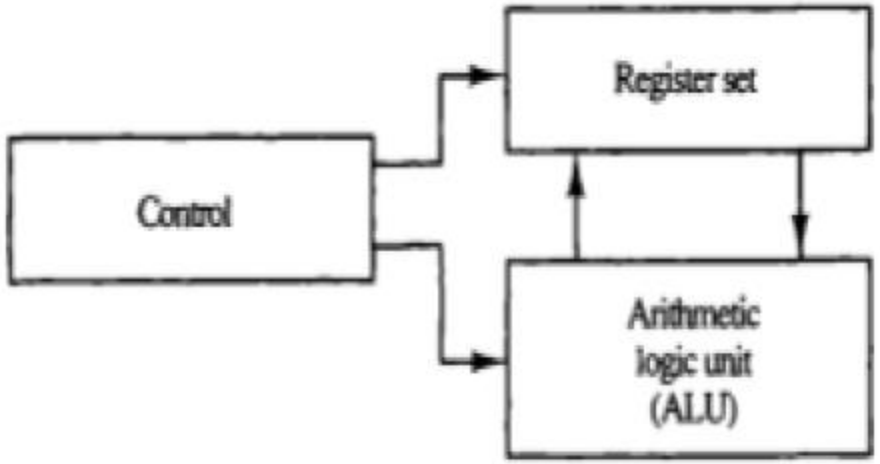
Unit-5
Central Processing Unit

Compiled by :- Er Nagendra karn

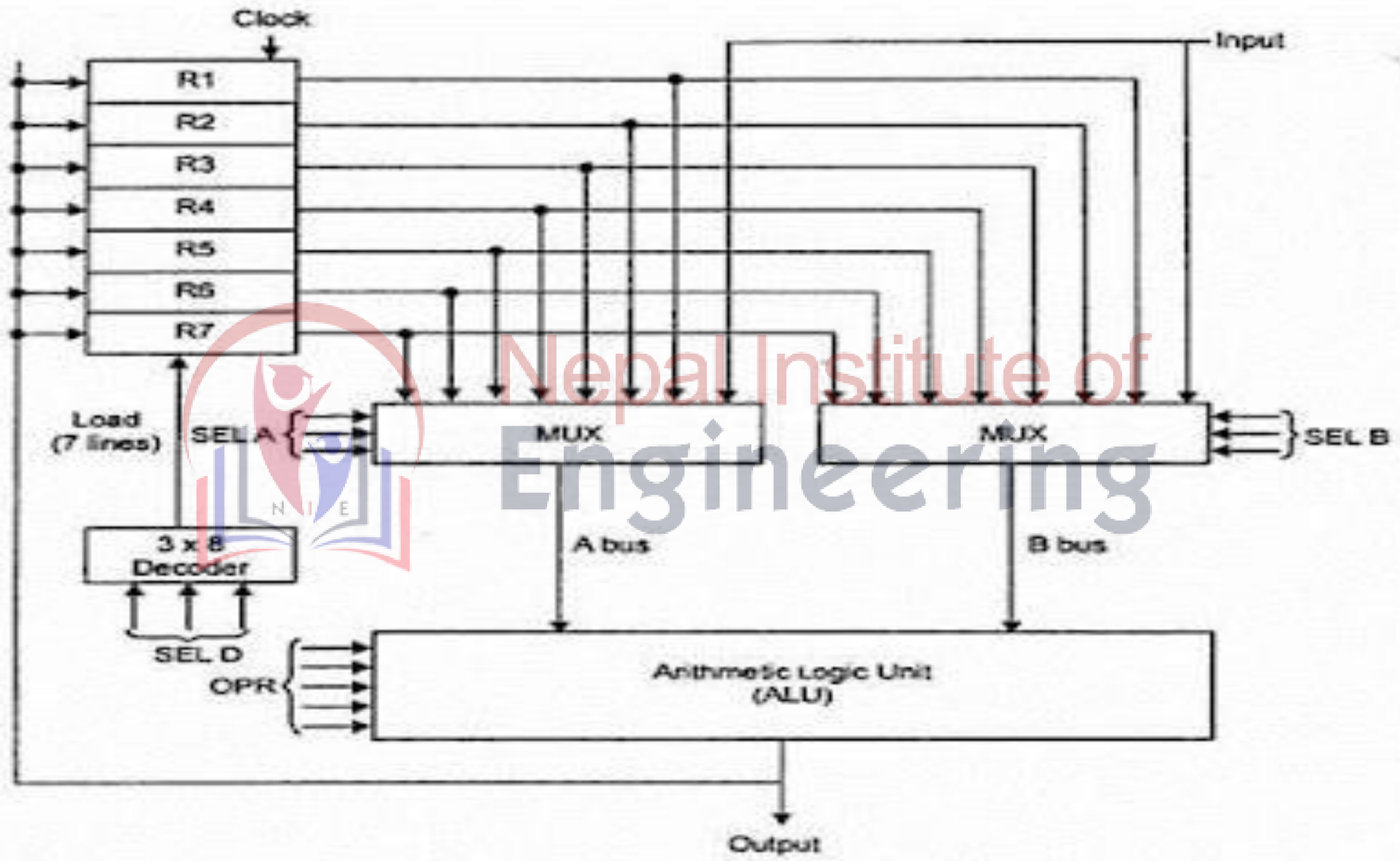
Introduction:-

- The part of the computer that performs the bulk of data-processing operations is called the central processing unit and is referred to as the CPU.
- The CPU is made up of three major parts, as shown in Fig.
  - ➢ The register set stores intermediate data used during the execution of the instructions.
  - ➢ The arithmetic logic unit (ALU) performs the required micro operations for executing the instructions.
  - ➢ The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.



General Register Organization :-

- A set of flip-flops forms a register. A register is a unique high-speed storage area in the CPU.
- They include combinational circuits that implement data processing. The information is always defined in a register before processing. The registers speed up the implementation of programs.
- When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system.
- The registers communicate with each other not only for direct data transfers, but also while performing various micro operations.
- The CPU bus system is managed by the control unit. The control unit explicit the data flow through the ALU by choosing the function of the ALU and components of the system.


Clock
Input
R1
R2
R3
R4
R5
R6
R7
Load (7 lines)
SEL A
MUX
MUX
SEL B
3 x 8 Decoder
A bus
B bus
SEL D
OPR
Arithmetic Logic Unit (ALU)
Output

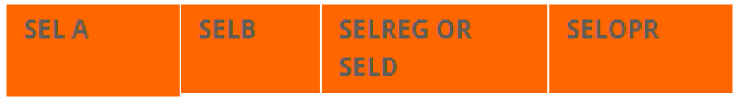
- The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system. For example, to perform the operation

  $R1 \leftarrow R2+R3$

- the control must provide binary selection variables to the following selector inputs:
- **MUX A Selector (SELA)** − It can place R2 into bus A.
- **MUX B Selector (SELB)** − It can place R3 into bus B.
- **ALU Operation Selector (OPR)** − It can select the arithmetic addition (ADD).
- **Decoder Destination Selector (SELD)** − It can transfers the result into R1.
- The multiplexers of 3-state gates are performed with the buses. The state of 14 binary selection inputs determines the control word. The 14-bit control word defines a micro-operation.

Control Word :-

- There are 14 binary selection inputs in the unit, and their combined value specifies a control word.



