UNIT – VI

INSTRUCTION SET AND ASSEMBLY LANGUAGE PROGRAMMING OF 8086

Addressing Modes of 8086:

Addressing mode indicates a way of locating data or operands. Depending up on the data type used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes or same instruction may not belong to any of the addressing modes.

The addressing mode describes the types of operands and the way they are accessed for executing an instruction. According to the flow of instruction execution, the instructions may be categorized as

1. Sequential control flow instructions and
2. Control transfer instructions.

Sequential control flow instructions are the instructions which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program. For example the arithmetic, logic, data transfer and processor control instructions are Sequential control flow instructions.

The control transfer instructions on the other hand transfer control to some predefined address or the address somehow specified in the instruction, after their execution. For example INT, CALL, RET & JUMP instructions fall under this category.

The addressing modes for Sequential and control flow instructions are explained as follows.

1. Immediate addressing mode:

In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

Example: MOV AX, 0005H.

In the above example, 0005H is the immediate data. The immediate data may be 8- bit or 16-bit in size.

2. Direct addressing mode:

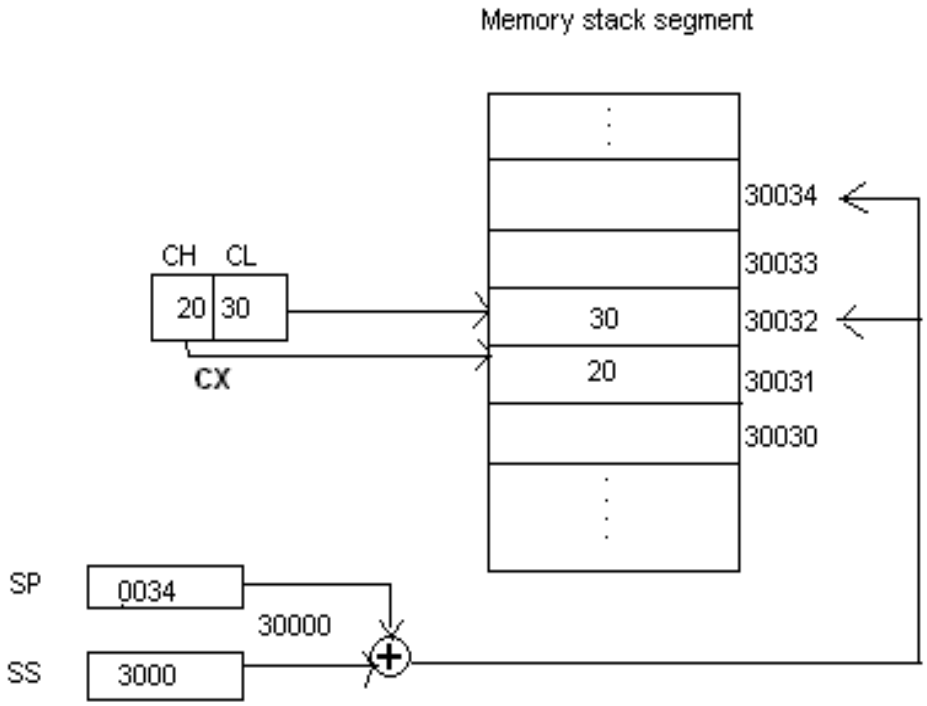
In the direct addressing mode, a 16-bit memory address (offset) directly specified in the instruction as a part of it.

2. PUSH instruction

The PUSH instruction decrements the stack pointer by two and copies the word from source to the location where stack pointer now points. Here the source must of word size data. Source can be a general purpose register, segment register or a memory location.

The PUSH instruction first pushes the most significant byte to sp-1, then the least significant to the sp-2.

Push instruction does not affect any flags.



Example:-

PUSH CX ; Decrements SP by 2, copy content of CX to the stack (figure shows execution of this instruction)

PUSH DS ; Decrement SP by 2 and copy DS to stack

3. POP instruction

The POP instruction copies a word from the stack location pointed by the stack pointer to the destination. The destination can be a General purpose register, a segment register or a memory location. Here after the content is copied the stack pointer is automatically incremented by two.

The execution pattern is similar to that of the PUSH instruction.

