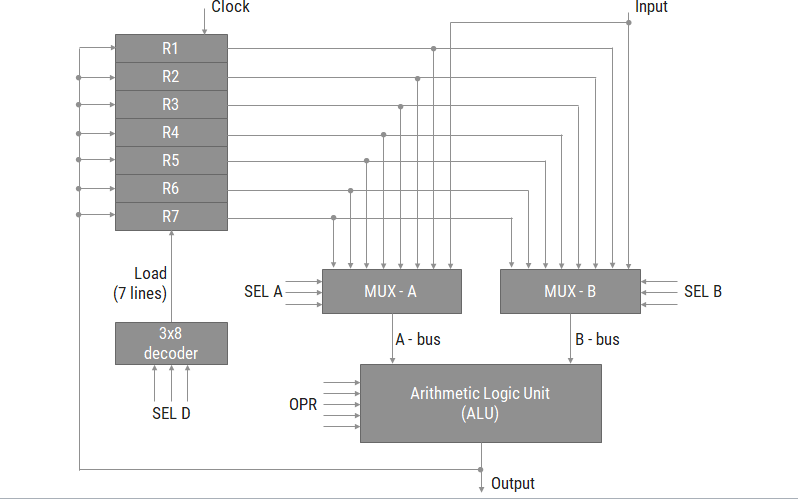
**UNIT-III**

**CENTRAL PROCESSING UNIT**

Contents:

* General registers Organization – Control Word, Examples of micro operation,
* Stack Organization – Register stack, Memory stack, Reverse Polish Notation,
* Evaluation of Arithmetic Expressions
* Instruction Formats – Three Address Instructions, Two Address Instructions
* One Address Instructions, Zero Address Instructions,
* Addressing Modes-Types
* Data Transfer Instructions,
* Data Manipulation Instructions
* Program Control – Conditional Branch instructions, Subroutine Call and Return.

**General Register Organization**

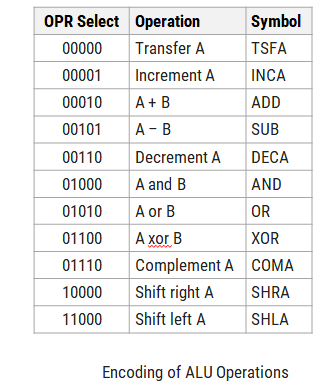
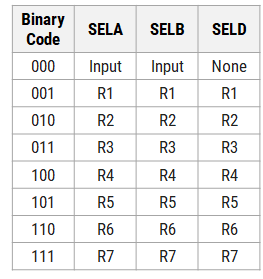


* Example: *R1 🡪 R2 + R3*
* To perform the above operation, the control must provide binary selection variables to the following selector inputs:

1. MUX A selector (SELA): to place the content of R2 into bus A.
2. MUX B selector (SELB): to place the content of R3 into bus B.
3. ALU operation selector (OPR): to provide the arithmetic addition A + B.
4. Decoder destination selector (SELD): to transfer the content of the output bus into R1.

