutine Nesting and the Processor Stack** :



**Parameter Passing:**







**ADDITIONAL INSTRUCTIONS:**

* The following instructions are move, load, store, clear, add, subtract, increment, decrement, branch, test bit, compare, call and return. These 13 instructions along with the addressing modes in table.
* To allow write routines to illustrate machine instruction sequencing, including branching, the subroutine structure and the basic memory mapped I/O operations.
* Even this small set of instructions has a number of redundancies.
* The load and store instructions can be replaced by Move, and the Increment / Decrement instructions can be replaced by Add and Subtract respectively. Clear can be replaced by a Move instruction containing an immediate operand of zero.
* Therefore only 8 instructions would have been sufficient for our purposes.
* But it is not unusual to have some redundancy in practical machine instruction sets.
* Certain simple operations can usually be accomplished in a number of different ways. Some alternatives may be more efficient than others.
* In this section we introduce a few more important instructions that are found in most instruction sets.

**Logic Instructions:**

* Logic operations such as AND, OR, and NOT applied to individual bits are the basic building blocks of digital circuits.
* Which is using instructions that apply operations to all bits of a word or byte independetly and parallel.
* For example,

Not dst (destination),

Complements all bits contained in the destination operand, changing 0’s to 1’s and 1’s to 0’s.

* That adding 1 to the 1’s complement of a signed positive number forms the negative version in 2’s complement representation.
* Example, +3 (0011) is converted to -3 (1101) by adding 1 to the 1’s complent of 0011. If 3 is contained in register R0.

Not R0

Add #1, R0

Achieve the conversion, Move computers have a single instruction

Nagate R0

**Shift and Rotate Instructions:**

**Shift Instructions:** Shift Instructions is shift an operand, bit by bit to the right or to the left, Directions of shift is dependent open the specific instructions

* There are many applications that require the bits of an operand to be shifted right or left some specified number of bit positions.
* For general operands, we use a logical shift
* For a number, we use a Arithmetic shift which preserve the sign of the number.

**Logical Shift:**

* Two Logical shift instructions are Shifting Left (LshiftL) and Shifting Right (LshiftR)
* These instructions shift an operand over a number of bit positions specified in a count operand contained in the instruction.
* The general form of a logical left shift

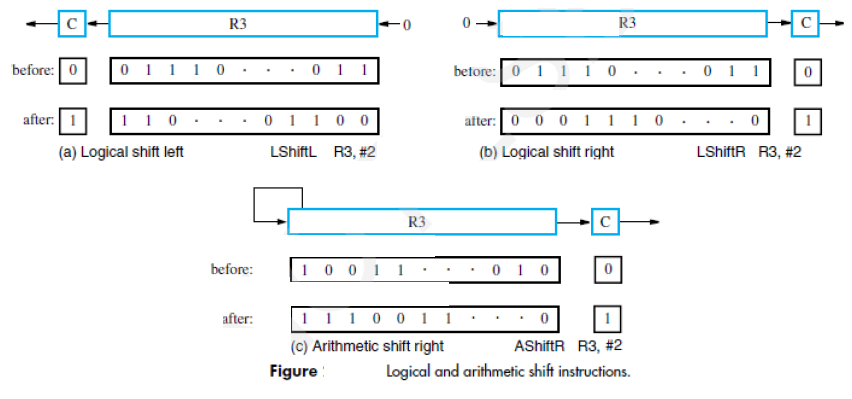
LshiftL count, dst

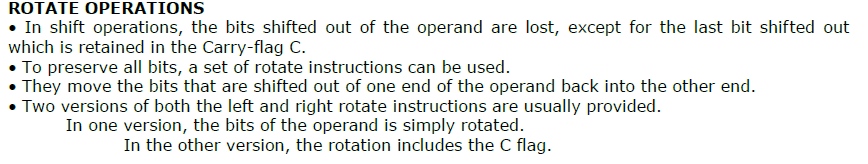
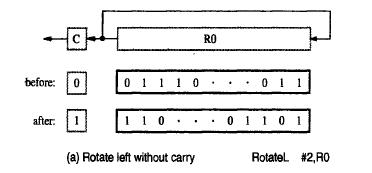
* The count operand may be given as an immediate operand or it may be contained in a processor register.
* Vacated positions are filled with zeros and the bits shifted out are passed through the carry flag C and then dropped.
* The logical shift right instruction, works in the same manner except that it shifts to the right.
* The general form of a logical right shift