SEL A	SELB	SELREG OR SELD	SELOPR

FORMATE OF CONTROL WORD

- The three bit of SELA select a source registers of the **a** input of the ALU.
- The three bits of SELB select a source registers of the **b** input of the ALU.
- The three bits of SELED or SELREG select a destination register using the decoder.
- The four bits of SELOPR select the operation to be performed by ALU.
- A control word of 14 bits is needed to specify a micro operation in the CPU. The control word for a given micro operation can be derived from the selection variables.

Encoding of Register Selection Field:-

Binary code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

For example, the subtract micro operation given by the statement

$$R1 \leftarrow R2 - R3$$

specifies R2 for the A input of the ALU, R3 for the B input of the ALU, R1 for the destination register, and an ALU operation to subtract A - B. Thus the control word is specified by the four fields and the corresponding binary value for each field is obtained from the encoding.

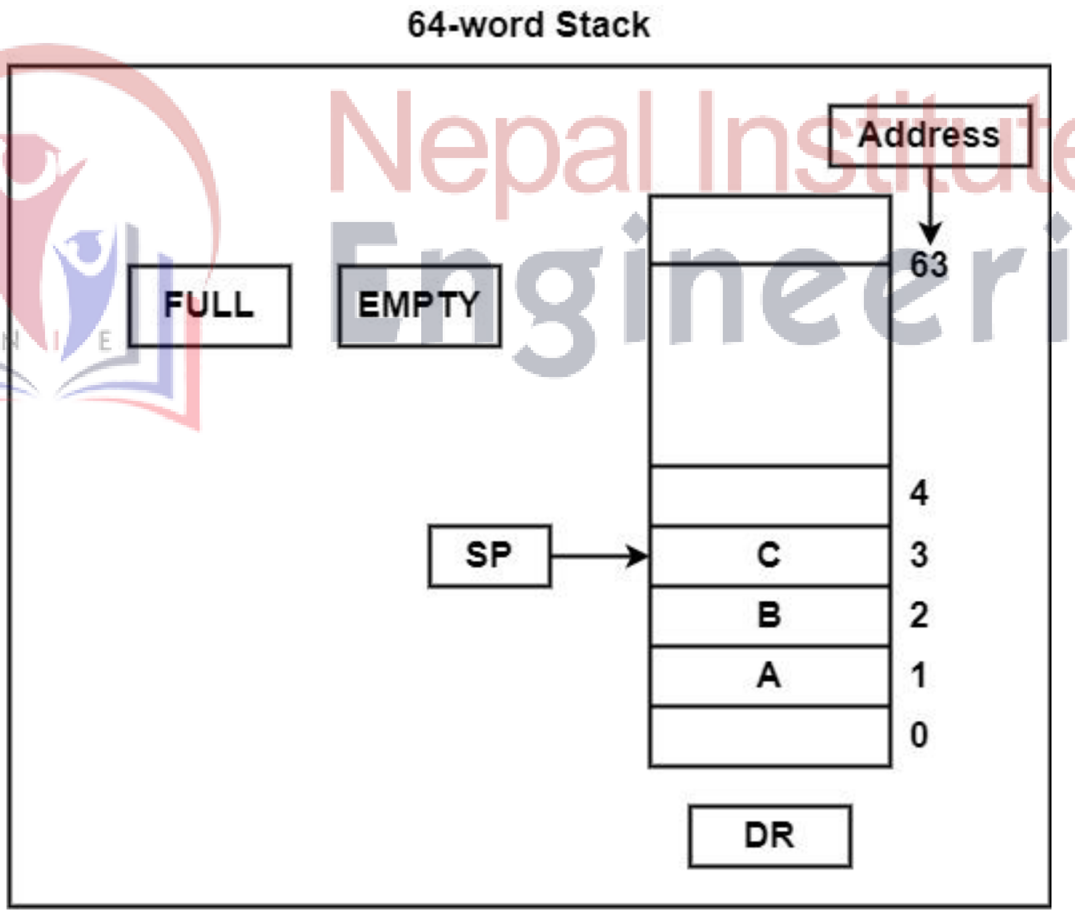
OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	ADD A+B	ADD
00101	SUB A-B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift Right A	SHRA
11000	Shift left A	SHLA

Stack Organization :-

- Stack is also known as the Last In First Out (LIFO) list. It is the most important feature in the CPU. It saves data such that the element stored last is retrieved first.
- A stack is a memory unit with an address register. This register influence the address for the stack, which is known as Stack Pointer (SP). The stack pointer continually influences the address of the element that is located at the top of the stack.
- It can insert an element into or delete an element from the stack. The insertion operation is known as push operation and the deletion operation is known as pop operation. In a computer stack, these operations are simulated by incrementing or decrementing the SP register.

Register Stack :-

- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure shows the organization of a 64-word register stack.



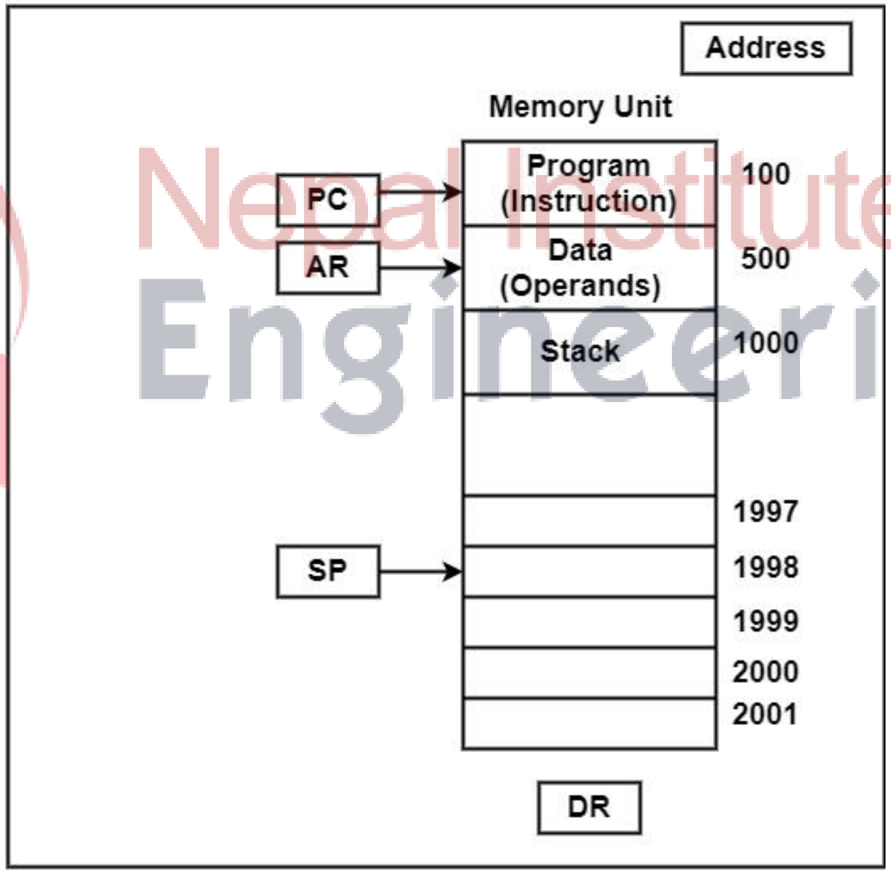
- The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C, in that order. Item C is on top of the stack so that the content of SP is now 3.
- To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2.
- To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack.
- Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation.

