Digital Design and Computer Organization(BCS302)

Module-3

Prepared by: Vinay G
Asst.Prof, Dept. of CSE
Gopalan college of Engineering,Bengaluru.

Basic Structure of Computers

Computer types:-

A computer can be defined as a fast electronic calculating machine that accepts the (data) digitized input information process it according to a list of internally stored instructions and produces the resulting output information.

List of instructions are called computer **program** and internal storage is called computer **memory**.

Types of computers

1. **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.
2. **Note book computers: -** These are compact and portable versions of PC
3. **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.
4. **Enterprise systems: -** These are used for business data processing in medium to large corporations that require much more computing power and storage capacity than work stations. Internets associated with servers have become a dominant worldwide source of all types of information.
5. **Super computers: -** These are used for large scale numerical calculations required in the applications like weather forecasting and aircraft design and simulation.

Functional units:-

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

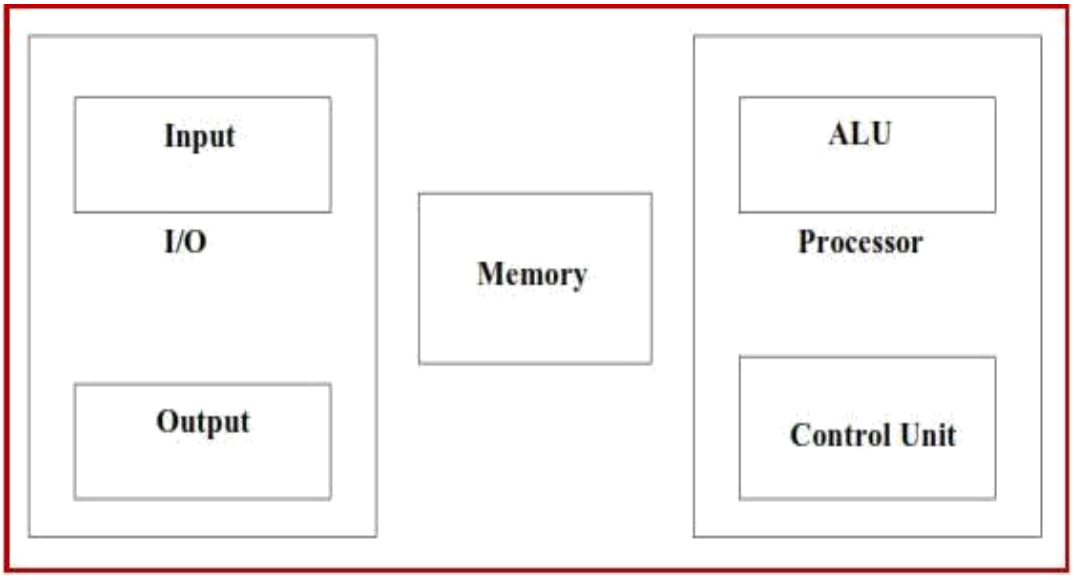

Figure: Functional units of computer

Functions of all computer are:

- Data PROCESSING
- Data STORAGE
- Data MOVEMENT

CONTROL-Coordinates How Information is Used

Information Handled by a Computer:-

Instructions/machine instructions

- Govern the transfer of information within a computer as well as between the computer and its I/O devices
- Specify the arithmetic and logic operations to be performed

Data

- Used as operands by the instructions
- Source program

Encoded in binary code – 0 and 1

- 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 to either memory or processor.

Example: Joysticks, trackballs, mouse, scanners etc are other input devices.

Memory unit: -

The function of the memory unit is store programs and data. It is basically to two types are

1. Primary memory
2. Secondary memory

1) **Primary memory: -** it is a fast memory that operates at the electronics speeds. Programs must be stored in the memory while they are 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 site called word.

To provide easy access to any 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 and programs have to be stored, particularly information that is accessed infrequently.

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

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.

Control unit:-

The control unit 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.

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. The operand at LOCA is fetched and added to the contents of R
3. 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 steps to execute the instructions can be enumerated as below:

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

The fig shows how memory & the processor can be connected. In addition to the ALU & the control circuitry,
the processor contains a number of registers used for several different purposes.
The instruction register (IR):- Holds the instructions that are 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.

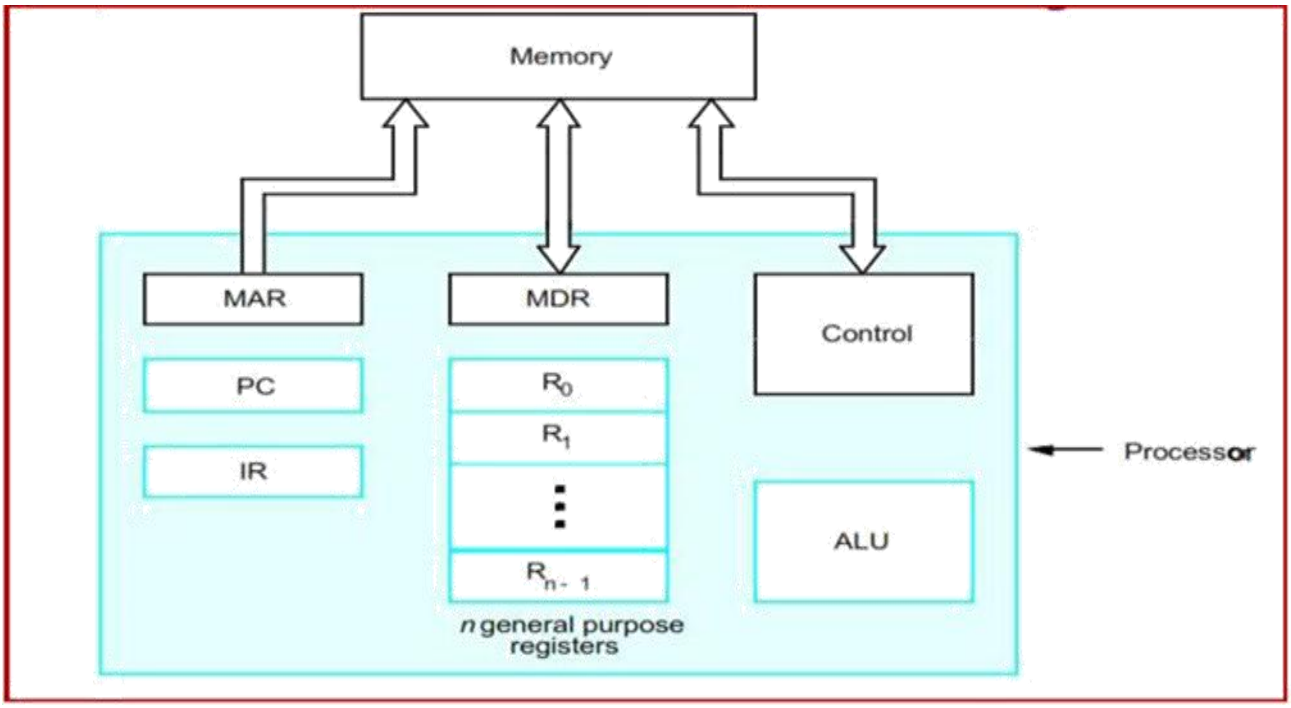

Figure : Connections between the processor and the memory

Operating steps are

1. Programs reside in the memory and usually get these through the input 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.

- Normal execution of a program may be pre-empted (temporarily interrupted) if some devices require urgent servicing, to do this one device raises an **Interrupt signal**.
- An interrupt is a request 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.

Bus structure

- A group of lines that serves a connecting path for several devices is called a **bus**.
- The simplest and most common way of interconnecting various parts of the computer is to use bus.
- 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.
- 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.