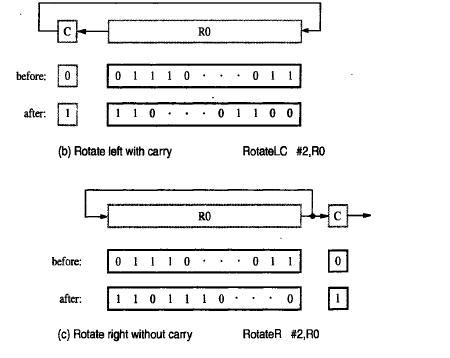
LshiftR count, dst

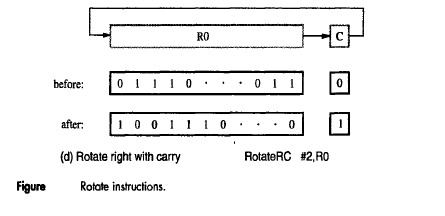
.**Arithmetic Shifts:**

* The 2’s-complement binary number representation that shifting a number one bit position to the left is equivalent to multiplying it by 2, and shifting it to the right is equivalent to dividing it by 2.
* Of course, overflow occurs on shifting left, and the remainder is lost in shifting right.
* Another important observation is that on a right shift the sign bit must be repeated as the fill – in bit for the vacated position.
* This requirement on right shifting distinguishes arithmetic shifts from logical shifts in which the fill – in bit is always 0. Otherwise the two types of shifts are very similar.

