ii. Loop Program:

In computer programming, a loop is a sequence of instructions that is continually repeated until a certain condition is reached. Typically, a certain process is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number. If it hasn't, the next instruction in the sequence is an instruction to return to the first instruction in the sequence and repeat the sequence. If the condition has been reached, the next instruction 'falls through' to the next sequential instruction or branches outside the loop. A loop is a fundamental programming idea that is commonly used in writing programs.

There are two types of loop programming structure. These are:

i. Descending Loop:

This loop starts with the ***final*** value of the counter then ***decremented*** by one and check whether the counter equal zero or not. If it is not zero then, return and continue the program. If it is reached zero, it will end the loop and continue the program.

Ex. From C++ a descending loop is like the following:

```
for (i=10; i > 0; i--)
```

In assembly language ***in general***, the **CX** and **CL** registers are used for the counter instead of the variables in the loop programs.

```
MOV CL,0Ah          or    MOV CX, 000Ah
.
*
.
.
DEC CL
JNZ *
.
(CONTINUE PROGRAM)
.
HLT
```

ii. Ascending Loop:

This loop starts with the ***least*** value of the counter then ***incremented*** by one and ***copamre*** whether the counter reaches its final value or not. If it is not reached then, return to the loop program and continue. If it is equal to the final value then, end the loop and continue the program.

Ex. From C++ a descending loop is like the following:

for (i=1; i ≤ 15; i++)

Again, in assembly language ***in general***, the **CX** and **CL** registers are used for the counter instead of the variables in the loop programs.

```
MOV CL,01h          or    MOV CX, 0001h
.
*
.
.
INC CL
CMP CL, 0Fh
JNZ *
.
(CONTINUE PROGRAM)
.
HLT
```

Ex. 4: Write an A.L.P. to find the average of a student of 8 degrees. The degrees stored in an M.L. starts at [5000h].

Sol.

```
MOV CL, 08h          ; Initialize the counter.
MOV AL, 00h          ; The summation of the degrees must equal zero.
MOV AH, 00h
MOV DI, 5000         ; Put the value of the 1st address in (DI or SI or BX)
                       i.e. use (DI, SI, and BX) as a pointer to the memory.
*: ADC AL, BYTEPTR[DI]    ; Summation for the degrees.
DEC CL
JNZ *
```

MOV BL, 08h
DIV BL ; calculate the average by dividing the summation
HLT which is now in (AX) by number of degrees (8)
degrees

Ex. 5: Write an A.L.P. to move a block of **12** bytes of data starting at offset address [3500h] to another block starting at offset address [7080h]. Assume that both blocks are in the same segment whose starting address is 1234h.

Sol.

MOV AX,1234h
MOV DS, AX
MOV SI, 3500h
MOV DI, 7080h
MOV CL, 0Ch
#: MOV DL, [SI]
MOV [DI], DL
INC SI
INC DI
DEC CL
JNZ #
HLT

H.W. 1: Write an A.L.P. that multiplicities a block of 20 bytes (vector of M.L.s) of data starts at an offset address [4000h] with another vector of M.L.s starts at an offset address of [8858h] then put the results in a third vector starts [8000h]. Assume that all of the three vectors are in the same segment that's its start address is DATASEGM.

Subroutines:

Subroutine is a sub program needed to perform a particular sub-task many times on different data values. Instead of putting this program in the main program, subroutine can be stored in a specified M.L. of the memory and called it using a **CALL** instruction. Whenever this sub-program needed to be executed, there is no need to write this program again and again, just call it and execute it.

When the program called a subroutine, executed it then, it should return to the main program. This operation can be done by putting the **RET** instruction at the end of the subroutine.

CALL Instruction:

The CALL instruction is a special branch instruction that performs the following operations:

- Push the contents of the next instruction address (IP) on the top of the stack.
- Update the stack pointer (SP).
- Branch to the target address specified by the CALL instruction.

Important Note: The CALL instruction is similar to the JMP instruction, that it is intrasegment and intersegment. The general format of the CALL instruction is shown in the following table:

Mnemonics	Meaning	Format	Operation	Flags affected
CALL	Subroutine Call	CALL Operand	Execution continues from the address of the subroutine specified by the operand.	None

Operand may be one of the: Near, Far, REG16, Mem16, Mem32).

Near Call: Execution of a near CALL causes the contents of the IP to be saved in the stack, and a new 16-bit value which corresponds to the address of the first subroutine instruction to be into IP register.

e.g. **CALL** 1453h

Far CALL: Execution of Far CALL causes the contents of IP and CS registers to be stored in the stack respectively and a new 32-bit value for code segment and offset to be loaded into IP and CS registers.

e.g. **CALL** 1781:2712h

Indirect CALL: (Reg16, Mem16, Mem32).

e.g.s **CALL** BX

CALL [BX]

RET Instruction:

The Return instruction (RET) is a special branch instruction that performs the following operations:

- Pop the return address from the top of the stack.
- Update the stack pointer (SP).

The RET instruction removes the 16-bit number (near return) from the stack and places it into IP, or removes a 32-bit number. (far return) and places it into IP and CS.

Subroutine Structure:

Main program

.

.

CALL SUB

Continue program

HLT

SUB.

Program instructions

.

.

RET

Ex.6: repeat Ex.5 using subroutine.

Sol.

MOV AX,1234h

MOV DS, AX

MOV SI, 3500h

MOV DI, 7080h

MOV CL, 0Ch

#:CALL Copy

INC SI

INC DI

DEC CL

JNZ #

HLT

Copy: MOV DL, [SI]

MOV [DI], DL

RET

H.W. 2: Repeat H.W. 1 using subroutine.