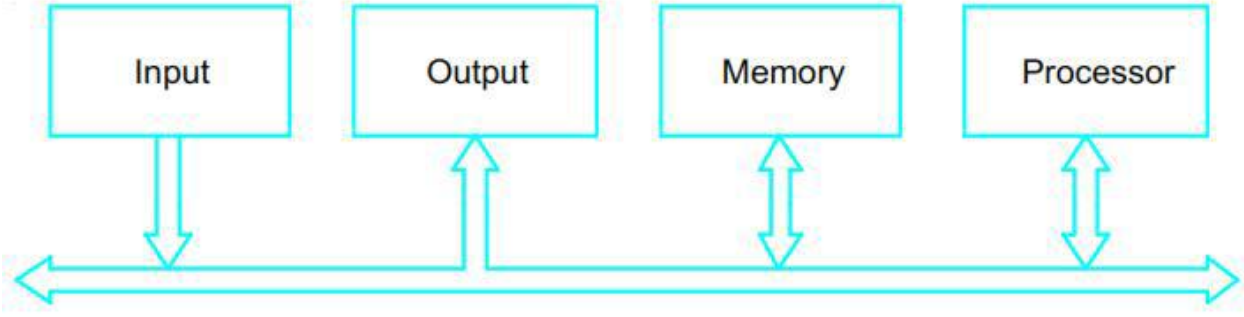

Figure : Single bus structure

Single bus structure is
1. Low cost
2. 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 and time leads to a bit of a problem. This is solved by using cache registers (i.e. 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.
    - Consider the transfer of an encoded character from processor to a character printer.
    - The processor sends character over the bus to the printer buffer.
    - Buffer is an electronic register which holds the information during the transfer of data.
    - After the buffer gets loaded the printer can start printing without future 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 encoded character which is in the buffer.
    - The printer is not available for further transfers until the previous task is completed.
    - During this time the processor can go for any other instruction with other devices.

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. For best performance, it is necessary to design the compiles, the machine instruction set, and the hardware in a coordinated way.
- The total time required to execute the program is elapsed time is a measure of the performance of the entire computer system. It is affected by the speed of the processor, the disk and the printer. The time needed to execute an instruction is called **the processor time**.

Just as 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 individual machine instructions. This hardware comprises the processor and the memory which are usually connected by the bus as shown in the figure.

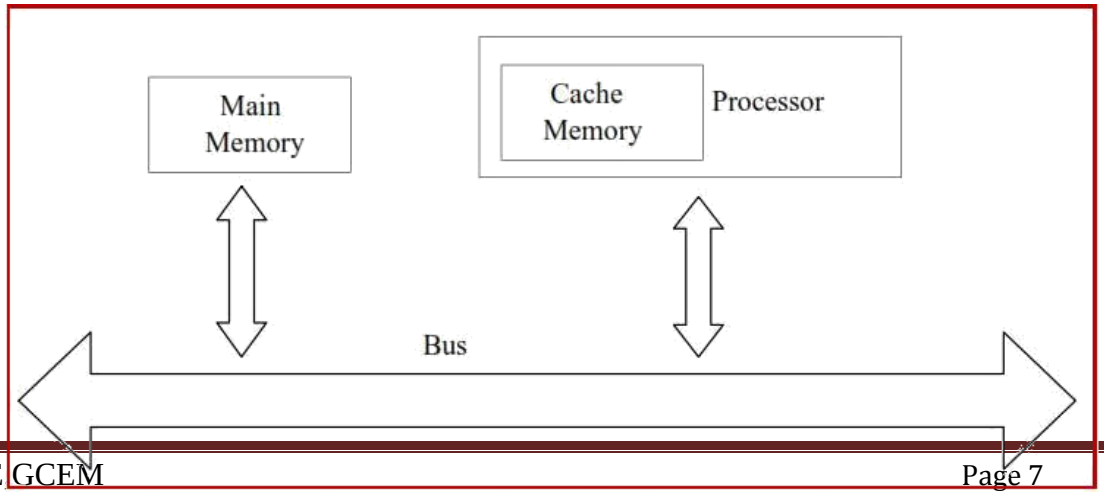

Figure: The processor cache

- Let us examine the flow of program instructions and data between the memories and the processor. At the start of execution, all program instructions and the required data are stored in the main memory.
- As the execution proceeds, instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache later if the same instruction or data item is needed a second time, it is read directly from the cache.
- The processor and relatively small cache memory can be fabricated on a single IC chip. The internal speed of performing the basic steps of instruction processing on chip is very high and is considerably faster than the speed at which the instruction and data can be fetched from the main memory.
- A program will be executed faster if the movement of instructions and data between the main memory and the processor is minimized, which is achieved by using the cache.

Processor clock

- Processor circuits are controlled by a timing signal called clock.
- The clock defines the 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 clock cycle is an important parameter that affects the processor performance
- Its inverse is the Clock rate, **R=1/p**. which is measured in cycles per second.
- Processor used in today's personal computer and work station has a clock rates that range from a few hundred million to over a billion cycles per second.

Basic performance equation

We now focus our attention on the processor time component of the total elapsed time.

- T – processor time required to execute a program that has been prepared in high-level language
- N – number of actual machine language instructions needed to complete the execution (note: loop)
- S – Average number of basic steps needed to execute one machine instruction. Each step completes in one clock cycle
- R – clock rate is cycles per second

$$T = \frac{N \times S}{R}$$

This is often referred to as the basic performance equation.
We must emphasize that N, S & R are not independent parameters changing one may affect another.

Introducing a new feature in the design of a processor will lead to improved performance only if the overall result is to reduce the value of T.

Clock rate

These are two possibilities for increasing the clock rate 'R'.

- Improving the IC technology makes logical circuit faster, which reduces the time of execution of basic steps. This allows the clock period P, to be reduced and the clock rate R to be increased.
- Reducing the amount of processing done in one basic step also makes it possible to reduce the clock period P..

Performance measurements

The performance measure is the time taken by the computer to execute a given bench mark.

- **Benchmark refers to standard task used to measure how well a processor operates**. To evaluate the performance of Computers, a non-profit organization known as SPEC-(System Performance Evaluation Corporation) selects and publishes application programs for different domains.
- Accordingly, it gives performance measure for a computer as the time required to execute a given benchmark program

$$\text{SPEC rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

If the SPEC rating = 50 Means that the computer under test is 50 times as fast as the ultra SPARC 10. This is repeated for all the programs in the SPEC suit, and the geometric mean of the result is computed.

Let SPECi be the rating for program 'i' in the suite. The overall SPEC rating for the computer is given by

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} \text{SPECi} \right)^{1/n}$$

Where 'n' = number of programs in suite.

Since actual execution time is measured the SPEC rating is a measure of the combined effect of all factors affecting performance, including the compiler, the OS, the processor, the memory of comp being tested.

Machine Instructions and Programs

Memory locations and addresses

- Number and character operands, as well as instructions, are stored in the memory of a computer. The memory consists of many millions of storage cells, each of which can store a bit of information having the value 0 or 1.
- Because a single bit represents a very small amount of information, bits are seldom handled individually. The usual approach is to deal with them in groups of fixed size.
- For this purpose, the memory is organized so that a group of n bits can be stored or retrieved in a single, basic operation. Each group of n bits is referred to as a word of information, and n is called the word length. The memory of a computer can be schematically represented as a collection of words.

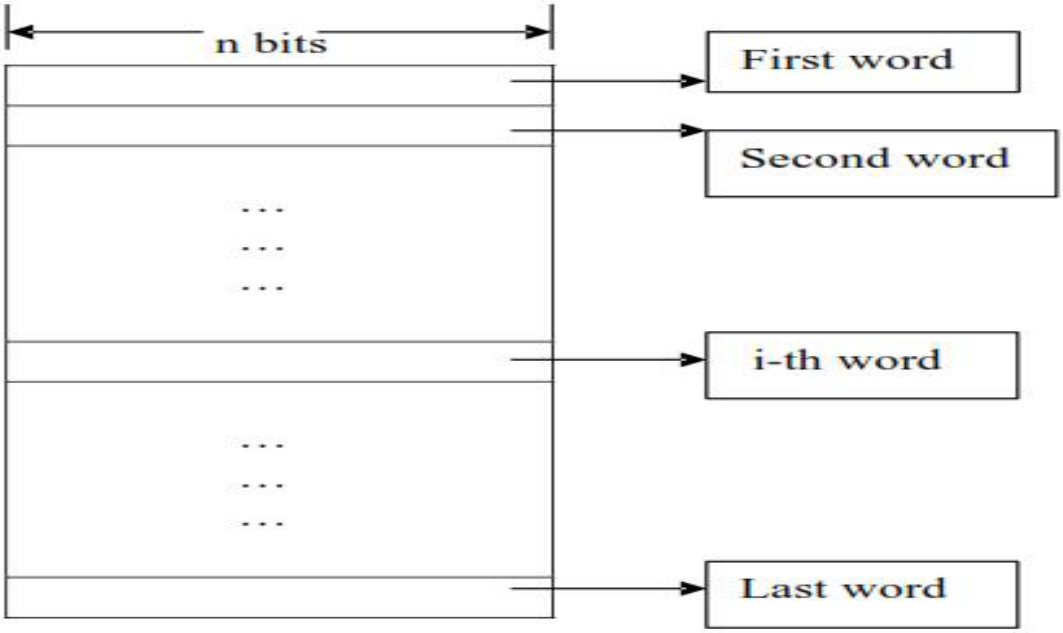

Figure : A signed integer

Modern computers have word lengths that typically range from 16 to 64 bits. If the word length of a computer is 32 bits, a single word can store a 32-bit 2's complement number or four ASCII characters, each occupying 8 bits. A unit of 8 bits is called a byte.

