Unit-II
8086 Microprocessor

Unit-II: 8086 Microprocessor

- Introduction to a Microprocessor.
- Introduction to 8-bit and 16-bit microprocessors.
- 8086- Architecture Functional diagram.
- Register Organization.
- Programming Model.
- Physical Memory Organization.
- Signal descriptions of 8086- common function signals.
- Non-pipelined & Pipelined machine cycle.
- Addressing modes, Instruction Format, instruction set of 8086.
- Introduction to 8086 Assembly Language Programming and a few sample programs of 8086.

- CPU: Fetching, Decoding and Execution
- I/O module
- System Bus: Data, Address, Control
- Relation of Address lines and memory locations (N=16, 2^16= 65,536 or 64K memory locations)
- Combined utilization of system bus:

  E.g: [18F8H] = 20H
- Memory module: Internal or External
- RAM / ROM

- Hardware: Physical device & circuitry
- Software: Programs
- Firmware: Program stored in ROM or other devices which maintain their information permanently

8086 Internal Architecture

Features of 8086

→Supply Voltage= +5V.

→16 bit Microprocessor.

→20 bit Address bus=1MB memory locations.

→Generates 16 bit I/O address=64K I/O ports.

→Provides 14 sixteen bit registers.

→Multiplexed address and data bus.

→Clock with 33% duty cycle.

→5MHz clock for 8086.

→8MHz clock for 8086-2.

→10MHz clock for 8086-1.

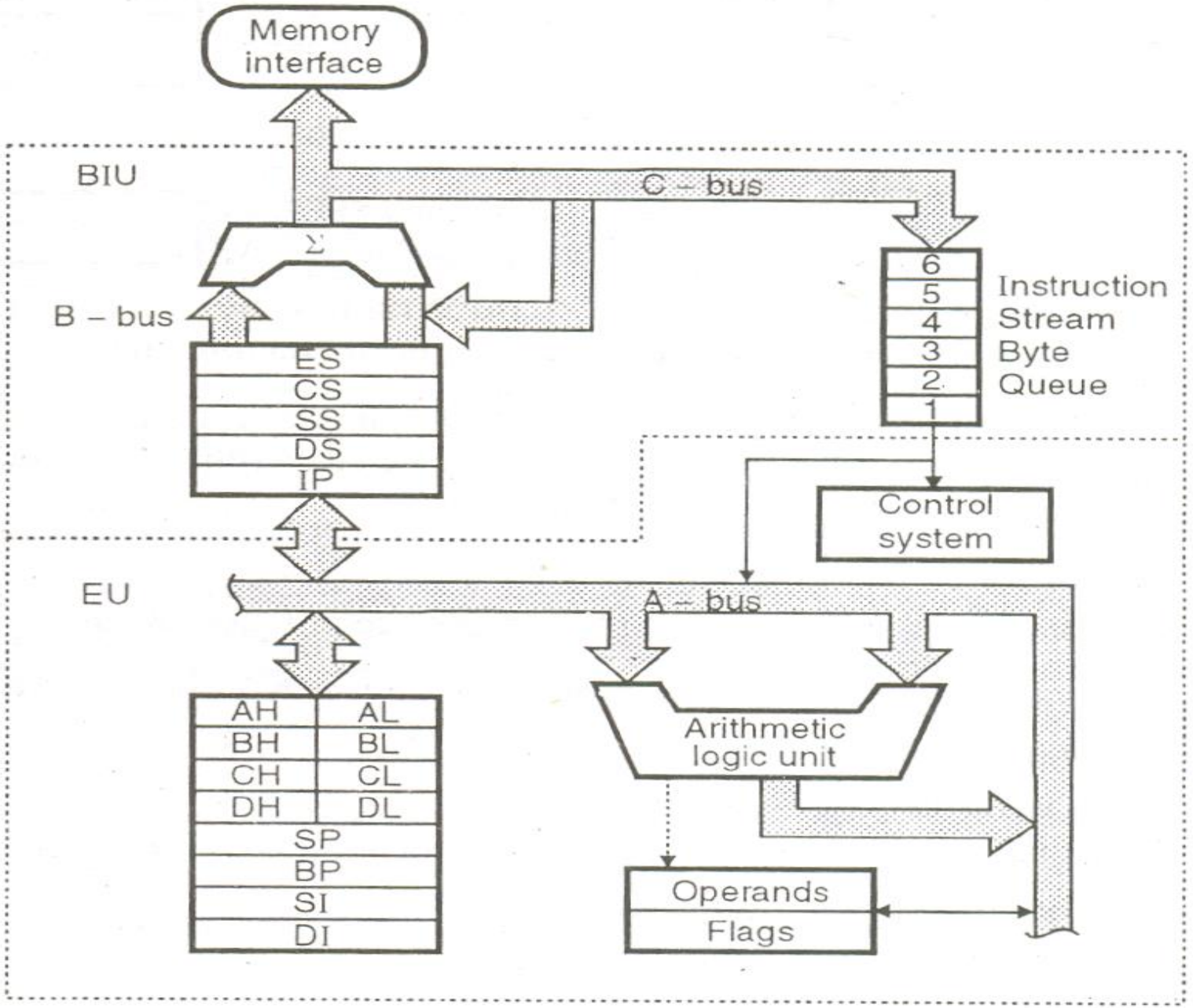
→Performs bit, byte word and block operations.

→Operates in two modes: Minimum and Maximum

→Supports Multiprogramming.

→6 Instruction byte queue.

