

## Unit 2

### 2.1 Read Write Timing Diagram

#### General Bus Operation

The 8086 has a combined address and data bus commonly referred as a time multiplexed address and data bus. The main reason behind multiplexing address and data over the same pins is the maximum utilization of processor pins and it facilitates the use of 40 pin standard DIP package. The bus can be demultiplexed using a few latches and transreceivers, whenever required.

Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T<sub>1</sub>, T<sub>2</sub>, T<sub>3</sub>, T<sub>4</sub>. The address is transmitted by the processor during T<sub>1</sub>. It is present on the bus only for one cycle. The negative edge of this ALE pulse is used to separate the address and the data or status information.

In maximum mode, the status lines S<sub>0</sub>, S<sub>1</sub> and S<sub>2</sub> are used to indicate the type of operation. Status bits S<sub>3</sub> to S<sub>7</sub> are multiplexed with higher order address bits and the BHE signal. Address is valid during T<sub>1</sub> while status bits S<sub>3</sub> to S<sub>7</sub> are valid during T<sub>2</sub> through T<sub>4</sub>.

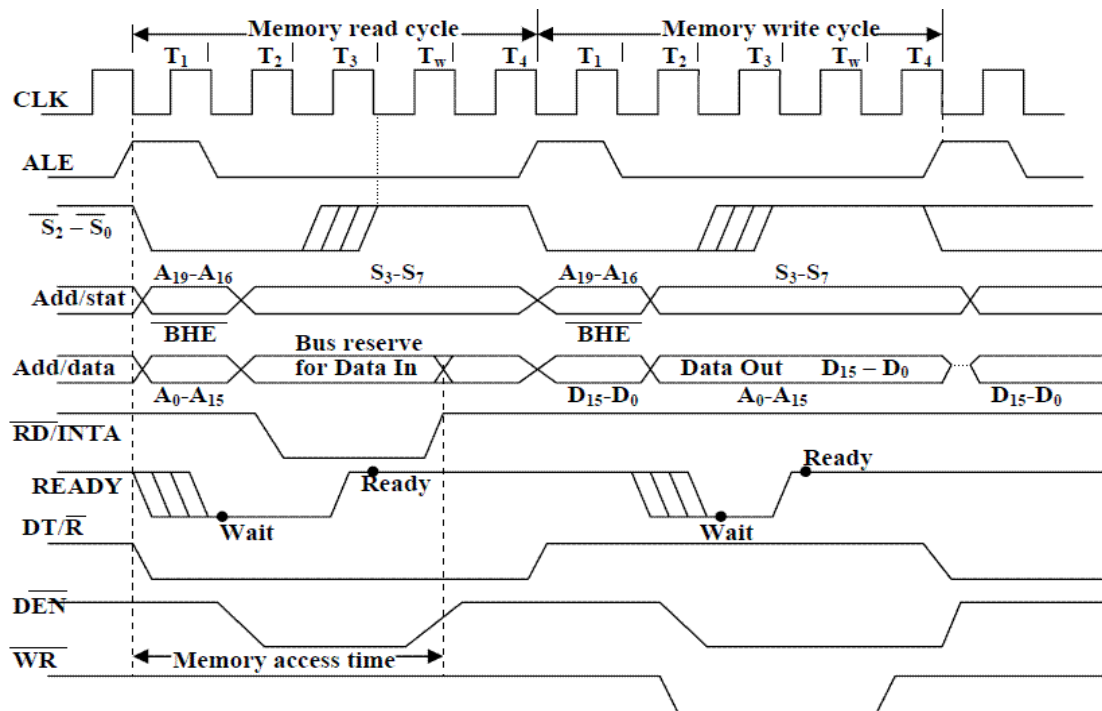


Fig. 2.1 General Bus Operation Cycle of 8086

### Maximum Mode

- i. In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
- ii. In this mode, the processor derives the status signal S2, S1, S0. Another chip called bus controller derives the control signal using this status information.
- iii. In the maximum mode, there may be more than one microprocessor in the system configuration.

### Minimum Mode

- i. In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.
- ii. In this mode, all the control signals are given out by the microprocessor chip itself.

## 2.2 Instruction Set of 8086

The 8086 instructions are categorized into the following main types.

- i. Data Copy / Transfer Instructions
- ii. Arithmetic and Logical Instructions
- iii. Branch Instructions
- iv. Loop Instructions
- v. Machine Control Instructions
- vi. Flag Manipulation Instructions
- vii. Shift and Rotate Instructions
- viii. String Instructions

### Data Copy / Transfer Instructions :

#### MOV :

This instruction copies a word or a byte of data from some source to a destination. The destination can be a register or a memory location. The source can be a register, a memory location, or an immediate number.

```
MOV AX,BX
MOV AX,5000H
MOV AX,[SI]
MOV AX,[2000H]
MOV AX,50H[BX]
MOV [734AH],BX
MOV DS,CX
MOV CL,[357AH]
```

Direct loading of the segment registers with immediate data is not permitted.

## PUSH : Push to Stack

This instruction pushes the contents of the specified register/memory location on to the stack. The stack pointer is decremented by 2, after each execution of the instruction.

E.g. PUSH AX

- PUSH DS
- PUSH [5000H]

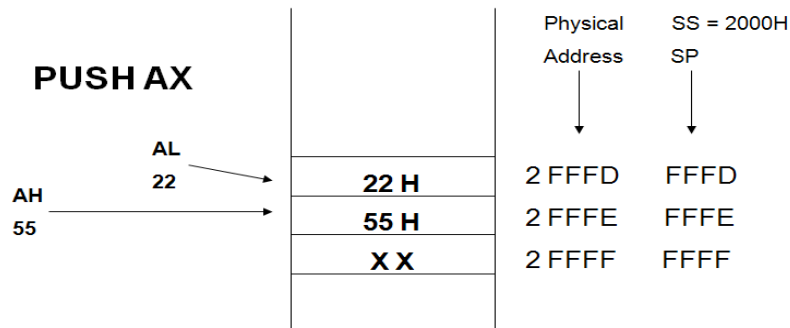


Fig. 2.2 Push Data to stack memory

## POP : Pop from Sack

This instruction when executed, loads the specified register/memory location with the contents of the memory location of which the address is formed using the current stack segment and stack pointer.

The stack pointer is incremented by 2

Eg. POP AX

POP DS

POP [5000H]

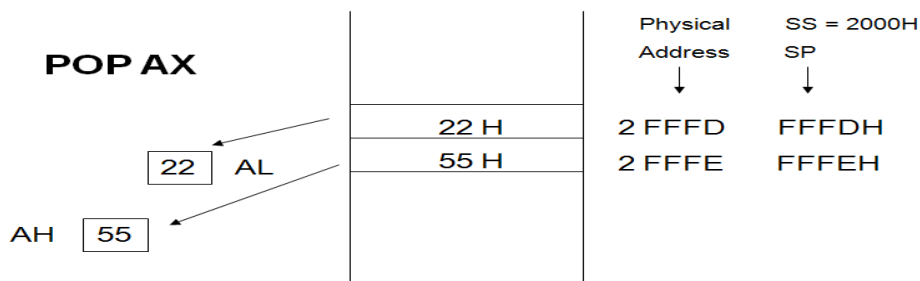


Fig. 2.3 Popping Register Content from Stack Memory

## XCHG : Exchange byte or word

This instruction exchange the contents of the specified source and destination operands

Eg. XCHG [5000H], AX

XCHG BX, AX

### **XLAT :**

Translate byte using look-up table

Eg. LEA BX, TABLE1

MOV AL, 04H

XLAT

Simple input and output port transfer Instructions:

### **IN:**

Copy a byte or word from specified port to accumulator.

Eg. IN AL,03H

IN AX,DX

### **OUT:**

Copy a byte or word from accumulator specified port.