- The push operation is implemented with the following sequence of micro operations;
- SP <- SP + 1 Increment stack pointer
- M[SP]<-DR Write item on top of the stack
- A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequence of micro operations:
- DR <--M[SP] Read item from the top of stack
- SP<--SP - 1 Decrement stack pointer

Memory Stack :-

- A stack can be implemented in a random-access memory attached to a CPU.
- The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer.
- The portion of computer memory is partitioned into three segments: program, data, and stack.
- The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data. The stack pointer SP points at the top of the stack.
- PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or pop items into or from the stack.

Memory Stack



- In the figure, the SP points to a beginning value ‘2001’. Therefore, the stack increase with decreasing addresses. The first element is saved at address 2000, the next element is saved at address 1999 and the last element is saved at address 1000.
- The data register can read an element into or from the stack. It can use push operation to insert a new element into the stack.
    - SP<-SP - 1
    - M[SP]<-DR
- A new item is deleted with a pop operation as follows:
    - DR <-M[SP]
    - SP<-SP + 1
- The top item is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack.

Program Interrupt:-

- Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request.
- Control returns to the original program after the service program is executed. The interrupt procedure is, in principle, quite similar to a subroutine call except for three variations:
- The interrupt is usually initiated by an internal or external signal rather than from the execution of an instruction (except for software interrupt);
- The address of the interrupt service program is determined by the hardware rather than from the address field of an instruction; and
- an interrupt procedure usually stores all the information necessary to define the state of the CPU rather than storing only the program counter.

Types of Interrupt :-

- There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:
- **External Interrupt :-** External interrupts come from input/output (I/0) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source. Examples that cause external interrupts are I/0 device requesting transfer of data, I/0 device finished transfer of data, elapsed time of an event, or power failure.
- **Internal Interrupt :-** Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called **traps**. Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation.
- The difference between internal and external interrupts is that the internal interrupt is initiated by some exceptional condition caused by the program itself rather than by an external event.

Software Interrupts:-

- External and internal interrupts are initiated from signals that occur in the hardware of the CPU.
- A software interrupt is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call.
- The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode.
- Certain operations in the computer may be assigned to the supervisor mode only, as for example, a complex input or output transfer procedure.
- A program written by a user must run in the user mode. When an input or output transfer is required, the supervisor mode is requested by means of a supervisor call instruction. This instruction causes a software interrupt.

Instruction Format :-

- Computer perform task on the basis of instruction provided. **An instruction format** defines layout of bits of an instruction.
- The most common fields are:
  - Operation field which specifies the operation to be performed like addition.
  - Address field which contain the location of operand, i.e., register or memory location.
  - Mode field which specifies how operand is to be founded.
- An instruction is of various length depending upon the number of addresses it contain. Generally, CPU organization are of three types on the basis of number of address fields:
- Single Accumulator organization
- General register organization
- Stack organization

1. Single Accumulator Organization :-

- All operations are performed with an accumulator register. The instruction format in this type of computer uses one address field. e.g. **ADD X**

2. General Register Organization :-

The instruction format in this type of computer needs three register address fields. e.g. **ADD R1,R2,R3** or **ADD R1,R2**.

3. Stack Organization :-

Computers with stack organization would have PUSH and POP instructions which require an address field. e.g. **PUSH X, ADD**

Instruction Format :-

- To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement

$$X = (A + B) * (C + D)$$

using zero, one, two, or three address instructions.

- **Three Address Instruction :-**
- Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.

ADD R1, A, B	**R1←M[A] + M[B]**
ADD R2, C, D	**R2←M[C] + M[D]**
MUL X, R1, R2	**M[X] ←R1*R2**

- The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

Two-Address Instruction :-

- Two-address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word.

MOV R1, A	**R1←M[A]**
ADD R1, B	**R1←R1 + M[B]**
MOV R2, C	**R2←M[C]**
ADD R2, D	**R2←R2 + M[D]**
MUL R1, R2	**R1←R1*R2**
MOV X, R1	**M[X] ←R1**

One-Address Instruction :-

- One-address instructions use an accumulator (AC) register for all data manipulation. here we will neglect the second register and assume that the AC contains the result of all operations.

LOAD A	**AC←M[A]**
ADD B	**AC←AC+M[B]**
STORE T	**M[T]←AC**
LOAD C	**AC←M[C]**
ADD D	**AC←AC+M[D]**
MUL T	**AC ←AC*M[T]**
STORE X	**M[X] ←AC**

- All operations are done between the AC register and a memory operand.

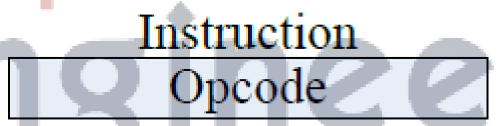
Zero-Address Instruction :-

- A stack-organized computer does not use an address field for the instructions **ADD** and **MUL**. The **PUSH** and **POP** instructions, however, need an address field to specify the operand that communicates with the stack.

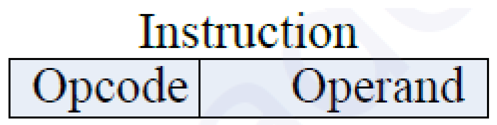
PUSH A	**TOS←A**
PUSH B	**TOS ←B**
ADD	**TOS ←(A+B)**
PUSH C	**TOS ←C**
PUSH D	**TOS ←D**
ADD	**TOS ←(C+D)**
MUL	**TOS ←(C+D)*(A+B)**
POP X	**M[X] ←TOS**

Addressing Mode :-

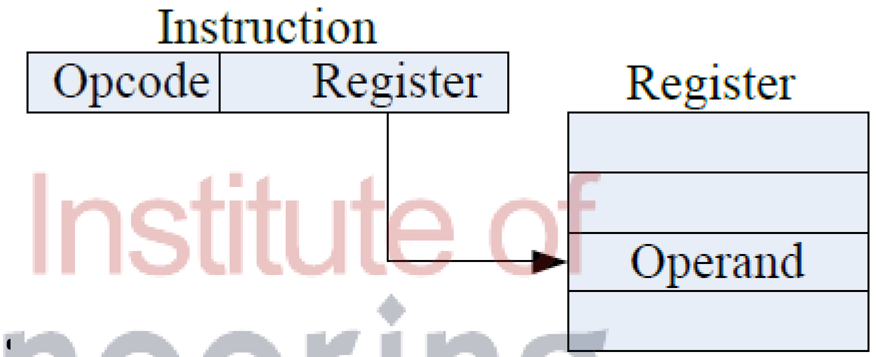
- **Implied Mode :-** The operand is hidden, and data to be operated is available in the instruction itself. e.g. CMA.
- All register reference instructions that use an accumulator are implied-mode instructions. Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.