Example:

POP CX ; Copy a word from the top of the stack to CX and increment SP by 2.

4. IN & OUT instructions

The IN instruction will copy data from a port to the accumulator. If 8 bit is read the data will go to AL and if 16 bit then to AX. Similarly OUT instruction is used to copy data from accumulator to an output port.

Both IN and OUT instructions can be done using direct and indirect addressing modes.

Example:

IN AL, 0F8H	;	Copy a byte from the port 0F8H to AL
MOV DX, 30F8H	;	Copy port address in DX
IN AL, DX	;	Move 8 bit data from 30F8H port
IN AX, DX	;	Move 16 bit data from 30F8H port
OUT 047H, AL	;	Copy contents of AL to 8 bit port 047H
MOV DX, 30F8H	;	Copy port address in DX
OUT DX, AL	;	Move 8 bit data to the 30F8H port
OUT DX, AX	;	Move 16 bit data to the 30F8H port

5. XCHG instruction

The XCHG instruction exchanges contents of the destination and source. Here destination and source can be register and register or register and memory location, but XCHG cannot interchange the value of 2 memory locations.

General Format

XCHG Destination, Source

Example:

XCHG BX, CX	;	exchange word in CX with the word in BX
XCHG AL, CL	;	exchange byte in CL with the byte in AL
XCHG AX, SUM[BX]	;	here physical address, which is DS+SUM+[BX]. The content at physical address and the content of AX are interchanged.

Arithmetic and Logic instructions

The arithmetic and logic logical group of instruction include,

1. **ADD instruction**

Add instruction is used to add the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected

General Format:

ADD Destination, Source

Example:

- ADD AL, 0FH ; Add the immediate content, 0FH to the content of AL and store the result in AL
- ADD AX, BX ; AX <= AX+BX
- ADD AX,0100H – IMMEDIATE
- ADD AX,BX – REGISTER
- ADD AX,[SI] – REGISTER INDIRECT OR INDEXED
- ADD AX, [5000H] – DIRECT
- ADD [5000H], 0100H – IMMEDIATE
- ADD 0100H – DESTINATION AX (IMPLICT)

2. **ADC: ADD WITH CARRY**

This instruction performs the same operation as ADD instruction, but adds the carry flag bit (which may be set as a result of the previous calculation) to the result. All the condition code flags are affected by this instruction. The examples of this instruction along with the modes are as follows:

Example:

- ADC AX,BX – REGISTER
- ADC AX,[SI] – REGISTER INDIRECT OR INDEXED
- ADC AX, [5000H] – DIRECT
- ADC [5000H], 0100H – IMMEDIATE
- ADC 0100H – IMMEDIATE (AX IMPLICT)

3. **SUB instruction**

SUB instruction is used to subtract the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected

General Format:

SUB Destination, Source

Example:

- SUB AL, 0FH ; subtract the immediate content, 0FH from the content of AL and store the result in AL
- SUB AX, BX ; AX <= AX-BX
- SUB AX,0100H – IMMEDIATE (DESTINATION AX)
- SUB AX,BX – REGISTER
- SUB AX,[5000H] – DIRECT
- SUB [5000H], 0100H – IMMEDIATE

4. SBB: SUBTRACT WITH BORROW

The subtract with borrow instruction subtracts the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand. Subtraction with borrow, here means subtracting 1 from the subtraction obtained by SUB, if carry (borrow) flag is set.

The result is stored in the destination operand. All the flags are affected (condition code) by this instruction. The examples of this instruction are as follows:

Example:

- SBB AX,0100H – IMMEDIATE (DESTINATION AX)
- SBB AX,BX – REGISTER
- SBB AX,[5000H] – DIRECT
- SBB [5000H], 0100H – IMMEDIATE

5. CMP: COMPARE

The instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location. For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere. The flags are affected depending upon the result of the subtraction. If both of the operands are equal, zero flag is set. If the source operand is greater than the destination operand,carry flag is set or else, carry flag is reset. The examples of this instruction are as follows:

Example:

- CMP BX,0100H – IMMEDIATE
- CMP AX,0100H – IMMEDIATE

□□□□□□□ CMP [5000H], 0100H – DIRECT

- CMP BX,[SI] – REGISTER INDIRECT OR INDEXED
- CMP BX, CX – REGISTER

6. INC & DEC instructions

INC and DEC instructions are used to increment and decrement the content of the specified destination by one. AF, CF, OF, PF, SF, and ZF flags are affected.

