BASIC STRUCTURE OF COMPUTERS

Explain the Functions of Processor Register with a block diagram? (8M)

OR

Explain the functions of following processor registers w.r.t a computer? (12M)

(i) MAR ii) MDR iii) IR iv) PC V) ALU

The processor contains number of registers used for several different purposes.

- ✓ The instruction register holds the instruction that is currently being executed. Its output is available to the control units which generates the timing signals that control the various processing elements involved in executing the instruction.
- ✓ The program counter is another specialized register. It keeps track of execution of program. It contains the memory address of next instruction to be fetched and executed. During the execution of the instruction to contents of PC are updated to correspond to the address of the next instruction to be executed.
- ✓ Finally two register facilitate communication with memory. These are the memory address registers and memory data register. The MAR holds the address of the location to be accessed. The MDR contains the data to be written into or read out of the addressed location.

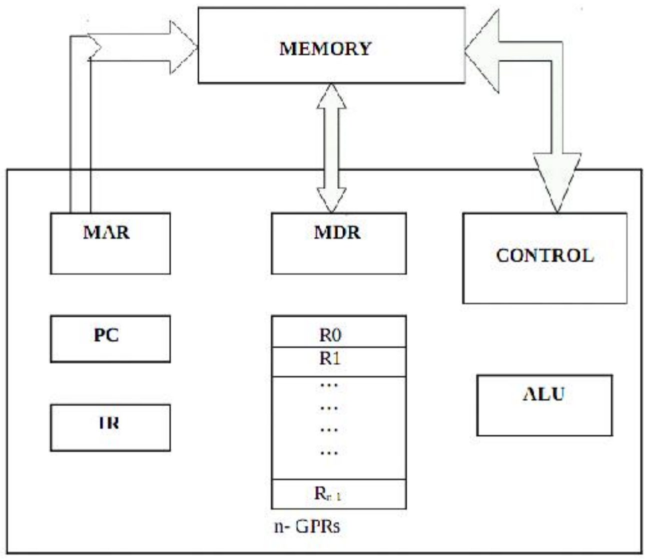


Fig b : Connections between the processor and the memory

Example:

If the instruction involves an operation to be performed by the ALU, the operands need to be fetched either from the memory or from the general purpose register. If the operand is available in the memory it has to be fetched by sending its address to the MAR and initiating a read cycle. When the operand has been read from the memory into the MDR, it is transferred from the MDR to the ALU. Then ALU performs the desired operation. If the result has to be stored in memory, then it is sent to MDR. The address of the location where the result is to be stored is sent the MAR and a write cycle is initiated. PC must be incremented to point to next instruction.

What is a Bus? Explain single bus and multiple Bus structures used to interconnect functional units in a computer system.

To form an operational system the different parts of computer must be connected in some organized way. When a word of data is transferred between units, all its bits are transferred in parallel that is the bits are transferred simultaneously over many wires or lines, one bit per line.

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

The simplest way to interconnect functional units is to use a single bus. All units are connected to this bus. Because the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time. The main virtue of the single-bus structure is its low cost and its flexibility for attaching peripheral devices.

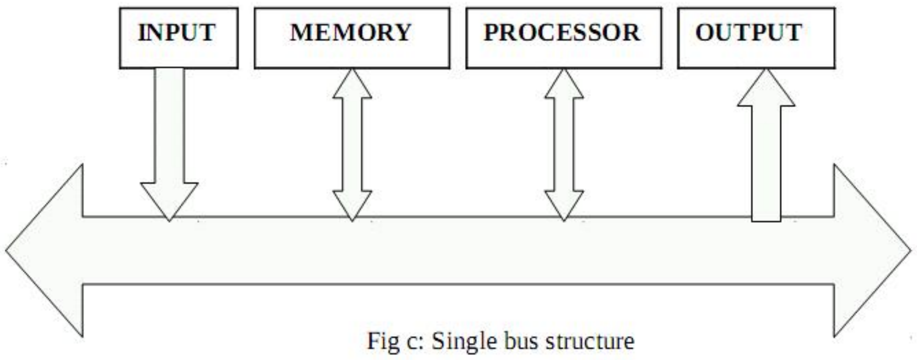


Fig c: Single bus structure

To achieve more concurrency in operations by allowing two or more transfers to be carried out at the same time we use multiple-bus structure. This leads to better performance but at an increased cost.

Some electromechanical devices, such as keyboards and printers, are relatively slow. Others like magnetic or optical disks are considerably faster. Because all these devices must communicate with each other over a bus, an efficient transfer mechanism that can be used to smooth out the difference in timing among processors, memories, and external devices is necessary.

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

How to prevent high speed processor from being locked to a slow input/output device during data transfer?

Some electromechanical devices, such as keyboards and printers, are relatively slow. Others like magnetic or optical disks are considerably faster. Because all these devices must communicate with each other over a bus, an efficient transfer mechanism that can be used to smooth out the difference in timing among processors, memories, and external devices is necessary.

A common approach is to include buffer registers with the devices to hold the information during transfers. To illustrate this technique, consider the transfer of an encoded character from a processor to a character printer .The processor sends the character over the bus to the printer buffer. Since the buffer is an electronic register, this transfer requires relatively little time. Once the buffer is loaded, the printer can start printing without further intervention by the processor. The bus and the processor are no longer needed and can be released for other activity. Thus buffer registers smooth out timing differences among processors, memories, and I/O devices. This allows the processor to switch rapidly from one device to another interweaving its processing activity with data involving several I/O devices.

What are the factors used to judge the performance of a computer? Explain any 3 of them.

Following are the factors used to judge the performance of a computer:

- Speed with which the programs are executed
- Performance of the compiler
- Total time to execute the program
- Processor clock
- Clock rate

The most important measure of the performance of a computer is how quickly it can execute programs. The speed with which a computer executes programs is affected by the design of its hardware and its machine language instructions. Because programs are usually written in a high-level language, performance is also affected by the compiler that translates programs into machine language. The total time required to execute the program is called elapsed time. This 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 etc.

1) PROCESSOR CLOCK Processor circuits are controlled by a timing signal called a clock. The clock defines regular time intervals, called clock cycles. To execute a machine instruction the processor divides the action to be performed into a sequence of basic steps, such that each step can be completed in one clock cycle. The length P of one clock cycle is an important parameter that affects processors performance. Its inverse is the clock rate, R=1/P, which is measured in cycles per second.

2) TOTAL ELAPSED TIME: Let T is the processor time required to execute a program that has been prepared in some high-level language. The compiler generates a machine language object program that corresponds to the source program. Assume that complete execution of the program requires the execution of N machine language instructions. The number N is the actual number of instruction executions and is not necessarily equal to the number of machine instructions in the object program. Suppose that the average number of basic steps needed to execute one machine instruction is S, where each basic step is completed in one clock cycle. If the clock rate is R cycles per second, the program execution time T is given by

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

This is often referred to as the basic performance equation. To achieve high performance, the computer designer must seek ways to reduce the value of T.

3) CLOCK RATE: Clock rate can be improved using 2 ways:
First, by improving IC technology to make the logic circuit faster and reduces the time needed to complete a basic step. This allows the clock period P to reduced and clock rate R to be increased.
Second, by reducing the amount of processing done in basic step reduces the clock period P.

What are the measures to improve the Performance of the computer?

- The most important measure of the computer is how quickly it can execute programs.
- The speed at which a computer executes a program is affected by the design of its hardware & machine language instructions.
- For best performance, it is necessary to design the compiler, the machine instruction set, and the hardware in a co-ordinated way.
- A program will be executed faster if the movement of instructions & data between the main memory & the processor is minimized, which is achieved by using the cache.
- The processor divides the machine instructions into a sequence of basic steps. Each step can be completed in one clock cycle. Therefore if the processor clock speed is high, it improves the speed of execution.
- By using the basic performance equation $T=(N*S)/R$, where T is the processor time required to execute the program, N is the actual number of instructions, S is the steps required to execute one machine instruction & R is clock rate in cycles per sec., we can increase the performance by reducing S which is done by having smaller number of basic steps & N by compiling the program into fewer machine instructions & increasing R by using a high frequency clock.
- By improving the IC technology, the clock rate can be increased.
- A substantial improvement in performance can be achieved by overlapping the execution of successive instructions by using PIPELINING.

Explain SPEC rating and its significance. (05 m)
System Performance Evaluation Corporation (SPEC) selects and publishes representative application programs for different application domains, together with test results for many commercially available computers. The program for selected range is compiled for the computer under test and the running time on a real computer is measured. The same program is also compiled and run on one computer selected as a reference.
For SPEC95, the reference is the SUN SPARC station 10/40. For SPEC2000, the reference computer is an UltraSPARC10 workstation with a 300-MHz UltraSPARC-IIi processor. The SPEC rating is computed as follows:

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

SPEC rating of 50 means that the computer under test is 50 times as fast as the UltraSPARC10 for this particular benchmark. Let $SPEC_i$ be the rating for program I in the suite. The overall SPEC rating for the computer is given by

$$SPEC\ rating = \left(\prod_{i=1}^{n} SPEC_i\right)^{\frac{1}{n}}$$

Where ‘n’ is the number of programs in the suite.
SIGNIFICANCE: Since the actual execution time is measured, hence the SPEC rating is a measure of the combined effect of all factors affecting performance, including the compiler, the operating system, the processor, and the memory of the computer being tested.

