INSTRUCTION AND INSTRUCTION SEQUENCING

A computer must have instruction capable of performing the following operations. They are

- Data transfer between memory and processor register.
- Arithmetic and logical operations on data.
- Program sequencing and control.
- I/O transfer.

2.1 REGISTER TRANSFER NOTATION

The possible locations in which transfer of information occurs are,

- Memory Location
- Processor register
- Registers in I/O sub-system.

Location	**Hardware Binary Address**	**Example**	**Description**
Memory	LOC,PLACE,A,VAR2	R1🡐 [LOC]	The contents of memory location are transferred to. the processor register
Processor	R0,R1,….	[R3]🡐[R1]+[R2]	Add the contents of register R1 &R2 and places .their sum into register R3.It is .called Register Transfer Notation.
I/O Registers	DATAIN,DATAOUT		Provides Status information

Assembly Language Notation:

Assembly Language Format	**Description**
Move LOC,R1	Transfers the contents of memory location to the processor register R1.
Add R1,R2,R3	Add the contents of register R1 & R2 and places their sum into register R3.

Basic Instruction Types:

Instruction Type	**Syntax**	**Example**	**Description**
Three Address	Operation Source1,Source2,Destination	Add A,B,C	Add values of variable A ,B & place the result into c.
Two Address	Operation Source, Destination	Add A,B	Add the values of A,B & place the result into B
One Address	Operation Operand	Add B	Content of accumulator add with content of B.

2.2 ADDRESSING MODES

The different ways in which the location of an operand is specified in an instruction is called as Addressing mode.

Generic Addressing Modes:

1. Immediate mode
2. Implied mode
3. Register mode
4. Direct mode or absolute mode
5. Indirect mode
6. Index mode
7. Base Register addressing mode
8. Relative mode
9. Auto-increment mode or decrement mode

Implementation of Variables and Constants:

Variables:

The value can be changed as needed using the appropriate instructions.
There are 2 accessing modes to access the variables. They are

- Register Mode
- Absolute Mode

Register Mode:

The operand is the contents of the processor register. The name(address) of the register is given in the instruction.

Absolute Mode(Direct Mode):

The operand is in new location. The address of this location is given explicitly in the instruction

Example: MOVE LOC,R2

The above instruction uses the register and absolute mode. The processor register is the temporary storage where the data in the register are accessed using register mode. The absolute mode can represent global variables in the program.

Mode	**Assembler Syntax**	**Addressing Function**
Register mode	Ri	**EA=Ri**
Absolute mode	LOC	**EA=LOC**

Where EA-Effective Address

Constants:
Address and data constants can be represented in assembly language using Immediate Mode.

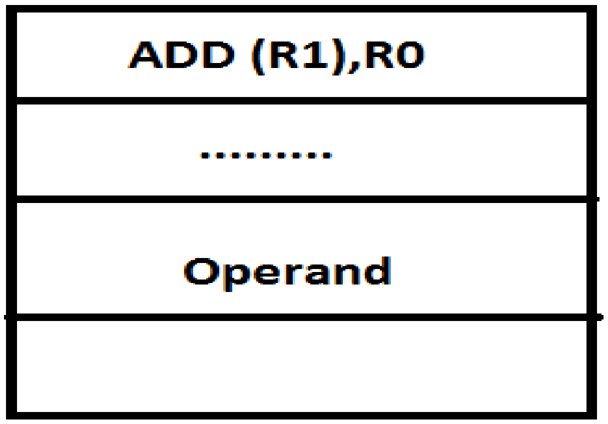
Immediate Mode.
The operand is given explicitly in the instruction.
Example: Move 200 immediate ,R0
It places the value 200 in the register R0.The immediate mode used to specify the value of source operand. In assembly language, the immediate subscript is not appropriate so # symbol is used.
It can be re-written as
Move #200,R0

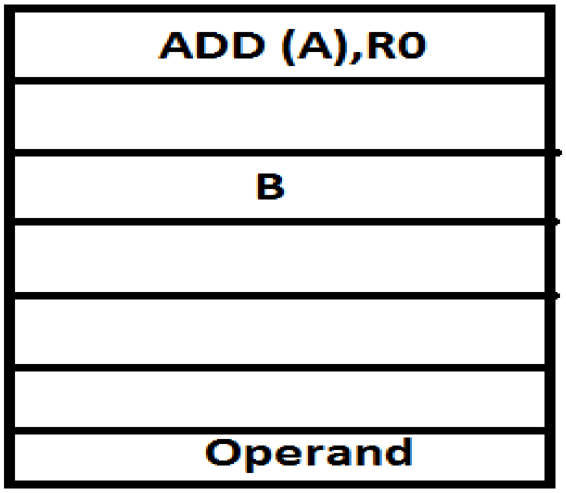
Assembly Syntax:	**Addressing Function**
Immediate #value	Operand =value

Indirection and Pointers

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

Indirect Mode:
The effective address of the operand is the contents of a register .We denote the indirection by the name of the register or new address given in the instruction.
Fig:Indirect Mode





Address of an operand(B) is stored into R1 register.If we want this operand,we can get it through register R1(indirection).
The register or new location that contains the address of an operand is called the **pointer.**

Mode Assembler	**Syntax**	**Addressing Function**
Indirect	Ri , LOC	EA=[Ri] or EA=[LOC]

Indexing and Arrays:

Index Mode:

- The effective address of an operand is generated by adding a constant value to the contents of a register.
- The constant value uses either special purpose or general purpose register.
- We indicate the index mode symbolically as,
  X(R_i)

Where **X** – denotes the constant value contained in the instruction
R_i – It is the name of the register involved.

The Effective Address of the operand is,
EA=X + [Ri]

Implied mode: In this mode,

- Operands are specified implicitly in the definition of the instruction.
- Example: The instruction complement accumulator
- Zero addressed instructions or stack organized computers are implied mode instructions.