Example:

- INC AL ; AL<= AL + 1
- INC AX ; AX<=AX + 1
- DEC AL ; AL<= AL – 1
- DEC AX ; AX<=AX – 1

7. AND instruction

This instruction logically ANDs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.

The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.

General Format:

AND Destination, Source

Example:

- AND BL, AL ; suppose BL=1000 0110 and AL = 1100 1010 then after the operation BL would be BL= 1000 0010.
- AND CX, AX ; CX <= CX AND AX

□ AND CL, 08 ; CL<= CL AND (0000 1000)

8. OR instruction

This instruction logically ORs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.

The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.

General Format:

OR Destination, Source

Example:

- OR BL, AL ; suppose BL=1000 0110 and AL = 1100 1010 then after the operation BL would be BL= 1100 1110.
- OR CX, AX ; CX <= CX AND AX
- OR CL, 08 ; CL<= CL AND (0000 1000)

9. NOT instruction

The NOT instruction complements (inverts) the contents of an operand register or a memory location, bit by bit. The examples are as follows:

Example:

- NOT AX (BEFORE AX= $(1011)_2$= (B) $_{16}$ AFTER EXECUTION AX= $(0100)_2$= $(4)_{16}$).

 NOT [5000H]

10. XOR instruction

The XOR operation is again carried out in a similar way to the AND and OR operation. The constraints on the operands are also similar. The XOR operation gives a high output, when the 2 input bits are dissimilar. Otherwise, the output is zero. The example instructions are as follows:

Example:

- XOR AX,0098H
- XOR AX,BX

 XOR AX,[5000H]

Shift / Rotate Instructions

Shift instructions move the binary data to the left or right by shifting them within the register or memory location. They also can perform multiplication of powers of 2^{+n} and division of powers of 2^{-n}.

There are two type of shifts logical shifting and arithmetic shifting, later is used with signed numbers while former with unsigned.

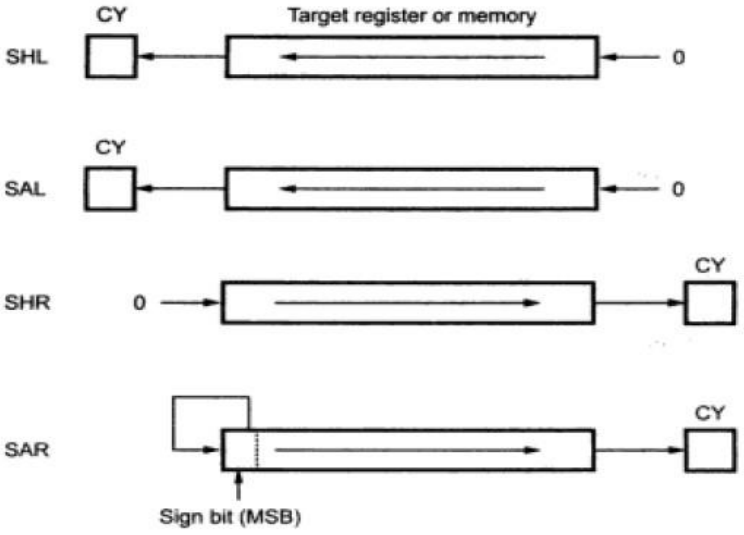


Fig.1 Shift operations

Rotate on the other hand rotates the information in a register or memory either from one end to another or through the carry flag.