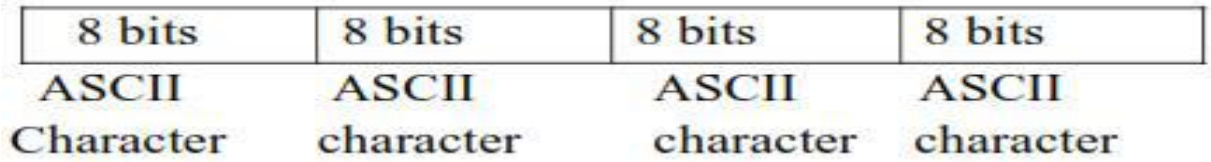

Figure : Four characters

- Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires distinct names or addresses for each item location. It is customary to use numbers from 0 through 2^K-1, for some suitable values of k, as the addresses of successive locations in the memory.
  - The 2^k addresses constitute the address space of the computer, and the memory can have up to 2^k addressable locations. 24-bit address generates an address space of 2^*24* (16,777,216) locations. A 32-bit address creates an address space of 2^32 or 4G (4 giga) locations.

Byte Addressability:-

- It is impractical to assign distinct addresses to individual bit locations in the memory.
- The most practical assignment is to have successive addresses refer to successive byte locations in the memory – byte-addressable memory.
- Byte locations have addresses 0, 1, 2, … If word length is 32 bits, they successive words are located at addresses 0, 4, 8,…

.

Big-Endian and Little-Endian Assignments:-

Big-Endian: lower byte addresses are used for the most significant bytes (the left most bytes) of the word

Little-Endian: opposite ordering. lower byte addresses are used for the less significant bytes (the rightmost bytes)of the word

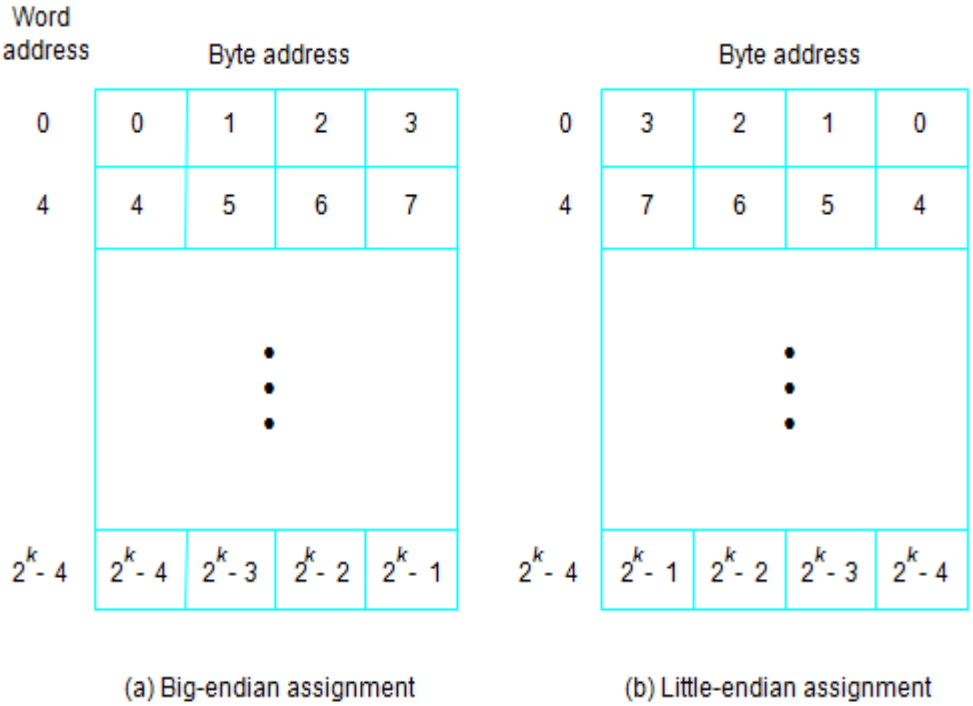

Figure Byte and word addressing

Word Alignment:-

- Address ordering of bytes
- Word alignment
  - Words are said to be aligned in memory if they begin at a byte addressing. that is a multiple of the num of bytes in a word.
    - 16-bit word: word addresses: 0, 2, 4,….
    - 32-bit word: word addresses: 0, 4, 8,….
    - 64-bit word: word addresses: 0, 8,16,….

Accessing Numbers, Characters, And Character Strings:-

- A number usually occupies one word. It can be accessed in the memory by specifying its word address.
- Similarly, individual characters can be accessed by their byte address.
- In many applications, it is necessary to handle character strings of variable length. The beginning of the string is indicated by giving the address of the byte containing its first character. Successive byte locations contain successive characters of the string.
  There are two ways to indicate the length of the string.
- A special control character with the meaning "end of string" can be used as the last character in the string, or a separate memory word location or processor register can contain a number indicating the length of the string in bytes.
- Word location or processor register can contain a number indicating the length of the string in bytes.

Memory operations

Both program instructions and data operands are stored in the memory. To execute an instruction, the processor control circuits must cause the word (or words) containing the instruction to be transferred from the memory to the processor. Operands and results must also be moved between the memory and the processor.

Thus, two basic operations involving the memory are needed, namely, Load (or Read or Fetch) and Store (or Write).

Load (or Read or Fetch)

- The load operation transfers a copy of the contents of a specific memory location, memory contents are unchanged.
- To start a Load operation, the processor sends the address of the desired location to the memory and requests that its contents are to be read.
- The memory reads the data stored at that address and sends them to the processor

Store operation:

- The store operation transfers an item of information from the processor to a specific memory location, destroying the former contents of that location.
- The processor sends the address of the desired location to the memory, together with the data to be written into that location.
- An information item of either one word or byte is transferred b/w Memory and Processor in a single operation

An information item of either one word or one byte can be transferred between the processor and the memory in a single operation. Actually, this is transfer in between the CPU register & main memory.

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 processor registers, or registers in the I/O subsystem..
- 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 of the location. Thus, the expression is

R1 ← [LOC]

Means that the contents of memory location LOC are transferred into processor register R1.

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 expression always denotes a value, and the left-hand side is the name of a location where the value is to be places, overwriting the old contents of that location.

Assembly Language Notation:-

- We can use assembly language format to represent machine instructions and programs. For example, an instruction that causes the transfer described above, 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 old contents of register R1 are overwritten.
- The second example of adding two numbers contained in processor registers R1 and R2 and placing their sum in R3 can be specified by the assembly language statement.

  Add R1, R2, R3

Basic instruction types

The operation of adding two numbers is a fundamental capability in any computer. The statement

- C = A + B

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. When the program containing this statement is compiled, the three variables, A, B and C, are assigned to distinct locations in the memory. We will use the variable names to refer to the corresponding memory location addresses.
According to address reference there three type of instruction:

- Three address instruction
- Two address instruction
- One address instruction

Three address instruction:

The general format for three address instruction is:

Operation source1,source2,destination

The address instruction can be represented symbolically to execute RTN i.e
C<- [A]+[B]

Add A,B,C

Where A,B,C are the variables. These variable names are assigned to the distinct location in the memory. In this instruction operand A and B are the source operand and operand C is the destination operand.
The number of bits required to represent such instruction include:

- Bits required to specify the three memory addresses of the three operands. If n-bits required to specify one memory address,3n bits are required to specify three memory address.
- Bits required to specify the operation.