8086 Internal Block Diagram

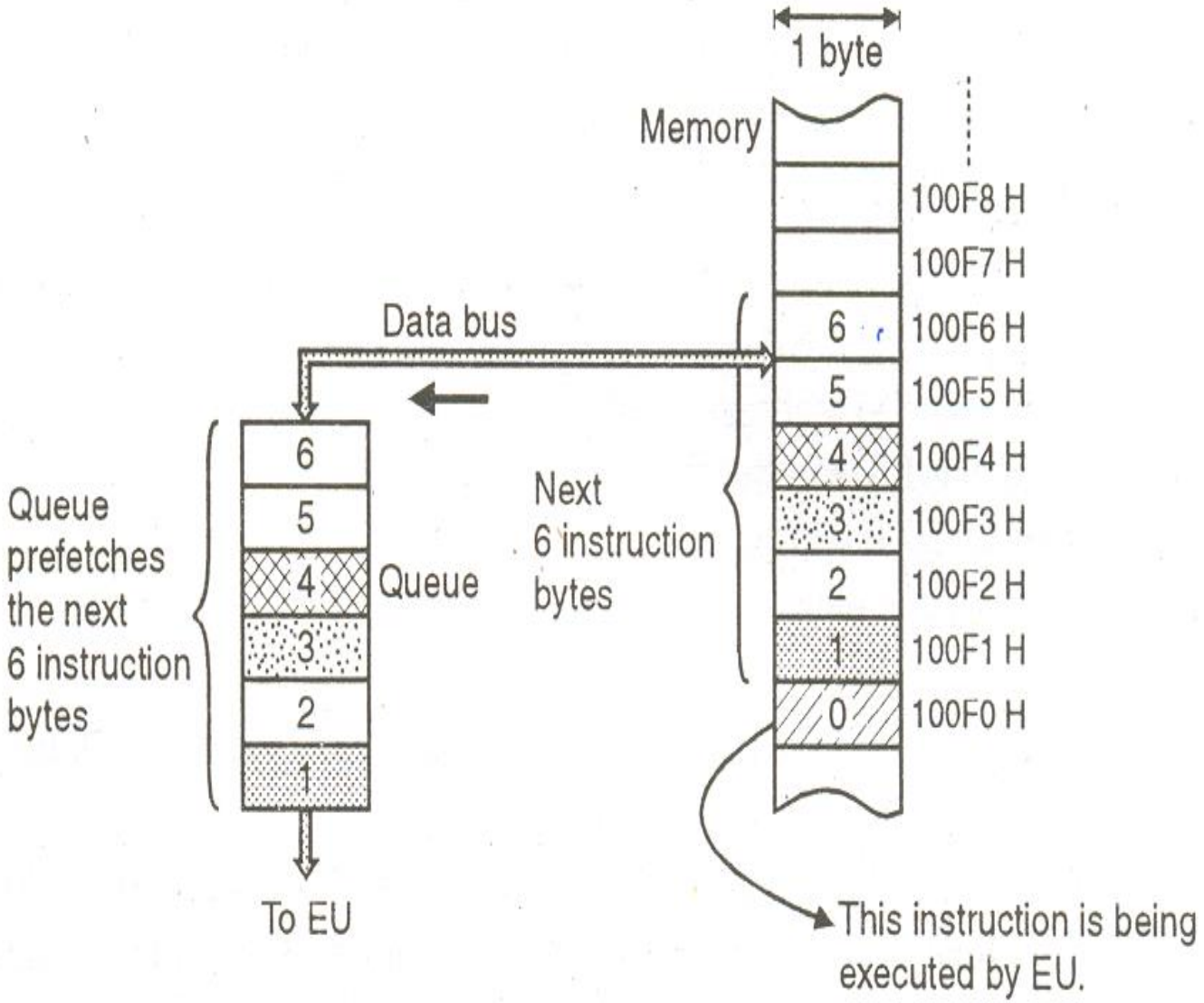


- BIU and EU are two functional units can work to increase system speed and hence throughput.
- Throughput is a measure of number of instructions executed per unit time.

Bus Interface Unit

- →Provides a full 16 bit bidirectional data bus and 20 bit address bus.
- →It sends address of the memory or I/O.
- →It fetches instructions from memory.
- →It reads data from memory/port.
- →It writes data into memory/port.
- →It supports instruction queuing.
- →It provides address relocation facility.