Write the basic performance equation? Explain the role of parameters on the performance of the computer. (5M)
Let 'T' be the processor time required to execute a program that has been prepared in some high-level language. The compiler generates a machine language object program that corresponds to the source program. The number N is the actual number of instruction execution and is not necessarily equal to the number of machine instructions in the object program. If the clock rate is R cycles per second, the program execution time is given by

$$T = \left(\frac{N*S}{R}\right)$$

This is often referred to as the basic performance equation.
<u>To achieve high performance,</u>

- Reduce the value of T which means reducing N&S and increasing R. N is reduced if the source program is compiled into fewer machine instructions. S is reduced if the instructions have a smaller number of basic steps to perform or if the execution of the instructions is overlapped. R can be increased by using a high frequency clock. N, S and 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.
- There are two possibilities for increasing the clock rate ‘R’. 1. Improving the IC technology makes logical circuit faster, which reduces the time needed to complete a basic step. This allows the clock period P, to be reduced and the clock rate R to be increased. 2. Reducing the amount of processing done in one basic step also makes it possible to reduce the clock period P.

Define the following:

a) **Processor clock** – Processor circuits are controlled by a timing signal called a clock. The clock defines regular time intervals, called clock cycles. To execute a machine instruction, the processor divides rhea action to be performed into 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 processor performance. Its inverse the clock rate, R=1/P which is measured in cycles per second.
b) **Clock rate** – the clock rate is the inverse of processor clock, P. R=1/P
c) **Elapsed time** – The total time required executing the program.
d) **Processor time** – The amount of the time taken by the processor to execute a program excluding I/O time needed by a program.

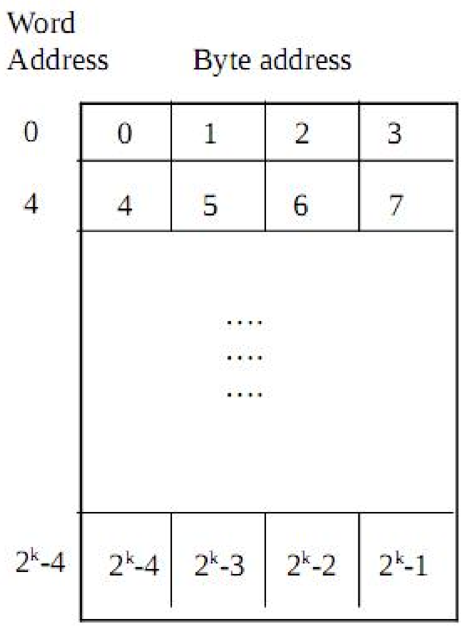
What is byte addressability? Explain 2ways that byte addresses are assigned across words. (Or) Explain a) Big endian assignment b) Little endian assignment

A byte is 8 bits, but the word length ranges from 16 to 64 bits. Byte addressable memory is the one where, successive addresses refer to successive memory locations in the memory. Byte locations have addresses 0, 1, 2… Thus, if the word length of the machine is 32 bytes, successive words are located at addresses 0, 4, 8 ..., with each word consisting of 4 byte.

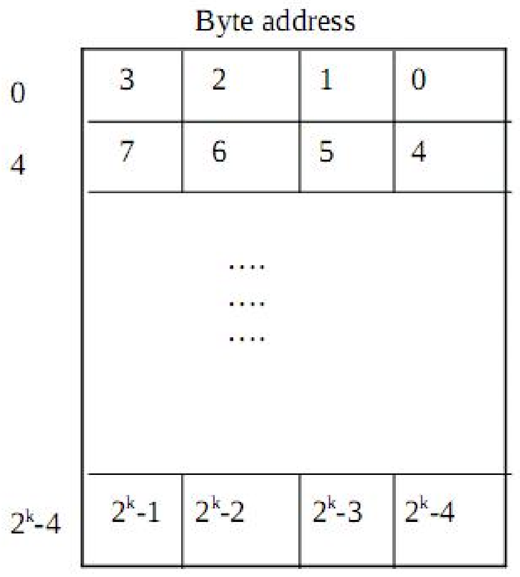
The two ways that byte addresses can be assigned across words are

i) **Big Endian assignment** - The name Big endian is used when lower byte addresses are used for the most significant bytes (the leftmost bytes) of the word.

ii) **Little Endian assignment** – The name Little endian is used for the opposite ordering, where the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word.



(a) Big-endian assignment



(b) Little-endian assignment

In both the cases byte addresses 0, 4, 8 . . . are taken as the addresses of successive words in the memory and are the addresses used when specifying memory read and write operations for words. In addition to this, it is also necessary to specify the labelling of bits within a byte or word. The same ordering is used for labelling bits within a byte, that is, b_7, b_6, …., b_0,from left to right.

What is word alignment of a m/c? What are the consecutive addresses of aligned words for 16, 32 and 64 bit word length m/c? Give consecutive addresses for each case. (5M)
If words in memory begin at address that is a multiple of the number of bytes in a word it is called as word alignment of a m/c.
The number of bytes in a word is a power of 2. Hence 16 bit word consists of 2 byte locations within it. 32 bit word consists of 4 byte locations and 64 bit word consists of 8 bytes in each location.

The consecutive addresses for:

i) 16 bit word length - (2 bytes), aligned words begin at byte addresses 0, 2, 4 ...
ii) 32-bit word length- (4 bytes), word boundaries occur at addresses 0, 4, 8 ...
iii) For a word length of 64(8 bytes), aligned words at bytes addresses 0,8,16.....

Explain 2 operations involving the memory. (4M)
The 2 basic operations involving memory are Load and Store.

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

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.

Show how the operation C=A+B can be implemented in a single accumulator computer by

i) 3-address instruction ii) 2-address instruction iii) 1-address instruction . (10M)

Using 1-address instruction: Here m/c instructions specify only one memory operand. When a second operand is needed it is understood implicitly to in a unique location. A processor register Accumulator is usually used for this purpose.

Thus C=A+B can be implemented as below:

Load A	(loads A into accumulator)
Add B	(adds B with accumulator and the result is available in Accumulator)
Store C	(from the accumulator store the result in to C)

Using 2-address instruction:

2-address instruction has the following form:

Operation Source, Destination

Thus C=A+B can be done like this:

Move A, R1	(move operand A to register R1)
Add B, R1	(add operand B to R1 and store the result in same register)
Move R1, C	(copy the result into the memory location C)

Using 3-address instruction:

A three address instruction has the form

Operation Source1, Source2, Destination

Thus C=A+B can be performed as below:

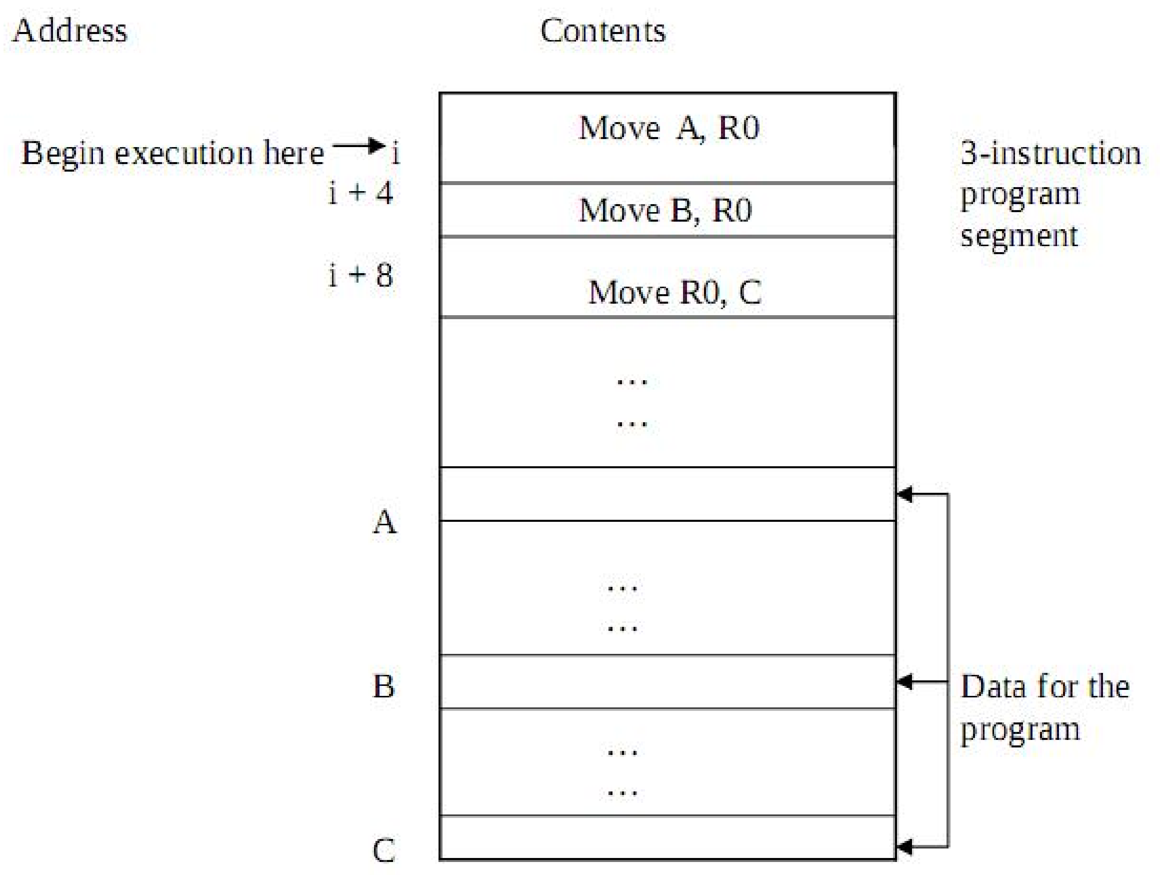
Add A, B, C (add A and B store the result into C)

What is straight line sequencing? Explain with an example?