Two address instruction:
The general format for two address instruction is:
Operation source1,destination
The address instruction can be represented symbolically to execute RTN i.e
C<- [A]+[B]
Add A,B
Where A,B are the variables. These variable names are assigned to the distinct location in the memory. In this instruction operand A is the source operand and operand B serve as both source and destination operand.
The number of bits required to represent such instruction include:

- Bits required specifying the two memory addresses of the two operands. If n-bits required to specify one memory address, 2n bits are required to specify two memory address.
- Bits required specifying the operation.

One address instruction:
The general format for two address instruction is:

Operation source1
The address instruction can be represented symbolically to execute RTN i.e

C<- [A]+[B]
Load A
Add B
Store C

Instruction loads the contents of variable A into the processor register called accumulator and add the contents of accumulator to register B and store the result back into accumulator. Then store the content of accumulator into register C
The number of bits required to represent such instruction include:

- Bits required to specify the one memory addresses of the one operands. If n-bits required to specify one memory address,1n bits are required to specify one memory address.
- Bits required to specify the operation

Example: Evaluate (A+B) * (C+D)

- Three-Address

```
1. ADD   A, B,R1    ; R1 ← M[A] + M[B]
2. ADD   C, D,R2    ; R2 ← M[C] + M[D]
3. MUL   R1,R2,X    ; M[X] ← R1 * R2
```

Example: Evaluate (A+B) * (C+D)

- Two-Address

```
1. MOV   A,R1       ; R1 ← M[A]
2. ADD   B,R1       ; R1 ← R1 + M[B]
3. MOV   C,R2       ; R2 ← M[C]
4. ADD   D,R2       ; R2 ← R2 + M[D]
5. MUL   R2, R1     ; R1 ← R1 * R2
6. MOV   R1,X       ; M[X] ← R1
```

Example: Evaluate (A+B) * (C+D)

- One-Address

```
1. LOAD   A         ; AC ← M[A]
2. ADD    B         ; AC ← AC + M[B]
3. STORE  T         ; M[T] ← AC
4. LOAD   C         ; AC ← M[C]
5. ADD    D         ; AC ← AC + M[D]
6. MUL    T         ; AC ← AC * M[T]
7. STORE  X         ; M[X] ← AC
```

Example: Evaluate (A+B) * (C+D)

- Zero-Address

```
1.  PUSH    A        ;TOS ← A
2.  PUSH    B        ;TOS ← B
3.  ADD              ;TOS ← (A + B)
4.  PUSH    C        ;TOS ← C
5.  PUSH    D        ;TOS ← D
6.  ADD              ;TOS ← (C + D)
7.  MUL              ;TOS ← (C+D)*(A+B)
8.  POP     X        ; M[X] ← TOS
```

➢ ***INSTRUCTION EXECUTION AND STRAIGHT-LINE SEQUENCING***

we used the task C ← [A] + [B] for illustration. Figure shows a possible program segment for this task as it appears in the memory of a computer.

- The memory is byte addressable, word length is 32 bits and the processor has a number of registers. We use a two address instruction format and instructions stored in successive memory 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.
- 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 (*i* in our example) 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.

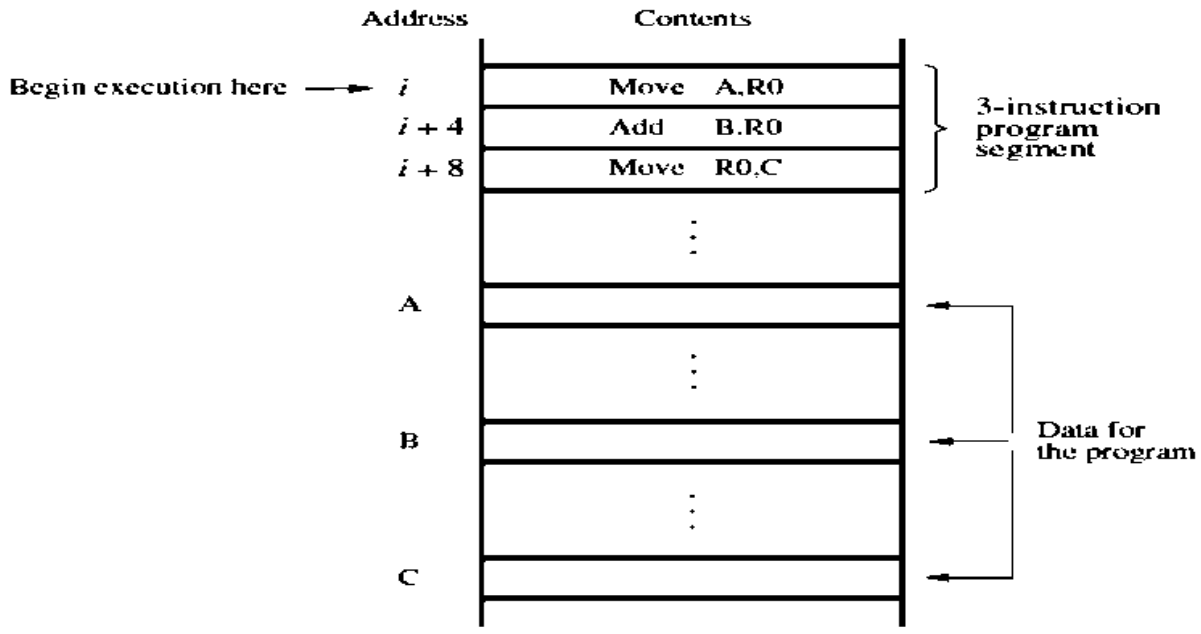

Figure: A Program for C←[A]+[B]

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. At the start of the second phase, called ***instruction execute***, 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. At some point during this two-phase procedure, the contents of the PC are advanced to point to the next instruction.

When the execute phase of an instruction is completed, the PC contains the address of the next instruction, and a new instruction fetch phase can begin. In most processors, the execute phase itself is divided into a small number of distinct phases corresponding to fetching operands, performing the operation, and storing the result

- **Branching**

Every time it is not possible to store a program in the consecutive memory location. After execution of decision making instruction we have to follow one of the two program sequence. IN such case we cannot use the straight –line sequencing. Here we have to use the branch instruction to transfer the program control from one straight-lone sequence to another straight line sequencing.

Consider the task of adding a list of *n* numbers. The program outlined in Figure 2.9 is a generalization of the program in above Figure.

The addresses of the memory locations containing the *n* numbers are symbolically given as NUM1, NUM2, . . . , NUM*n*, 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 2.10. 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 to R0.