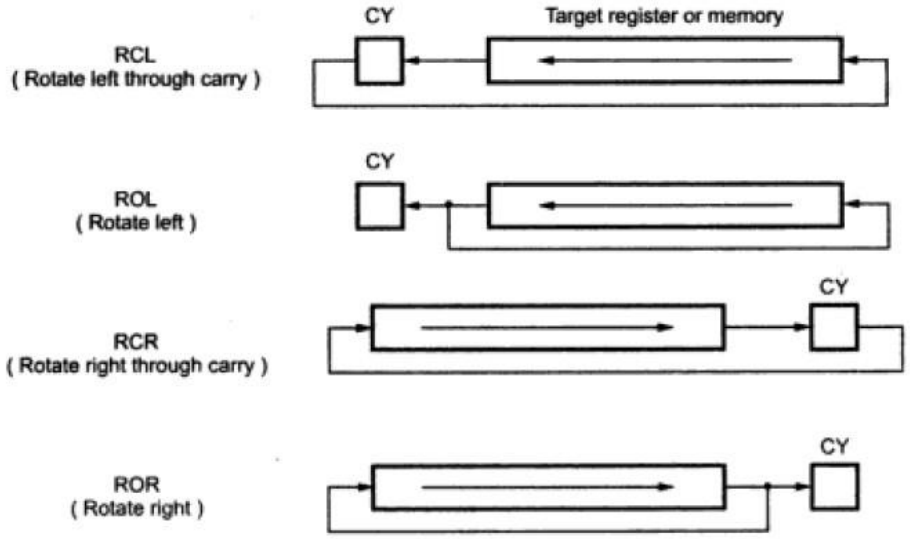


Fig.2 Rotate operations

SHL/SAL instruction

Both the instruction shifts each bit to left, and places the MSB in CF and LSB is made 0. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected.

General Format:

SAL/SHL destination, count

Example:

MOV BL, B7H ; BL is made B7H

SAL BL, 1 ; shift the content of BL register one place to left.

Before execution,

CY	B7	B6	B5	B4	B3	B2	B1	B0
0	1	0	1	1	0	1	1	1

□□□□□□□□

After the execution,

CY		B7	B6	B5	B4	B3	B2	B1	B0 1
	0	1	1	0	1	1	1	0	

1. SHR instruction

This instruction shifts each bit in the specified destination to the right and 0 is stored in the MSB position. The LSB is shifted into the carry flag. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected

General Format: SHR destination, count

Example:

MOV BL, B7H ; BL is made B7H

SHR BL, 1 ; shift the content of BL register one place to the right.

Before execution,

B7	B6	B5	B4	B3	B2	B1	B0	CY
1	0	1	1	0	1	1	1	0

□□□□□□□□

After execution,

B7	B6	B5	B4	B3	B2	B1	B0	CY
0	1	0	1	1	0	1	1	1

2. ROL instruction

This instruction rotates all the bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected

General Format: ROL
destination, count
Example:

MOV BL, B7H ; BL is made B7H

ROL BL, 1 ; rotates the content of BL register one place to the left.

Before execution,

CY	B7	B6	B5	B4	B3	B2	B1	B0
0	1	0	1	1	0	1	1	1

□□□□□□□□□(B7)

After the execution,

CY	B7	B6	B5	B4	B3	B2	B1	B0
1	0	1	1	0	1	1	1	1

3. ROR instruction

This instruction rotates all the bits in a specified byte or word to the right some number of bit positions. LSB is placed as a new MSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected

General Format: ROR
destination, count
Example:

MOV BL, B7H ; BL is made B7H

ROR BL, 1 ; shift the content of BL register one place to the right.

Before execution,

B7	B6	B5	B4	B3	B2	B1	B0	CY
1	0	1	1	0	1	1	1	0

(B0)□□□□□□□□□

After execution,

B7	B6	B5	B4	B3	B2	B1	B0	CY
1	1	0	1	1	0	1	1	1

4. RCR instruction

This instruction rotates all the bits in a specified byte or word to the right some number of bit positions along with the carry flag. LSB is placed in a new CF and previous carry is placed in the new MSB. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected

General Format: RCR destination, count

Example:

```
MOV BL, B7H        ;    BL is made B7H

RCR BL, 1          ;    shift the content of BL register one place to the right.
```

Before execution,

B7	B6	B5	B4	B3	B2	B1	B0	CY
1	0	1	1	0	1	1	1	0

(CY)□□□□□□□□□

After execution,

B7	B6	B5	B4	B3	B2	B1	B0	CY
0	1	0	1	1	0	1	1	1

Program control transfer instructions

There are 2 types of such instructions. They are:

1. Unconditional transfer instructions – CALL, RET, JMP
2. Conditional transfer instructions – J condition

1. CALL instruction

The CALL instruction is used to transfer execution to a subprogram or procedure. There are two types of CALL instructions, near and far.