Significance of Instruction Queue



Pipelining

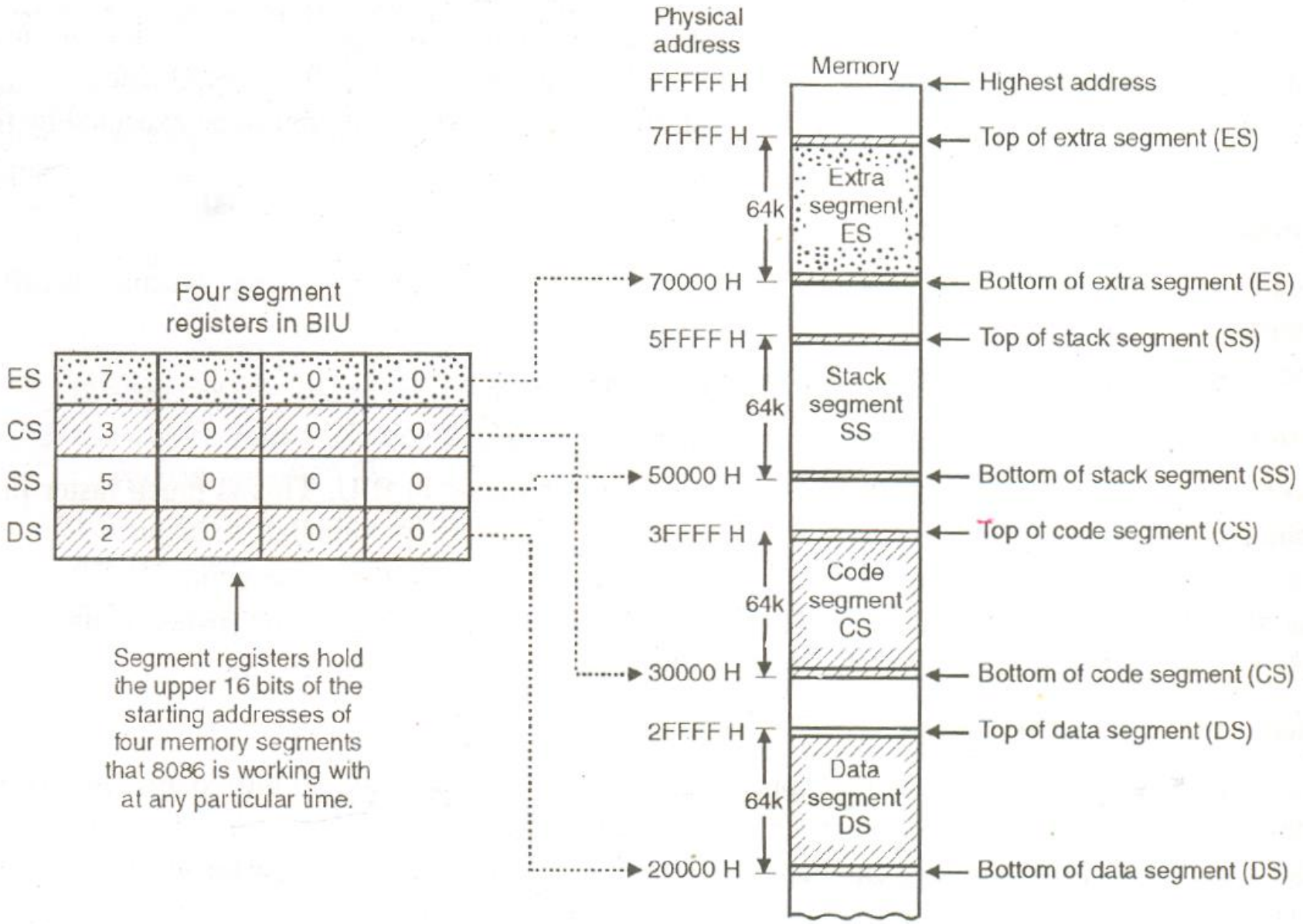
- The process of fetching the next instruction when the present instruction is being executed is called as pipelining.
- Pipelining has become possible due to the use of queue.
- Eliminates the waiting time of EU and speed up the processing.

- **Queue in JMP and CALL instructions?**
- **Why is the 8086 queue only 6 byte long?**

Segment Registers

- PA is 20 bits to access 1MB memory locations.
- Registers are of 16 bits only.
- 1MB memory is segmented into 64KB segments and 4 segments are active at a time.

Segment registers



Rules for segmentation.

- 4 segments can overlap for small programs.
- In a min. system, all 4 segments can start at the address 00000H.
- Segment can begin at any memory address which is divisible by 16.