Eg. OUT 03H, AL

OUT DX, AX

### **LEA :**

Load effective address of operand in specified register.

[reg]      offset portion of address in DS

Eg. LEA reg, offset

### **LDS:**

Load DS register and other specified register from memory.

[reg]      [mem]

[DS]      [mem + 2]

Eg. LDS reg, mem

### **LES:**

Load ES register and other specified register from memory.

[reg]      [mem]

[ES]      [mem + 2]

Eg. LES reg, mem

## **Flag transfer instructions:**

### **LAHF:**

Load (copy to) AH with the low byte the flag register.

[AH] ← [Flags low byte]

Eg. LAHF

### **SAHF:**

Store (copy) AH register to low byte of flag register.

$[\text{Flags low byte}] \leftarrow [\text{AH}]$

Eg. SAHF

### **PUSHF:**

Copy flag register to top of stack.

$[\text{SP}] \leftarrow [\text{SP}] - 2$

$[[\text{SP}]] \leftarrow [\text{Flags}]$

Eg. PUSHF

### **POPF :**

Copy word at top of stack to flag register.

$[\text{Flags}] \leftarrow [[\text{SP}]]$

$[\text{SP}] \leftarrow [\text{SP}] + 2$

## **Arithmetic Instructions:**

The 8086 provides many arithmetic operations: addition, subtraction, negation, multiplication and comparing two values.

### **ADD :**

The add instruction adds the contents of the source operand to the destination operand.

Eg. ADD AX, 0100H  
 ADD AX, BX  
 ADD AX, [SI]  
 ADD AX, [5000H]  
 ADD [5000H], 0100H  
 ADD 0100H

### **ADC : Add with Carry**

This instruction performs the same operation as ADD instruction, but adds the carry flag to the result.

Eg. ADC 0100H  
 ADC AX, BX  
 ADC AX, [SI]  
 ADC AX, [5000]  
 ADC [5000], 0100H

### **SUB : Subtract**

The subtract instruction subtracts the source operand from the destination operand and the result is left in the destination operand.

Eg. SUB AX, 0100H  
 SUB AX, BX  
 SUB AX, [5000H]  
 SUB [5000H], 0100H

### **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

Eg. SBB AX, 0100H  
 SBB AX, BX  
 SBB AX, [5000H]  
 SBB [5000H], 0100H

### **INC : Increment**

This instruction increases the contents of the specified Register or memory location by 1. Immediate data cannot be operand of this instruction.

Eg. INC AX  
 INC [BX]  
 INC [5000H]

### **DEC : Decrement**

The decrement instruction subtracts 1 from the contents of the specified register or memory location.

Eg. DEC AX  
 DEC [5000H]

### **NEG : Negate**

The negate instruction forms 2's complement of the specified destination in the instruction. The destination can be a register or a memory location. This instruction can be implemented by inverting each bit and adding 1 to it.

Eg. NEG AL  
 AL = 0011 0101 35H Replace number in AL with its 2's complement  
 AL = 1100 1011 = CBH

### **CMP : Compare**

This 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

Eg. `CMP BX, 0100H`  
`CMP AX, 0100H`  
`CMP [5000H], 0100H`  
`CMP BX, [SI]`  
`CMP BX, CX`

### **MUL :Unsigned Multiplication Byte or Word**

This instruction multiplies an unsigned byte or word by the contents of AL.

Eg. `MUL BH` ; (AX) (AL) x (BH)  
`MUL CX` ; (DX)(AX) (AX) x (CX)  
`MUL WORD PTR [SI]` ; (DX)(AX) (AX) x ([SI])

### **IMUL :Signed Multiplication**

This instruction multiplies a signed byte in source operand by a signed byte in AL or a signed word in source operand by a signed word in AX.

Eg. `IMUL BH`  
`IMUL CX`  
`IMUL [SI]`

### **CBW : Convert Signed Byte to Word**

This instruction copies the sign of a byte in AL to all the bits in AH. AH is then said to be sign extension of AL.

Eg. `CBW`  
`AX= 0000 0000 1001 1000` Convert signed byte in AL signed word in AX.  
Result in `AX = 1111 1111 1001 1000`

### **CWD : Convert Signed Word to Double Word**

This instruction copies the sign of a word in AX to all the bits in DX. DX is then said to be sign extension of AX.

Eg. `CWD`  
Convert signed word in AX to signed double word in DX : AX  
`DX= 1111 1111 1111 1111`  
Result in `AX = 1111 0000 1100 0001`

### **DIV : Unsigned division**

This instruction is used to divide an unsigned word by a byte or to divide an unsigned double word by a word.

Eg. `DIV CL` ; Word in AX / byte in CL  
; Quotient in AL, remainder in AH  
`DIV CX` ; Double word in DX and AX / word  
; in CX, and Quotient in AX,  
; remainder in DX

### AAA : ASCII Adjust After Addition

The AAA instruction is executed after an ADD instruction that adds two ASCII coded operand to give a byte of result in AL. The AAA instruction converts the resulting contents of AL to a unpacked decimal digits.

```
Eg.  ADD CL, DL    ; [CL] = 32H = ASCII for 2
      ; [DL] = 35H = ASCII for 5
      ; Result [CL] = 67H
      MOV AL, CL    ; Move ASCII result into AL since
      ; AAA adjust only [AL]
      AAA           ; [AL]=07, unpacked BCD for 7
```

### AAS : ASCII Adjust AL after Subtraction

This instruction corrects the result in AL register after subtracting two unpacked ASCII operands. The result is in unpacked decimal format. The procedure is similar to AAA instruction except for the subtraction of 06 from AL.

### AAM : ASCII Adjust after Multiplication

This instruction, after execution, converts the product available in AL into unpacked BCD format.

```
Eg.  MOV AL, 04    ; AL = 04
      MOV BL, 09    ; BL = 09
      MUL BL        ; AX = AL*BL ; AX=24H
      AAM           ; AH = 03, AL=06
```

### AAD : ASCII Adjust before Division

This instruction converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be made before dividing the two unpacked BCD digits in AX by an unpacked BCD byte. In the instruction sequence, this instruction appears before DIV instruction.

```
Eg.  AX 05 08
      AAD result in AL 00 3A 58D = 3A H in AL
```

The result of AAD execution will give the hexadecimal number 3A in AL and 00 in AH. Where 3A is the hexadecimal Equivalent of 58 (decimal).

### DAA : Decimal Adjust Accumulator

This instruction is used to convert the result of the addition of two packed BCD numbers to a valid BCD number. The result has to be only in AL.

```
Eg.  AL = 53    CL = 29
      ADD AL, CL  ; AL (AL) + (CL)
      ; AL 53 + 29
      ; AL 7C
      DAA         ; AL 7C + 06 (as C>9)
      ; AL 82
```



### **DAS : Decimal Adjust after Subtraction**

This instruction converts the result of the subtraction of two packed BCD numbers to a valid BCD number. The subtraction has to be in AL only.

Eg. AL = 75, BH = 46  
 SUB AL, BH ; AL 2 F = (AL) - (BH)  
 ; AF = 1  
 DAS ; AL 2 9 (as F > 9, F - 6 = 9)

### **Logical Instructions**

#### **AND : Logical AND**

This instruction bit by bit ANDs the source operand that may be an immediate register or a memory location to the destination operand that may be a register or a memory location. The result is stored in the destination operand.

Eg. AND AX, 0008H  
 AND AX, BX

#### **OR : Logical OR**

This instruction bit by bit ORs the source operand that may be an immediate, register or a memory location to the destination operand that may be a register or a memory location. The result is stored in the destination operand.

Eg. OR AX, 0008H  
 OR AX, BX

#### **NOT : Logical Invert**

This instruction complements the contents of an operand register or a memory location, bit by bit.