Immediate Mode: In this mode,

1. An operand is specified in the instruction itself.
2. An immediate mode instruction has an operand field rather than an address field.
   Example: MOVE x,20
3. It is useful for initializing registers with constant value.
4. When this mode is used in register operation then it is known as “Register mode”

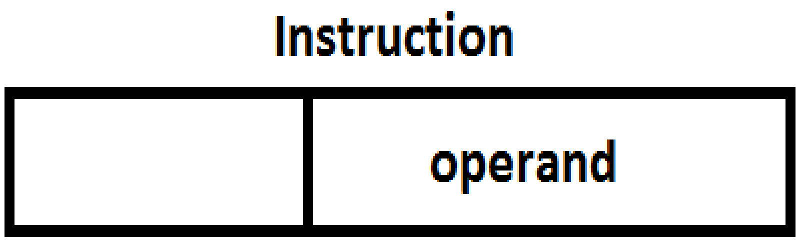
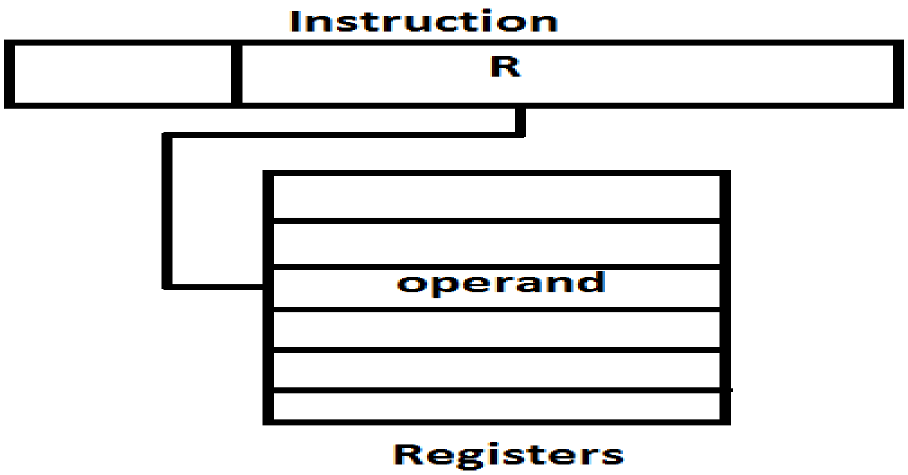


Fig:Immediate mode

Register Mode: In this mode,

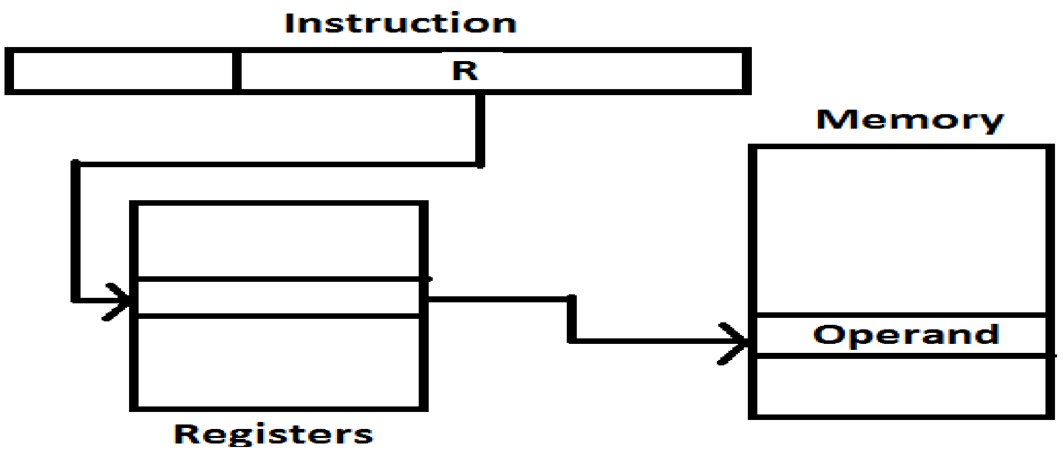
1. The operands in register that reside with in the cpu
2. The particular register is selected from a register field in the instruction.



Register Indirect Mode: In this mode,

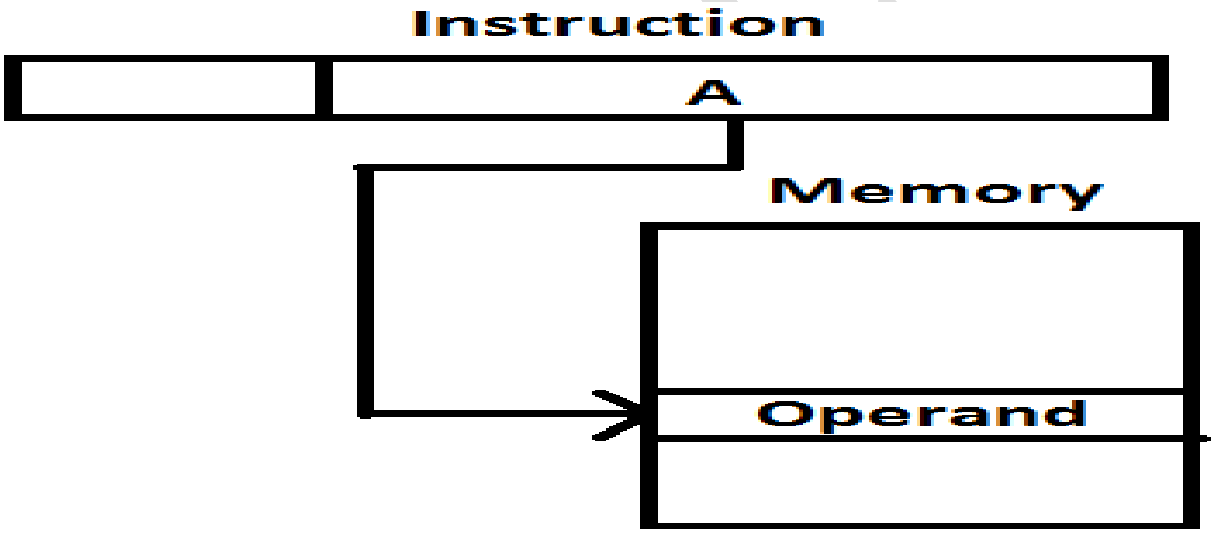
3. The instruction specifies a register in the CPU whose contents gives the address of the operand.
4. The selected register contains the address of the operand rather than the operand itself.