Assume that the computer allows one memory operand per instruction and has a number of processor registers. We assume that the word length is 32 bits and the memory is byte, addressable. The thee instruction of program are in successive word locations, sorting at location i. Since each instruction is 4 bytes long , the second and third instruction 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 controls 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 the next 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, call 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 involves fetching operands from the memory or from processor or from the processor registers, performing an arithmetic or logic operation, and storing the result in the destination location.



What are conditional flags? Explain 4 commonly used flags (5M)

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

This is accomplished by recording the required information in individual bits, often called the conditional 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 operations performed. Four commonly used flags are

1. N(negative):set to 1 if the result is negative; otherwise cleared to 0.
2. Z(zero) :set to 1 if the result is 0;otherwise cleared to 0.
3. V(overflow) :set to 1 if arithmetic overflow occurs; otherwise cleared to 0.
4. c(carry) :set to 1 if a carry out results from the operation; otherwise cleared to 0.

The N and the Z flags indicate whether the result of an arithmetic or logic operation is zero or negative. Then N and Z flags may also be affected by instructions that transfer data such as Move, Load or Store. This makes it possible for a later conditional branch instruction to cause a branch based on the sign and value of the operand that was moved.

The V flag indicates whether the overflow has taken place. Overflow occurs when the result of arithmetic operation is outside the range of values that can be represented by the no of bits available for the operands. The processor sets the V flag to allow the programmer to test whether overflow has occurred and branch to an appropriate routine that corrects the problem. Instructions
such as BranchIfOverflow are provided for this purpose.

The C flag is set to 1 if a carry occurs from the most significant bit position during an arithmetic operation.This flag makes it possible to perform arithmetic operations on operands that are longer than the word length of the processor.

The instruction Branch>0 is an example of the branch instruction that tests one or more conditional flags.It causes a branch if the value tested is neither negative nor equal to zero. That is, branch is taken if neither N nor Z is 1.Many other conditional branch instructions are provided to enable a variety of conditions to be tested. The conditions are given as logic expressions involving the condition code flags.

MACHINE INSTRUCTIONS AND PROGRAMS

Define an addressing mode. Explain the following addressing modes with example: Indirect, Index, Relative and Auto increment. (5M)

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

<u>1. Relative addressing</u>:- X(PC) can be used to address location that is X bytes away from the location presently pointed to by the program counter. The effective address is determined by the index mode using the program counter in place of general purpose register Ri. This mode can be used to access data operands. But, it’s most common use is to specify the target address in branch instructions. An instruction such as

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.

2. Auto increment mode:-The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically to point to the next item in a list,(Ri)+

```
            Move        N, R1
            Move        #NUM1, R2
            Clear       R0
LOOP        Add         (R2)+,R0
            Decrement   R1
            Branch>0    LOOP
            Move        R0, SUM
```

The auto increment mode is used in the program.

3. Index mode:-The effective address of the operand is generated by adding the constant value to the contents of the register. The register may be either a special register provided for this purpose, or may be any one of a set of a general purpose registers in the processor. In either case, it is referred as index register. Index mode symbolically represented as X (Ri) where X denotes the constant value contained in the instruction and Ri is the name of the register involved. The effective address of the operand is given by EA =X+[Ri].

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

Use of indirect addressing to add n numbers

```
                  Move        N,R1
                  Move        #NUM, R2
                  Clear       R0

LOOP              Add         (R2), R0
                  Add         #4, R2
                  Decrement   R1
                  Branch>0    LOOP
                  Move        R0, NUM
```

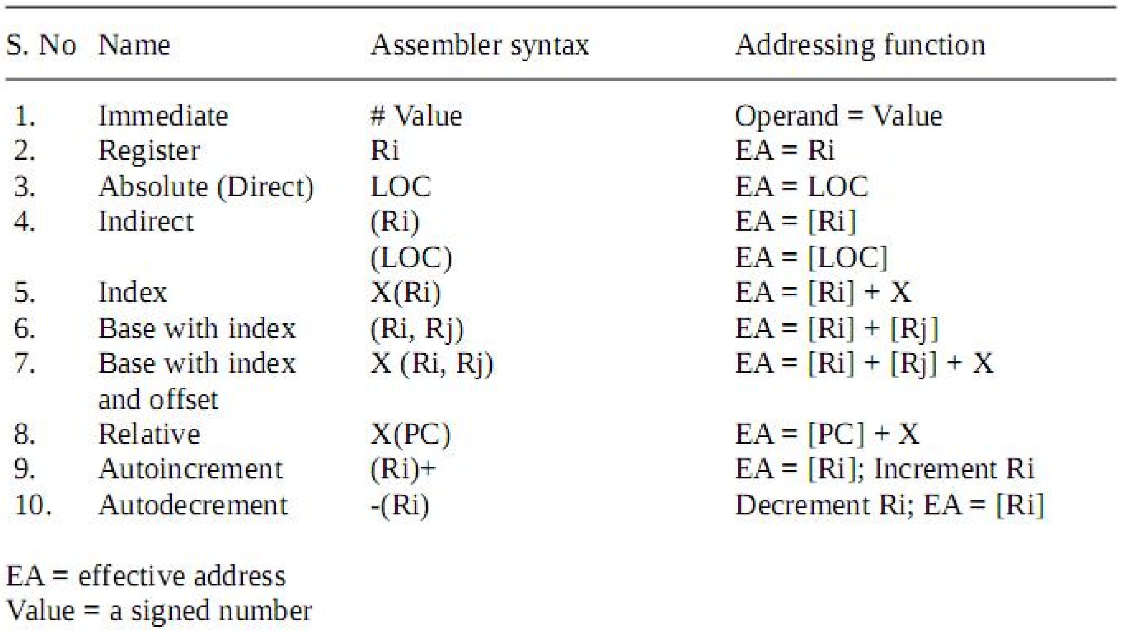
Write a program that can evaluate the expression (A*B) + (C*D) in a single accumulator processor. Assume that the processor has load, store, multiply and add instruction and all values fit in the accumulator. (5M)

```
LOAD      A
MUL  B
STORE     RESULT
LOAD      C
MUL D
ADD    RESULT
STORE     RESULT
```

Explain all the generic addressing modes with assembler syntax. -12M.

Following table shows all the Generic addressing modes

Table 2.1 Generic addressing modes



S. No	Name	Assembler syntax	Addressing function
1.	Immediate	# Value	Operand = Value
2.	Register	Ri	EA = Ri
3.	Absolute (Direct)	LOC	EA = LOC
4.	Indirect	(Ri)	EA = [Ri]
		(LOC)	EA = [LOC]
5.	Index	X(Ri)	EA = [Ri] + X
6.	Base with index	(Ri, Rj)	EA = [Ri] + [Rj]
7.	Base with index and offset	X (Ri, Rj)	EA = [Ri] + [Rj] + X
8.	Relative	X(PC)	EA = [PC] + X
9.	Autoincrement	(Ri)+	EA = [Ri]; Increment Ri
10.	Autodecrement	-(Ri)	Decrement Ri; EA = [Ri]

EA = effective address
Value = a signed number

Explanations:

- ✓ Immediate mode-The operand is given explicitly in the instruction.
  For example, the instruction

  Move 200immediate, R0

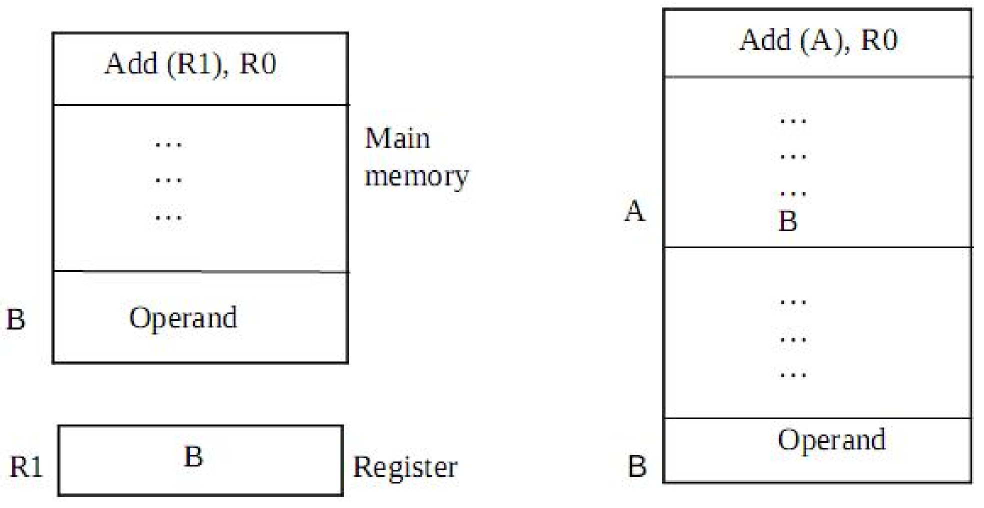
  Places the value 200 in register R0.The sharp sign (#) is also used to indicate the value is used as an immediate operand.