Eg. NOT AX  
 NOT [5000H]

#### **XOR : Logical Exclusive OR**

This instruction bit by bit XORs the source operand that may be an immediate, register or a memory location to the destination operand that may be a register or a memory location. The result is stored in the destination operand.

Eg. XOR AX, 0098H  
 XOR AX, BX

#### **TEST : Logical Compare Instruction**

The TEST instruction performs a bit by bit logical AND operation on the two operands. The result of this ANDing operation is not available for further use, but flags are affected.

Eg. TEST AX, BX  
 TEST [0500], 06H

### **SAL/SHL : SAL / SHL destination, count.**

SAL and SHL are two mnemonics for the same instruction. This instruction shifts each bit in the specified destination to the left and 0 is stored at LSB position. The MSB is shifted into the carry flag. The destination can be a byte or a word.

It can be in a register or in a memory location. The number of shifts is indicated by count.

Eg.    SAL CX, 1  
       SAL AX, CL

### **SHR : SHR destination, count**

This instruction shifts each bit in the specified destination to the right and 0 is stored at MSB position. The LSB is shifted into the carry flag. The destination can be a byte or a word.

It can be a register or in a memory location. The number of shifts is indicated by count.

Eg.    SHR CX, 1  
       MOV CL, 05H  
       SHR AX, CL

### **SAR : SAR destination, count**

This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a copy of the old MSB is put in the MSB position. The LSB will be shifted into CF.

Eg.    SAR BL, 1  
       MOV CL, 04H  
       SAR DX, CL

### **ROL Instruction : ROL destination, count**

This instruction rotates all 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.

Eg.    ROL CX, 1  
       MOV CL, 03H  
       ROL BL, CL

### **ROR Instruction : ROR destination, count**

This instruction rotates all 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.

Eg.    ROR CX, 1  
       MOV CL, 03H  
       ROR BL, CL

### **RCL Instruction : RCL destination, count**

This instruction rotates all bits in a specified byte or word some number of bit positions to the *left along with the carry flag*. MSB is placed as a new carry and previous carry is place as new LSB.

Eg.    RCL CX, 1  
          MOV CL, 04H  
          RCL AL, CL

### **RCR Instruction : RCR destination, count**

This instruction rotates all bits in a specified byte or word some number of bit positions to the right *along with the carry flag*. LSB is placed as a new carry and previous carry is place as new MSB.

Eg.    RCR CX, 1  
          MOV CL, 04H  
          RCR AL, CL

### **ROR Instruction : ROR destination, count**

This instruction rotates all 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.

Eg.    ROR CX, 1  
          MOV CL, 03H  
          ROR BL, CL

### **RCL Instruction : RCL destination, count**

This instruction rotates all bits in a specified byte or word some number of bit positions to the *left along with the carry flag*. MSB is placed as a new carry and previous carry is place as new LSB.

Eg.    RCL CX, 1  
          MOV CL, 04H  
          RCL AL, CL

### **RCR Instruction : RCR destination, count**

This instruction rotates all bits in a specified byte or word some number of bit positions to the right *along with the carry flag*. LSB is placed as a new carry and previous carry is place as new MSB.

Eg.    RCR CX, 1  
          MOV CL, 04H  
          RCR AL, CL

### **Branch Instructions :**

Branch Instructions transfers the flow of execution of the program to a new address specified in the instruction directly or indirectly. When this type of instruction is executed, the CS and IP registers get loaded with new values of CS and IP corresponding to the location to be transferred.

The Branch Instructions are classified into two types

- i. Unconditional Branch Instructions.
- ii. Conditional Branch Instructions.

### **Unconditional Branch Instructions :**

In Unconditional control transfer instructions, the execution control is transferred to the specified location independent of any status or condition. The CS and IP are unconditionally modified to the new CS and IP.

#### **CALL : Unconditional Call**

This instruction is used to call a Subroutine (Procedure) from a main program. Address of procedure may be specified directly or indirectly.

There are two types of procedure depending upon whether it is available in the same segment or in another segment.

- i. Near CALL i.e.,  $\pm 32K$  displacement.
- ii. For CALL i.e., anywhere outside the segment.

On execution this instruction stores the incremented IP & CS onto the stack and loads the CS & IP registers with segment and offset addresses of the procedure to be called.

#### **RET: Return from the Procedure.**

At the end of the procedure, the RET instruction must be executed. When it is executed, the previously stored content of IP and CS along with Flags are retrieved into the CS, IP and Flag registers from the stack and execution of the main program continues further.

#### **INT N: Interrupt Type N.**

In the interrupt structure of 8086, 256 interrupts are defined corresponding to the types from 00H to FFH. When INT N instruction is executed, the type byte N is multiplied by 4 and the contents of IP and CS of the interrupt service routine will be taken from memory block in 0000 segment.

### **INTO: Interrupt on Overflow**

This instruction is executed, when the overflow flag OF is set. This is equivalent to a Type 4 Interrupt instruction.

### **JMP: Unconditional Jump**

This instruction unconditionally transfers the control of execution to the specified address using an 8-bit or 16-bit displacement. No Flags are affected by this instruction.

### **IRET: Return from ISR**

When it is executed, the values of IP, CS and Flags are retrieved from the stack to continue the execution of the main program.

### **LOOP : LOOP Unconditionally**

This instruction executes the part of the program from the Label or address specified in the instruction upto the LOOP instruction CX number of times. At each iteration, CX is decremented automatically and JUMP IF NOT ZERO structure.

```
Example:  MOV CX, 0004H
          MOV BX, 7526H
          Label 1 MOV AX, CODE
          OR  BX, AX
          LOOP Label 1
```

## **Conditional Branch Instructions**

When this instruction is executed, execution control is transferred to the address specified relatively in the instruction, provided the condition implicit in the Opcode is satisfied. Otherwise execution continues sequentially.

### **JZ/JE Label**

Transfer execution control to address 'Label', if ZF=1.

### **JNZ/JNE Label**

Transfer execution control to address 'Label', if ZF=0

### **JS Label**

Transfer execution control to address 'Label', if SF=1.

### **JNS Label**

Transfer execution control to address 'Label', if SF=0.

### **JO Label**

Transfer execution control to address 'Label', if OF=1.

**JNO Label**

Transfer execution control to address 'Label', if OF=0.

**JNP Label**

Transfer execution control to address 'Label', if PF=0.

**JP Label**

Transfer execution control to address 'Label', if PF=1.

**JB Label**

Transfer execution control to address 'Label', if CF=1.

**JNB Label**

Transfer execution control to address 'Label', if CF=0.

**JCXZ Label**

Transfer execution control to address 'Label', if CX=0

**Conditional LOOP Instructions.****LOOPZ / LOOPE Label**

Loop through a sequence of instructions from label while ZF=1 and CX=0.

**LOOPNZ / LOOPNE Label**

Loop through a sequence of instructions from label while ZF=1 and CX=0.

**String Manipulation Instructions**

A series of data byte or word available in memory at consecutive locations, to be referred as Byte String or Word String. A String of characters may be located in consecutive memory locations, where each character may be represented by its ASCII equivalent.

The 8086 supports a set of more powerful instructions for string manipulations for referring to a string, two parameters are required.

- I. Starting and End Address of the String.
- II. Length of the String.

The length of the string is usually stored as count in the CX register. The incrementing or decrementing of the pointer, in string instructions, depends upon the Direction Flag (DF) Status. If it is a Byte string operation, the index registers are updated

by one. On the other hand, if it is a word string operation, the index registers are updated by two.

### **REP : Repeat Instruction Prefix**

This instruction is used as a prefix to other instructions, the instruction to which the REP prefix is provided, is executed repeatedly until the CX register becomes zero (at each iteration CX is automatically decremented by one).