Advantage: The address field of the instruction uses fewer bits to select a register rather than memory.



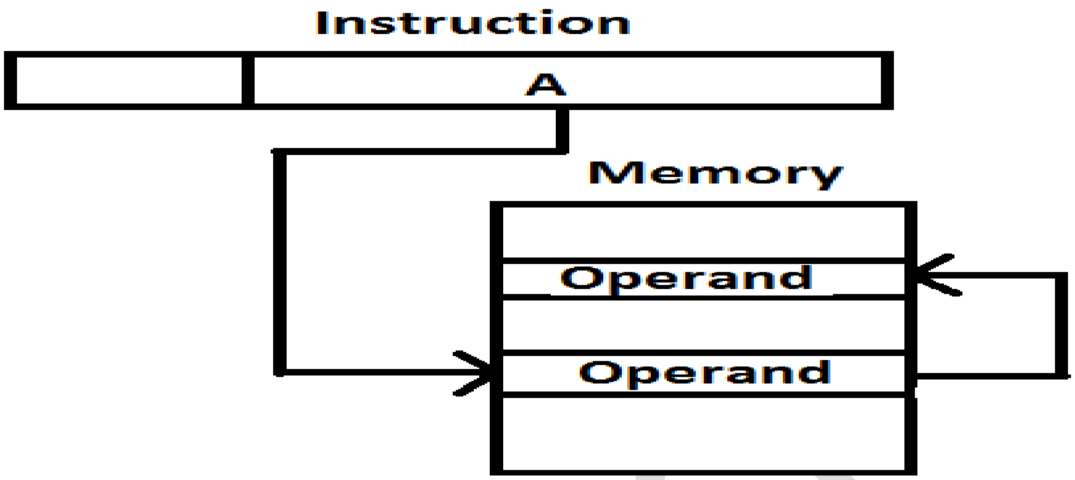
Direct Address Mode: In this mode,

1. The effective address is equal to the address part of the instruction. as shown below



In direct Address Mode: In this mode: In this mode

1. The address field of the instruction gives the address where the effective address is stored in memory.



Relative Address mode: In this mode,

The contents of program counter is added to the address part of the instruction in order to obtain the effective address.

Effective Address(EA)=M[PC]+Address

Example: PC=824

Address=24

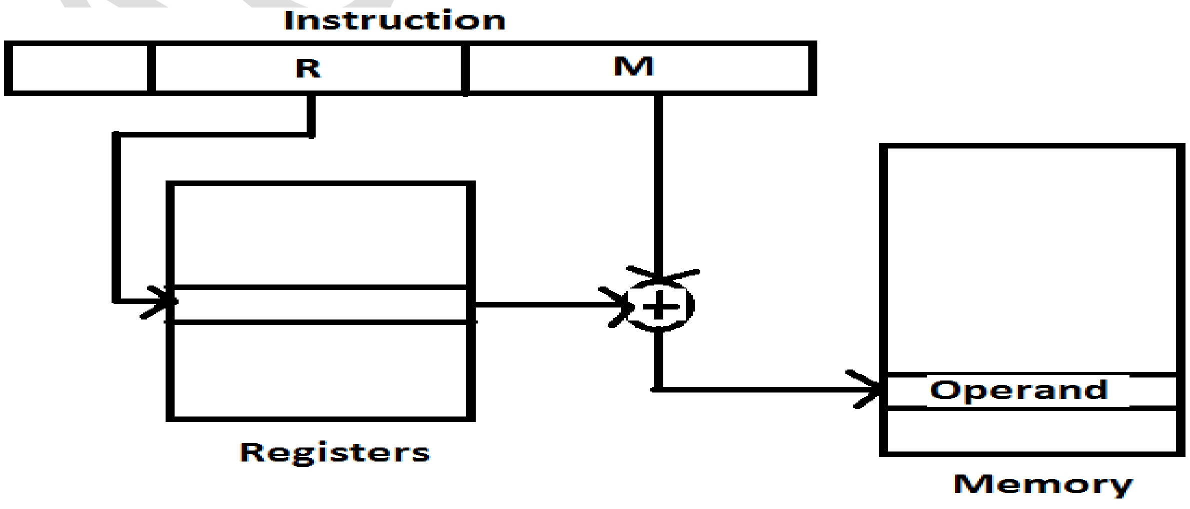
PC value after fetch Phase=824+1=825

EA=825+24=849

Displacement Mode: In This mode,

1. The content of base register is added to the address part of the instruction. It is used in a computer to facilitate the relocation of program in memory