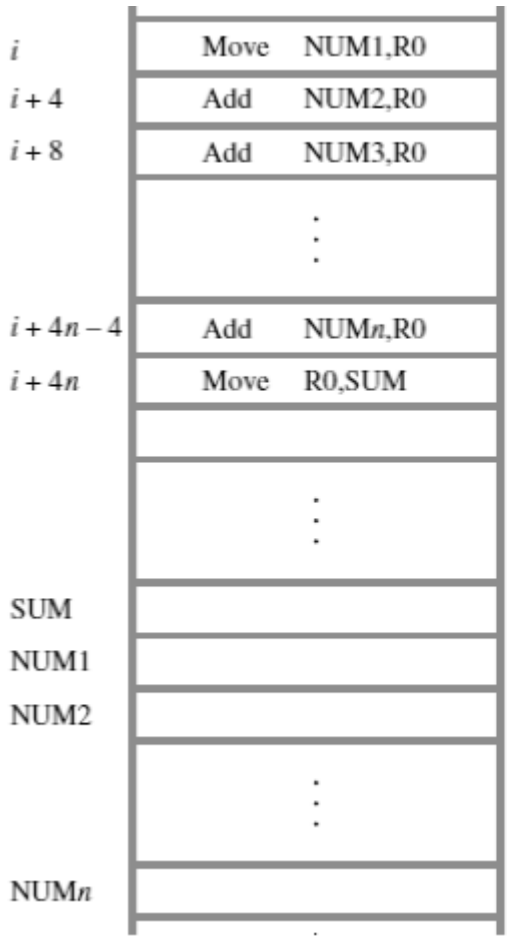

Figure: A straight-line program for adding n numbers

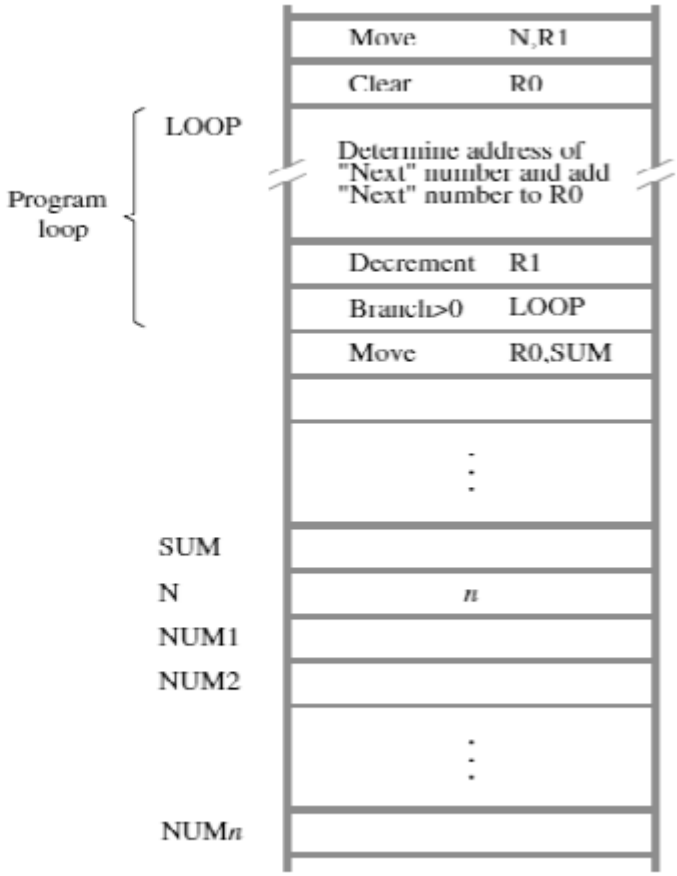

Figure: Using a loop to add n numbers

- **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.

- **Carry/borrow flag**: The carry bit is set when the summation of two 8 bit number is greater than 1111 1111.A borrow bit is generated when a large number is subtracted from a smaller number.

- **Zero flag**: The zero bit is set when the contents of register are zero after any operation. This happens not only when we decrement the register, but also any arithmetic and logical operation.

- **Negative or sign flag**: In 2's complement arithmetic, the most significant bit is a sign bit. If the bit is logic 1,then number is negative number, otherwise a positive number.
- **Auxiliary Flag**:The auxiliary carry bit of status register is set when an addition in the 4 bit cause a carry into the fifth bit.
- **Overflow flag**:This flag is set if the result of signed operation is too large to fit in the number of bit available to represent it.
- **Parity flag**:When the result of an operation leaves the even number of 1's then parity is set.

Generating Memory Addresses

Let us return to fig b. The purpose of the instruction block at LOOP is to add a different number from the list during each pass through the loop. Hence, the Add instruction in the block must refer to a different address during each pass. How are the addresses to be specified? The memory operand address cannot be given directly in a single Add instruction in the loop. Otherwise, it would need to be modified on each pass through the loop.

The instruction set of a computer typically provides a number of such methods, called addressing modes. While the details differ from one computer to another, the underlying concepts are the same.

Addressing Modes

Programs are normally written in a high-level language, which enables the programmer to use constants, local and global variables, pointers, and arrays. ***The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.***

Table: Generic addressing modes

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

EA = effective address
Value = a signed number

Implementation of variables 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. Thus, the value can be changed as needed using appropriate instructions.

1) **REGISTER MODE -** The operand is the contents of a processor register; the name (address) of the register is given in the instruction.

   Move r1,r2

   The effective address of register mode is **Ea=Ri**

2) **ABSOLUTE 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).

   Move LOC, R2

   The effective address of absolute mode is **Ea=Loc**

3) **IMMEDIATE MODE –** The operand is given explicitly in the instruction. For example, the instruction **Move #200, R0**

Places the value 200 in register R0. Clearly, the immediate mode is only used to specify the value of a source operand. Using a subscript to denote the immediate mode is not appropriate in assembly languages. 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

The effective address of immediate mode is

Operand=value

4) INDIRECTION AND POINTERS:-

In the addressing modes that follow, the instruction does not give the operand or its address explicitly, Instead, it provides information from which the memory address of the operand can be determined. We refer to this address as the effective address (EA) of the operand.

Indirect mode – The effective address of the operand is the contents of a register or Memory location whose address appears in the instruction.

Move (r1),r2

For example: Move #num,r1

Move (r1),r2

In the above instruction, the address of the num is copied into the register r1, whenever second instruction is executed ,the content of the num is moved to the register r2**.**