- i. REPE / REPZ - repeat operation while equal / zero.
- ii. REPNE / REPNZ - repeat operation while not equal / not zero.

These are used for CMPS, SCAS instructions only, as instruction prefixes.

### **MOVSb / MOVSw : Move String Byte or String Word**

Suppose a string of bytes stored in a set of consecutive memory locations is to be moved to another set of destination locations. The starting byte of source string is located in the memory location whose address may be computed using SI (Source Index) and DS (Data Segment) contents.

The starting address of the destination locations where this string has to be relocated is given by DI (Destination Index) and ES (Extra Segment) contents.

### **CMPS : Compare String Byte or String Word**

The CMPS instruction can be used to compare two strings of byte or words. The length of the string must be stored in the register CX. If both the byte or word strings are equal, zero Flag is set.

The REP instruction Prefix is used to repeat the operation till CX (counter) becomes zero or the condition specified by the REP Prefix is False.

### **SCAN : Scan String Byte or String Word**

This instruction scans a string of bytes or words for an operand byte or word specified in the register AL or AX. The String is pointed to by ES:DI register pair. The length of the string is stored in CX. The DF controls the mode for scanning of the string. Whenever a match to the specified operand, is found in the string, execution stops and the zero Flag is set. If no match is found, the zero flag is reset.

### **LODS : Load String Byte or String Word**

The LODS instruction loads the AL / AX register by the content of a string pointed to by DS : SI register pair. The SI is modified automatically depending upon DF, If it is a byte transfer (LODSB), the SI is modified by one and if it is a word transfer (LODSW), the SI is modified by two. No other Flags are affected by this instruction.

### **STOS : Store String Byte or String Word**

The STOS instruction Stores the AL / AX register contents to a location in the string pointer by ES : DI register pair. The DI is modified accordingly, No Flags are affected by this instruction.

The direction Flag controls the String instruction execution, The source index SI and Destination Index DI are modified after each iteration automatically. If DF=1, then the execution follows autodecrement mode, SI and DI are decremented automatically after each iteration. If DF=0, then the execution follows autoincrement mode. In this mode, SI and DI are incremented automatically after each iteration.

### **Flag Manipulation and a Processor Control Instructions**

These instructions control the functioning of the available hardware inside the processor chip. These instructions are categorized into two types:

1. Flag Manipulation instructions.
2. Machine Control instructions.

#### **Flag Manipulation instructions**

The Flag manipulation instructions directly modify some of the Flags of 8086.

- i. CLC – Clear Carry Flag.
- ii. CMC – Complement Carry Flag.
- iii. STC – Set Carry Flag.
- iv. CLD – Clear Direction Flag.
- v. STD – Set Direction Flag.
- vi. CLI – Clear Interrupt Flag.
- vii. STI – Set Interrupt Flag.

#### **Machine Control instructions**

The Machine control instructions control the bus usage and execution

- i. WAIT – Wait for Test input pin to go low.
- ii. HLT – Halt the process.
- iii. NOP – No operation.
- iv. ESC – Escape to external device like NDP
- v. LOCK – Bus lock instruction prefix.



## **Addressing Modes**

### **Addressing modes of 8086**

When 8086 executes an instruction, it performs the specified function on data. These data are called its operands and may be part of the instruction, reside in one of the internal registers of the microprocessor, stored at an address in memory or held at an I/O port, to access these different types of operands, the 8086 is provided with various addressing modes (Data Addressing Modes).

### **Data Addressing Modes of 8086**

The 8086 has 12 addressing modes. The various 8086 addressing modes can be classified into five groups.

- A. Addressing modes for accessing immediate and register data (register and immediate modes).
- B. Addressing modes for accessing data in memory (memory modes)
- C. Addressing modes for accessing I/O ports (I/O modes)
- D. Relative addressing mode
- E. Implied addressing mode

## **8086 ADDRESSING MODES**

### **A. Immediate addressing mode:**

In this mode, 8 or 16 bit data can be specified as part of the instruction.

OP Code	Immediate Operand
---------	-------------------

Example 1 : MOV CL, 03 H  
Moves the 8 bit data 03 H into CL

Example 2 : MOV DX, 0525 H  
Moves the 16 bit data 0525 H into DX

In the above two examples, the source operand is in immediate mode and the destination operand is in register mode.

A constant such as “VALUE” can be defined by the assembler EQUATE directive such as VALUE EQU 35H

Example : MOV BH, VALUE  
Used to load 35 H into BH

### **Register addressing mode :**

The operand to be accessed is specified as residing in an internal register of 8086. Fig. below shows internal registers, any one can be used as a source or destination operand, however only the data registers can be accessed as either a byte or word.

Register	Operand sizes	
	Byte (Reg 8)	Word (Reg 16)
Accumulator	AL, AH	Ax
Base	BL, BH	Bx
Count	CL, CH	Cx
Data	DL, DH	Dx
Stack pointer	-	SP
Base pointer	-	BP
Source index	-	SI
Destination index	-	DI
Code Segment	-	CS
Data Segment	-	DS
Stack Segment	-	SS
Extra Segment	-	ES

**Example 1 :** MOV DX (Destination Register) , CX (Source Register)  
Which moves 16 bit content of CS into DX.

**Example 2 :** MOV CL, DL  
Moves 8 bit contents of DL into CL

MOV BX, CH is an illegal instruction.

\* The register sizes must be the same.

### **B. Direct addressing mode :**

The instruction Opcode is followed by an effective address, this effective address is directly used as the 16 bit offset of the storage location of the operand from the location specified by the current value in the selected segment register.

### **The default segment is always DS.**

The 20 bit physical address of the operand in memory is normally obtained as

PA = DS : EA

But by using a segment override prefix (SOP) in the instruction, any of the four segment registers can be referenced,

$$PA = \left\{ \begin{array}{c} CS \\ DS \\ SS \\ ES \end{array} \right\} : \{ \text{Direct Address} \}$$

The Execution Unit (EU) has direct access to all registers and data for register and immediate operands. However the EU cannot directly access the memory operands. It must use the BIU, in order to access memory operands.

In the direct addressing mode, the 16 bit effective address (EA) is taken directly from the displacement field of the instruction.

**Example 1 : MOV CX, START**

If the 16 bit value assigned to the offset START by the programmer using an assembler pseudo instruction such as DW is 0040 and [DS] = 3050. Then BIU generates the 20 bit physical address 30540 H.

The content of 30540 is moved to CL  
The content of 30541 is moved to CH

**Example 2 : MOV CH, START**

If [DS] = 3050 and START = 0040  
8 bit content of memory location 30540 is moved to CH.

**Example 3 : MOV START, BX**

With [DS] = 3050, the value of START is 0040.

Physical address : 30540

MOV instruction moves (BL) and (BH) to locations 30540 and 30541 respectively.

**Register indirect addressing mode :**

The EA is specified in either pointer (BX) register or an index (SI or DI) register. The 20 bit physical address is computed using DS and EA.

Example : MOV [DI], BX

← register indirect

If [DS] = 5004, [DI] = 0020, [Bx] = 2456 PA=50060.

The content of BX(2456) is moved to memory locations 50060 H and 50061 H.

$$PA = \begin{Bmatrix} CS \\ DS \\ SS \\ ES \end{Bmatrix} = \begin{Bmatrix} BX \\ SI \\ DI \end{Bmatrix}$$

**Based addressing mode:**

$$PA = \begin{Bmatrix} CS \\ DS \\ SS \\ ES \end{Bmatrix} : \begin{Bmatrix} BX \\ \text{or} \\ BP \end{Bmatrix} + \text{displacement}$$

when memory is accessed PA is computed from BX and DS when the stack is accessed PA is computed from BP and SS.

**Example :** MOV AL, START [BX]

or

MOV AL, [START + BX]

← based mode

EA : [START] + [BX]

PA : [DS] + [EA]

The 8 bit content of this memory location is moved to AL.