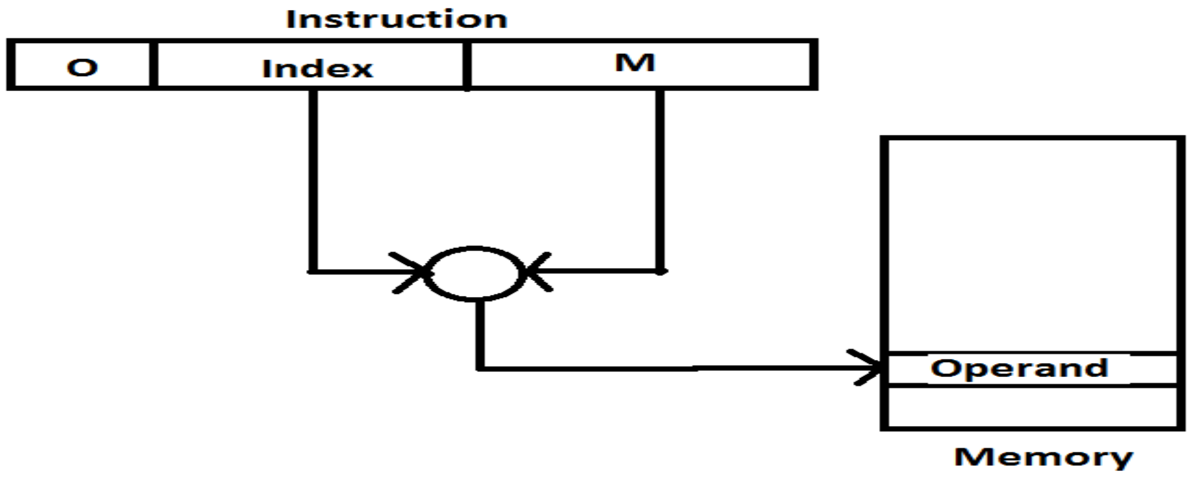
EA=M[Base Register]+Addres



Indexed Addressing Mode: In this mode,

2. The contents of index register is added to the to the address part of the instruction to obtain effective address.
3. The index register is a special CPU register that contain an other index value.

EA=Index register+Address field



Auto increment OR Decrement Mode: In this mode

When the address stored in the register refers to table of data in memory, it is necessary to increment or decrement register after every access to the table.

Effective Address= it is defined to be memory address obtained from the computation dictated by the given addressing mode.

It is the address where control branches in a response to a branch type instruction.

2.3 BASIC INPUT OUTPUT OPERATIONS

Input/output operations specifies the way how data are transferred between memory of a computer and outside world. I/O operations are essential and have significant effect on the performance of computer.

Consider a task that read character input from the keyboard and produces character output on a display screen. This can be performed by using program-controlled I/O.

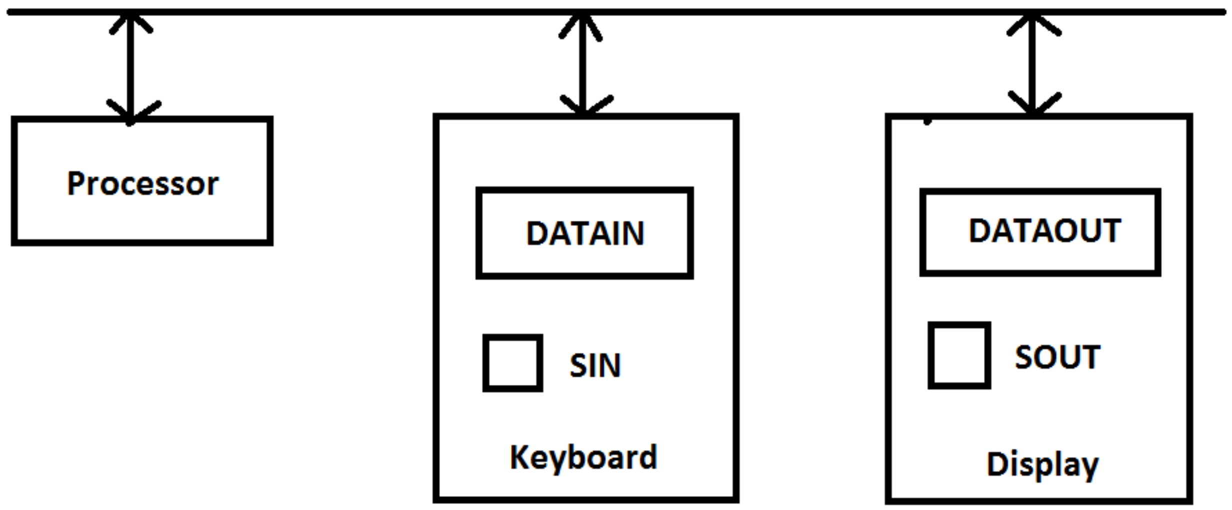


Figure**: Bus connection for processor, keyboard and Display**

A solution to this problem is as follows: On output, the processor sends the first character and then waits for a signal from the display that the character has been received. It then sends the second character and so on. Input is sent from the keyboard in similar way: the processor waits for the signal indicated that a character key has been struck and that its code is available in some buffer register associated with the keyboard.

The keyboard and display are separate devices as shown in figure. The action of striking the key on keyboard does not automatically cause the corresponding character to be displayed on the screen. One block of instruction in I/O program transfers the characters into processor, and an other associated block of instructions cause the character is to be displayed.

Consider a problem of moving character code from keyboard to the processor. striking a key stores the corresponding character code in 8 -bit buffer register associated with the keyboard. Let us call this register DATAIN. to inform the processor that the valid character is in DATAIN , a status control flag, 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 SIN is again set 1 and the process repeats.

Same process takes place when characters are transferred from the processor to the display. A buffer register ,DATAOUT, and a status control flag, SOUT, are used for this transfer. When SOUT equals to 1, the display is ready to receive a character. Under program control , the processor monitors SOUT and when SOUT set to 1.The processor transfer a character code to DATAOUT. The transfer of character code to DATAOUT clears SOUT to 0.When the display device is ready to receive second character, SOUT is Again set to 1.The buffer registers DATAIN and DATAOUT and the status flags SIN and SOUT are part of circuitry commonly known as **Device Interface**

Example:1

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

READWAIT   branch to READWAIT if SIN=0

                    Input from DATAIN to R1

The branch operation is usually implemented by two machine instructions. The first instruction tests the status flag and second performs branching.