  Move #200, R0
- ✓ Register mode-The operand is the contents of the processor register; the name (address) of the register is given in the instruction.
  Example : Add R1, R2
- ✓ Absolute (Direct) Mode-The operand is in memory location; the Address Of this location is given explicitly in the instruction.(In some assembly languages the mode is called Direct) The instruction

  Move LOC, R2

  Processor registers are used as temporary storage Locations where the data in a register are accessed using the register mode.
- ✓ Indirect mode-The effective address of the operand is the contents of a register or memory location whose address appears in the instruction.

Fig (a) Through a general-purpose register (b) Through a memory location



✓ Index mode-The effective address of the operand is generated by adding a constant value to the content of a register. The register used may be general-purpose register or special register provided for this purpose.

The symbolic representation

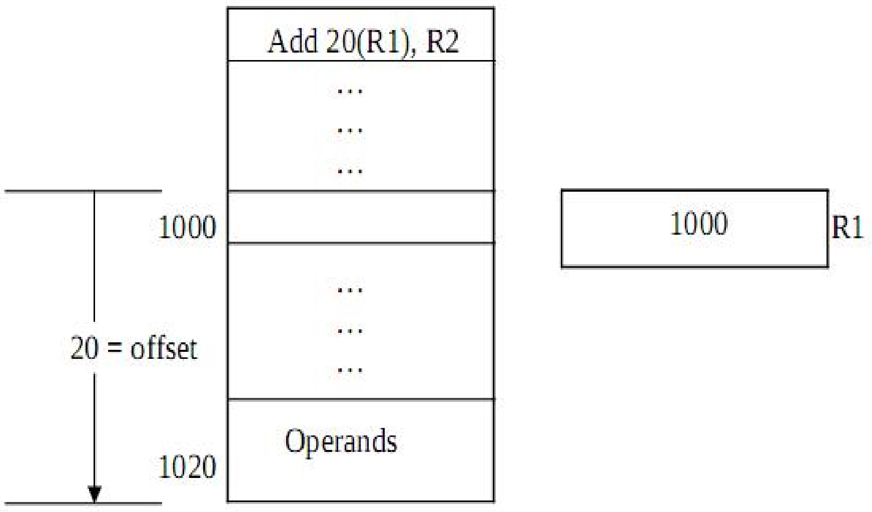
X(Ri)

Where X is a constant value in instruction and Ri is the name of the register.

The effective address of the operand is given by

EA=X + [Ri]

Fig (a) Offset is given as a constant



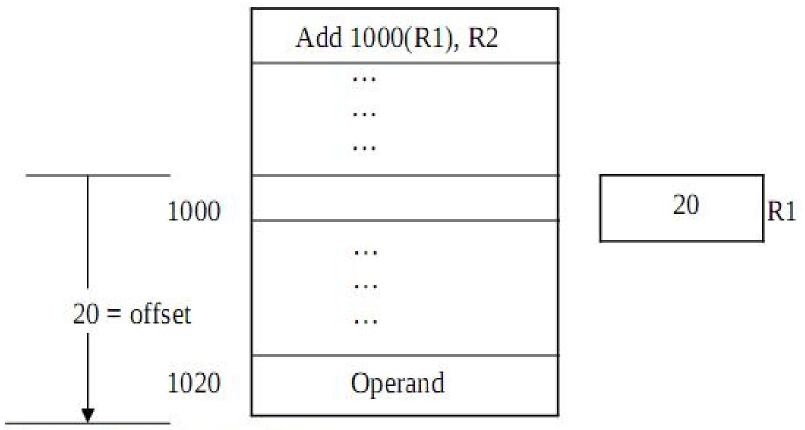


Fig (b) Offset is in the index register

A second register may be used to contain the offset X, in which case we can write the index mode as

(Ri,Rj)

The effective address is the sum of the contents of Ri and Rj. The second register is generally called the base register, and the addressing mode is called **Base with index mode**.

Another version of index mode uses two register plus a constant.

X(Ri,Rj)

The effective address is the sum of the constant X and the contents of Ri and Rj. The addressing mode is called **Base with index and offset mode**.

- ✓ Relative mode-The effective address is determined by the index mode using the program counter in place of general-purpose register Ri. This mode can be used to access data operands. But, it's most common use is to specify the target address in branch instructions. An instruction such as

  Branch>0 LOOP

  causes program execution to go to the branch target location identified by the name LOOP if the branch condition is satisfied.
- ✓ Auto increment mode-The effective address of the operand is the contents of a register Specified in the instruction. After accessing the operand, the contents of the register are automatically incremented to point to the next item in a list. The auto increment mode is written as

  (Ri)+
- ✓ Auto decrement mode-The contents of a register specified in the instruction are first Automatically decremented and then used as a effective address of the operand. The auto decrement mode is written as

  - (Ri)

  In this mode, operands are accessed in descending address order.

4) Write a program to add N numbers using indirect addressing mode. (5 marks)

```
                 Move            N, R1
                 Move            #NUM, R2
                 Clear           R0
     LOOP        Add             (R2), R0
                 Add             #4, R2
                 Decrement       R1
                 Branch > 0      LOOP
                 Move            R0, SUM
```

The difference between index mode, base with index mode, and base with index and offset Mode ?

INDEX MODE
The effective address of the operand is generated by adding a constant value to the contents of a register. The register may be either a special register provided for this purpose, or may be any one of a set of general purpose registers in the processor. In either case, it is referred to as index register. The index mode is symbolically represented as: x (Ri) where x denotes the constant value contained in the instruction and Ri is the name of the register involved. The effective address of the operand is given by

$$EA=x + [Ri]$$

The contents of the index register are not changed in the process of generating the effective address. In an assembly language program, the constant x may be given either as an explicit number or as a symbolic name representing a numerical value.

BASE WITH INDEX MODE
The effective address is the sum of the contents of registers Ri and Rj.the second register is usually called the base register. The second register may be used to contain the offset x, in which case it is written the index mode as(Ri,Rj). This form of indexed addressing provides more flexibility in accessing operands, because both components of the effective address can be changed.

BASE WITH INDEX AND OFFSET
Another version of the index mode uses two registers plus a constant which can be denoted as x(Ri,Rj) . In this case, the effective address is the sum of the constant x and the contents of the registers Ri and Rj. The added flexibility is useful in accessing multiple components inside each item in a record, where the beginning of an item is specified by the(Ri,Rj) part of the addressing mode.
In other words, this mode implements a three-dimensional array.

Write a program to add N numbers using auto increment mode .

```
MOVE        N, R1               ; move the content of N to R1
MOVE       #NUM1, R2            ; immediate address mode move NUM1 to R2
Clear       R0                  ; clear   R0   register
LOOP   ADD   (R2) +, R0         ;(R2)+   automatically points to the next item in the list
Decrement   R1                  ; R1 is decremented
Branch>0    LOOP                ; Branch>0 till n not equals to zero, LOOP is repeated.
MOVE         R0, SUM            ; move the contents of R0 to SUM
```

Explain the EQU, ORIGIN, DATAWORD, RESERVE assembler directives with an example for each.

In addition to providing a mechanism for representing instructions in a program, the assembly language allows the programmer to specify other information needed to translate the source program into the object program. Suppose that the name SUM is used to represent the value 200. This fact may be conveyed to the assembler program through a statement such as

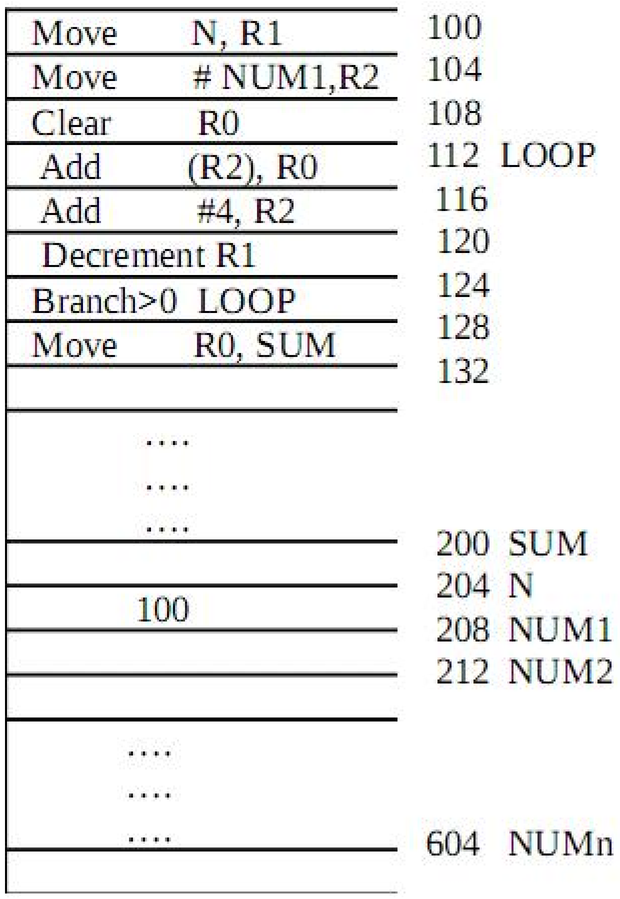
SUM EQU 200

This statement does not denote an instruction that will be executed when the object program is run, in fact; it will not even appear in the object program. It simply informs the assembler that the name SUM should be replaced by the value 200 whenever it appears in the program. Such statements, called assembler directives (or commands), are used by the assembler while it translates a source program into an object program.

An assembler directive is a message to the assembler that tells the assembler something it needs to know in order to carry out the assembly process. It will not appear in the object program.