* *Control Word:*

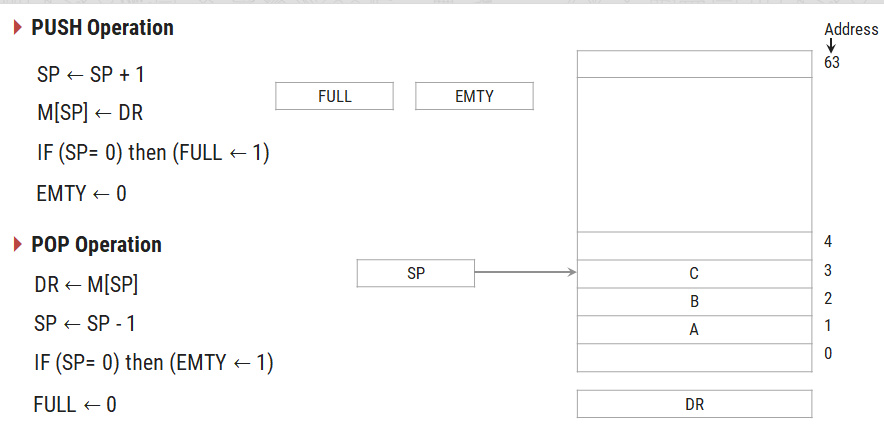




**Stack Organization**

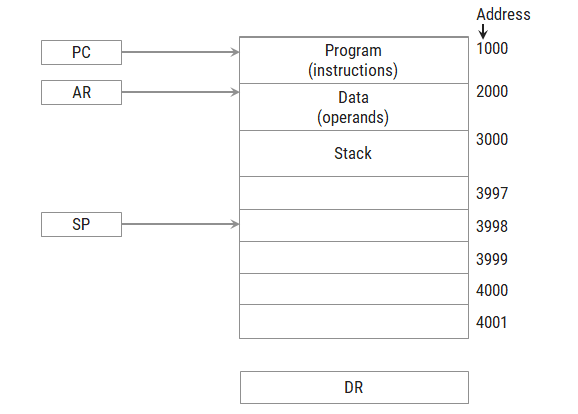
* A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved (LIFO).
* The register that holds the address for the stack is called a *stack pointer (SP)* because its value always points at the top item in the stack.
* The physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted or deleted.
* There are two types of stack organization
  1. Register stack – built using registers
  2. Memory stack – logical part of memory allocated as stack

**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.
* In a 64-word stack, the stack pointer contains 6 bits because 26 = 64.
* Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary).
* The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items.
* DR is the data register that holds the binary data to be written into or read out of the stack**.**

**Memory Stack**

****

PUSH Operation

SP ← SP - 1

M[SP] ← DR

Pop operation

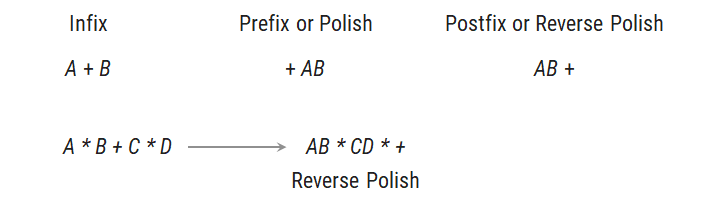
DR ← M[SP]

SP ← SP + 1

* 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.
* Figure shows a portion of computer memory partitioned into three segments: program, data, and stack.
* The program counter PC points at the address of the next instruction in the program which is used during the fetch phase to read an instruction.
* The address registers AR points at an array of data which is used during the execute phase to read an operand.
* The stack pointer SP points at the top of the stack which is used to push or pop items into or from the stack.
* We assume that the items in the stack communicate with a data register DR.

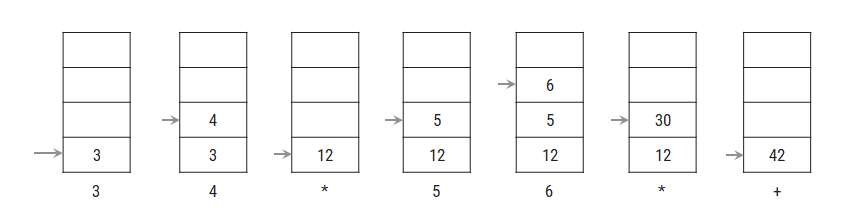
**Reverse Polish Notation**

* The common mathematical method of writing arithmetic expressions imposes difficulties when evaluated by a computer.
* The Polish mathematician Lukasiewicz showed that arithmetic expressions can be represented in prefix notation as well as postfix notation



**Evaluation of Arithmetic Expression**

(3 \* 4) + (5 \* 6) ----🡪3 4 \* 5 6 \* + ---🡪 42



**Instruction format**

* Instruction Fields

OP-code field - specifies the operation to be performed

Address field - designates memory address(s) or a processor register(s)

Mode field - specifies the way the operand or the effective address is determined

* **The number of address fields in the instruction format depends on the internal organization of CPU**
* - The three most common CPU organizations:

**Single accumulator organization:**

ADD X /\* AC ←AC + M[X] \*/

**General register organization:**

ADD R1, R2, R3 /\* R1 ← R2 + R3 \*/

ADD R1, R2 /\* R1 ←R1 + R2 \*/

MOV R1, R2 /\* R1 ←R2 \*/

ADD R1, X /\* R1 ←R1 + M[X] \*/

**Stack organization:**

PUSH X /\* TOS ← M[X] \*/

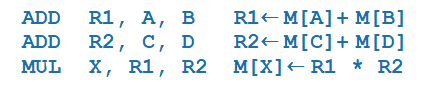
ADD

Instructions are categorized into different formats with respect to the operand fields in the instructions.