### Indexed addressing mode:

$$PA = \left\{ \begin{array}{c} CS \\ DS \\ SS \\ ES \end{array} \right\} : \left\{ \begin{array}{c} SI \\ \text{or} \\ DI \end{array} \right\} + 8 \text{ or } 16\text{bit displacement}$$

**Example** : MOV BH, START [SI]  
PA : [SART] + [SI] + [DS]

The content of this memory is moved into BH.

### Based Indexed addressing mode:

$$PA = \left\{ \begin{array}{c} CS \\ DS \\ SS \\ ES \end{array} \right\} : \left\{ \begin{array}{c} BX \\ \text{or} \\ BP \end{array} \right\} + \left\{ \begin{array}{c} SI \\ \text{or} \\ DI \end{array} \right\} + 8 \text{ or } 16\text{bit displacement}$$

**Example** : MOV ALPHA [SI] [BX], CL  
If [BX] = 0200, ALPHA – 08, [SI] = 1000 H and [DS] = 3000

Physical address (PA) = 31208

8 bit content of CL is moved to 31208 memory address.

### String addressing mode:

The string instructions automatically assume SI to point to the first byte or word of the source operand and DI to point to the first byte or word of the destination operand. The contents of SI and DI are automatically incremented (by clearing DF to 0 by CLD instruction) to point to the next byte or word.

**Example** : MOV S BYTE  
If [DF] = 0, [DS] = 2000 H, [SI] = 0500,  
[ES] = 4000, [DI] = 0300

Source address : 20500, assume it contains 38

PA : [DS] + [SI]

Destination address : [ES] + [DI] = 40300, assume it contains 45

After executing MOV S BYTE,

$\left. \begin{array}{l} [40300] = 38 \\ [SI] = 0501 \\ [DI] = 0301 \end{array} \right\} \text{ incremented}$

### C. I/O mode (direct) :

Port number is an 8 bit immediate operand.

**Example** : OUT 05 H, AL  
Outputs [AL] to 8 bit port 05 H

### I/O mode (indirect):

The port number is taken from DX.

**Example 1** : INAL, DX

If [DX] = 5040

8 bit content by port 5040 is moved into AL.

Example 2 : IN AX, DX

Inputs 8 bit content of ports 5040 and 5041 into AL and AH respectively.

**D. Relative addressing mode:**

Example : JNC START

If CY=0, then PC is loaded with current PC contents plus 8 bit signed value of START, otherwise the next instruction is executed.

**E. Implied addressing mode:**

Instruction using this mode have no operands.

Example : CLC which clears carry flag to zero.

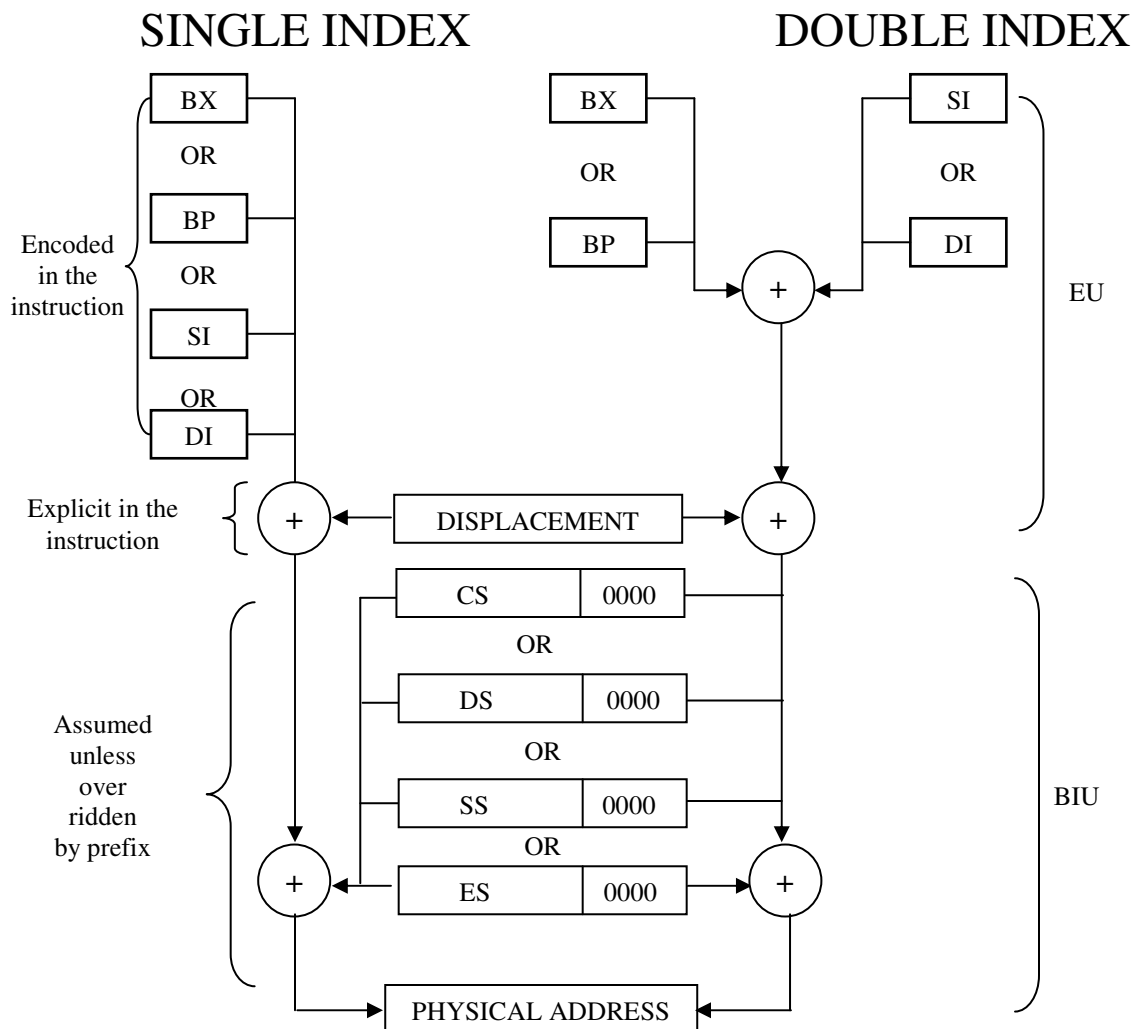


Fig.3.1 : Summary of 8086 Addressing Modes

## Special functions of general-purpose registers:

### AX & DX registers:

In 8 bit multiplication, one of the operands must be in AL. The other operand can be a byte in memory location or in another 8 bit register. The resulting 16 bit product is stored in AX, with AH storing the MS byte.

In 16 bit multiplication, one of the operands must be in AX. The other operand can be a word in memory location or in another 16 bit register. The resulting 32 bit product is stored in DX and AX, with DX storing the MS word and AX storing the LS word.

**BX register** : In instructions where we need to specify in a general purpose register the 16 bit effective address of a memory location, the register BX is used (register indirect).

**CX register** : In Loop Instructions, CX register will be always used as the implied counter.

In I/O instructions, the 8086 receives into or sends out data from AX or AL depending as a word or byte operation. In these instructions the port address, if greater than FFH has to be given as the contents of DX register.

**Ex :** IN AL, DX

DX register will have 16 bit address of the I/P device

### **Physical Address (PA) generation :**

Generally Physical Address (20 Bit) = Segment Base Address (SBA)  
+ Effective Address (EA)

### **Code Segment :**

Physical Address (PA) = CS Base Address  
+ Instruction Pointer (IP)

Data Segment (DS)

PA = DS Base Address + EA can be in BX or SI or DI

Stack Segment (SS)

PA + SS Base Address + EA can be SP or BP

Extra Segment (ES)

PA = ES Base Address + EA in DI

### **Instruction Format :**

The 8086 instruction sizes vary from one to six bytes. The OP code occupies six bytes and it defines the operation to be carried out by the instruction.