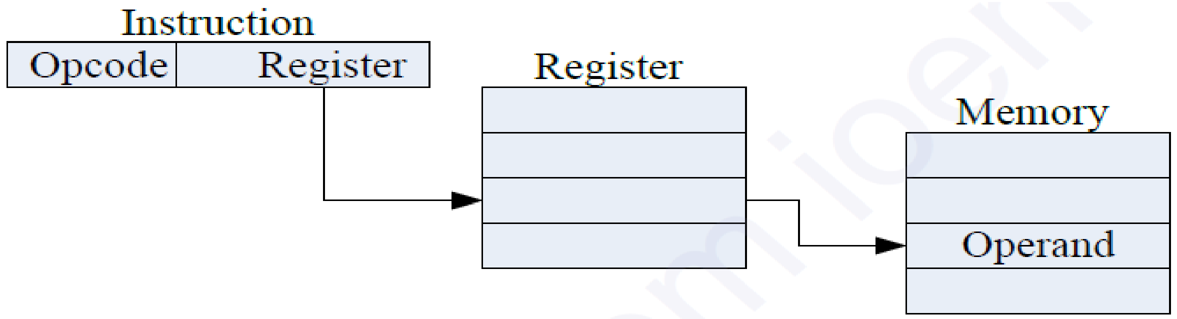
- **Immediate Mode :-** In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field.



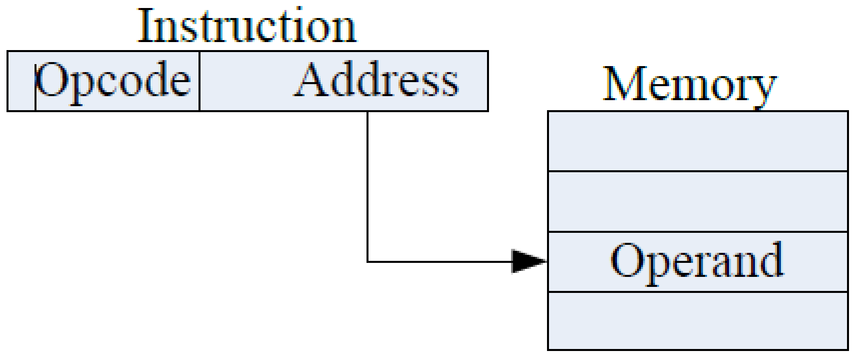
- **Register Direct Addressing Mode :-**
- In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. e.g. MOV A,B



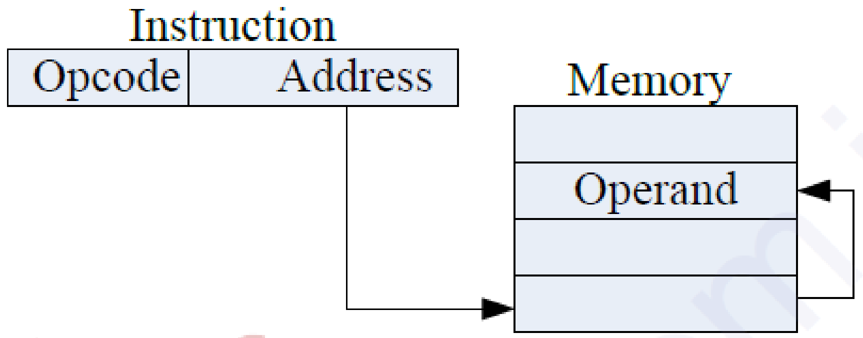
- **Register Indirect Addressing Mode :**
- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the operand itself.



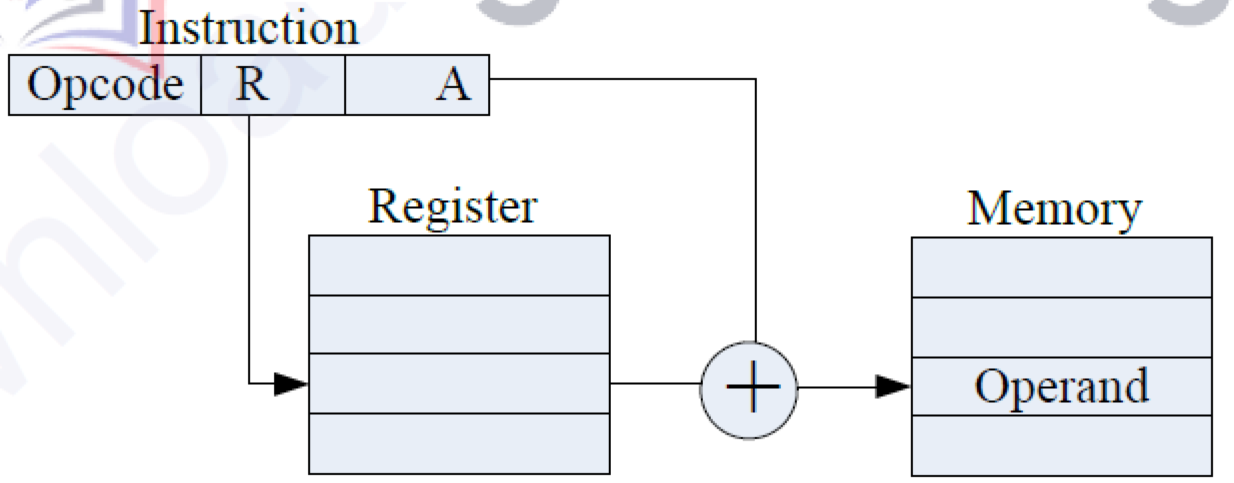
- **Auto Increment or Auto Decrement mode :-**
- This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
- After accessing the operand, the contents of this register are automatically incremented to the next value.
- Before accessing the operand, the contents of this register are automatically decremented and then the value is accessed.
- **Direct Addressing Mode :-**
- Here, the operand resides in memory and its address is given directly by the address field of the instruction. In a branch-type instruction the address field specifies the actual branch address.



- **Indirect Addressing Mode :-**
- In this mode the address field of the instruction gives the address where the effective address is stored in memory.



- **Displacement Addressing Mode :-**
- A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing.
- The address field of instruction is added to the content of specific register in the CPU.



Effective Address (EA) = A + (R)

- **Relative addressing Mode :-** In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
- **Indexed addressing Mode :-** In this mode the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value.
- **Base Register addressing Mode :-** In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.

Data Transfer Manipulation :-

- Data transfer instructions cause transfer of data from one location to another without changing the binary information. The most common transfer are between the
  - Memory and Processor registers
  - Processor registers and input output devices
  - Processor registers themselves
- **Typical Data transfer Instructions :-**

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
PUSH	PUSH
Pop	POP

Data Manipulation Instructions :-

- Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. These instructions perform arithmetic, logic and shift operations.
- Arithmetic Instructions :-

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2’s complement)	NEG

- **Logical and Bit manipulation Instructions :-**
- Logical instructions perform binary operations on strings of bits stored in registers.
- They are useful for manipulating individual bits or a group of bits that represent binary-coded information.