Example:1

Sequence of operations used for transferring output to the display is

READWAIT   branch to READWAIT if SOUT=0

                    Output from R1 to DATAOUT

2.4 STACKS AND QUEES

Stack is a LIFO list.in stack the elements are added only at one end called top.push and pop operations are used to work with the stack.

- Push operation places an element on to the top of the stack.
- Pop operations delete an element from the top of the stack.

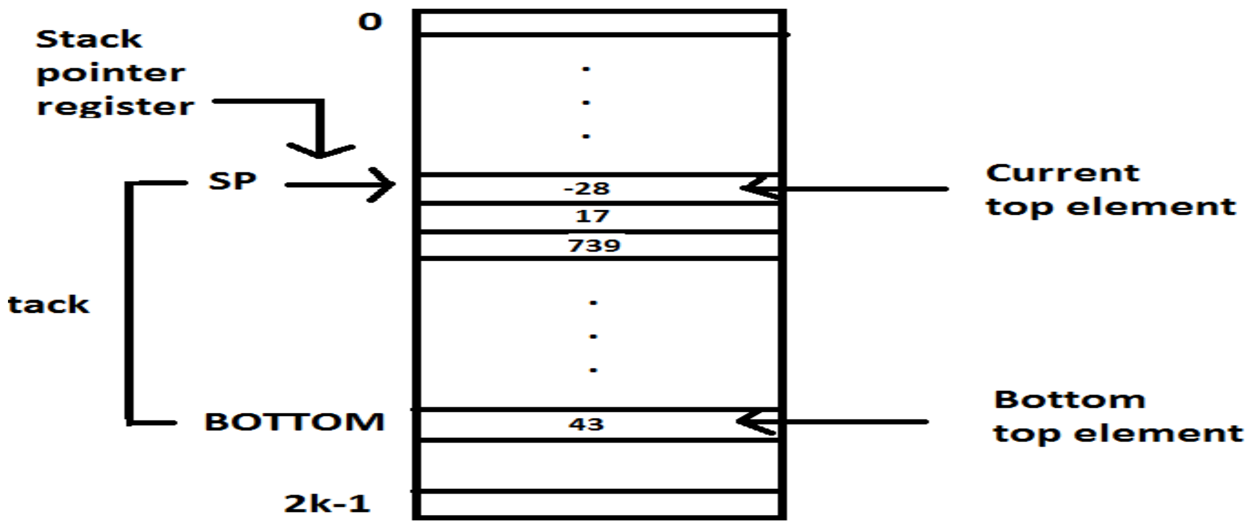


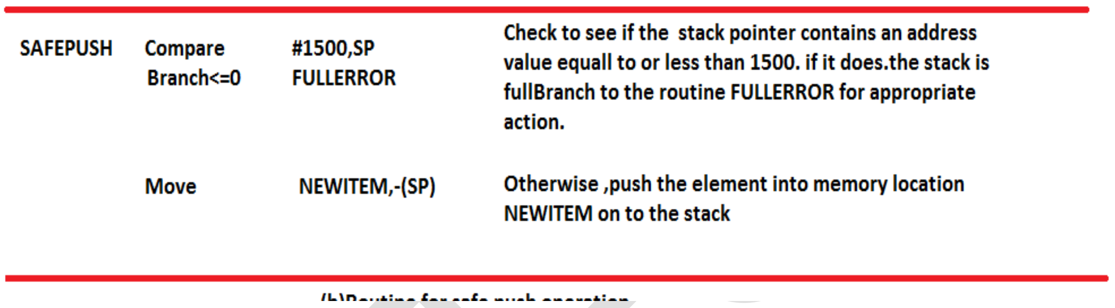
Figure: A stack of words in the memory

Figure above shows the stack of word data items in the memory of a computer. it contains 43 as bottom element and -28 at the top. stack pointer (SP)register is a processor register contains the address of the stack of the top. if we assume byte addressable memory with 32 bit word length, the push operation is as follows.

```
Subtract    #4,SP

Move  NEWITEM ,SP
```

The above two instructions move the word from location NEWITEM into the top of the stack.



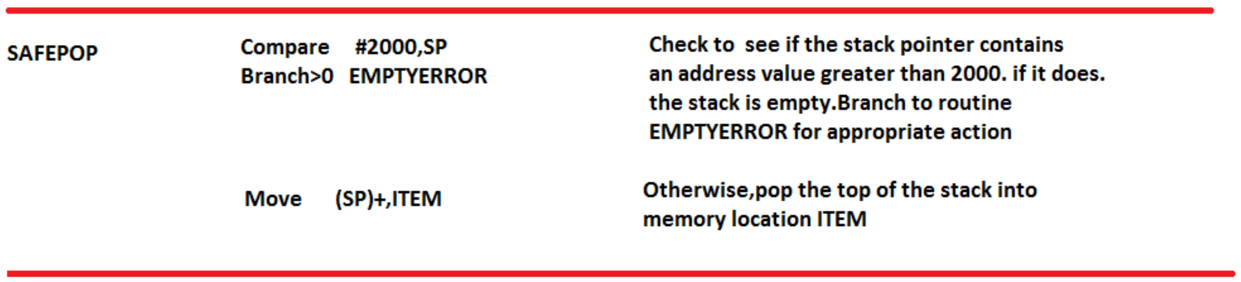
SAFEPUSH	Compare Branch<=0	#1500,SP FULLERROR	Check to see if the stack pointer contains an address value equall to or less than 1500. if it does.the stack is fullBranch to the routine FULLERROR for appropriate action.
	Move	NEWITEM,-(SP)	Otherwise ,push the element into memory location NEWITEM on to the stack