Register Direct bit (D) occupies one bit. It defines whether the register operand in byte 2 is the source or destination operand.

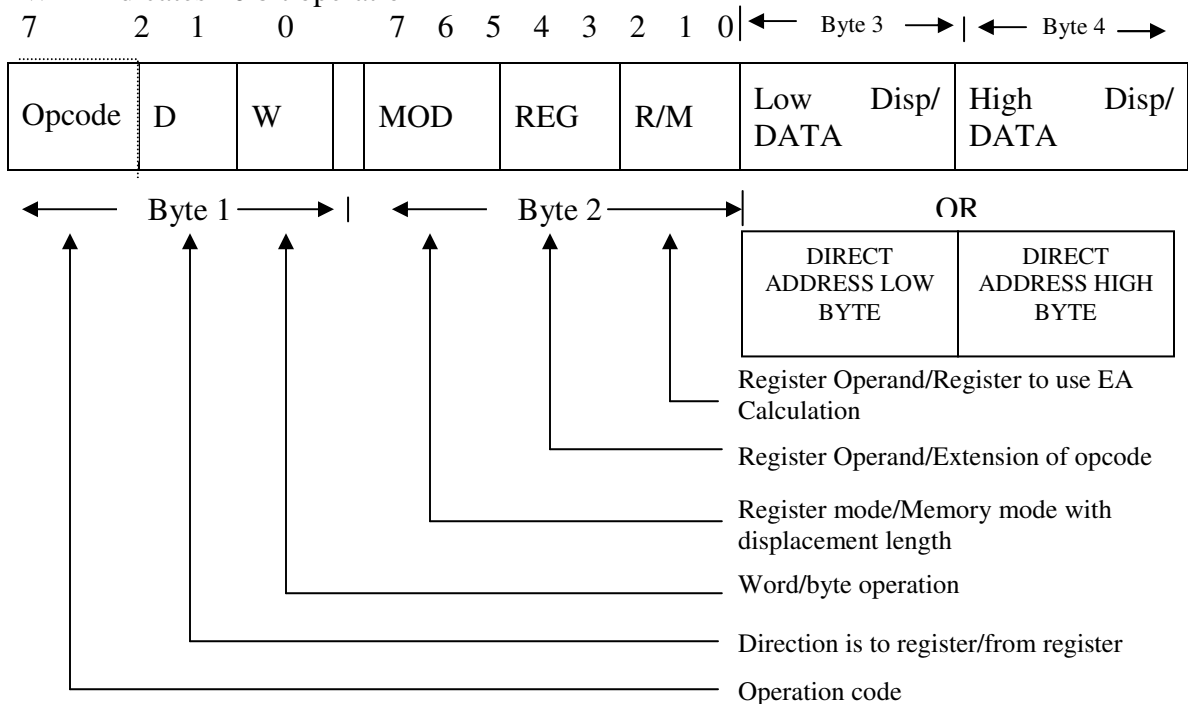
D=1 Specifies that the register operand is the destination operand.

D=0 indicates that the register is a source operand.

Data size bit (W) defines whether the operation to be performed is an 8 bit or 16 bit data

W=0 indicates 8 bit operation

W=1 indicates 16 bit operation



The second byte of the instruction usually identifies whether one of the operands is in memory or whether both are registers.

This byte contains 3 fields. These are the mode (MOD) field, the register (REG) field and the Register/Memory (R/M) field.

MOD (2 bits)	Interpretation
00	Memory mode with no displacement follows except for 16 bit displacement when R/M=110
01	Memory mode with 8 bit displacement
10	Memory mode with 16 bit displacement
11	Register mode (no displacement)

Register field occupies 3 bits. It defines the register for the first operand which is specified as source or destination by the D bit.

REG	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

The R/M field occupies 3 bits. The R/M field along with the MOD field defines the second operand as shown below.

#### MOD 11

R/M	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

#### Effective Address Calculation

R/M	MOD=00	MOD 01	MOD 10
000	(BX) + (SI)	(BX)+(SI)+D8	(BX)+(SI)+D16
001	(BX)+(DI)	(BX)+(DI)+D8	(BX)+(DI)+D16
010	(BP)+(SI)	(BP)+(SI)+D8	(BP)+(SI)+D16
011	(BP)+(DI)	(BP)+(DI)+D8	(BP)+(DI)+D16
100	(SI)	(SI) + D8	(SI) + D16
101	(DI)	(DI) + D8	(DI) + D16
110	Direct address	(BP) + D8	(BP) + D16
111	(BX)	(BX) + D8	(BX) + D16

In the above, encoding of the R/M field depends on how the mode field is set. If MOD=11 (register to register mode), this R/M identifies the second register operand.

MOD selects memory mode, then R/M indicates how the effective address of the memory operand is to be calculated. Bytes 3 through 6 of an instruction are optional fields that normally contain the displacement value of a memory operand and / or the actual value of an immediate constant operand.

#### Example 1 : MOV CH, BL

This instruction transfers 8 bit content of BL



### Into CH

The 6 bit Opcode for this instruction is  $100010_2$  D bit indicates whether the register specified by the REG field of byte 2 is a source or destination operand.

D=0 indicates BL is a source operand.

W=0 byte operation

In byte 2, since the second operand is a register MOD field is  $11_2$ .

The R/M field =  $101$  (CH)

Register (REG) field =  $011$  (BL)

Hence the machine code for MOV CH, BL is

10001000 11 011 101

Byte 1 Byte2

= 88DD<sub>16</sub>

### Example 2 : SUB Bx, (DI)

This instruction subtracts the 16 bit content of memory location addressed by DI and DS from Bx. The 6 bit Opcode for SUB is  $001010_2$ .

D=1 so that REG field of byte 2 is the destination operand. W=1 indicates 16 bit operation.

MOD = 00

REG = 011

R/M = 101

The machine code is  $\frac{0010}{2} \frac{1011}{B} \frac{0001}{1} \frac{1101}{D}$

**2B1D<sub>16</sub>**

MOD / R/M	Memory Mode (EA Calculation)			Register Mode	
	00	01	10	W=0	W=1
000	(BX)+(SI)	(BX)+(SI)+d8	(BX)+(SI)+d16	AL	AX
001	(BX) + (DI)	(BX)+(DI)+d8	(BX)+(DI)+d16	CL	CX
010	(BP)+(SI)	(BP)+(SI)+d8	(BP)+(SI)+d16	DL	DX
011	(BP)+(DI)	(BP)+(DI)+d8	(BP)+(DI)+d16	BL	BX
100	(SI)	(SI) + d8	(SI) + d16	AH	SP
101	(DI)	(DI) + d8	(DI) + d16	CH	BP
110	d16	(BP) + d8	(BP) + d16	DH	SI
111	(BX)	(BX) + d8	(BX) + d16	BH	DI

### Summary of all Addressing Modes

#### Example 3 : Code for MOV 1234 (BP), DX

Here we have specify DX using REG field, the D bit must be 0, indicating the DX is the source register. The REG field must be 010 to indicate DX register. The W bit must be 1 to indicate it is a word operation. 1234 [BP] is specified using MOD value of 10 and R/M value of 110 and a displacement of 1234H. The 4 byte code for this instruction would be 89 96 34 12H.

Opcode	D	W	MOD	REG	R/M	LB displacement	HB displacement
100010	0	1	10	010	110	34H	12H

**Example 4 :** Code for MOV DS : 2345 [BP], DX

Here we have to specify DX using REG field. The D bit must be 0, indicating that Dx is the source register. The REG field must be 010 to indicate DX register. The w bit must be 1 to indicate it is a word operation. 2345 [BP] is specified with MOD=10 and R/M = 110 and displacement = 2345 H.