The assembler should know:

1. How to interpret the names.
2. Where to place the instructions in the memory.
3. Where to place the data operands in the memory.



Memory arrangement for the program to add N numbers

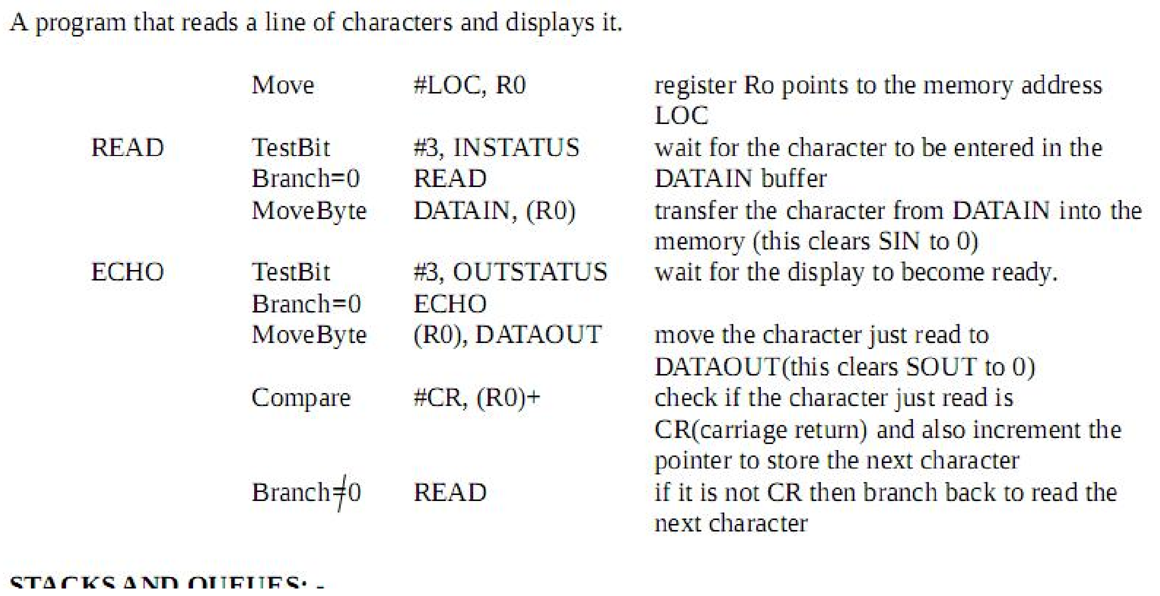
Assembly language representation for the program to add N numbers

	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

EQU informs the assembler about the value of SUM. ORIGIN tells the assembler where in the memory to place the data block that follows. In this case the location specified has the address 204.
The DATAWORD directive is used to inform the assembler that this location is to be loaded with the value 100. It states that the data value 100 is to be placed in the memory word at address 204.
The label is assigned a value equal to the address of that location. The RESERVE directive declares that the memory block of 400 bytes is to be reserved for data and the name NUM1 is to be associated with address 208.The second ORIGIN directive specifies that the instruction of the object program are to be loaded in the memory starting at address 100.

Why loader is needed? Explain
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 in the memory. Executing the loader performs a sequence of input operations needed to transfer the machine language program from disk into a specified place in the memory. The loader must know the length of the program and the address in the memory where it will be stored.

For a simple example of i/o operations involving keyboard and display device, write an assembly language program that reads 1 line from keyboard, stores it in memory buffer and echos it back to the display



A program that reads a line of characters and displays it.

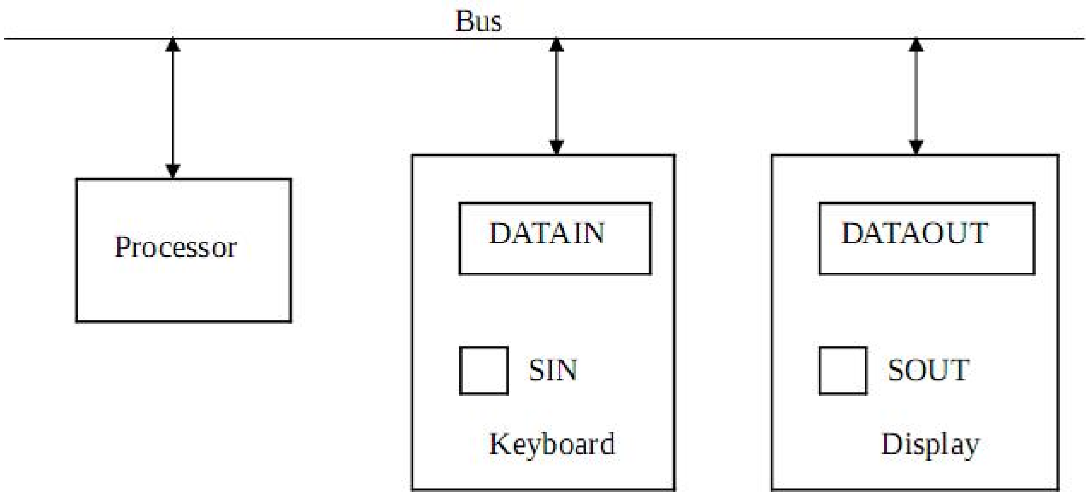
```
            Move        #LOC, R0         register Ro points to the memory address
                                         LOC
READ        TestBit     #3, INSTATUS     wait for the character to be entered in the
            Branch=0    READ             DATAIN buffer
            MoveByte    DATAIN, (R0)     transfer the character from DATAIN into the
                                         memory (this clears SIN to 0)
ECHO        TestBit     #3, OUTSTATUS    wait for the display to become ready.
            Branch=0    ECHO
            MoveByte    (R0), DATAOUT    move the character just read to
                                         DATAOUT(this clears SOUT to 0)
            Compare     #CR, (R0)+       check if the character just read is
                                         CR(carriage return) and also increment the
                                         pointer to store the next character
            Branch≠0    READ             if it is not CR then branch back to read the
                                         next character
```

STACKS AND QUEUES: -

Explain the usage of datain, dataout registers and sin, sout status control flags in i/o devices.

Consider the problem of moving a character code from the keyboard to the processor. Striking key stores the corresponding character code in an 8-bit buffer register associated with the keyboard. This register is DATAIN. To inform the processor that a valid character is in DATAIN, a status control flag, SIN, is set to 1. A program monitors SIN, and when SIN is set to 1, the processor reads the contents of DATAIN. When the character is transferred to the processor, SIN is automatically cleared to 0. If a second character is entered at the keyboard, SIN is again set to 1 and the process repeats.

An analogous process takes place when characters are transferred from the processor to the display buffer register, DATAOUT, and a status control flag, SOUT are used for this transfer. When SOUT equals 1, the display is ready to receive a character .under program control, the processor monitors SOUT and when SOUT is set to 1, the processor transfers a character code to DATAOUT. The transfer of a character to a DATAOUT clears SOUT to 0; when the display device is ready to receive a second character, SOUT is again set to 1.



Write the sequence for READWAIT and WRITEWAIT for pgm controlled I/O.

In order to perform I/O transfers, we need machine instructions that can check the state of status flags and transfer the data between the processor and the I/O device. These instructions are similar in format to those used for moving data between the processor and the memory. For example, the processor can monitor the keyboard status flag SIN and transfer a character from DATAIN to reg. 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 second performs the branch. Although the details vary from computer to computer, 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. An example is

```
 WRITEWAIT          Branch to WRITEWAIT if SOUT=0
                    Output from R1 to DATAOUT
```

Again the Branch operation is normally implemented by two machine instructions. The wait loop is executed repeatedly until the status flag SOUT is set to 1 by display when it is free to receive a character. The output operations transfers a character from R1 to DATAOUT to be displayed, and it cleans SOUT to 0.

13) WRITE THE SEQUENCE OF OPERATIONS NEEDED FOR READWAIT AND WRITEWAIT FOR MEMORY MAPPED I/O.

Many computers use an argument called 'memory mapped i/o' in which some memory address values are used to refer to peripheral device buffer registers, such as DATAIN and DATAOUT. There are two status flags namely SIN and SOUT. It is more common to include SIN and SOUT in device status registers, one for each of the 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
```

The Testbit instruction tests the state of one bit in the destination location, where the bit position to be tested is indicated by the first operand. if the bit tested is equal to 0, then the condition of the branch instruction is true, and a branch is made to the beginning of the wait loop. When the device is ready, i.e, when the bit tested becomes equal to 1, the data are read from the input buffer or written into the output buffer.

WRITE THE ROUTINES FOR SAFEPUSH AND SAFEPOP OPERATIONS & EXPLAIN THE SAME

There is every possibility that while pushing and popping operations are performed the, stack pointer may go out of the memory which is allocated for the stack. In order to avoid such situations SAFEPUSH & SAFEPOP are implemented. The routines for SAFEPUSH & SAFEPOP operations are as shown below:

Routine for safe pop operation

```
SAFEPOP    Compare     #2000, SP      check to see if the stack pointer contains an address
           Branch >0   EMPTYERROR     value greater than 2000. if it does , the stack is
                                      empty. Branch to the routine EMPTYERROR.
           Move        (SP)+, ITEM    otherwise, pop the top of the stack into memory
                                      location ITEM.
```

Routine for safe push operation

