7/1/2022 8086 Microprocessor.

② MOVSW (Move String Word)

→ Moves a string, one word (16-bit) at a time from source memory DS:SI to memory destination ES:DI.

→ Value of SI register and DI register is incremented by 2 based on the value of direction flag (DF)

if DF=0 then SI = SI+2 ; DI = DI+2

if DF=1 then SI = SI-2 ; DI = DI-2.

→ This instruction has no operand.

→ The number of string words to moved is loaded in CX register.

For eg:

```
MOV AX, 4000H
MOV DS, AX
MOV AX, 6000H
MOV ES, AX
MOV CX, 0005H   <-- length of string.
MOV SI, 0002H
MOV DI, 0100H
CLD
REP MOVSW
HLT.
```

③ STOSB (Store String byte)

↳ This instruction moves a string byte at a time from AL register to destination memory address ES:DI.

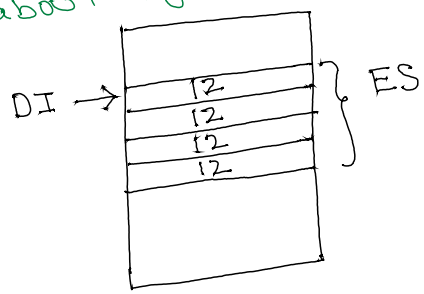
↳ DI is then incremented or decremented by 1 based on Direction flag (DF).

↳ if DF=0 then DI = DI + 1

if DF=1 then DI = DI − 1

For eg:

```
MOV CX, 0004H   ← counter which tells about length of string.
MOV AX, 6000H
MOV ES, AX
MOV DI, 0010H
MOV AL, 12H
CLD
REP STOSB
```



④ STOSW (Store String Word)

↳ This instruction moves a string one word at a time from AX register (Source) to destination memory address ES:DI

↳ DI is then incremented or decremented by 2 based on Direction flag (DF).

↳ if DF=0 then DI = DI + 2

if DF=1 then DI = DI − 2

↳ CX register will hold the length of string

⑤ LODSB (Load String byte)

↳ This instruction moves a string one byte at a time from source memory DS:SI to AL register. → destination

↳ SI is then incremented or decremented by 1 based on direction flag (DF).

⑥ LODSW (Load String Word)

↳ This instruction moves a string one word at a time from source memory DS:SI to AX register. → destination.

↳ SI is then incremented or decremented by 2 based on DF

⑦ CMPSB (Compare String byte)

↳ The byte or 8-bit data at DS:SI is compared with the byte or 8-bit data at ES:DI and the flags are set accordingly.

↳ Both SI and DI are incremented or decremented based on the value of DF.

⑧ CMPSW (Compare String Word)

↳ The word or 16-bit data at DS:SI is compared with the word or 16-bit data at ES:DI and the flags are set accordingly.

↳ Both SI and DI are incremented by 2 or decremented by 2 based on the value of DF.

REPEAT Instruction

The string instructions are used to operate on large blocks of data. To refer a string two parameters are required :-

(i) Starting and End address of the string

(ii) length of string.

The starting address is generally refered by DS:SI while the ending address is pointed by ES:DI.

The Length of string is stored as count in the CX register.

After each iteration, the incrementing and decrementing of SI and DI depends upon the direction flag (DF), the counter is decremented by one. (CX = CX-1)

→ To perform string instruction repeatedly REP (repeat) instruction is used as prefix with the string instruction.

→ Hence, the string instruction with the REP prefix is executed again and again till the CX register becomes Zero. If CX becomes Zero the execution proceeds to the next instruction in sequence.

For eg: REP MOVSW

↑ prefix

REP / REPE / REPZ (Repeat String instruction)

This prefix forces a string operation to be repeated as long as CX is not equal to 0. CX is decremented once for each repetition.

REPNE / REPNZ (Repeat string while not Zero)

This prefix forces a string operation to be repeated till $CX \neq 0$ and $ZF \neq 0$

($CX \neq 0$ & $ZF \neq 0$)

Processor Control Instructions

→ CLC (clear carry flag)

$CF \leftarrow 0$

→ CMC (complement carry flag)

$CF \leftarrow \overline{CF}$

→ STC (set carry flag)

$CF \leftarrow 1$

→ CLD (clear direction flag)

$DF \leftarrow 0$

→ STD (Set direction flag)

$DF \leftarrow 1$

→ CLI (clear interrupt flag)

IF ← 0

→ STI (set interrupt flag)

IF ← 1

→* HLT (Halt instruction)

HLT instruction is used to ask the μp to stop execution. The μp goes into infinite series of NOP instruction until some external interrupt occurs.

→* NOP (No operation)

When NOP instruction is executed, this instruction doesnot allow the processor to perform any operation for one instruction cycle. only the value of IP is incremented by one to point to next instruction.

ASSEMBLER DIRECTIVES & 8086 ASSEMBLY LANGUAGE PROGRAMMING

* An assembler is a program which converts an assembly language program into the equivalent machine code.
* Assembler directives are predefined alphabetical strings which helps the assembler to understand the assembly-language programs properly and generate the machine codes.
* Each 8086 assembly-language program will consists of :-
  1. Assembler Directives
  2. 8086 Instructions.
* The 8086 Instructions are translated to machine codes whereas directives are not translated to machine codes. Hence, directives are often refered as pseudo-codes.
* Pseudo-codes or pseudo-instructions or assembler directives are statements for the assembler to carry out the assembly process i.e. generation of machine codes.
* Using directives, the programmer mention constants, variables, logical names of segments, types of different modules, and end of file for a assembly language program.

The following directives are commonly used in 8086 assembly-language programs:

→ DB : Define byte

→ DW : Define word

→ ASSUME : Assume logical Segment Name

→ END : End of program

→ ENDP : End of Procedure

→ ENDS : End of Segment

→ EQU : Equate

→ ORG : Origin

→ SEG : Segment of a Label

→ PROC : Procedure

→ LABEL : Label

→ SEGMENT : Indicates start of a logical Segment

8086 ASSEMBLY LANGUAGE PROGRAM EXAMPLE

- .asm is extension for assembly language program

Program1.asm

```
ASSUME CS:CODE DS:DATA
DATA   SEGMENT
       DATA1 DW 3567H
       DATA2 DW 1235H
DATA   ENDS
CODE   SEGMENT
LABEL  MOV AX, DATA
       MOV DS, AX
       MOV AX, DATA1
       MOV BX, DATA2
       ADD AX, BX
       HLT
CODE   ENDS
       END LABEL
```

CS → CODE
DS → DATA

Data Segment Start

Data segment.

Data segment end

Code segment start.

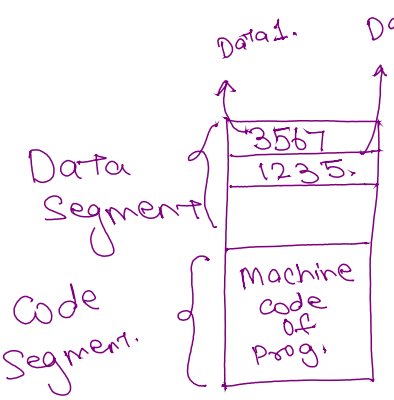
part of code Segment.

code segment end.

DATA1 → name given to a constant value 3567H

DATA2 → name given to a constant value 1235H.

Assembler.



For 8086 we will use.

MASM assembler.

DOS based assembler