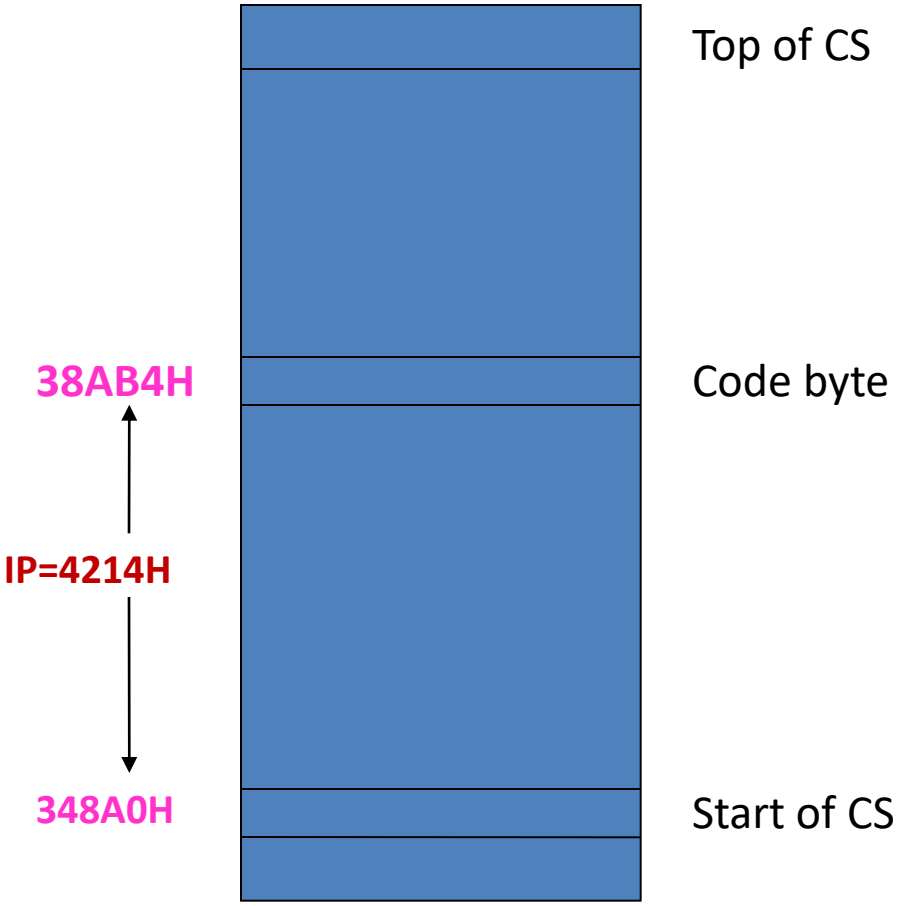
Advantages of segmentation

- It allows the mem. to 1MB even though the address associated with individual instruction is only 16 bit.
- It allows instruction code, data, stack and portion of program to be more than 64KB long by using more than one code, data, stack segment and extra segment.

Advantages of segmentation (cntd…)

- Facilitates use of separate memory areas for program, data and stack.
- Permits a program or its data to be put in different areas of memory, each time the program is executed i.e program can be relocated which is very useful in multiprogramming.

Instruction Pointer.



Base Address =348A0H

IP = 4214H

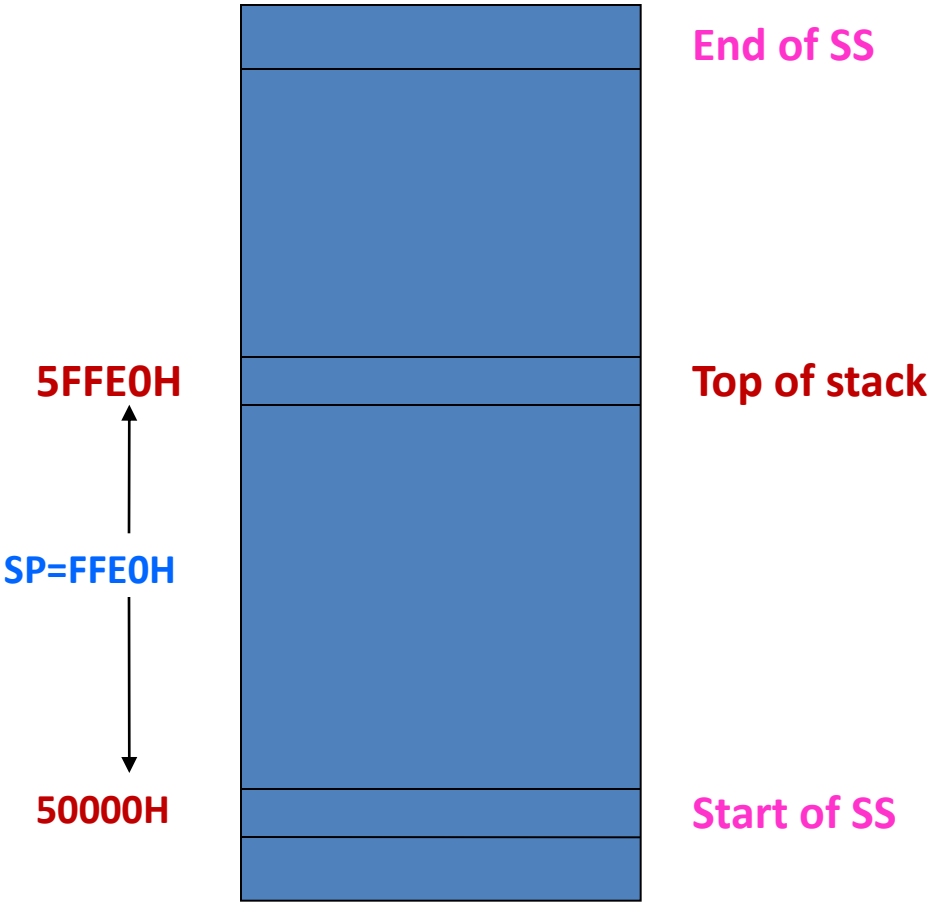
Physical Address=38AB4H

Segment Base : Offset

348A : 4214

CS : IP

Execution Unit:
Stack Segment & Stack Pointer



Base Address =50000H

SP = FFE0H

Physical Address=5FFE0H

Segment Base : Offset

5000 : FFE0

SS : SP

Pointers and Index Registers in EU

- Base Pointer BP.
- Source Index SI.
- Destination Index DI.

→Can be used as a general purpose registers and used to hold the 16 bit offset of data word in one of the segments.

Source Index Register

- Used for holding the offset of a data word in the data segment.
- PA Generation:

DS Reg. → 2 0 0 0 0

SI(offset)→ + 1 F 2 3

PA in DS→ 2 1 F 2 3

Base Pointer Register

- Used for holding the offset relative to stack segment.
- BP is used whenever we pass a parameter by way of stack.
- Use as an offset register in the addressing mode called Base addressing mode.