The effective address of indirect mode is

Ea=(Ri) //register

Ea=(Loc) //memory

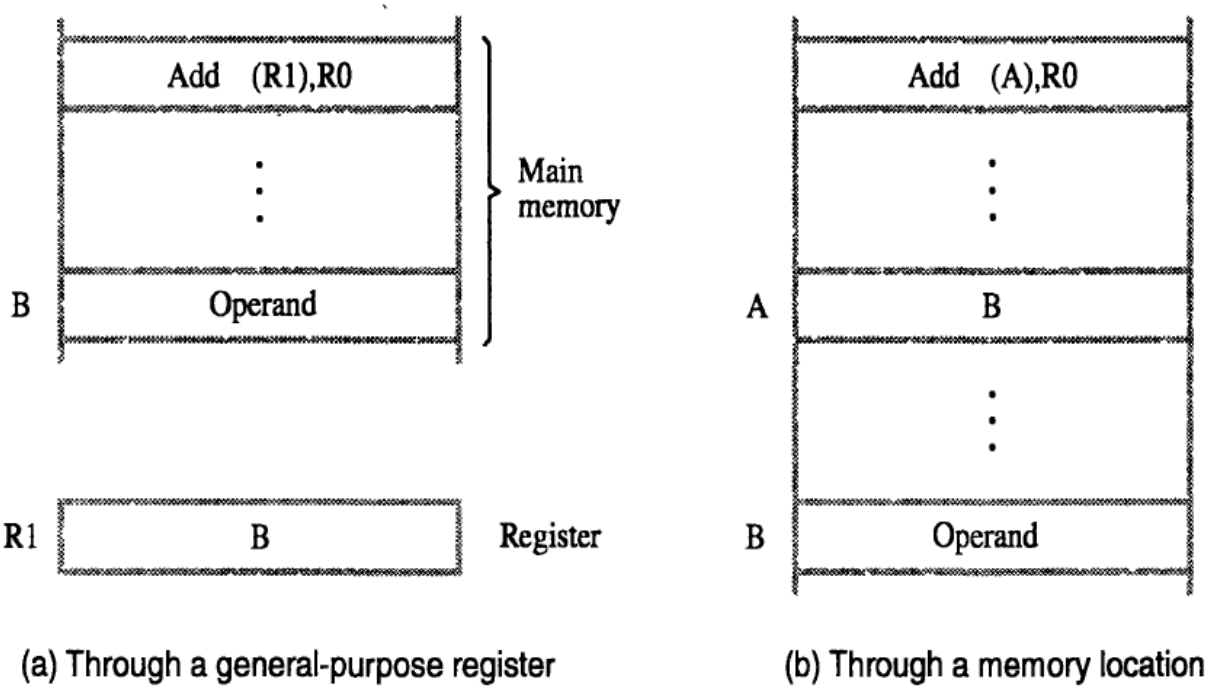

Figure: Indirect Addressing

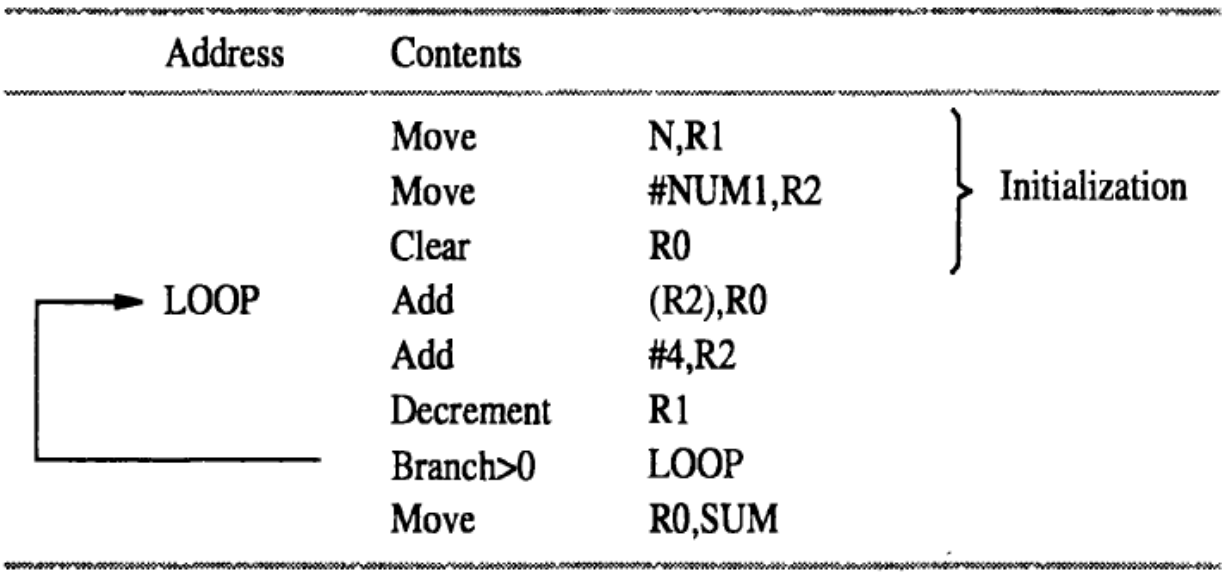

Figure: Use of indirect addressing in the program

In the program shown 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 R1 and uses the immediate addressing mode to place the address value NUM1, which is the address of the first number in the list, into R2.

Then it clears R0 to 0. The first two instructions in the loop implement the unspecified instruction block starting at LOOP. The first time through the loop, the Instruction Add (R2), R0 fetches the operand at location NUM1and adds it to R0. The second Add instruction adds 4 to the contents of the pointer R2, so that it will contain the address value NUM2 when the above instruction is executed in the second pass through the loop.

As another example of pointers, consider the C-language statement

A = *B; Where B is a pointer variable

This statement may be compiled into

Move B, R1
Move (R1), A

Using indirect addressing through memory, the same action can be achieved with

Move (B), A

Indirect addressing through registers is used extensively. The above program shows the flexibility it provides. Also, when absolute addressing is not available, indirect addressing through registers makes it possible to access global variables by first loading the operand's address in a register.

INDEXING AND ARRAYS:-

A different kind of flexibility for accessing operands is useful in dealing with lists and arrays.

Index mode – the effective address of the operand is generated by adding a constant value to the contents of a register

Move X(r1),r2

The register use may be either a special register provided for this purpose, or, more commonly, it may be any one of a set of general-purpose registers in the processor.

In either case, it is referred to as 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 + [Rj]**

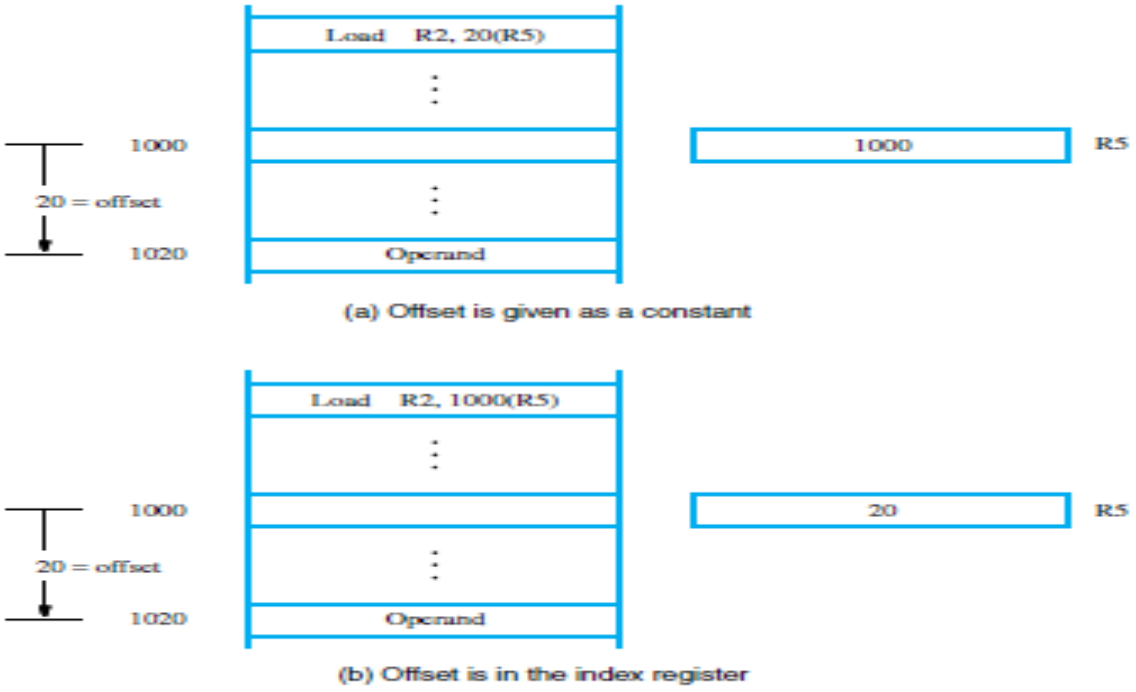

Figure: Indexed addressing

Above Fig illustrates two ways of using the Index mode. In fig a, the index register, R1, contains the address of a memory location, and the value X defines an offset (also called a displacement) from this address to the location where the operand is found. An alternative use is illustrated in fig (5). Here, the constant X corresponds to a memory address, and the contents of the index register define the offset to the operand.

In either case, the effective address is the sum of two values; one is given explicitly in the instruction, and the other is stored in a register**.**

RELATIVE ADDRESSING:-

Relative mode: The effective address is determined by the index mode using program counter in place of the general purpose processor register.

EA=[pc]+x

This addressing mode commonly used to specify the target address in the branch instruction. For example; **Branch > 0 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

Additional modes

Auto increment mode: The effective address of the operand is the content of a register specified in the instruction. After accessing the operand ,the contents of this register are incremented to the address the next instruction