```
SAFEPUSH   Compare     #1500, SP      check to see if the stack pointer contains an address
           Branch >0   EMPTYERROR     value equal to or less than 1500. if it does , the
                                      stack is full. Branch to the routine FULLERROR.
           Move        (SP)+, ITEM    otherwise, push the element in memory
                                      location NEWITEM onto the stack.
```

Explain the operation of stack with an example. Give any three differences between stack and queue.

A stack is a list of data elements usually words or bytes with accessing restriction that elements can be added or removed at one end of the list only. This end is called top of the stack and the other end is called bottom. This structure is sometimes referred to as a pushdown stack. Data stored in the memory of the computer can be organized as stack, with successive elements occupying successive memory locations.

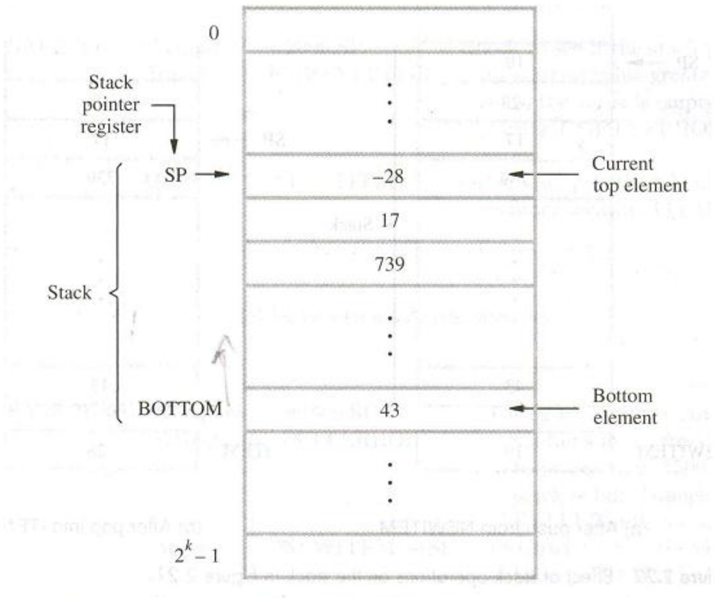


Figure 2.21 A stack of words in the memory.

Above figure shows a stack of word data items in the memory of the computer. It contains numerical values with 43 at bottom and -28 at the top. A processor register is used to keep track of the address of the element of the stack that is at the top at any given time. This register is called stack pointer(SP).

The push operation can be implemented as

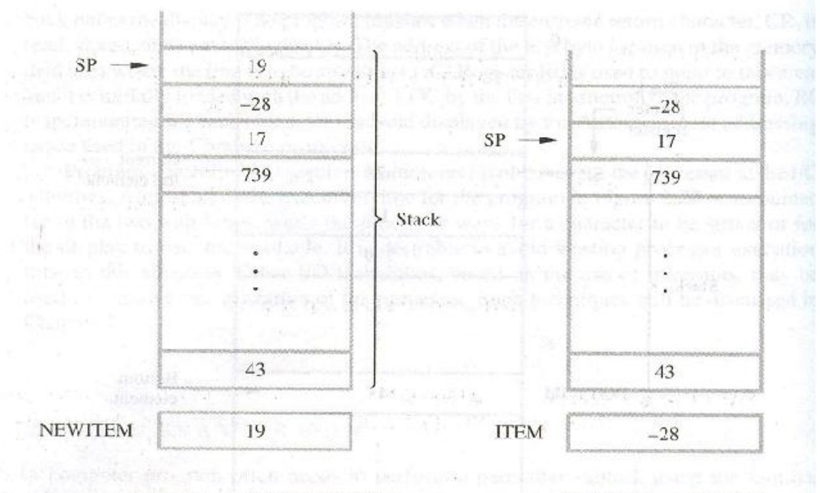
```
Subtract #4,SP
 Move     NEWITEM,(SP)
```

where the subtract instruction subtracts the source operand 4 from the destination operand contained in SP and places the result in SP. These two instructions move the word from location NEWITEM onto the top of the stack, decrementing the stack pointer by 4 before the move.

The pop operation can be implemented as

```
Move (SP),ITEM
 Add  #4,SP
```

These two instructions move the top value from the stack into location item and then increment the stack pointer by 4 so that it points to the new top element.



(a) After push from NEWITEM (b) After pop into ITEM

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

Differences between stack and queue

Stack	**Queue**
One end of stack is fixed while the other end rises and falls as data are pushed and popped	Both ends of a queue move to higher addresses as data are added at the back and removed from the front.
A single pointer is needed to point to the top of the stack at any given time	Two pointers are required to keep track of the two ends of the queue.
Care must be taken to detect when the region assigned to stack is either completely full or completely empty.	One way to limit a queue to a fixed region in the memory is to use a circular buffer

.

what is a subroutine linkage? Explain with an example subroutine linkage using linkage registers.

It is possible to include the block of instructions (subroutine) at the required place in the program. So as to save space one copy of the subroutine is placed in the memory, and any program which requires this subroutine will branch to its starting memory location. This is known as *calling* of subroutine which is done by a branch operation called as call instruction. After execution of the subroutine the control should return back to the program which called it and this is done by executing a Return statement. Since the subroutine may be called from diff. places in the calling program, provision must be made to return to the appropriate location. While the call instruction is being executed, the PC will be updated with the starting address of the subroutine and it will no longer contain the location of the next instruction which has to be executed once the return statement is encountered. Hence the contents of the PC have to be saved before it is updated and enable correct returning of execution to the calling program. The return address is hence saved in a register called as ***link register***. When the task of the subroutine is completed, the Return instruction returns to the calling program by indirectly branching through the link register.

The way which the computer makes it possible to call and return from subroutines is referred to as its *subroutine linkage* method.

Hence

The call instruction performs following operations-

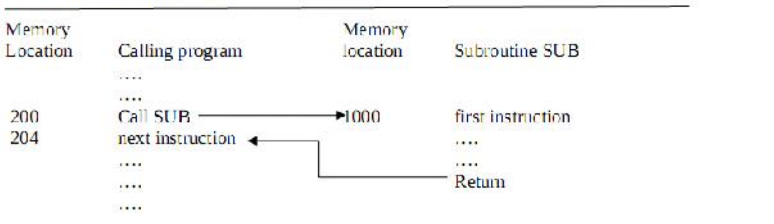
- Store contents of PC in link register.
- Branch to the staring address of the subroutine.

The Return instruction performs following operations

- Branch to the address contained in the link register.

Following figure illustrates the same.

Fig a illustrates this procedure



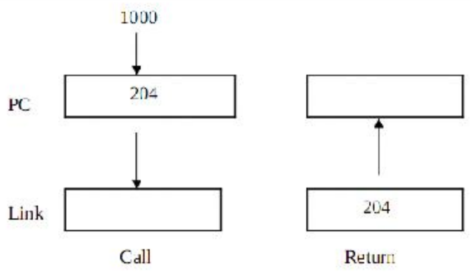


Fig b Subroutine linkage using a link register

What are different operations performed during call instruction and return instruction?

In a program it is necessary to perform a particular subtask more than once on different data values. Such a subtask is called as SUBROUTINE.

Only one copy of the instructions that constitute the subroutine is placed in memory to save space, and any program that requires the use of subroutine simply branches to its starting location. This is called as calling the subroutine and the instruction that performs this is called CALL INSTRUCTION.

The operations are: a) store the contents of the PC in the link register
b) Branch to the target address specified by the instruction

After the program is executed, the subroutine is said to return to the program that called it by executing a RETURN INSTRUCTION.

The operations are: a) branch to the address contained in the link register

Consider the following possibilities for saving the return address of a subroutine

i. **in a processor register**
ii. **in a memory location**
iii. **on a stack**

Which of the following possibilities support the subroutine nesting and which supports subroutine recursion? Why?

A common programming practice, called subroutine nesting, is to have one subroutine call another. In this case the return address of the second call is also stored in the link register destroying its previous contents. Hence it is essential to save the contents of the link register in some other location before calling another subroutine. Otherwise the return address of the first subroutine will be lost.

Subroutine nesting can be carried out to any depth. Eventually the last subroutine called completes its computations and returns to the subroutine that called it. The return address needed for this first return is the last one generated in the nested call sequence. That is, return addresses are generated and used in a last -in –first- out order. This suggests that the return address associated with subroutine calls should be pushed onto a stack. Many processors do this automatically as one of the operations performed by the Call instruction. A particular register is designated as the stack pointer .SP, to be used in this operation .The stack pointer points to the stack called the processor stack and loads the subroutine address into the PC. The return instruction pops the return address from the processor stack into the PC.

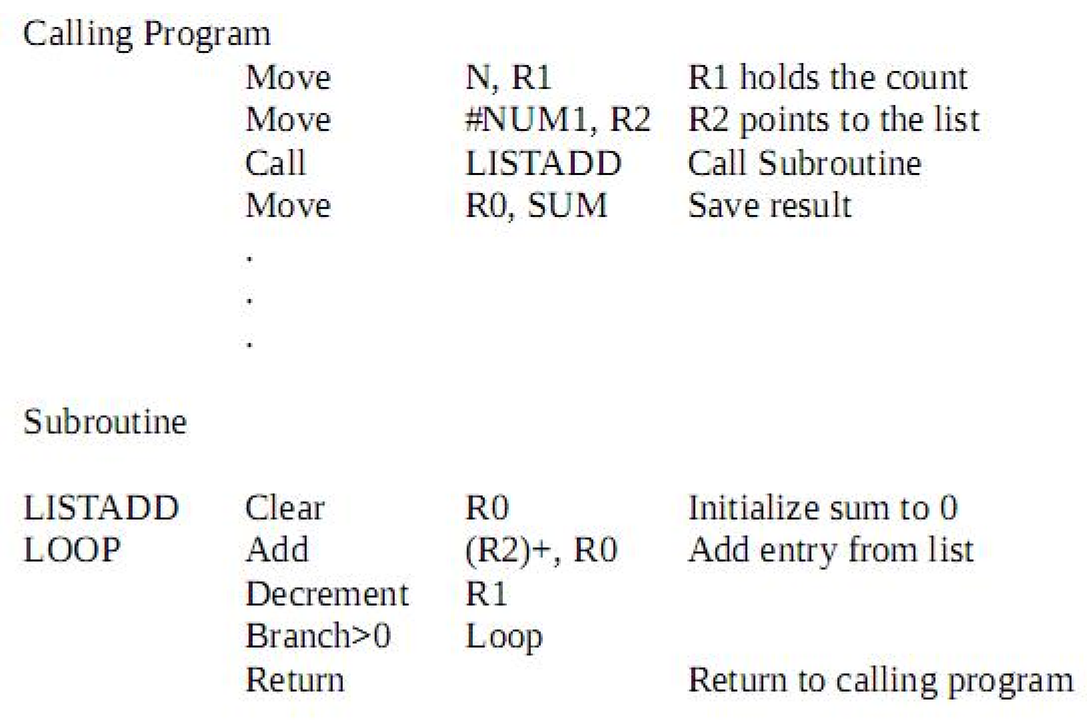
Thus

a) Processor register supports neither nesting nor recursion.
b) Memory location supports nesting, because different Call instructions will save the return address at different memory locations. But Recursion is not supported.
c) Stack supports both nesting and recursion.

Explain how the parameters are passed to a subroutine? WAP to add a list of N-numbers stored in a memory, which calls a subroutine namely LISTADD. Trace the program with suitable example.

When calling a subroutine, a program must provide to the subroutine the parameters, that is, the Operands or the addresses to be in the computation. Later the subroutine returns the other parameters, in this case, the result of the computation. This exchange of information between a calling program and a subroutine is referred to as parameter passing. Parameter passing may be accomplished in several ways. The parameters may be placed in registers or in memory locations, where they can be accessed by the subroutine. Alternatively, the parameters may be placed on the processor stack used for saving the return address.

a) Following program multiplies a list N-numbers stored in a memory. Uses processor to pass the parameters.

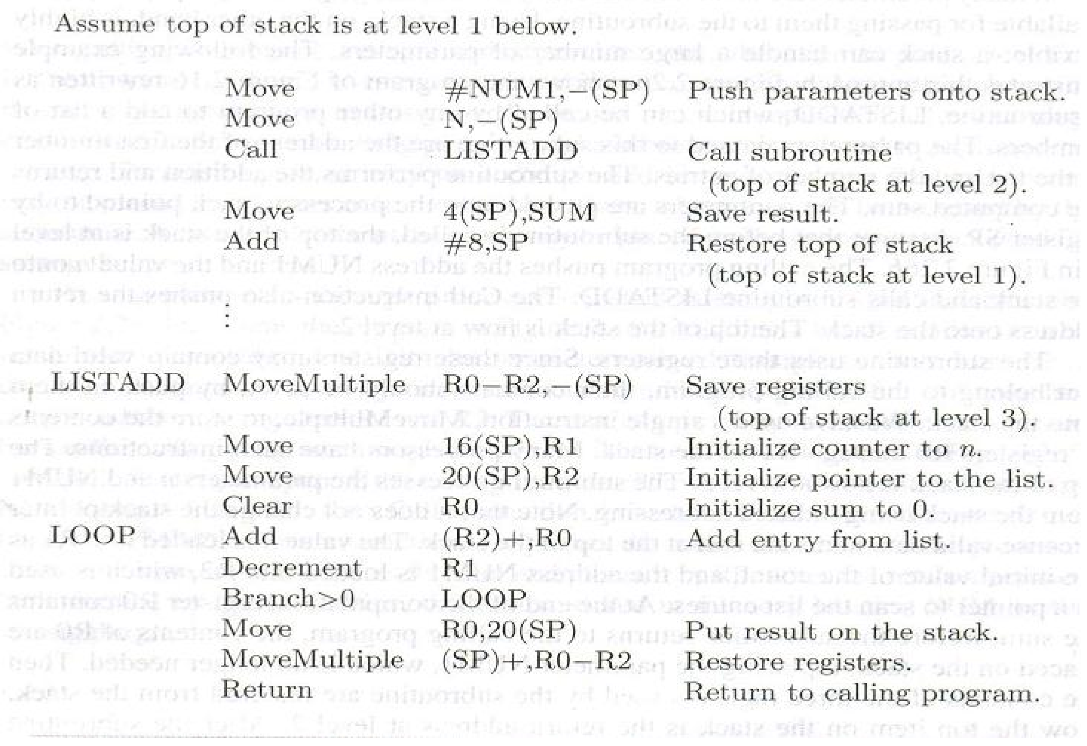