Destination Index register

- Used for holding 16 bit offset of a data word in Extra segment.
- SI and DI registers are used for the string related instructions.
- E.g: [3000H]= 10, [3001H]=20, [3002H]=30

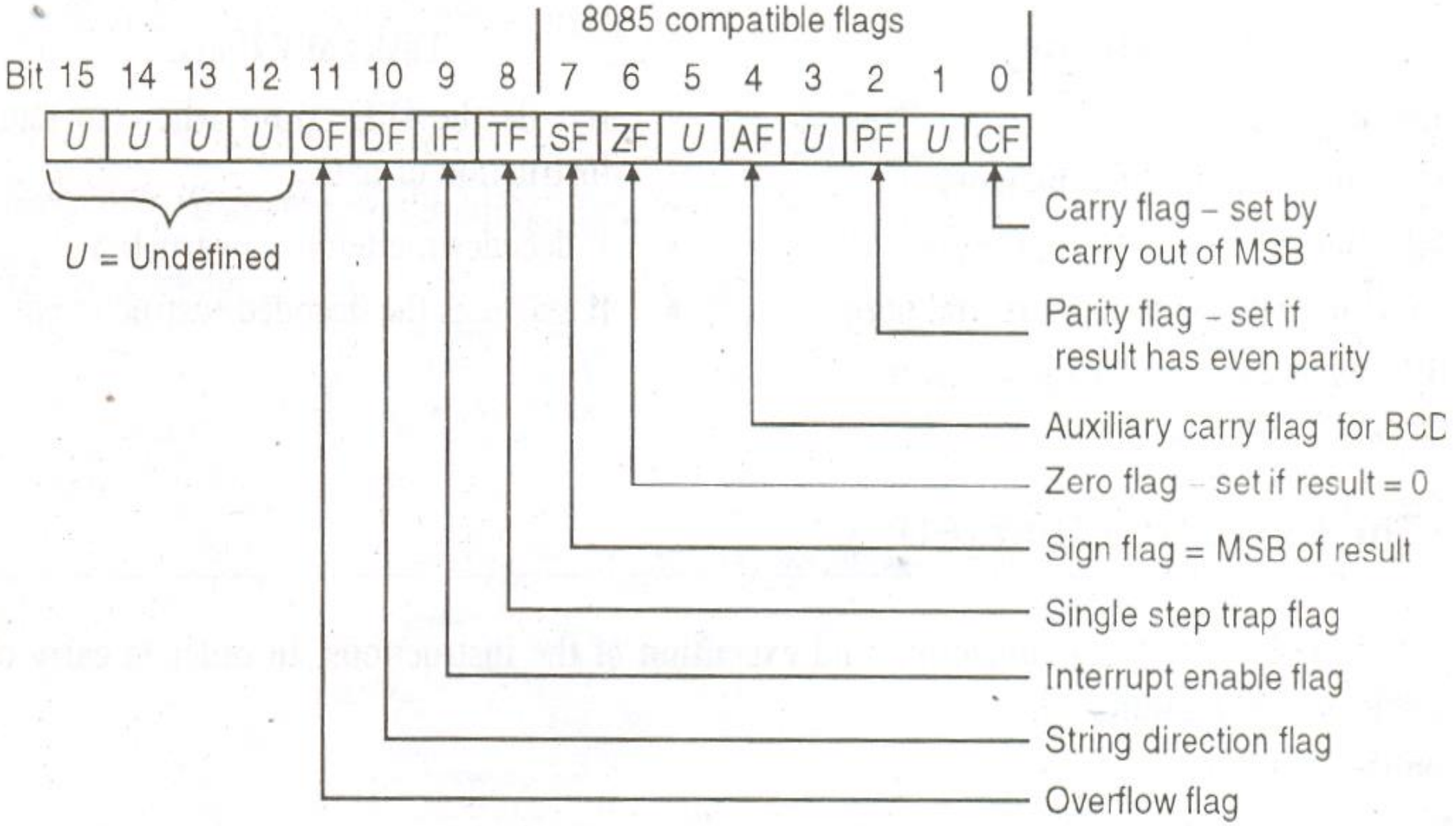
  [4000H], [4001H], [4002H]

→ Memory to Memory transfer

- The main function of EU is decoding and execution of instructions.
- Different units of EU:

- Control Unit
- Decoder.
- ALU.
- Flag Registers.
- General purpose Registers.

Flag Registers



Flags

- **Condition Flags**

→**OF**

→**SF**

→**ZF**

→**AF**

→**PF**

→**CF**

- **Control Flags**

→**DF**

→**IF**

→**TF**

FLAG	SET	RESET
Overflow	OV	NV
Direction	DN	UP
Sign	NG	PL
Interrupt	EI	DI
Zero	ZR	NZ
Carry	CY	NC

Control Flags

- **Trap Flag (TF):**

  1→Trap ON, CPU generates internal interrupt after each instruction

  0→ Trap OFF.

- **Interrupt Flag (IF):**

  1→Enabled,CPU recognize external (maskable) interrupt request

  0→ Disabled

- **Direction Flag (DF):** used for string instructions(SI & DI Registers)