1. Three Address Instructions
2. Two Address Instruction
3. One Address Instruction
4. Zero Address Instruction

**Three Address Instruction**

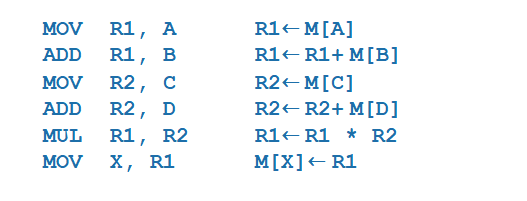
* Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.
* The program in assembly language that evaluates X = (A + B) \* (C + D) is shown below.



* The advantage of 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.

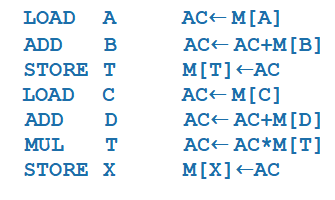
**Two Address Instruction**

* 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.
* The program to evaluate X = (A + B) \* (C + D) is as follows:



**One Address Instruction**

* One address instructions use an implied accumulator (AC) register for all data manipulation.
* For multiplication and division these is a need for a second register.
* However, here we will neglect the second register and assume that the AC contains the result of all operations.
* The program to evaluate X = (A + B) \* (C + D) is



**Addressing Modes**

* The way of choosing operands during program execution is dependent on addressing modes of the instruction.
* Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:
  1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
  2. To reduce the number of bits in the addressing field of the instruction.

There are basic 10 addressing modes supported by the computer

1. Implied Mode
2. Immediate Mode
3. Register Mode
4. Register Indirect Mode
5. Autoincrement or Autodecrement Mode
6. Direct Address Mode
7. Indirect Address Mode
8. Relative Address Mode
9. Indexed Addressing Mode
10. Base Register Addressing Mode

**Implied Mode**

* Operands are specified *implicitly* in the definition of the instruction.
* For example, the instruction “complement accumulator (CMA)” is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.
* In fact, all register reference instructions that use an accumulator and zero address instructions are implied mode instructions

**Immediate Mode**

* Operand is specified in the instruction itself.
* In other words, an immediate-mode instruction has an operand field rather than an address field.
* The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
* Immediate mode of instructions is useful for initializing register to constant value.
* E.g. MOV R1, 05H
* instruction copies immediate number 05H to R1 register.

**Register Mode**

* Operands are in registers that reside within the CPU.
* The particular register is selected from a register field in the instruction.
* E.g. MOV AX, BX

move value from BX to AX register

**Register Indirect Mode**

* In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
* Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.
* The advantage of this mode is that address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.
* E.g. MOV [R1], R2
* value of R2 is moved to the memory location specified in R1**.**

**Autoincrement or Autodecrement Mode**

* This is similar to the register indirect mode expect that the register is incremented or decremented after (or before) its value is used to access memory.
* When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction**.**

**Direct Address Mode**

* In this mode the effective address is equal to the address part of the instruction.
* The operand resides in memory and its address is given directly by the address field of the instruction.
* E.g. ADD 457

**Indirect Address Mode**

* In this mode the address field of the instruction gives the address where the effective address is stored in memory.
* Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

**Relative Address 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.
* The address part of the instruction is usually a signed number which can be either positive or negative.
* **Effective address = address part of instruction + content of PC**

**Indexed Addressing Mode**

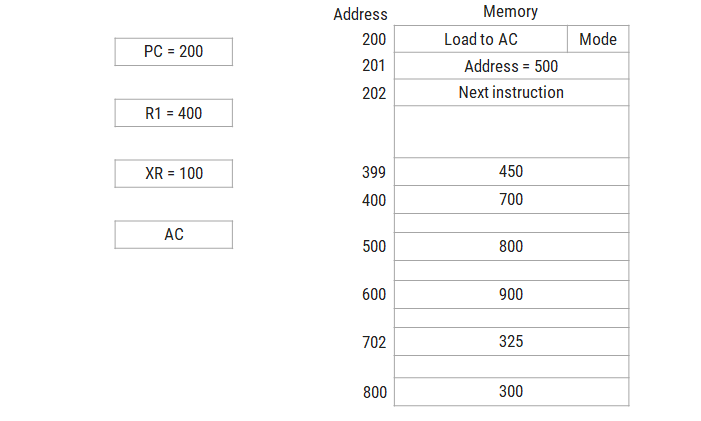
* In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
* The indexed register is a special CPU register that contain an index value.
* The address field of the instruction defines the beginning address of a data array in memory.
* Each operand in the array is stored in memory relative to the beginning address.