For example: **Mov (r2)+,r0**

The effective address of auto increment mode is **EA=[Ri]**

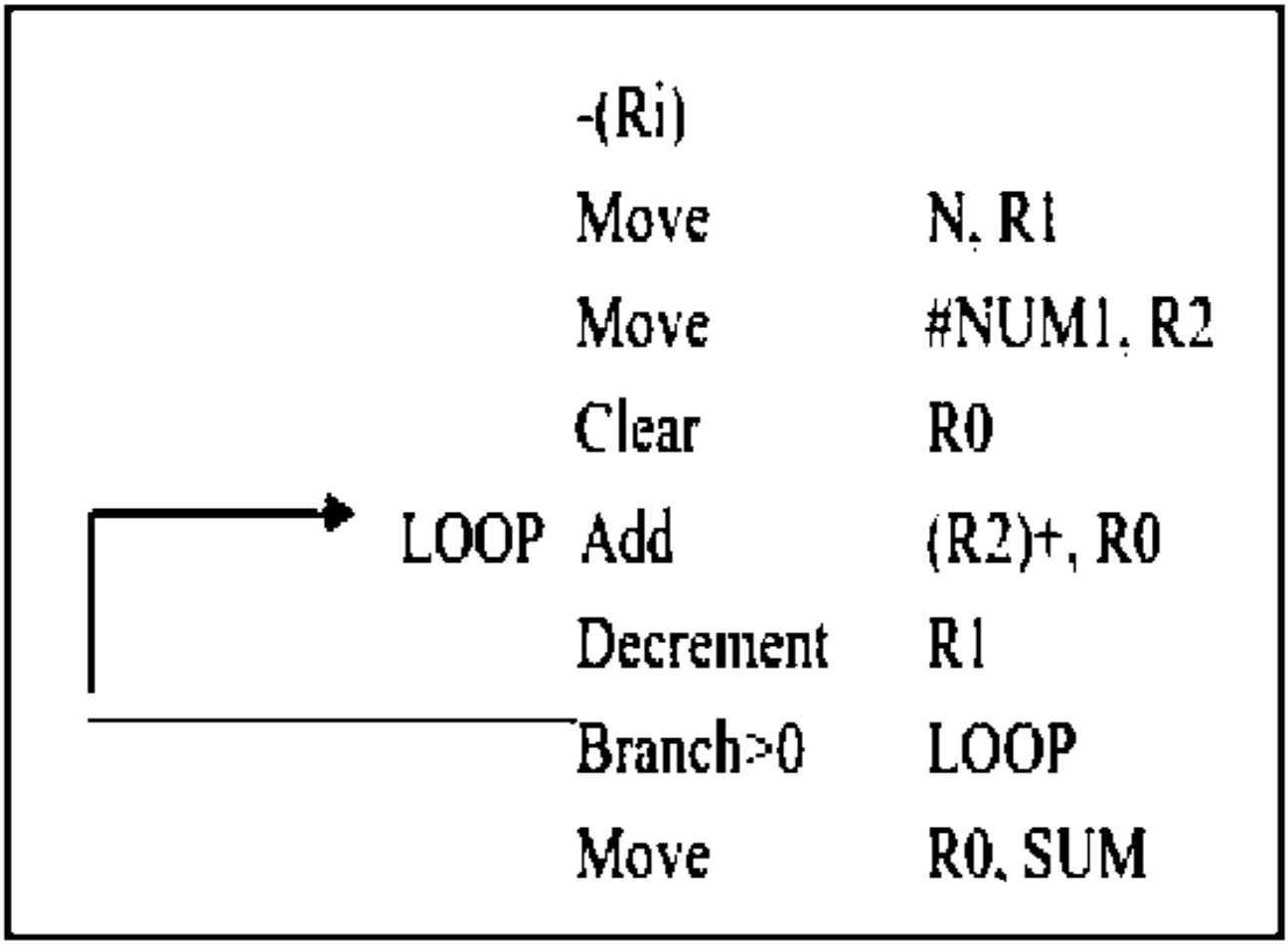

Fig:- The Auto increment addressing mode used in program

Auto decrement mode – The contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand. Notation is –(Ri)

Assembly language:

- Machine instructions are represented by patterns of 0s and 1s. Such patterns are awkward to deal with when discussing or preparing programs. Therefore, we use symbolic names to represent the patterns. So far, we have used normal words, such as Load, Store, Add, and Branch, for the instruction operations to represent the corresponding binary code patterns.
- When writing programs for a specific computer, such words are normally replaced by acronyms called ***mnemonics***, such as LD, ST, ADD, and BR.
- A shorthand notation is also useful when identifying registers, such as R3 for register 3. Finally, symbols such as LOC may be defined as needed to represent particular memory locations.
- A complete set of such symbolic names and rules for their use constitutes a programming language, generally referred to as an ***assembly language***.
- The set of rules for using the mnemonics and for specification of complete instructions and programs is called the ***syntax* of the language**.

- Programs written in an assembly language can be automatically translated into a sequence of machine instructions by a program called an ***assembler*.** The assembler program is one of a collection of utility programs that are a part of the system software of a computer.
- The user program in its original alphanumeric text format is called a ***source program,*** and the assembled machine-language program is called an ***object program***.

To make programming easier, usually write program in assembly language. They then translate the assembly level language to machine level language. so that it can be loaded into memory and executed.

The assembly text is usually divided into fields, separated by space and tabs. The format for typical line from assembly language program can be given as

Labels: Mnemonic Operand1, Opreand2 ; comment

- The first field, which is optional, is the label field, used to specify symbolic labels.
- The second field is mnemonic, which is compulsory. All instruction must contain a mnemonic.
- The third and following fields are operand. The presence of the operands depends on the instruction. Some instruction have no operands, some have two or one operands.

Assembler Directives:

There are some instructions in assembly language program which are not a part of processor instruction set. These instructions are instruction to the assembler, linker, loader. Then these instruction are called **assembler directives**

The assembly language requires assembler directives for performing following basic function.

- To indicate starting location of the memory where the data block is stored and starting location of the memory where code is stored.
- To define different types of variables or to set aside one or more storage location of corresponding data type in memory.
- To indicate the assembler about the value of the variables.
- To indicate start and end of subroutine program

The commonly used directives are :

ORIGIN: This directive tells assembler that where to place the data block in the memory or where to start loading of object program in the memory. That is ORIGIN directive specifies the starting memory location for data and code.

DB,DW,DD,DQ,DT: These directive are used to define different types of variables or to set aside one or more storage location of the corresponding data types in the memory. These are know as data control directives.

DB-define byte
DD-define double-word
DW-define word
DQ-define Quad-word
DT=define ten bytes

EQU directives: The EQU directive is used to redefine a data name or variable with another data name, variable or immediate value. The directive should be define in a program before it is referenced.

NUM EQU 200

ST EQU 'hello'

PROC directives: The procedure in the programs can be defined by PROC directive. The procedure name must be present, must be unique and must follow name conventions for the language. After PROC directive the term NEAR or FOR are issued to specify the type of the procedure.

ENDP directive: ENDP directive is used along with the PROC directive .The ENDP define the end of procedure.

RESERVE Directive: RESERVE directive is used to reserve the number of memory location for the given variable.

NUM RESERVE 200

As the assembler scans through a source programs, it keeps track of all names and the numerical values that Correspond to them in a symbol table. Thus, when a name appears a second time, it is replaced with its value from the table. A problem arises when a name appears as an operand before it is given a value. For example,

This happens if a forward branch is required. A simple solution to this problem is to have the assembler scan through the source program twice.

- During the first pass, it creates a complete symbol table. At the end of this pass, all names will have been assigned numerical values. The assembler then goes through the source program a second time and substitutes values for all names from the symbol table. Such an assembler is called a **two-pass assembler**.

The assembler stores the object program on a magnetic disk. The object program must be loaded into the memory of the computer before it is executed. For this to happen, another utility program called a loader must already be in the memory.

When the object program begins executing, it proceeds to completion unless there are logical errors in the program. The user must be able to find errors easily. The assembler can detect and report syntax errors. To help the user find other programming errors, the system software usually includes a debugger program. This program enables the user to stop execution of the object program at some points of interest and to examine the contents of various processor registers and memory locations.

	Memory address label	Operation	Addressing or data information
Assembler directives	SUM	EQU	200
		ORIGIN	204
	N	DATAWORD	100
	NUM1	RESERVE	400
		ORIGIN	100
Statements that generate machine instructions	START	MOVE	N,R1
		MOVE	#NUM1,R2
		CLR	R0
	LOOP	ADD	(R2),R0
		ADD	#4,R2
		DEC	R1
		BGTZ	LOOP
		MOVE	R0,SUM
Assembler directives		RETURN	
		END	START

Figure: Assembly language representation for the program

Number Notation:-

When dealing with numerical values, it is often convenient to use the familiar decimal notation. Of course, these values are stored in the computer as binary numbers. In some situations, it is more convenient to specify the binary patterns directly. Most assemblers allow numerical values to be specified in different ways, using conventions that are defined by the assembly language syntax.

- Consider, for example, the number 93, which is represented by the 8-bit binary number 01011101. If this value is to be used an immediate operand, it can be given as a decimal number, as in the instructions.

ADD #93, R1

Or as a binary number identified by a prefix symbol such as a percent sign, as in
ADD #%01011101, R1

- Binary numbers can be written more compactly as hexadecimal, or hex, numbers, in which four bits are represented by a single hex digit. In hexadecimal representation, the decimal value 93 becomes 5D. In assembly language, a hex representation is often identified by a dollar sign prefix. Thus, we would write

ADD #$5D, R1

Basic input and output operations:

Consider 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 transfer of data between keyboard and processor and display device is called Input /Output data transfer or I/O data transfer
- The rate of data transfer from the keyboard to a computer is limited by the typing speed of the user, which is unlikely to exceed a few characters per second.
- The rate of output transfers from the computer to the display is much higher. Due to the speed between these devices we have to synchronization mechanism for proper transfer of data between them.

Figure shows the **typical bus connection for processor, keyboard and display**. The DATAIN and DATAOUT are the register by which the processor reads the contents from keyboard and sends the data for display respectively. SIN and SOUT are status bit used to synchronize data transfer between keyboard and processor and data transfer between display and processor respectively.