A **near CALL** is a call to a procedure which is in the same code segment as the

CALL instruction. 8086 when encountered a near call, it decrements the SP by 2 and copies the offset of the next instruction after the CALL on the stack. It loads the IP with the offset of the procedure then to start the execution of the procedure.

A **far CALL** is the call to a procedure residing in a different segment. Here value of CS and offset of the next instruction both are backed up in the stack. And then branches to the procedure by changing the content of CS with the segment base containing procedure and IP with the offset of the first instruction of the procedure.

Example:

Near call

```
CALL PRO          ;   PRO is the name of the procedure

CALL CX           ;   Here CX contains the offset of the first instruction of
                      the procedure, that is replaces the content of IP with the
                      content of CX
```

Far call

```
CALL DWORD PTR[8X]  ;   New values for CS and IP are fetched from four
                        memory locations in the DS. The new value for CS is fetched
                        from [8X] and [8X+1], the new IP is fetched from
                        [8X+2] and [8X+3].
```

2. RET instruction

RET instruction will return execution from a procedure to the next instruction after the CALL instruction in the calling program. If it was a near call, then IP is replaced with the value at the top of the stack, if it had been a far call, then another POP of the stack is required. This second popped data from the stack is put in the CS, thus resuming the execution of the calling program. RET instruction can be followed by a number, to specify the parameters passed.

RET instruction does not affect any flags.

General format:

RET

Example:

```
p1 PROC          procedure declaration.

   MOV
   AX
RET              return to caller.

p1 ENDP
```

3. JMP instruction

This is also called as unconditional jump instruction, because the processor jumps to the specified location rather than the instruction after the JMP instruction. Jumps can be **short jumps** when the target address is in the same segment as the JMP instruction or **far jumps** when it is in a different segment.

General Format:

JMP<targetaddress>

Example:

```
        MOV AL,05H          ;

        JMP label1          ;       jump over to label

        MOV
AL, 00H     ;
label1:     MOV
[2000H], AL;
        RET                 ;
```

4. Conditional Jump (J cond)

Conditional jumps are always short jumps in 8086. Here jump is done only if the condition specified is true/false. If the condition is not satisfied, then the execution proceeds in the normal way.

Example:

There are many conditional jump instructions like JC : Jump on carry (CF=set)

JNC : Jump on non carry (CF=reset)

JZ : Jump on zero (ZF=set)

JNO : Jump on overflow (OF=set)

Etc

5. Iteration control instructions

These instructions are used to execute a series of instructions some number of times. The number is specified in the CX register, which will be automatically decremented in course of iteration. But here the destination address for the jump must be in the range of -128 to 127 bytes.

Example:

Instructions here are:-

LOOP : loop through the set of instructions until CX is 0
LOOPE/LOOPZ : here the set of instructions are repeated until CX=0 or ZF=0
LOOPNE/LOOPNZ: here repeated until CX=0 or ZF=1

Machine Control Instructions

1. **HLT instruction**

The HLT instruction will cause the 8086 microprocessor to fetching and executing instructions.

The 8086 will enter a halt state. The processor gets out of this Halt signal upon an interrupt signal in INTR pin/NMI pin or a reset signal on RESET input.

General form:-

HLT

2. **WAIT instruction**

When this instruction is executed, the 8086 enters into an idle state. This idle state is continued till a high is received on the TEST input pin or a valid interrupt signal is received. Wait affects no flags. It generally is used to synchronize the 8086 with a peripheral device(s).

3. **ESC instruction**

This instruction is used to pass instruction to a coprocessor like 8087. There is a 6 bit instruction for the coprocessor embedded in the ESC instruction. In most cases the 8086 treats ESC and a NOP, but in some cases the 8086 will access data items in memory for the coprocessor

4. **LOCK instruction**

In multiprocessor environments, the different microprocessors share a system bus, which is needed to access external devices like disks. LOCK

Instruction is given as prefix in the case when a processor needs exclusive access of the system bus for a particular instruction. It affects no flags.