**Effective address = address part of instruction + content of index register**

**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.
* A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.
* The base register addressing mode is used in computers to facilitate the relocation of programs in memory.
* **Effective address = address part of instruction + content of base register**

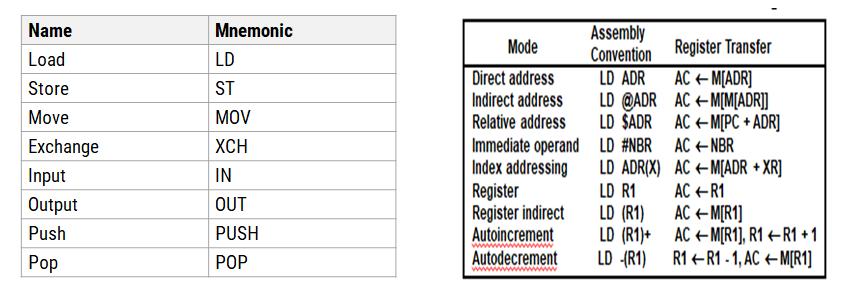
**Addressing Modes (Example)**

****

**Data transfer and manipulation**

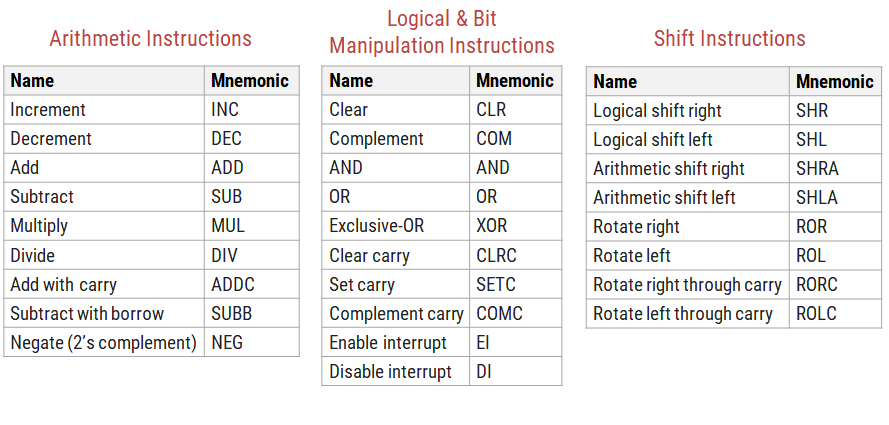
**Data transfer instructions**

* Data transfer instructions move data from one place in the computer to another without changing the data content.
* The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.

****

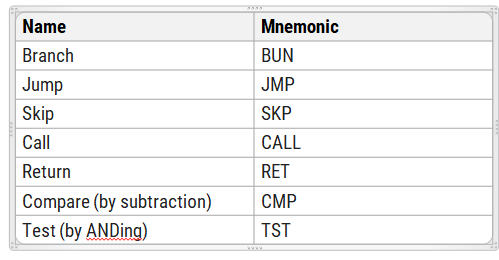
**Data Manipulation Instructions**

* The data manipulation instructions in a typical computer are usually divided into three basic types:
  1. Arithmetic instructions
  2. Logical and bit manipulation instructions
  3. Shift instructions

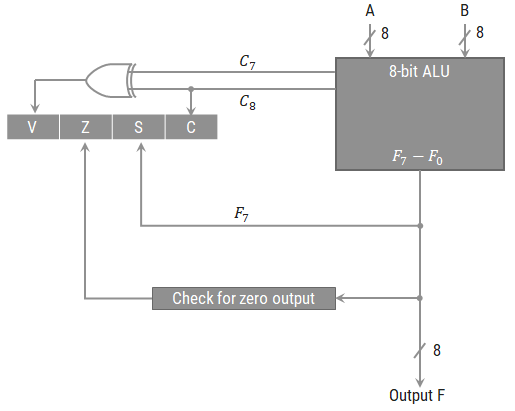


**Program Control**

* A program control type of instruction, when executed, may change the address value in the program counter and cause the flow of control to be altered.
* The change in value of the program counter as a result of the execution of a program control instruction causes a break in the sequence of instruction execution.