(b)Routine for safe push operation

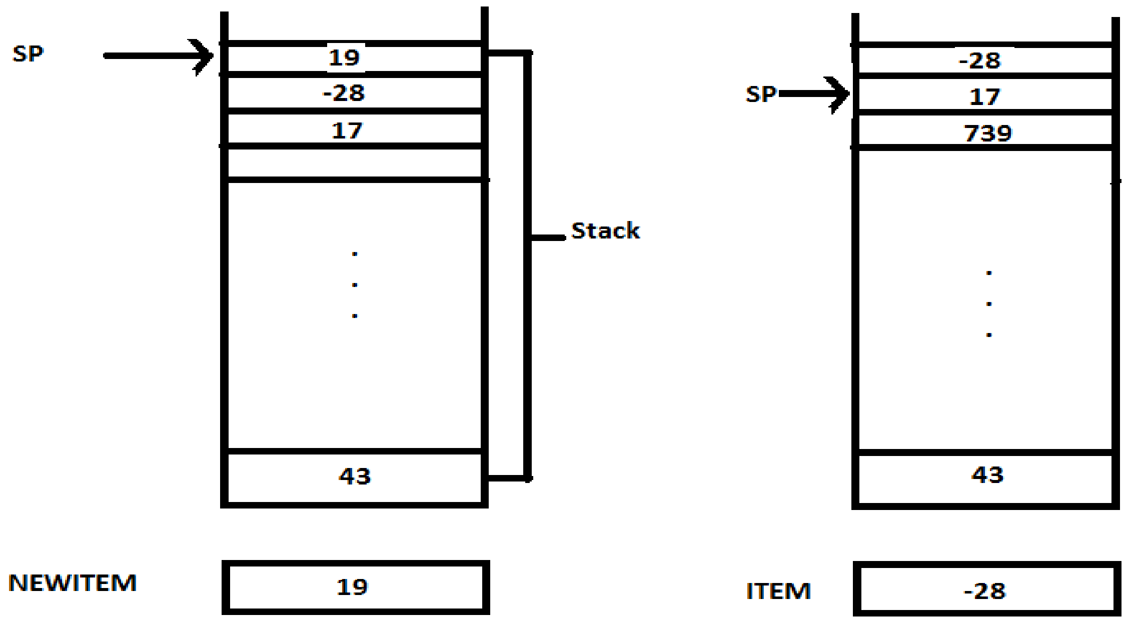
Pop opration:

```
Move (SP),ITEM

Add   #4,SP
```



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



a) After PUSH from NEWITEM b)After POP into ITEM

Figure: Effect of stack operations on the stack in figure

If the processor has auto increment and auto decrement addressing modes then the push operation can be performed by the single instruction

Move NEWITEM,-(SP)

push operation can be performed by the single instruction

Move (SP)+,ITEM

QUEUE

A useful data structure similar to stack is called queue. Data are stored in and retrieved from a queue on first-in-first-out(FIFO) basis. Queue grows in the direction of increasing addresses in the memory. New data are added at the back (high-address-end) and retrieved from the front (low-address-end) of the queue.

Operations on Queue:

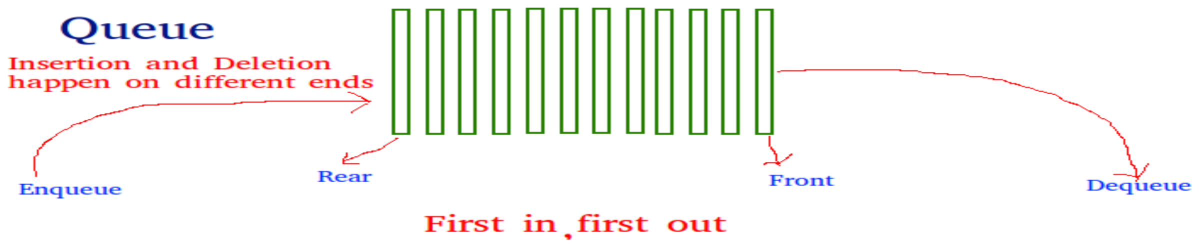
Mainly the following four basic operations are performed on queue:

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.

Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.

Front: Get the the front item from queue.

Rear: Get the last item from queue.



Applications of Queue:

Queue is used when things don't have to be processed immediately, but have to be processed in **F**irst **I**n**F**irst **O**ut order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

1) When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
2) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

2.5 LOGIC INSTRUCTIONS

Logical instructions perform binary operations on strings of bits stored in registers. They are useful in manipulating individual bits or group of bits that represent binary coded information. they consider each bit of the operand separately and treat them as Boolean value.

The basic logical instructions

Name	Mnemonic
Clear	CLR
Compliment	COM
AND	AND
OR	OR
Ex-or	X-OR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

- The clear operand causes the operand to be replaced by zero's
- The complement operation produces one's complement by inverting all the bits of the component.

Example: a=1001
Complement of a=0110

- AND operation is used to clear a bit or a group of bits of a operand. For any Boolean variable x x AND 0=0 x AND 1=x
- The OR operation is used to set a bit or selected group of bits of an operand. variable x x+1=1 and x+0=x
- XOR is used to selectively complement bits of an operand.