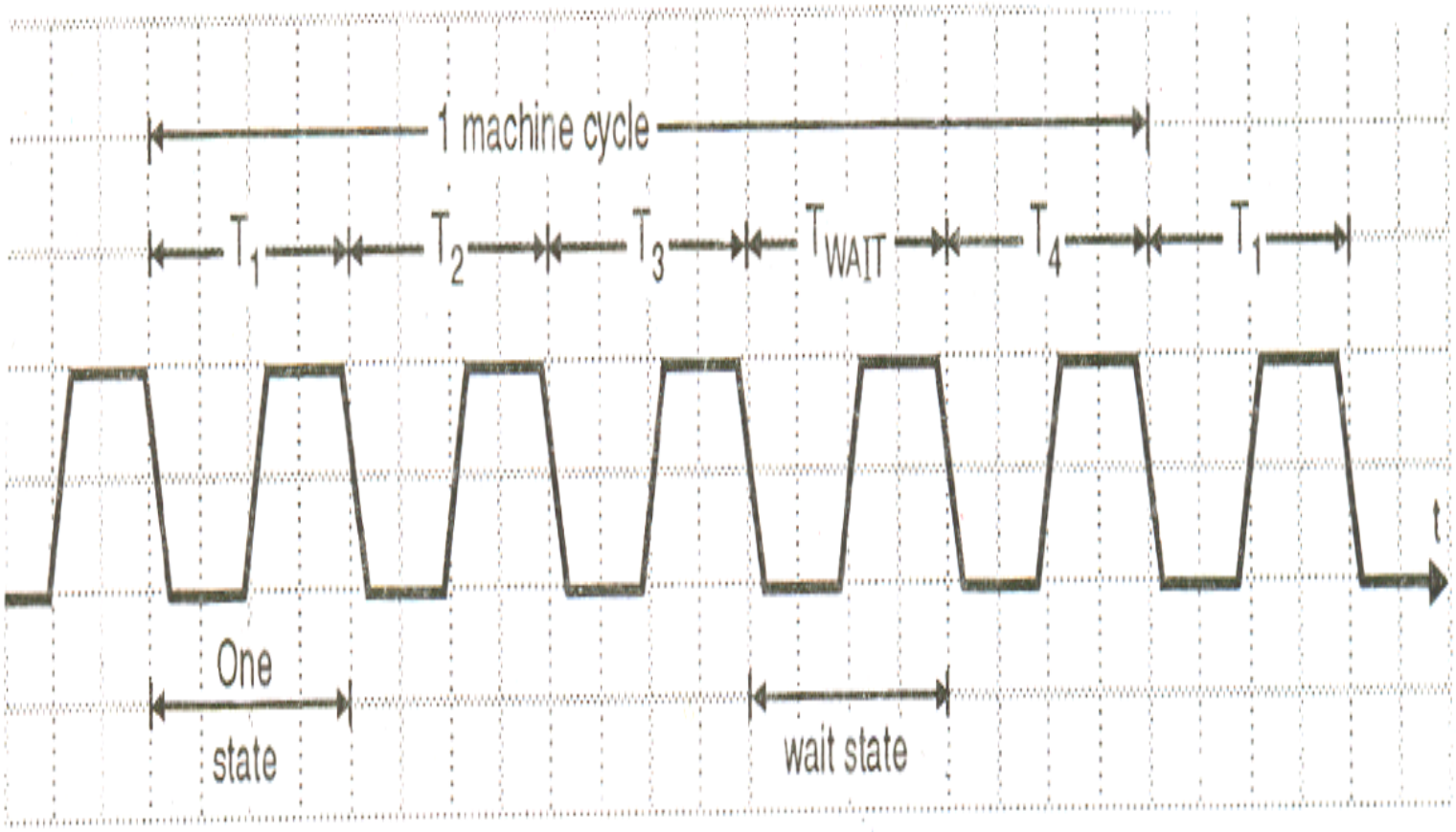
  1→Down,process string from higher address to lower address.

  0→UP, process string from lower address to higher address.

General Purpose Register

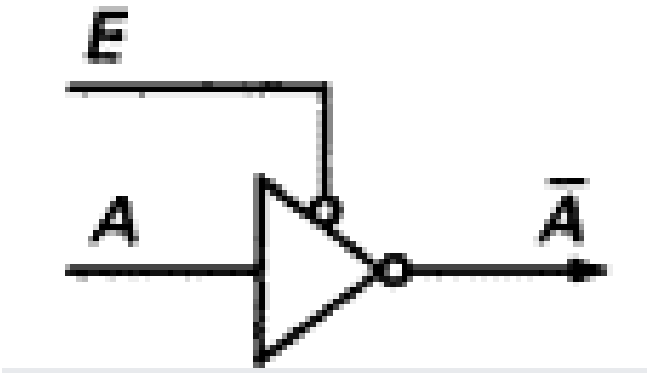
- AX→AH & AL (Accumulator)
- BX→BH & BL(for indirect addressing)
- CX→CH & CL(Count register)
- DX→DH & DL(Data register): used together with AX for word size MUL and DIV operations, it also hold the port no. for the IN & OUT instructions.
- Why to use general purpose registers inside EU?

8086 Timing Diagrams
Clock, State, M/c Cycle, Instruction Cycle



Tri state Gate:

useful in implementing a Bus structure

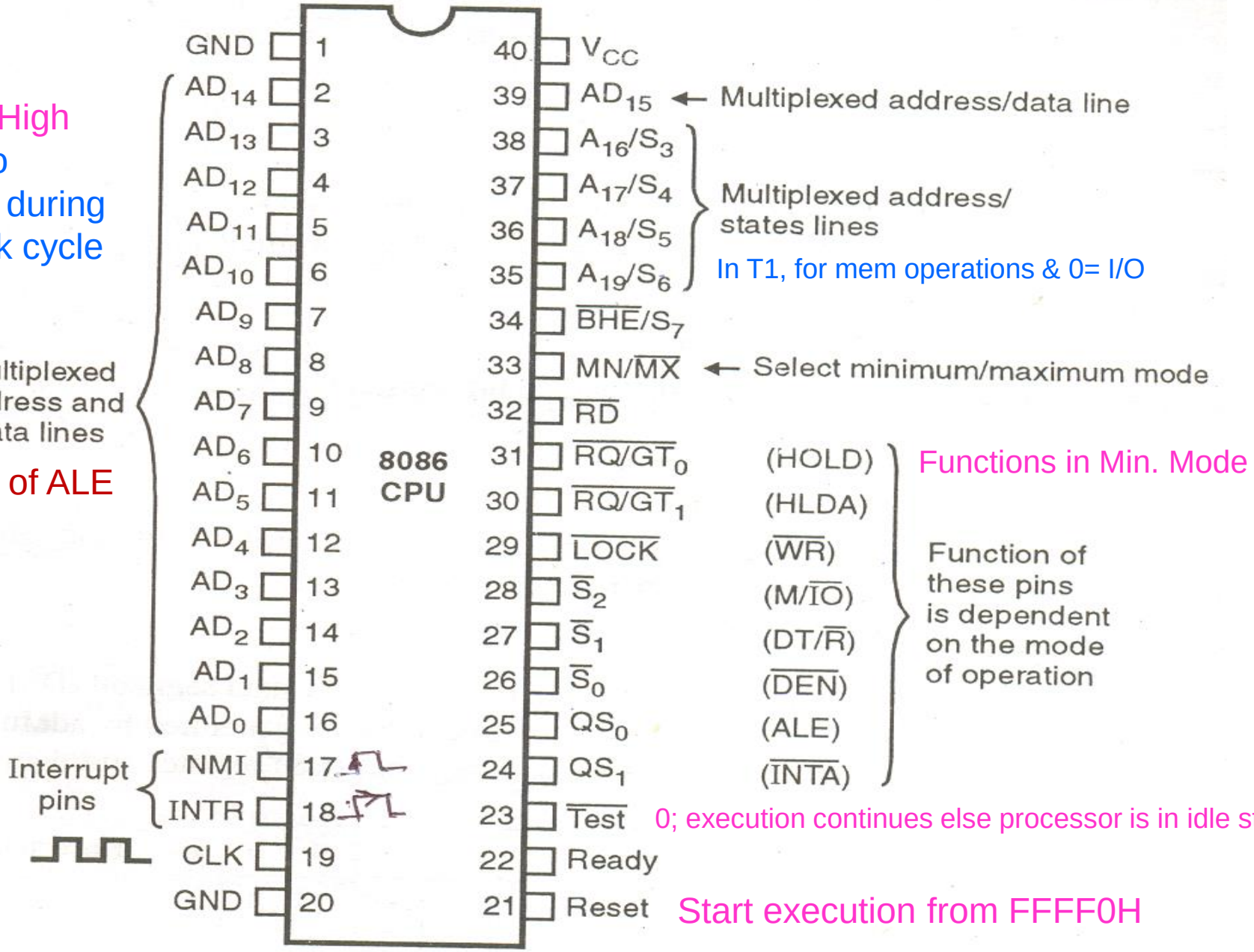


E	A	Y=A’
0	0	0
0	1	1
1	0	Tristate
1	1	Tristate

8086 Signals

- Signal having common functions in Minimum and Maximum mode
- Signal having special functions for Minimum mode
- Signal having special functions for Maximum mode