Whenever BP is used to generate the Effective Address (EA), the default segment would be SS. In this example, we want the segment register to be DS, we have to provide the segment override prefix byte (SOP byte) to start with. The SOP byte is 001 SR 110, where SR value is provided as per table shown below.

SR	Segment register
00	ES
01	CS
10	SS
11	DS

To specify DS register, the SOP byte would be 001 11 110 = 3E H. Thus the 5 byte code for this instruction would be 3E 89 96 45 23 H.

SOP	Opcode	D	W	MOD	REG	R/M	LB disp.	HD disp.
3EH	1000 10	0	1	10	010	110	45	23

Suppose we want to code MOV SS : 2345 (BP), DX. This generates only a 4 byte code, without SOP byte, as SS is already the default segment register in this case.

**Example 5 :**

Give the instruction template and generate code for the instruction ADD OFABE [BX], [DI], DX (code for ADD instruction is 000000)

ADD OFABE [BX] [DI], DX

Here we have to specify DX using REG field. The bit D is 0, indicating that DX is the source register. The REG field must be 010 to indicate DX register. The w must be 1 to indicate it is a word operation. FABE (BX + DI) is specified using MOD value of 10 and R/M value of 001 (from the summary table). The 4 byte code for this instruction would be

Opcode	D	W	MOD	REG	R/M	16 bit disp.	
000000	0	1	10	010	001	BEH	FAH

=01 91 BE FAH

### Example 6 :

Give the instruction template and generate the code for the instruction MOV AX, [BX]

(Code for MOV instruction is 100010)

AX destination register with D=1 and code for AX is 000 [BX] is specified using 00 Mode and R/M value 111

It is a word operation

Opcode	D	W	Mod	REG	R/M	
100010	1	1	00	000	111	=8B 07H

### Questions :

- Write a note on segment registers.
- List the rules for segmentation.
- What are the advantages of using segmentation?
- What do you mean by index registers?
- What is the function of SI and DI registers?
- Explain the addressing modes of 8086 with the help of examples.
- What do you mean by segment override prefix?
- Write a short notes on i) Instruction formats ii) Instruction execution timing
- Determine and explain the addressing modes of the following 8086 instructions.
  - PUSH BX
  - CALL BX
  - JMP DWORD PTR 6200 [BX]
  - OR OABCD [BX] [SI], CX
  - INT 0
- Give the instruction template and generate code for the instruction ADD OFABE [BX] [DI], DX (code for ADD instruction is 000 000)
- Explain the special functions performed by general purpose registers of 8086.
- Give the instruction template and generate the code for the instruction MOV AX, [BX].

### Data Transfer Instructions :

The MOV instruction is used to transfer a byte or a word of data from a source operand to a destination operand. These operands can be internal registers of the 8086 and storage locations in memory.

Mnemonic	Meaning	Format	Operation	Flags affected
MOV	Move	MOV D, S	(S) → (D)	None

Destination	Source	Example
Memory	Accumulator	MOV TEMP, AL
Accumulator	Memory	MOV AX, TEMP
Register	Register	MOV AX, BX
Register	Memory	MOV BP, Stack top
Memory	Register	MOV COUNT [DI], CX
Register	Immediate	MOV CL, 04
Memory	Immediate	MOV MASK [BX] [SI], 2F
Seg. Register	Reg 16	MOV ES, CX
Seg. Register	Mem 16	MOV DS, Seg base
(Word Operation) Reg 16	Seg Reg	MOV BP SS
(Word Operation) Memory 16	Seg Reg	MOV [BX], CS

MOV instruction cannot transfer data directly between a source and a destination that both reside in external memory.

### INPUT/OUTPUT INSTRUCTIONS :

**IN acc, port** : In transfers a byte or a word from input port to the AL register or the AX register respectively. The port number may be specified either with an immediate byte constant, allowing access to ports numbered 0 through 255 or with a number previously placed in the DX register allowing variable access (by changing the value in DX) to ports numbered from 0 through 65,535.

In Operands	Example
acc, immB	IN AL, 0E2H (OR) IN AX, PORT
acc, DX	IN AX, DX (OR) IN AL, DX

**OUT port, acc** : Out transfers a byte or a word from the AL register or the AX register respectively to an output port. The port numbers may be specified either with an immediate byte or with a number previously placed in the register DX allowing variable access.

No flags are affected.

In Operands	Example
Imm 8, acc	OUT 32, AX (OR) OUT PORT, AL
DX, acc	OUT DX, AL (OR) OUT DX, AX

### XCHG D, S :

Mnemonic	Meaning	Format	Operation	Flags affected
XCHG	Exchange	XCHGD,S	(D) $\leftrightarrow$ (S)	None

Destination	Source	Example
Accumulator	Reg 16	XCHG, AX, BX
Memory	Register	XCHG TEMP, AX
Register	Register	XCHG AL, BL

In the above table register cannot be a segment register

Example : For the data given, what is the result of executing the instruction.

XCHG [SUM], BX

((DS) + SUM)  $\leftrightarrow$  (BX)

if (DS) = 0200, SUM = 1234

PA = 02000 + 1234 = 03234

ASSUME (03234) = FF [BX] = 11AA

(03235) = 00

(03234)  $\leftrightarrow$  (BL)

(03235)  $\leftrightarrow$  (BH)

We get (BX) = 00FF

(SUM) = 11AA

### XLAT (translate):

This instruction is useful for translating characters from one code such as ASCII to another such as EBCDIC, this is no operand instruction and is called an instruction with implied addressing mode.

The instruction loads AL with the contents of a 20 bit physical address computed from DS, BX and AL. This instruction can be used to read the elements in a table where BX can be loaded with a 16 bit value to point to the starting address (offset from DS) and AL can be loaded with the element number (0 being the first element number) no flags are affected.

XLAT instruction is equivalent to

MOV AL, [AL] [BX]

AL  $\leftarrow$  [(AL) + (BX) + (DS)]

### Example :

Write a program to convert binary to gray code for the numbers 0 to F using translate instruction.

Let the binary number is stored at 0350 and its equivalent gray code is stored at 0351 after the program execution. Look up table is as follows.

Memory	Data	Data in look up table
0300	00	Exampe:
0301:	01	If (0350) = 03
0302	03	Result (0351) = 02
0303	02	
.		
.		
.		
030F	08	

MOV BX, 0300 : Let BX points to the starting address of the look up table.

MOV SI, 0350 : Let SI points to the address of binary numbers

LOD SB : Load the string byte into AL register.

XLAT : Translate a byte in AL from the look up table stored in the memory pointed by BX.

MOV [SI+1], AL : Move the equivalent gray code to location SI+1  
INT20

### Flag Control Instructions :

Mnemonic	Meaning	Operation	Flags affected
LAHF	Load AH from flags	(AH) ← Flags	None
SAHF	Store AH into flags	(flags) ← (AH)	SF,ZF,AF,PF,CF
CLC	Clear carry flag	(CF) ← 0	CF
STC	Set carry flag	(CF) ← 1	CF
CMC	Complement carry flag	(CF) ← (CF) $\overline{\phantom{x}}$	CF
CLI	Clear interrupt flag	(IF) ← 0	IF
STI	Set interrupt flag	(IF) ← 1	IF

**Fig. : Flag control Instructions**

The first two instructions LAHF and SAHF can be used either to read the flags or to change them respectively notice that the data transfer that takes place is always between the AH register and flag register. For instance, we may want to start an operation with certain flags set or reset. Assume that we want to preset all flags to logic 1. To do this we can first load AH with FF and then execute the SAHF instruction.

**Example :** Write an instruction sequence to save the contents of the 8086's flags in memory location MEM1 and then reload the flags with the contents of memory location MEM2. Assume that MEM1 and MEM2 are in the same data segment defined by the current contents of DS.