2.6 SHIFT AND ROTATE INSTRUCTIONS

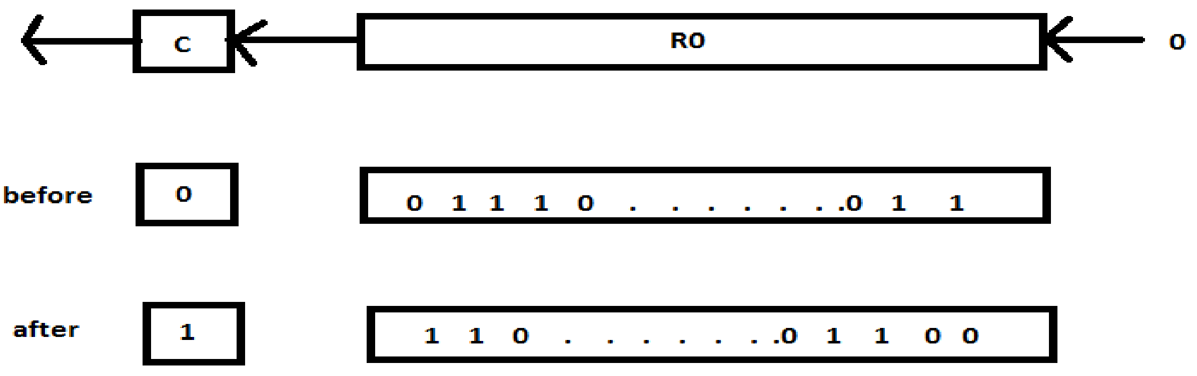
There are many applications that require the bits of an operand to be shifted right or left some specified number of bit positions.
Logical shifts
There are two types of logical shifts

1. Logical shift left(LshiftL)
2. Logical shift right(LshiftR)

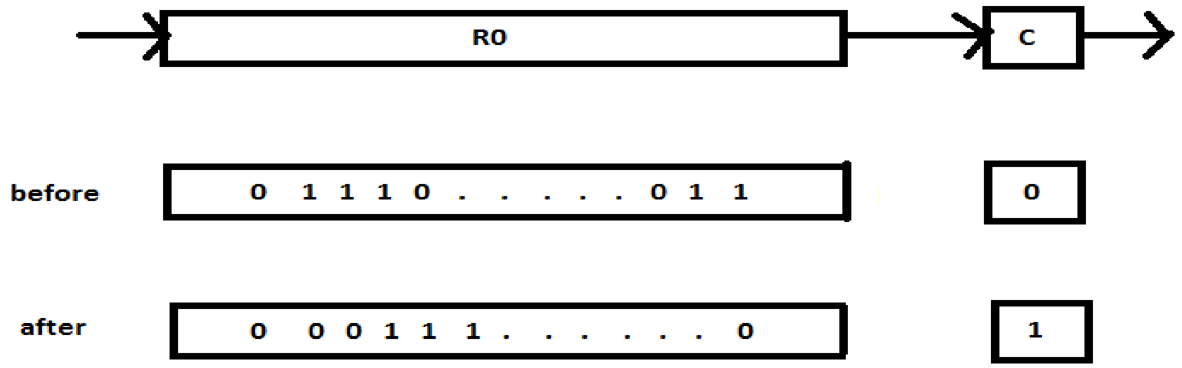
Logical shift left(LshiftL): it shifts the bits towards left. Here carry bit will be preserved during operation. The general form of logical shift left is

LShiftL count ,dst

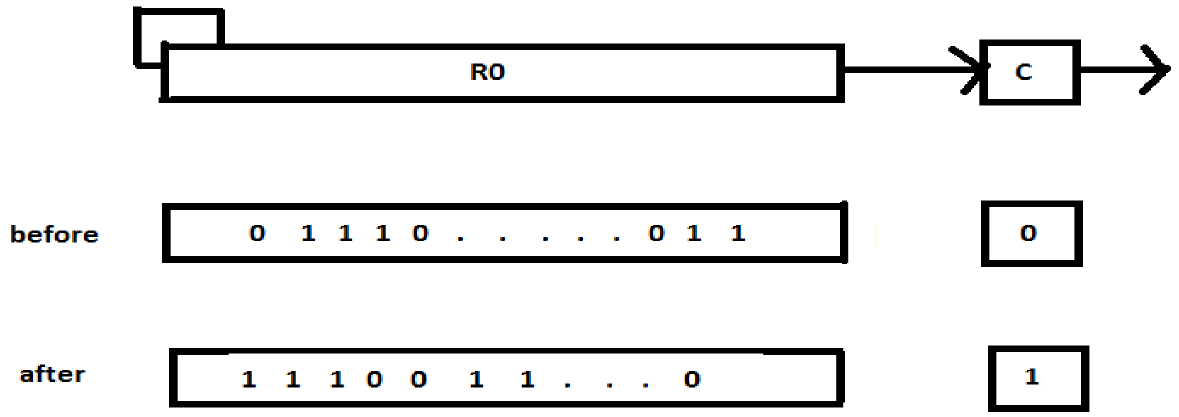
In the above format count specifies no of shifts and dst specifies the destination operand on which shift has to be taken place.



(a) Logical shift left LShiftL #2,R0



(b) logical Shift Right LShiftR #2 ,R0



(c) Arithmetc shift right AshiftR #2,R0

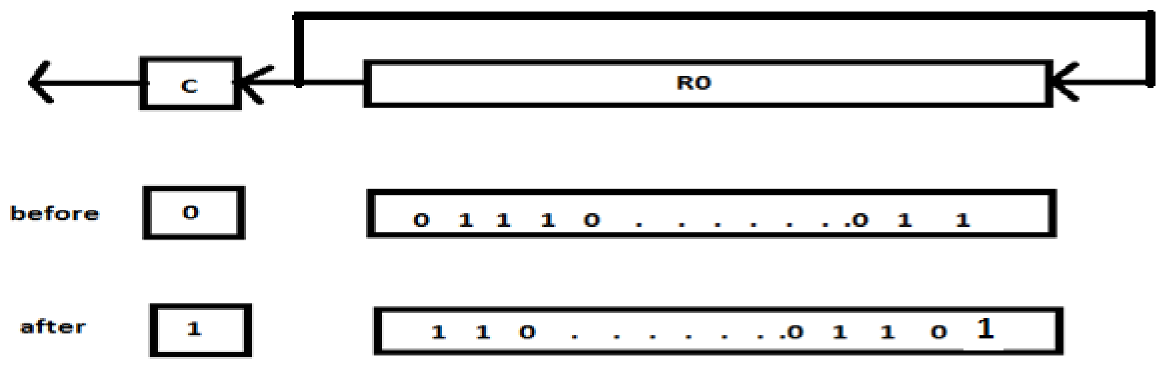
Figure: Logial and Arithmetic Shift Instructions