```
Calling Program
            Move        N, R1        R1 holds the count
            Move        #NUM1, R2    R2 points to the list
            Call        LISTADD      Call Subroutine
            Move        R0, SUM      Save result
            .
            .
            .

Subroutine

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

The size of the list n, contained in memory location N and the address NUM1 of the first number are passed through the registers R1 and R2. The sum computed by the subroutine is passed back to the calling program through register R0. After the return operation is performed by the subroutine, the sum is stored in the memory location SUM by the calling program.

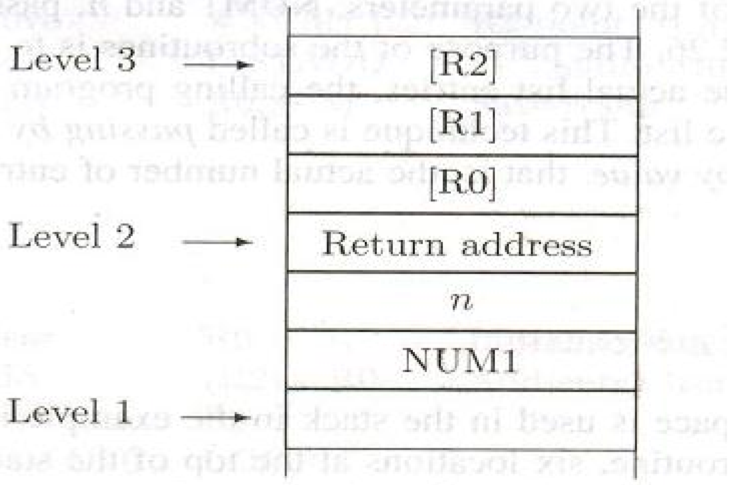
If many parameters are involved, there may not be enough general-purpose registers available for passing them to the subroutine. Using a stack, on the other hand, is highly flexible; a stack can handle a large number of parameters. The parameters passed to this subroutine are the address of the first number in the list and the number of entries. The subroutine performs the addition and returns the computed sum. The parameters are pushed onto the processor stack pointed to by register SP.