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear Carry	CLRC
Set carry	SETC
Complement Carry	COMC
Enable interrupt	EI
Disable interrupt	DI

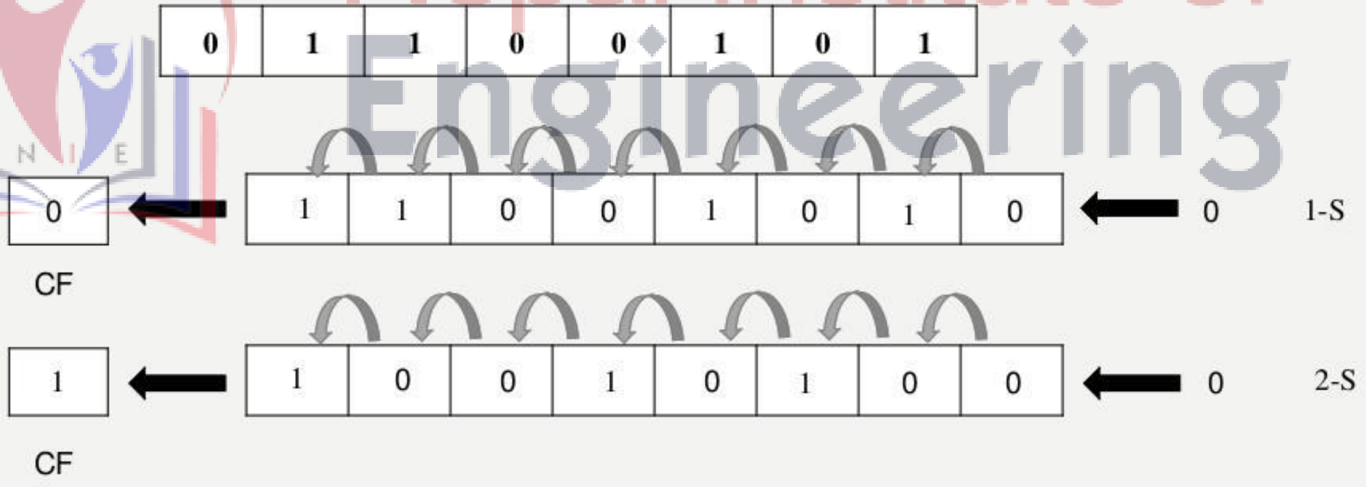
Shift Operations :-

- Shifts are operations in which the bits of a word are moved to the left or right.
- The bit shifted in at the end of the word determines the type of shift used.
- Shift instructions may specify either **logical shifts, arithmetic shifts, or rotate-type** operations.

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through Carry	RORC
Rotate left through Carry	ROLC

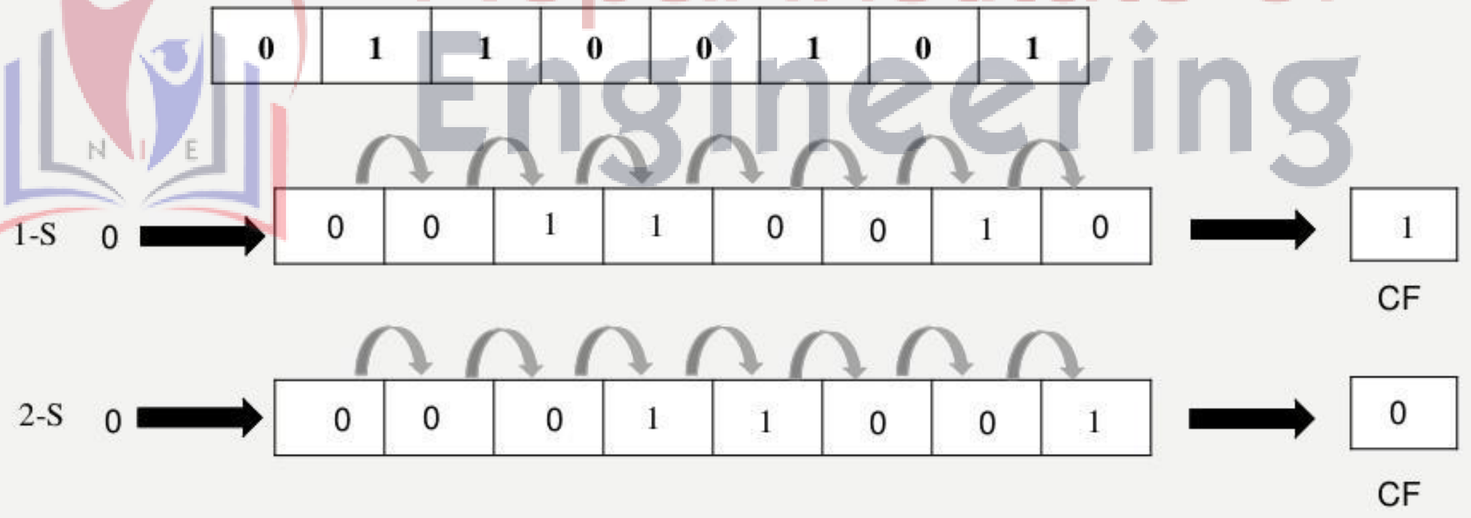
SHIFT INSTRUCTION: SHL AND SAL

- The SHL (shift left) instruction shifts the bits in the destination to the left.
- A 0 is shifted into the right most bit position and the msb is shifted into CF.



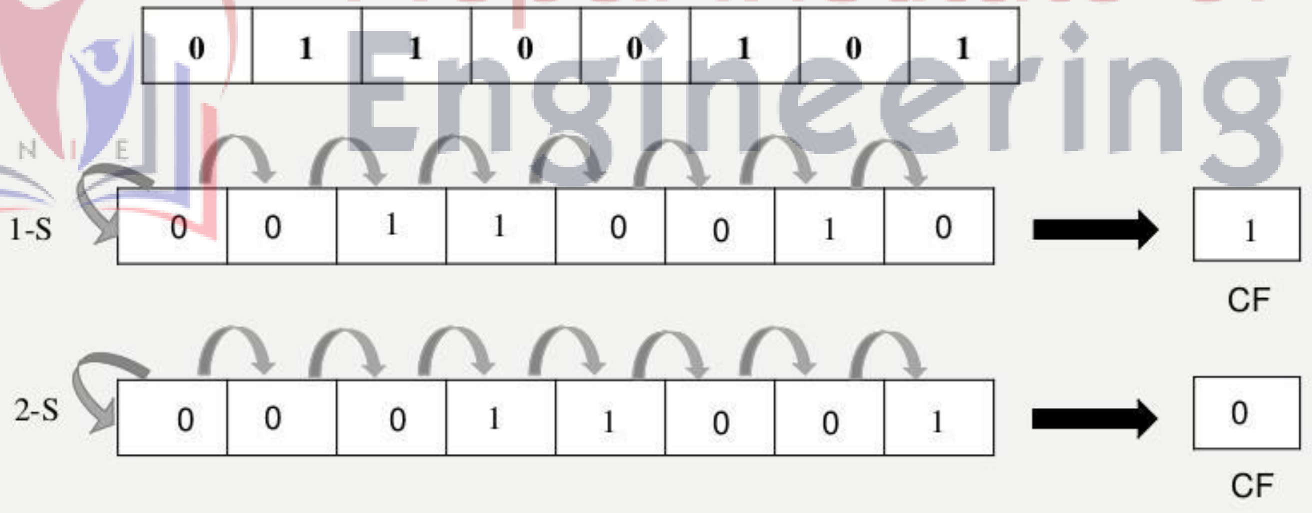
SHIFT INSTRUCTION: SHR

- The instruction SHR(shift right) performs right shift on the destination operand.
- A 0 is shifted into the msb position ,and the rightmost bit is shifted into CF.