LAHF	:	Load current flags in AH register
MOV (MEM1), AH	:	Save in (MEM1)
MOV AH, (MEM2)	:	Copy the contents of (MEM2)
SAHF	:	Store AH contents into the flags.

### Strings and String Handling Instructions :

The 8086 microprocessor is equipped with special instructions to handle string operations. By string we mean a series of data words or bytes that reside in consecutive memory locations. The string instructions of the 8086 permit a programmer to implement operations such as to move data from one block of memory to a block elsewhere in memory. A second type of operation that is easily performed is to scan a string and data elements stored in memory looking for a specific value. Other examples are to compare the elements and two strings together in order to determine whether they are the same or different.

#### Move String : MOV SB, MOV SW:

An element of the string specified by the source index (SI) register with respect to the current data segment (DS) register is moved to the location specified by the destination index (DI) register with respect to the current extra segment (ES) register.

The move can be performed on a byte (MOV SB) or a word (MOV SW) of data. After the move is complete, the contents of both SI & DI are automatically incremented or decremented by 1 for a byte move and by 2 for a word move. Address pointers SI and DI increment or decrement depends on how the direction flag DF is set.

Example : Block move program using the move string instruction

```
MOV AX, DATA SEG ADDR
MOV DS, AX
MOV ES, AX
MOV SI, BLK 1 ADDR
MOV DI, BLK 2 ADDR
```

```

MOV CK, N
CDF    ; DF=0
NEXT :
MOV SB
LOOP NEXT
HLT

```

### Load and store strings : (LOD SB/LOD SW and STO SB/STO SW)

**LOD SB:** Loads a byte from a string in memory into AL. The address in SI is used relative to DS to determine the address of the memory location of the string element.

$(AL) \leftarrow [(DS) + (SI)]$

$(SI) \leftarrow (SI) \pm 1$

**LOD SW :** The word string element at the physical address derived from DS and SI is to be loaded into AX. SI is automatically incremented by 2.

$(AX) \leftarrow [(DS) + (SI)]$

$(SI) \leftarrow (SI) \pm 2$

**STO SB :** Stores a byte from AL into a string location in memory. This time the contents of ES and DI are used to form the address of the storage location in memory

$[(ES) + (DI)] \leftarrow (AL)$

$(DI) \leftarrow (DI) \pm 1$

**STO SW :**  $[(ES) + (DI)] \leftarrow (AX)$

$(DI) \leftarrow (DI) \pm 2$

Mnemonic	Meaning	Format	Operation	Flags affected
MOV SB	Move String Byte	MOV SB	$((ES)+(DI)) \leftarrow ((DS)+(SI))$ $(SI) \leftarrow (SI) \mp 1$ $(DI) \leftarrow DI \mp 1$	None
MOV SW	Move String Word	MOV SW	$((ES)+(DI)) \leftarrow ((DS)+(SI))$ $((ES)+(DI)+1) \leftarrow ((DS)+(SI)+1)$ $(SI) \leftarrow (SI) \mp 2$ $(DI) \leftarrow (DI) \mp 2$	None
LOD SB / LOD SW	Load String	LOD SB/ LOD SW	$(AL) \text{ or } (AX) \leftarrow ((DS)+(SI))$ $(SI) \leftarrow (SI) \mp 1 \text{ or } 2$	None



STOSB/ STOSW	Store String	STOSB/ STOSW	$((ES)+(DI)) \leftarrow (AL) \text{ or } (AX)$ $(DI) \leftarrow (DI) \mp 1 \text{ or } 2$	None
-----------------	-----------------	-----------------	--	------

Example : Clearing a block of memory with a STOSB operation.

```
MOV AX, 0
MOV DS, AX
MOV ES, AX
MOV DI, A000
MOV CX, OF
CDF
```

```
AGAIN :   STO SB
          LOOP NE AGAIN
```

NEXT :

Clear A000 to A00F to 00<sub>16</sub>

### Repeat String : REP

The basic string operations must be repeated to process arrays of data. This is done by inserting a repeat prefix before the instruction that is to be repeated.

Prefix REP causes the basic string operation to be repeated until the contents of register CX become equal to zero. Each time the instruction is executed, it causes CX

to be tested for zero, if CX is found to be nonzero it is decremented by 1 and the basic string operation is repeated.

Example : Clearing a block of memory by repeating STOSB

```
MOV AX, 0
MOV ES, AX
MOV DI, A000
MOV CX, OF
CDF
REP STOSB
NEXT:
```

The prefixes REPE and REPZ stand for same function. They are meant for use with the CMPS and SCAS instructions. With REPE/REPZ the basic compare or scan operation

can be repeated as long as both the contents of CX are not equal to zero and zero flag is 1.

REPNE and REPNZ works similarly to REPE/REPZ except that now the operation is repeated as long as CX≠0 and ZF=0. Comparison or scanning is to be performed as long as the string elements are unequal (ZF=0) and the end of the string is not yet found (CX≠0).

Prefix	Used with	Meaning
REP	MOVS STOS	Repeat while not end of string CX≠0
REPE/ REPZ	CMPS SCAS	CX≠0 & ZF=1
REPNE/REPNZ	CMPS SCAS	CX≠0 & ZF=0

Example :

```

CLD      ; DF =0
MOV AX, DATA SEGMENT ADDR
MOV DS, AX
MOV AX, EXTRA SEGMENT ADDR
MOV ES, AX
MOV CX, 20
MOV SI, OFFSET MASTER
MOV DI, OFFSET COPY
REP MOVSB

```

Moves a block of 32 consecutive bytes from the block of memory locations starting at offset address MASTER with respect to the current data segment (DS) to a block of locations starting at offset address copy with respect to the current extra segment (ES).

### Auto Indexing for String Instructions :

SI & DI addresses are either automatically incremented or decremented based on the setting of the direction flag DF.

When CLD (Clear Direction Flag) is executed DF=0 permits auto increment by 1.

When STD (Set Direction Flag) is executed DF=1 permits auto decrement by 1.

Mnemonic	Meaning	Format	Operation	Flags affected
CLD	Clear DF	CLD	$(DF) \leftarrow 0$	DF
STD	Set DF	STD	$(DF) \leftarrow 1$	DF

### 1. LDS Instruction:

LDS register, memory (Loads register and DS with words from memory)

This instruction copies a word from two memory locations into the register specified in the instruction. It then copies a word from the next two memory locations into the DS register. LDS is useful for pointing SI and DS at the start of the string before using one of the string instructions. LDS affects no flags.

Example 1 :LDS BX [1234]

Copy contents of memory at displacement 1234 in DS to BL. Contents of 1235H to BH. Copy contents at displacement of 1236H and 1237H is DS to DS register.

Example 2 : LDS, SI String – Pointer

$(SI) \leftarrow [\text{String Pointer}]$

$(DS) \leftarrow [\text{String Pointer} + 2]$

DS, SI now points at start and desired string

### 2. LEA Instruction :

Load Effective Address (LEA register, source)

This instruction determines the offset of the variable or memory location named as the source and puts this offset in the indicated 16 bit register.

LEA will not affect the flags.

Examples :

LEA BX, PRICES

Load BX with offset and PRICES in DS

LEA BP, SS : STACK TOP

Load BP with offset of stack-top in SS

LEA CX, [BX] [DI]

Loads CX with EA :  $(BX) + (DI)$

### 3. **LES instruction :**

LES register, memory

Example 1: LES BX, [789A H]

(BX)  $\leftarrow$  [789A] in DS

(ES)  $\leftarrow$  [789C] in DS

Example 2 : LES DI, [BX]

(DI)  $\leftarrow$  [BX] in DS

(ES)  $\leftarrow$  [BX+2] in DS