ROTATE OPERATIONST:

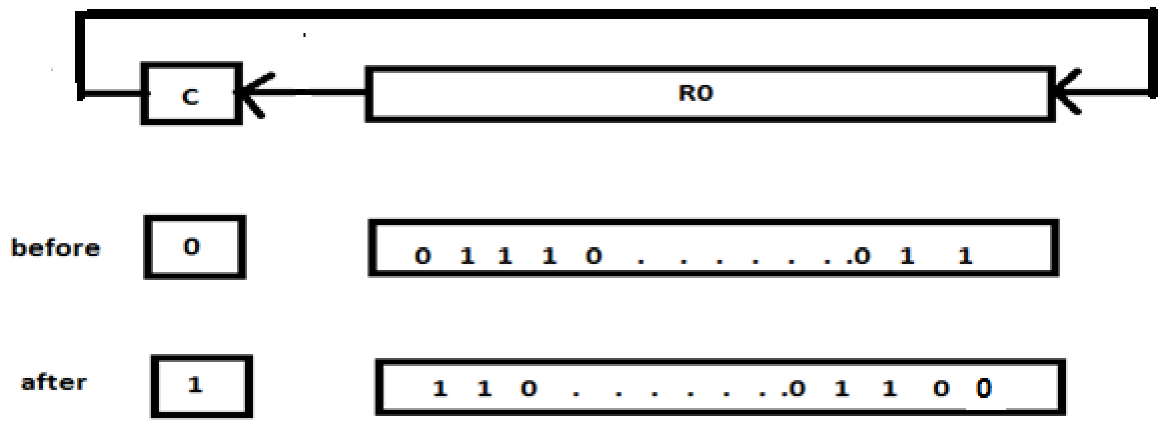
In shift operations the bits shifted out of the operand are lost, except for the lost bit shifted out which is retained in the carry flag C. To preserve all the bits set of rotate instructions are used. They move the bits that are shifted out of one end of the operand into the other end.

There are different rotations are

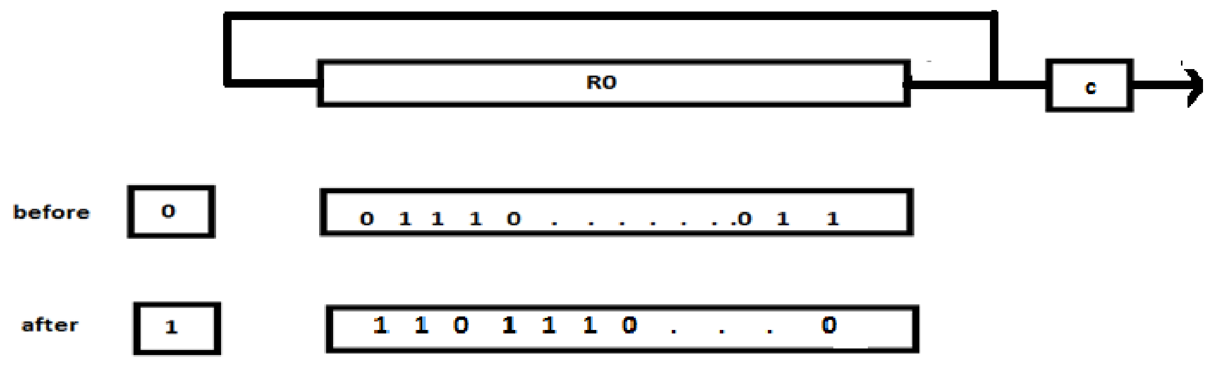
1. Rotate Left without carry(RotateL)
2. Rotate Left with carry(RotateLC)
3. Rotate Right without carry(RotateR)
4. Rotate Right with carry(RotateRc)



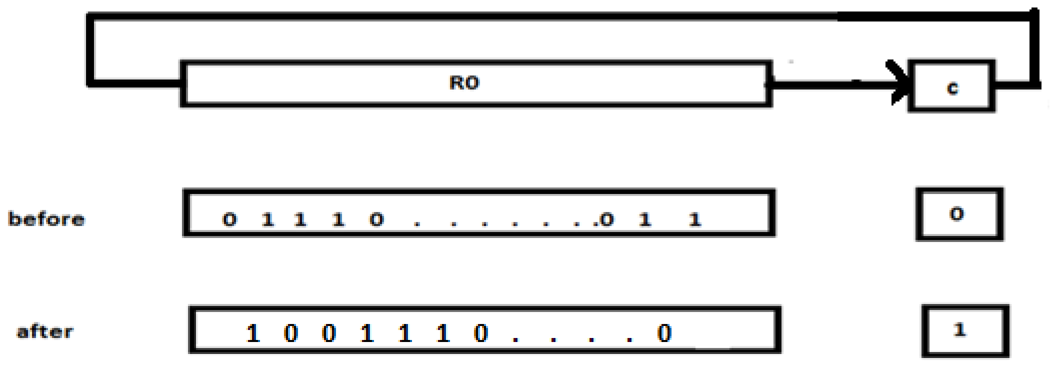
(a) Rotate left with out carry RotateL #2,R0



(a) Rotate left with carry RotateLC #2,R0



(c) Rotate right with out carry RotateR #2, R0



(d)Rotate Right with carry RotateRC #2, R0