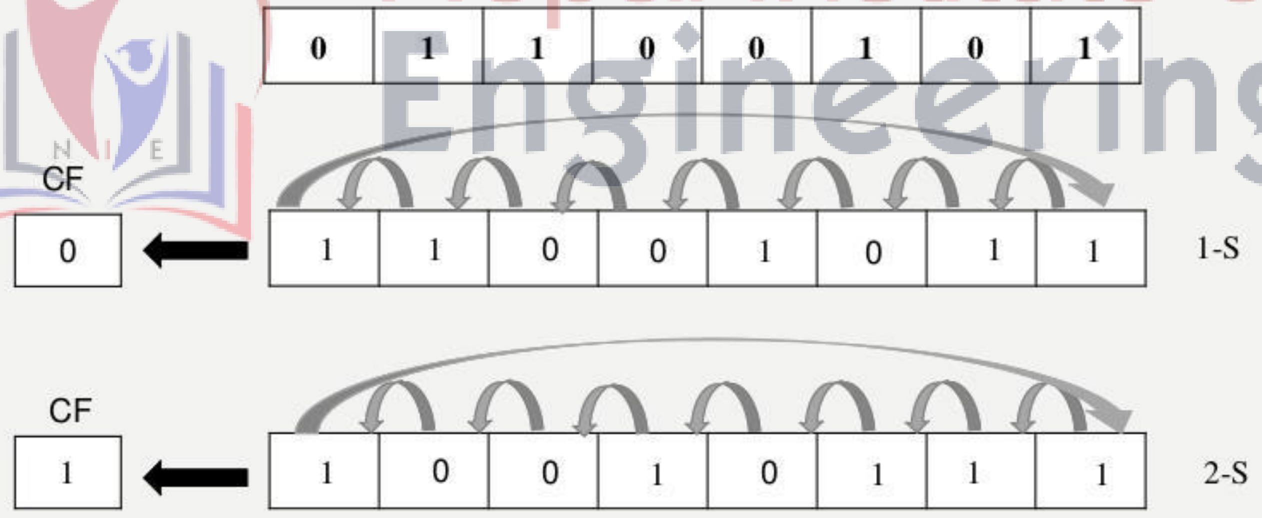
SHIFT INSTRUCTION: SAR

- The SAR instruction (shift arithmetic right) operates like SHR, with one difference: the msb retains its original value.



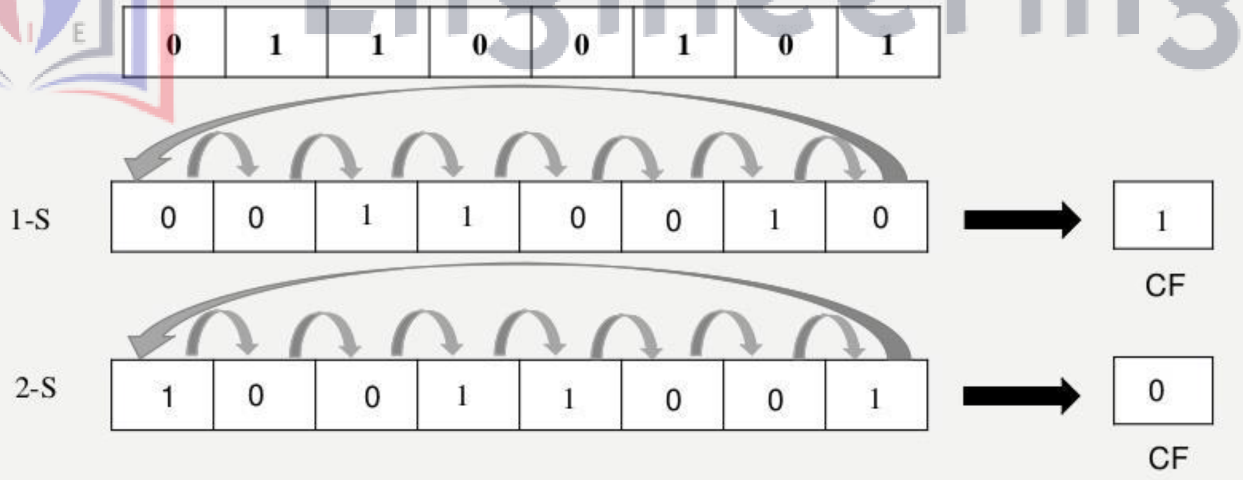
ROTATE INSTRUCTION: ROL

- The instruction ROL (rotate left)shifts bits to the left .The msb is shifted into the rightmost bit . The CF also gets the bit shifted out of the msb .



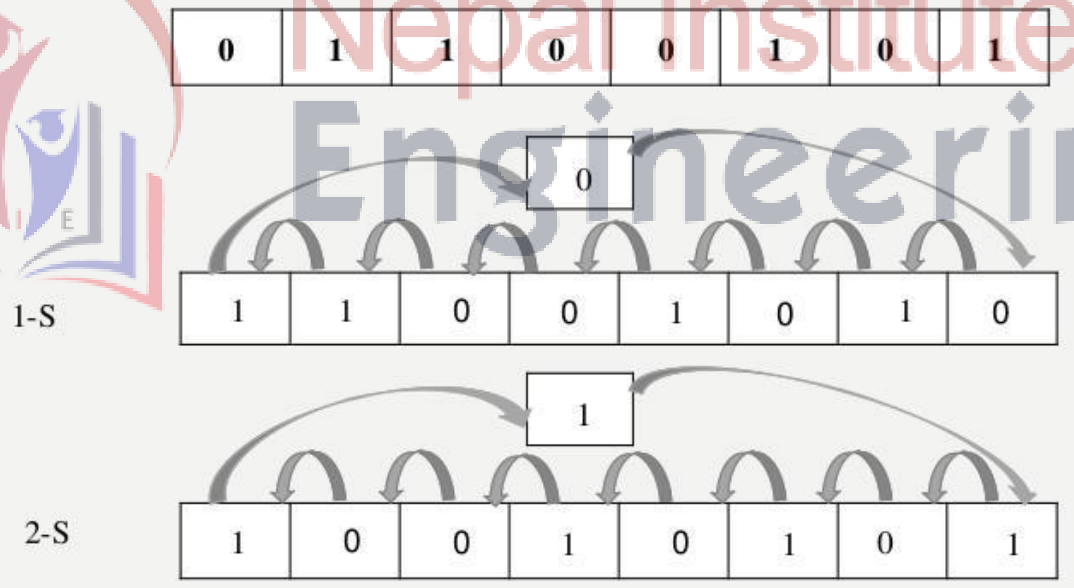
ROTATE INSTRUCTION: ROR

- The instruction ROR (rotate right) works just like ROL , except that the bits are rotate to the right .The rightmost bit is shifted into the msb , and also into The CF.