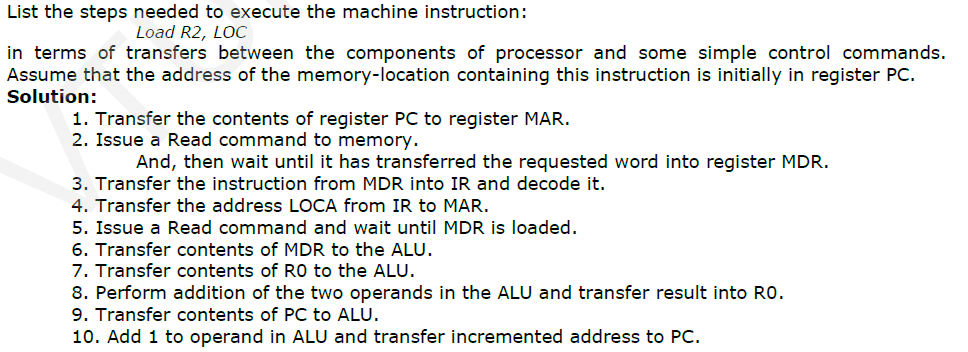


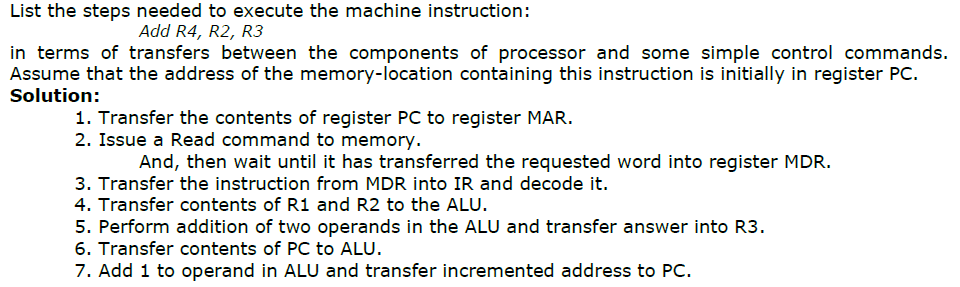
 

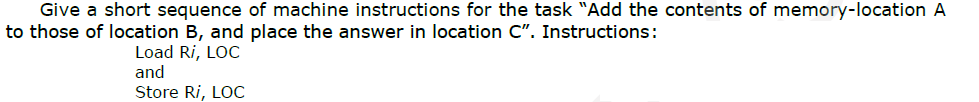


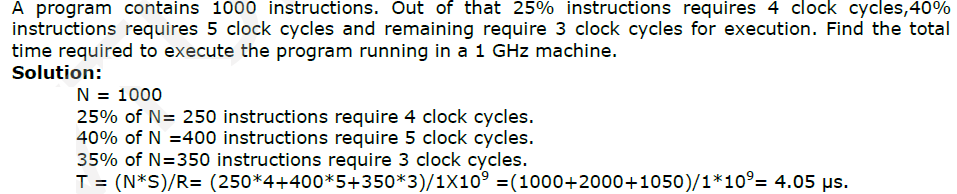


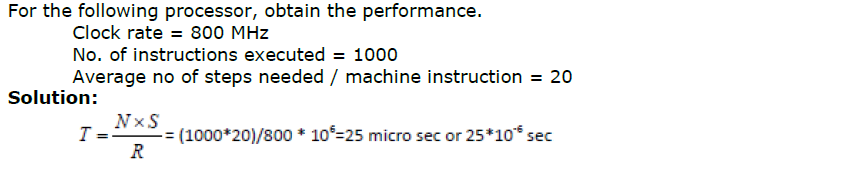
**PROBLEMS**

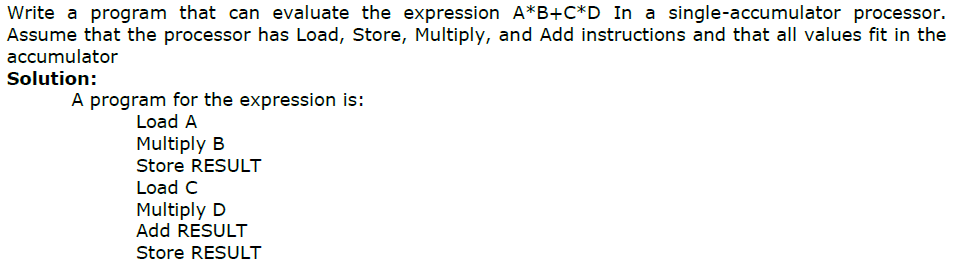
**1.**

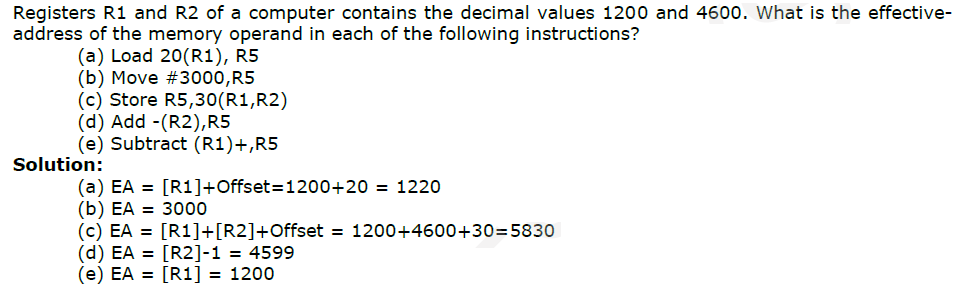
**2.**

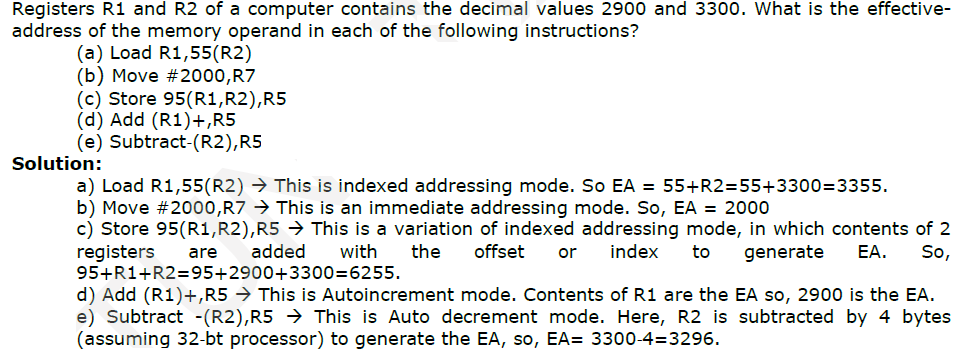
**3.**

**4.**

**5.**

**6.**

**7.**

**8.**

**\_\_\_\_\_\_\_\_\*\*\*\*\*\_\_\_\_\_\_\_\_**