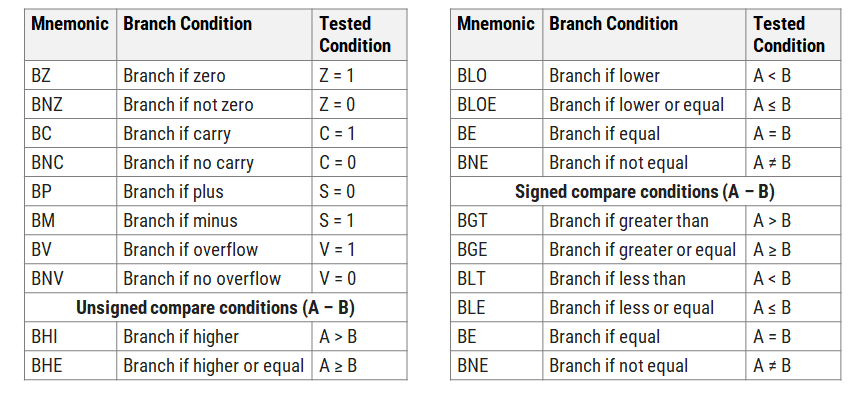


**Status Bit Conditions**



* Bit C (carry) is set to 1 if the end carry *C*8 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*7 is 1. It is set to 0 if the bit is 0.
* Bit Z (zero) is set to 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. This is the condition for an overflow when negative numbers are in 2’s complement.

**Conditional Branch Instructions**

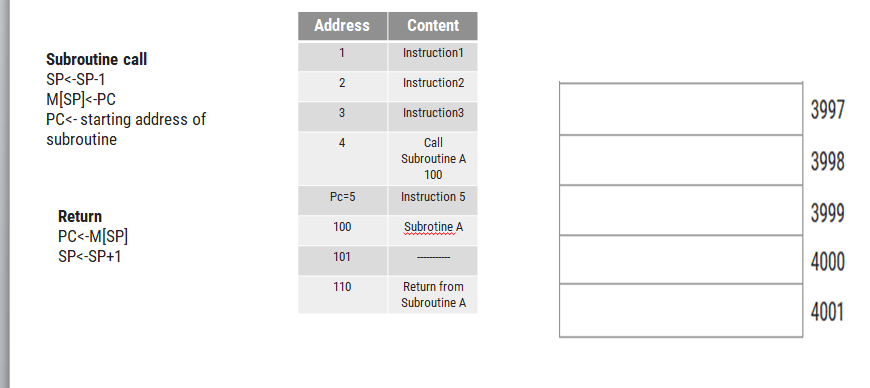
****

**Subroutine call and return**

* A subroutine(function) is a sub program in the main program that performs a specific task
* During the execution of the main program a subroutine may be called at various points in the program
* Each time a subroutine is called, a branch is executed to the beginning of the subroutine to start executing its set of instructions. After the subroutine has been executed, a branch is made back to the main program.
* The instruction is executed by performing two operations:

1. the address of the next instruction available in the program counter (the return address) is stored in memory stack so the subroutine knows where to return, and
2. 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.

****

**Exercise**

* A bus-organized CPU similar to general register organization has 16 registers with 32 bits In each , an ALU and a destination decoder
* How many multiplexers are there in the A bus, and what is the size of each multiplexer?

1. Each register has 32 bits i.e., 32 multiplexers.

No.of registers is 16 and size of multiplexer is 16x1

How many selection Inputs are needed for MUX A and MUX B?

. How many inputs and outputs are there In the decoder?

. How many inputs and outputs are there in the ALU for data, lnduding input and output carries?

1. Two inputs-32bits + 32bits + 1 --65 inputs Output –32+1 = 33 data output lines

Formulate a control word for the system assuming thai the ALU has 35 operations

SELA+SELB+SELD+OPR(35 operations-26)=4+4+4+6=18 bits

* 2) An instruction is stored at location 300 with its address field at location 301. The address field has the value 400. The processor register R1 contains the number 200. Evaluate the effective address if the addressing mode of the instruction is a)direct b)immediate c)relative d)register indirect e)index with R1 as the index register.

3) Consider the two 8-bit numbers A =01000001 and B =10000100.

* Give the decimal equivalent of each number assuming that (1) they are unsigned, and (2) they are signed.
* Add the two binary numbers and interpret the sum assuming that the numbers are (1) unsigned, and (2) signed.
* Determine the values of the C, Z, S, and V status bits after the addition.
* List the conditional branch instructions from Table 8-11 that will have a true condition.

4) 