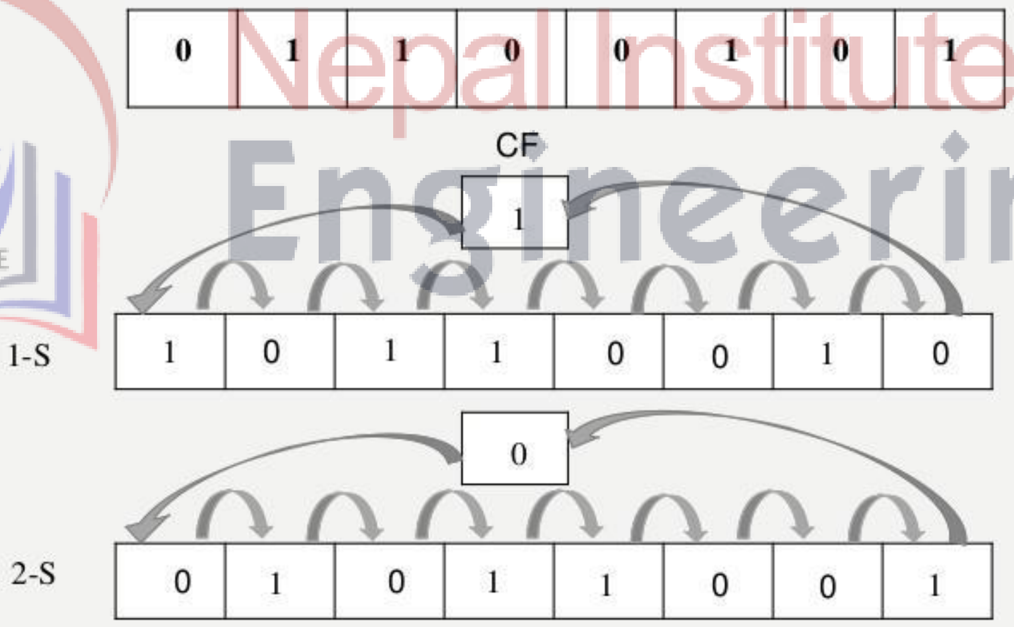
ROTATE INSTRUCTION: RCL

- The instruction **RCL** (Rotate through carry left) shifts the bits of the destination to the left. The msb is shifted into the CF, and the previous value of CF is shifted into the rightmost bit.



ROTATE INSTRUCTION: RCR

The instruction RCR(Rotate through carry right) works just like RCL, except that the bits are rotated to the right.



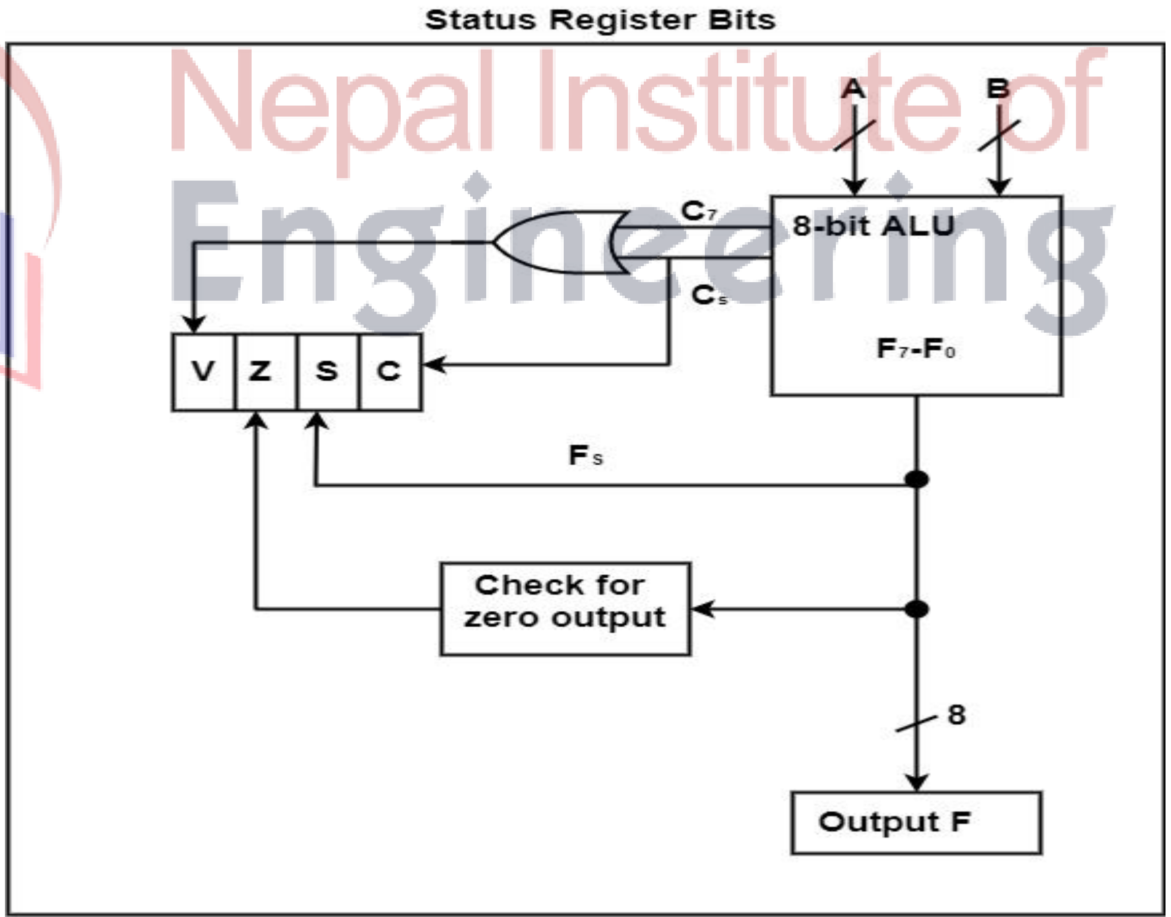
Program Control Instructions :-

- The program control instructions provide decision making capabilities and change the path taken by the program when executed in computer.
- These instructions specify conditions for altering the content of the program counter. The change in value of program counter as a result of execution of program control instruction causes a break in sequence of instruction execution.
- This is an important feature in **digital computers**, as it provides control over the flow of program execution and a capability for branching to different program segments.
- Some program control instructions are:

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by Anding)	TST

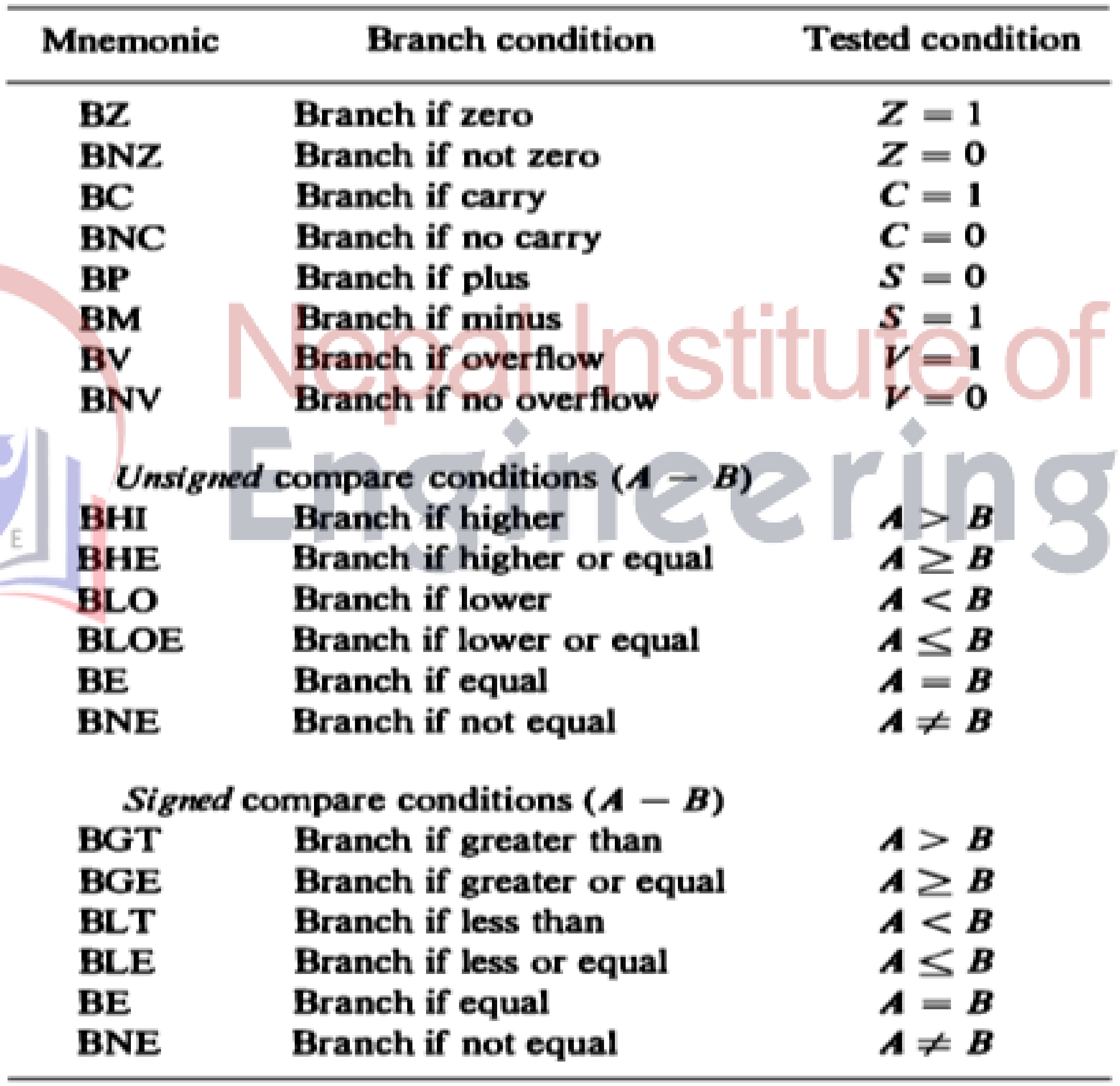
Status Bit Conditions :-

- **Status bits** are also called **condition-code bits** or **flag bits**.
- Figure below shows the block diagram of an 8-bit ALU with a 4-bit status register. The four status bits are symbolized by C, S, Z, and V.



- The bits are set or cleared as a result of an operation performed in the ALU.
- **Bit C (carry)** is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry is 0.
- **Bit S (sign)** is set to 1 if the highest-order bit F, is 1. It is set to 0 if the bit is 0.
- **Bit Z (zero)** is set to 1 if the output of the ALU contains all O's. !tis cleared to 0 otherwise. In other words, Z = 1 if the output is zero and Z = 0 if the output is not zero.
- **Bit V (overflow)** is set to 1 if the exclusive-OR of the last two carries is equal to 1, and cleared to 0 otherwise.

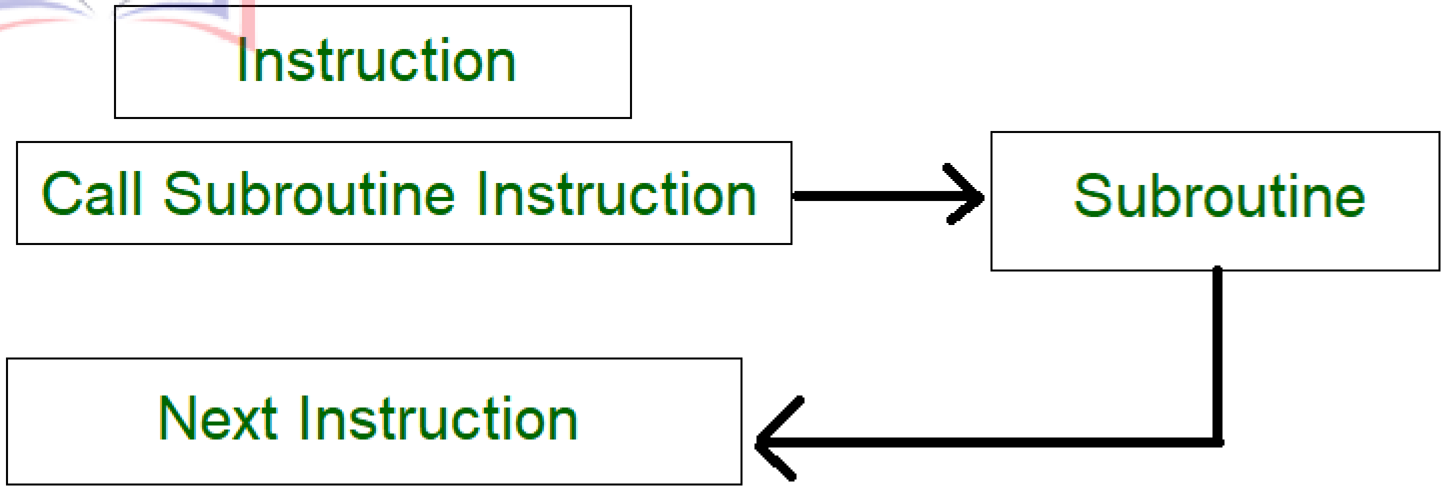
Conditional Branch Instructions :-



Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
Unsigned compare conditions ($A - B$)		
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
Signed compare conditions ($A - B$)		
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

Subroutine Call & Return :-

- A set of instructions that are used repeatedly in a program can be referred to as Subroutine.
- Only one copy of this Instruction is stored in the memory. When a Subroutine is required it can be called many times during the Execution of a particular program.
- A *call Subroutine Instruction* calls the Subroutine. Care Should be taken while returning a Subroutine as Subroutine can be called from a different place from the memory.



- A **call subroutine** instruction consists of an operation code together with an address that specifies the beginning of the subroutine. The instruction is executed by performing two operations:
    - ➢ the address of the next instruction available in the program counter (the return address) is stored in a temporary location so the subroutine knows where to return, and
    - ➢ control is transferred to the beginning of the subroutine.
- The last instruction of every subroutine, commonly called **return** from subroutine, transfers the **return address** from the temporary location into the program counter.
- This results in a transfer of program control to the instruction whose address was originally stored in the temporary location.

Reduced Instruction Set Format :-

- A computer uses fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often. It is classified as a reduced instruction set computer (**RISC**).
- RISC concept – an attempt to reduce the execution cycle by simplifying the instruction set
- Small set of instructions – mostly register to register operations and simple load/store operations for memory access
- Each operand – brought into register using load instruction, computations are done among data in registers and results transferred to memory using store instruction
- Simplify instruction set and encourages the optimization of register manipulation
- May include immediate operands, relative mode etc.

Characteristics of a RISC Architecture :-

- Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format
- Single-cycle instruction execution
- Hardwired rather than microprogrammed control

Complex Instructions Set Computer :-

- A computer with a large number of instructions is classified as a complex instruction set computer (CISC).
- One reason for the trend to provide a complex instruction set is the desire to simplify the compilation and improve the overall computer performance.
- The essential goal of CISC architecture is to attempt to provide a single machine instruction for each statement that is written in a high-level language.
- **Major Characteristics of CISC Computer :-**
- A large number of instructions-typically from 100 to 250 instructions.
- Some instructions that perform specialized tasks and are used infrequently.
- A large variety of addressing modes-typically from 5 to 20 different modes.
- Variable-length instruction formats.
- A relatively large number of registers in the processor unit.