Example:

LOCK XCHG SEMAPHORE, AL :The XCHG instruction requires two bus accesses. The lock prefix prevents another processor from taking control of thesystem bus between the 2 accesses

5. NOP instruction

At the end of NOP instruction, no operation is done other than the fetching and decoding of the instruction. It takes 3 clock cycles. NOP is used to fill in time delays or to provide space for instructions while trouble shooting. NOP affects no flags.

Flag manipulation instructions

1. STC instruction

This instruction sets the carry flag. It does not affect any other flag.

2. CLC instruction

This instruction resets the carry flag to zero. CLC does not affect any other flag.

3. CMC instruction

This instruction complements the carry flag. CMC does not affect any other flag.

4. STD instruction

This instruction is used to set the direction flag to one so that SI and/or DI can be decremented automatically after execution of string instruction. STD does not affect any other flag.

5. CLD instruction

This instruction is used to reset the direction flag to zero so that SI and/or DI can be incremented automatically after execution of string instruction. CLD does not affect any other flag.

6. STI instruction

This instruction sets the interrupt flag to 1. This enables INTR interrupt of the 8086. STI does not affect any other flag.

7. CLI instruction

This instruction resets the interrupt flag to 0. Due to this the 8086 will not respond to an interrupt signal on its INTR input. CLI does not affect any other flag.

String Instructions

1. MOVS/MOVSB/MOVSW

These instructions copy a word or byte from a location in the data segment to a location in the extra segment. The offset of the source is in SI and that of destination is in DI. For multiple word/byte transfers the count is stored in the CX register.

When direction flag is 0, SI and DI are incremented and when it is 1, SI and DI are decremented.

MOVS affect no flags. MOVSB is used for byte sized movements while MOVSW is for word sized.

Example:

```
CLD              ;    clear the direction flag to auto increment SI and DI
   MOV AX, 0000H ;
MOV DS, AX       ;    initialize data segment register to 0
   MOV ES, AX    ;    initialize extra segment register to 0
   MOV SI, 2000H ;    Load the offset of the string1 in SI
   MOV DI, 2400H ;    Load the offset of the string2 in DI
   MOV CX, 04H   ;    load length of the string in CX
REP MOVSB        ;    decrement CX and MOVSB until CX will be 0
```

2. REP/REPE/REP2/REPNE/REPNZ

REP is used with string instruction; it repeats an instruction until the specified condition becomes false.

Example:

```
    REP              =>   CX=0
REPE/REPZ
                     =>   CX=0 OR ZF=0
    REPNE/REPNZ      =>    CX=0 OR ZF=1
```

3. LODS/LODSB/LODSW

This instruction copies a byte from a string location pointed to by SI to AL or a word from a string location pointed to by SI to AX.LODS does not affect any flags. LODSB copies byte and LODSW copies word.

Example:

```
CLD                    ; clear direction flag to auto increment SI  MOV SI,
OFFSET S_STRING        ;  point SI at string
LODS S_STRING          ;
```

4. STOS/STOSB/STOSW

The STOS instruction is used to store a byte/word contained in AL/AX to

the offset contained in the DI register. STOS does not affect any flags. After copying the content DI is automatically incremented or decremented, based on the value of direction flag.

Example:

```
MOV DL, OFFSET D_STRING ;     assign DI with destination address.

STOS D_STRING                 ;  assembler uses string name to determine byte or
                                 word, if byte then AL is used and if of word size, AX is used.
```

5. CMPS/CMPSB/CMPSW

CMPS is used to compare the strings, byte wise or word wise. The comparison is affected by subtraction of content pointed by DI from that pointed by SI. The AF, CF, OF, PF, SF and ZF flags are affected by this instruction, but neither operand is affected.

Example:

```
MOV SI, OFFSET F_STRING   ;     point first string

MOV DI, OFFSET            ;     point second string
S_STRING MOV CX, 0AH
CLD                       ;     set the counter as 0AH

REPE CMPSB                ;     clear direction flag to auto increment
                                repeatedly compare till unequal or counter =0
```