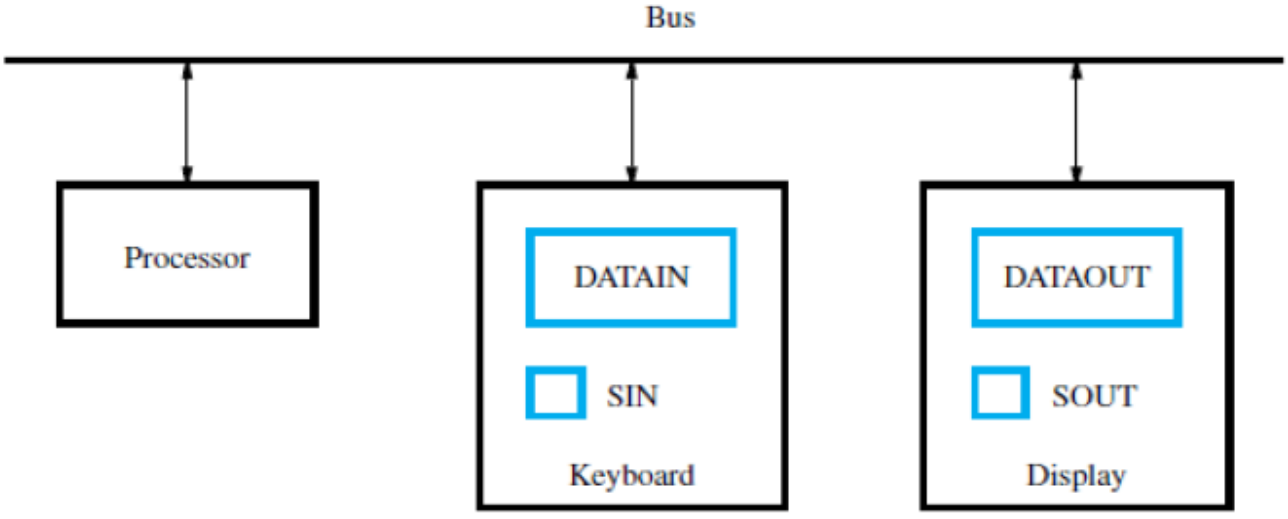

Figure: Bus connection for processor, keyboard, and display

- When a key is pressed, the corresponding character code is stored in the DATAIN register and SIN status bit is set to indicate that the valid character code is available in the DATAIN register. Under the program control processor check the SIN bit and when it fiends SIN =1, it read the contents of the DATAIN register. After the completion of read operation SIN is automatically cleared to 0.and process repeats.
- When the character is too transferred from the processor to the display, DATAOUT register and SOUT status bit are used. Under program control, processor checks SOUT bit. If SOUT=1indicates that the display is ready to receive character. Therefore, when processor wants to transfer data to the DATAOUT register and clears SOUT status to 0 and process repeats

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. This section will describe stacks, as well as a closely related data structure called a queue.
- Data operated on by a program can be organized in a variety of ways.We have already encountered data structured as lists. Now, we consider an important data structure known as a 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.
- Another descriptive phrase, last-in-first-out (LIFO) stack, is also used to describe this type of storage mechanism; the last data item placed on the stack is the first one removed when retrieval begins
- . The terms push and pop are used to describe placing a new item on the stack and removing the top item from the stack, respectively.

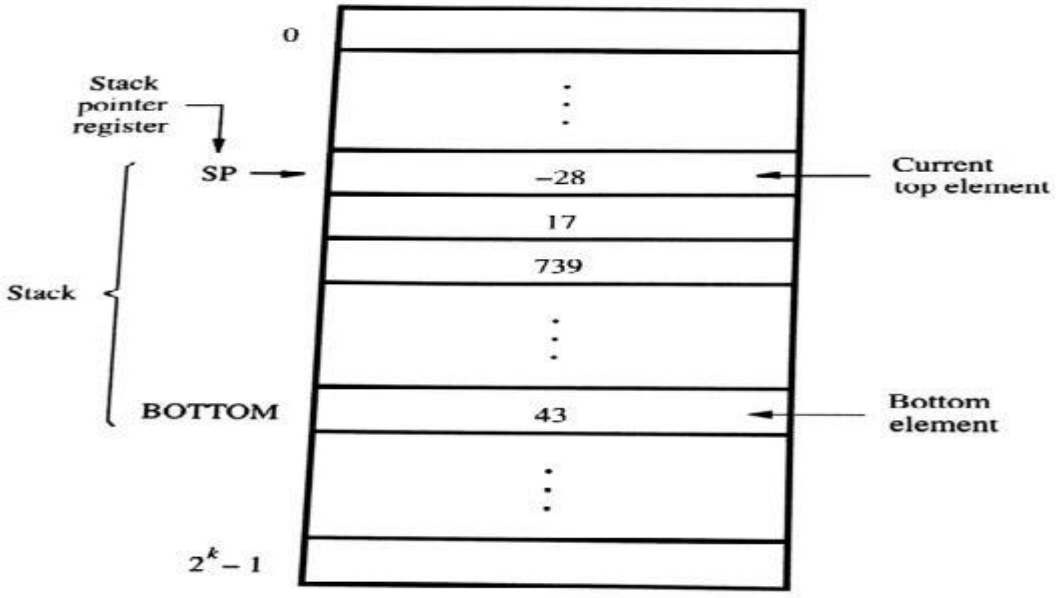

Figure: A stack of words in the memory

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. is either completely full or completely empty.

Figure shows an example of a stack of word data items. The stack contains numerical values, with 43 at the bottom and −28 at the top. The stack pointer, SP, is used to keep track of the address of the element of the stack that is at the top at any given time.

If we assume a byte-addressable memory with a 32-bit word length, the push operation can be implemented as

Subtract SP, SP, #4
Store R*j*, (SP)

where the Subtract instruction subtracts 4 from the contents of SP and places the result in SP. Assuming that the new item to be pushed on the stack is in processor register R*j*, the Store instruction will place this value on the stack. These two instructions copy the word from R*j* onto the top of the stack, decrementing the stack pointer by 4 before the store (push) operation.

The pop operation can be implemented as

Load R*j*, (SP)
Add SP, SP, #4

These two instructions load (pop) the top value from the stack into register R*j* and then increment the stack pointer by 4 so that it points to the new top element. Below Figure shows the effect of each of these operations on the stack in next Figure.

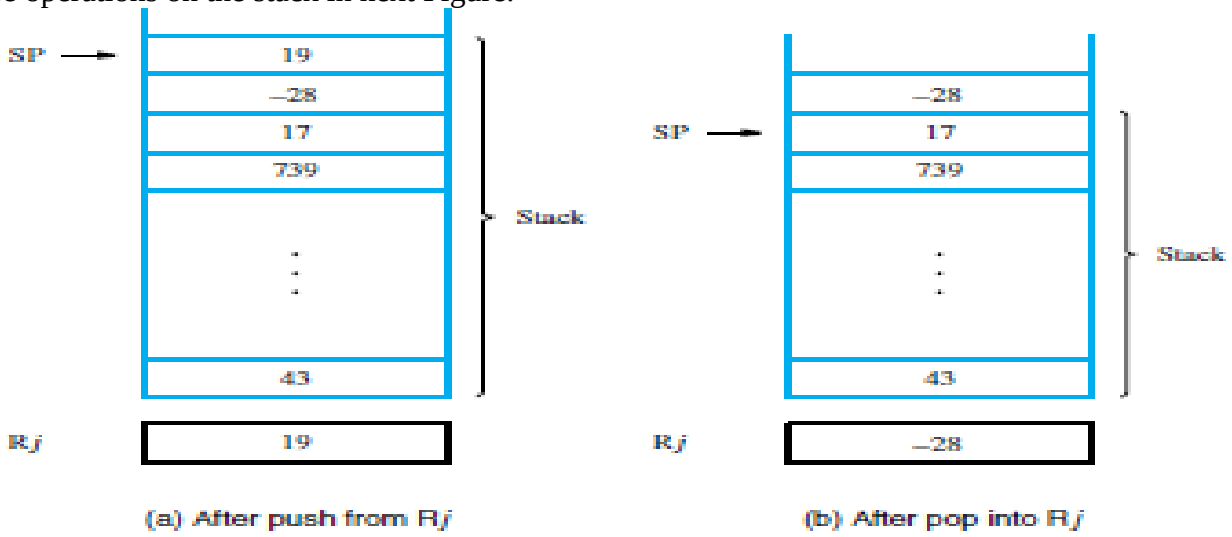

Figure: Effect of stack operation on the stack in previous figure