b) Following program adds a list N-numbers stored in a memory. Uses stack to pass the parameters.



```
Assume top of stack is at level 1 below.

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

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

Following figure shows the top of stack at various times:



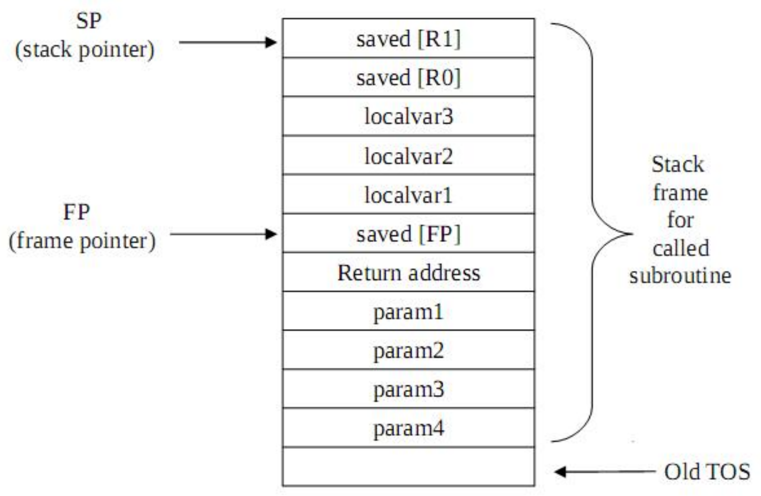
What is stack frame? Illustrate the use of stack frame in the mechanism for implementing subroutine?

The locations constitute a private works space for the subroutine, created at the time the subroutine is entered and freed up when then subroutine return control to the calling program. Such space is called a stack frame.

In addition to the stack pointer (SP), it is useful to have another pointer register, called the frame pointer (FP), for convenient access to the parameters passed to the subroutine and to the local memory variables used by the subroutine. These local variables are only used within the subroutine so it is appropriate to allocate space for them in the stack frame associated with the subroutine.

We assume that 4 parameters are passed to the subroutine; three local variables are used in the subroutine and registers needs to be saved because they will also be used within the subroutine. After these instructions are executed both SP and FP point to the saved FP contents. Space for the three local variables is now allocated on the stack by executing the instruction .finally the contents of the processor registers are saved by pushing them on to the stack at this point the stack frame has been set up. The subroutine now executes its task. When the task is completed the subroutine pops the saved values into those registers remove the local variables from the stack frame by executing the instructions. The calling program is responsible for removing the parameters from the stack frame, some of which may be results passed back by the subroutine. This stack pointer now points to the old of the stack and we are back to where we started.

Fig a A subroutine stack frame example.



Explain the following instructions with example

There are many instructions that require the bits of an operand to be shifted right or left some specified right or left some specified number of bit positions. The details of how the shifts are performed depend on whether the operand is a signed number or some more general binary-coded information. For general operands, we use logical shift.

1. <u>Logical shift</u>

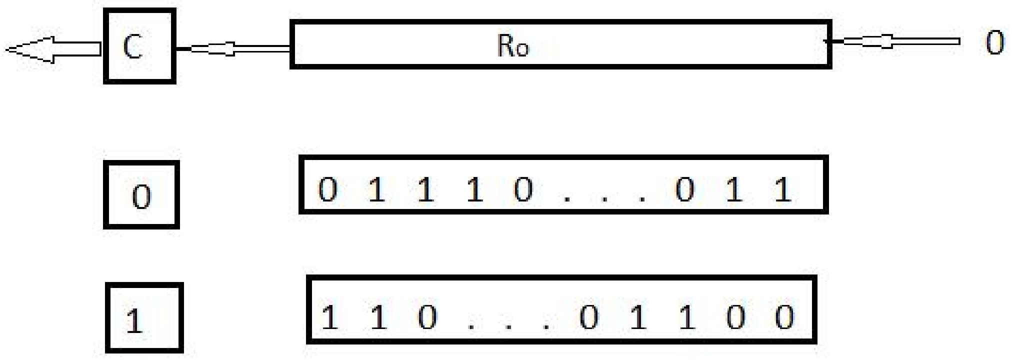
Two logical shift instructions are needed, one for shifting left(LShiftL) and another for shifting right(LShiftR). These instructions shift an operand over a number of bit positions specified in a count operand contained in the instruction.

The general form of a logical left shift instruction is

LShiftL count,dst

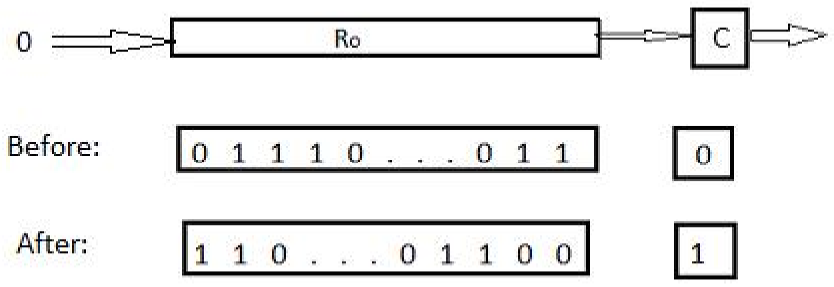
The count operand may be given as an immediate operand, or it may be contained in a processor register. To complete the description of the left shift operation, we need to specify the bit values brought in to the vacated positions at the right end of the destination operand, and to determine what happens to the bits shifted out of the left end. Vacated positions are filled with zeros, and the bits shifted out are passed through the carry flag , C , and then dropped. Involving the C flag in shifts is useful in performing arithmetic operations on large numbers that occupy more than one word.

The following figure shows the example of shifting the contents of the register R_0 left by two bit positions.



Logical shift left LShiftL #2,R0

The logical shift right instruction, LShiftR , works in the same manner except that it shifts to the right , the following figure demonstrates the right shift operation . .

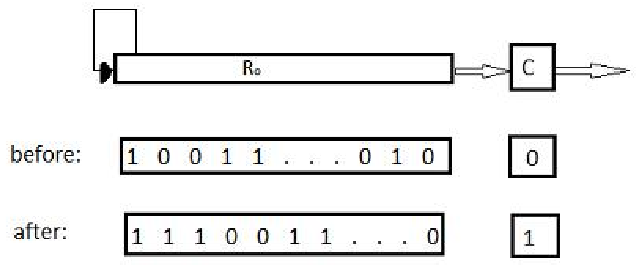


Logical Shift Right LShiftR,R #2,R0

2. <u>Arithmetical Shifts</u>

A 2's-complement binary number representation reveals 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.

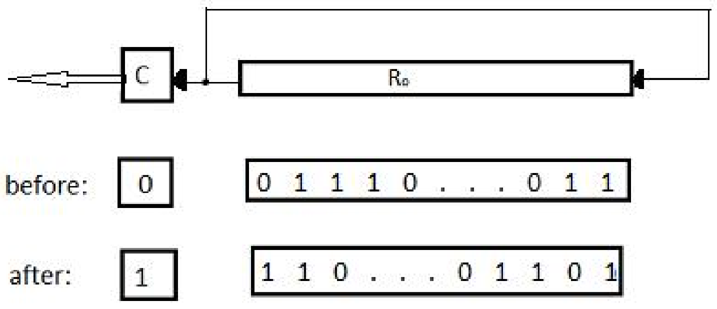
Of course, overflow may occur while shifting left and the remainder is lost while shifting right. Another important observation is that on a right shift the sign bit must be repeated as the fill-in bit for the vacated positions. This requirement on right shifting distinguishes arithmetic shifts from the logical shifts in which the fill-in bit is always 0. Otherwise, the two types of shifts are very similar.
An example of the arithmetic shift i.e., AShiftR , is shown in the figure.



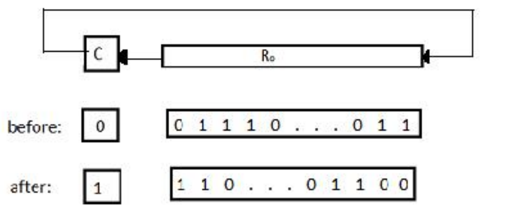
Arithmetic Shift Right AShiftR #2,R₀

3. Rotate Operations

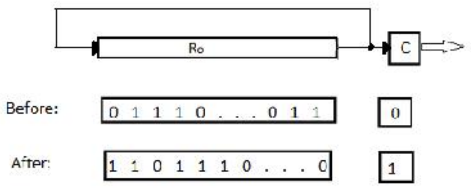
In the shift operations, the bits shifted out of the operand are lost, except for the last bit shifted out which is retained in the carry flag C. To preserve all bits, a set of rotate instructions can be used. They move the bits that are shifted out of one end of the operand back to the other end. Two versions of both the left and right rotate instructions are usually provided. In one version, the bits of the operand are simply rotated. In other version, the rotation includes the C flag. The following figure shows the left and right rotation operations with and without the C flag being included in the rotation.



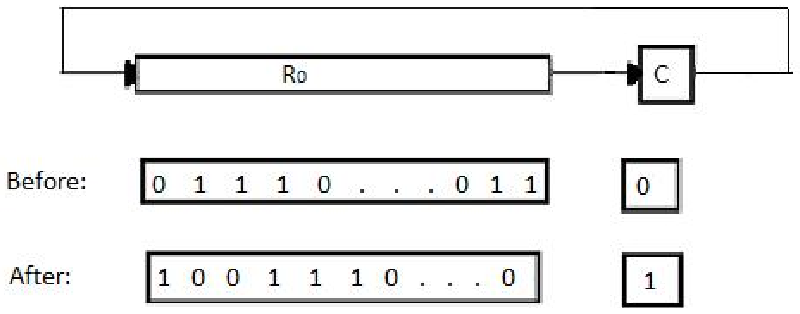
Rotate Left without carry RotateL #2,R₀



Rotate Left with Carry RotateLC #2,R0



Rotate Right without Carry RotateR #2,R0



Rotate Right with Carry RotateRC #2,R0

Write a routine to compare whether the most significant char is Z or not

```
MSBCMP          AND    #$FF000000, R0
                CMP    #$5A000000, R0
                BRANCH=0 SUCCESS
```

Write a routine to pack 2 BCD digits and explain with an example

A routine that packs two BCD digits

```
Move        #LOC, R0       R0 points to data.
MoveByte    (R0)+, R1      Load first byte into R1.
LshiftL     #4, R1         shift left by 4 bit positions.
MoveByte    (R0), R2       Load second byte into R2.
And         #$F, R2        Eliminate high order bits.
Or          R1, R2         Concatenate the BCD digits.
MoveByte    R2, PACKED     Store the result.
```

Explain the following

(1) one word instruction

For a one word instruction there is an 8-bit OP-code field and two 7-bit fields for specifying the source and destination operands. The 7-bit field identifies the addressing mode and the register involved. The "Other info" field specifies the index value or an immediate operand.

e.g.

Move 24(Ro), R5

The instruction requires 16 bits to denote the OP-code and the two registers and some bits to express that the source operand uses the Index addressing mode and that the index value is 24

(a) One-word instruction

Opcode	Source	Dest	Other info

(2) Two word instruction

for a two word instruction there is an 8-bit OP-code field and two 7-bit fields for specifying the source and destination operands. The 7-bit field identifies the addressing mode and the register invovled. The "Other info " field specifies the index value or an immediate operand which also includes a second word which contains the full memory address. e.g.

Move R2, LOC

the instruction requires 18 bits to denote the OP-code,the addressing modes and the register and the second word contains the full memory address of LOC.

(b) Two-Word instruction

Opcode	Source	Dest	Other info
Memory address/Immediate operand			

(3) Three operand instruction

For a three operand instruction there is an OP-code field and three register fields and the "Other info" field specifies the index value or an immediate operand.

e,g.

Add R1,R2,R3

which performs the operation

R3<-[R1]+[R2]

(c) Three-operand instruction

Op code	Ri	Rj	Rk	Other info