8086 Pin Diagram



Active High
Float to tristate during INT.Ack cycle

Multiplexed address and data lines

Role of ALE

Interrupt pins

← Multiplexed address/data line

Multiplexed address/ states lines

In T1, for mem operations & 0= I/O

← Select minimum/maximum mode

Functions in Min. Mode

Function of these pins is dependent on the mode of operation

0; execution continues else processor is in idle state

Start execution from FFFF0H

Significance of status line S4 and S3

S4	S3	Indications
0	0	Alternate Data
0	1	Stack
1	0	Code or None
1	1	Data

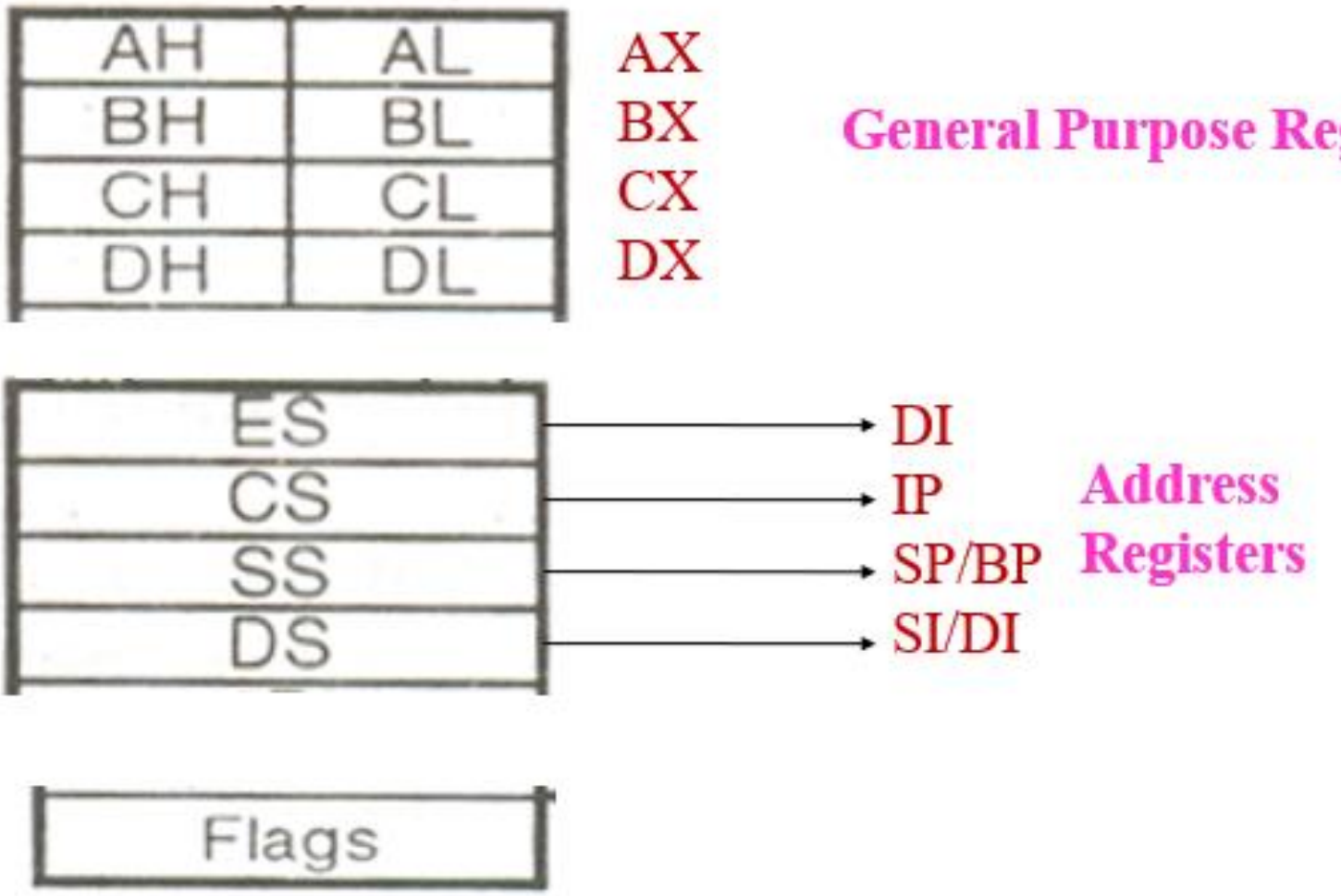
Relation between BHE’ and A0

BHE’	A0	Indications
0	0	Whole word: Read/Write word from even address
0	1	Upper byte: Read/Write word from odd address
1	0	Lower byte: Read/Write word from even address
1	1	None: No read/Write operation

Programmer's model of 8086

- The register organization of 8086/8088 which is accessible for the user / programmer is known as programmer's model of 8086.

Programmer's model of 8086



Addressing Modes

- The different ways that a processor can access data are referred to as Addressing Modes.
- The classification is:

→Data Addressing mode

→Program memory Addressing mode.

→Stack memory Addressing mode

Addressing Modes

- Register; Immediate
- Memory:-

→Direct →Reg.Indirect →Based →Indexed →Based Indexed

- Relative:-

→Reg. relative→Relative based→Relative Indexed → Relative based Indexed

- String Addressing; Implied Addressing
- I/O Addressing:-

→Direct port addressing

→Indirect port addressing

Data Addressing modes

- For accessing immediate and register data
- For accessing data in memory
- For accessing I/O ports

Addressing mode For accessing immediate and register data

- Register Addressing mode:

  MOV AL,BL

→Both source and destination operands are in 8086 register.

- Immediate Addressing mode:

MOV AL,20H

Addressing modes for accessing data in memory

- When EU needs to access memory locations,it sends an offset value to the BIU.
- This offset is called EA(Effective Address).
- EA is displacement of the desired location from the segment base.
- The BIU generates 20 bit PA .

- There are 6 ways to specify EA in instruction:-

→Direct Addressing

→Reg. Indirect Addressing

→Based Addressing

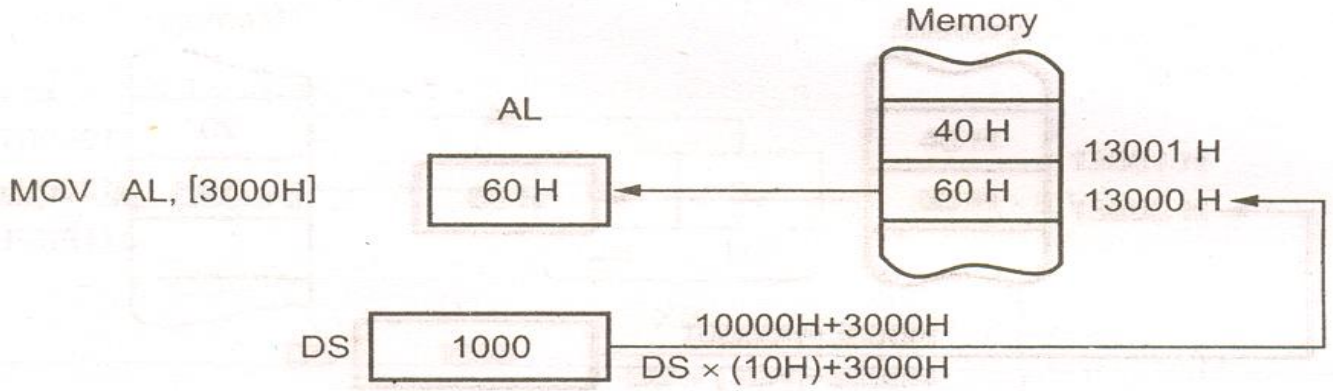
→Indexed Addressing

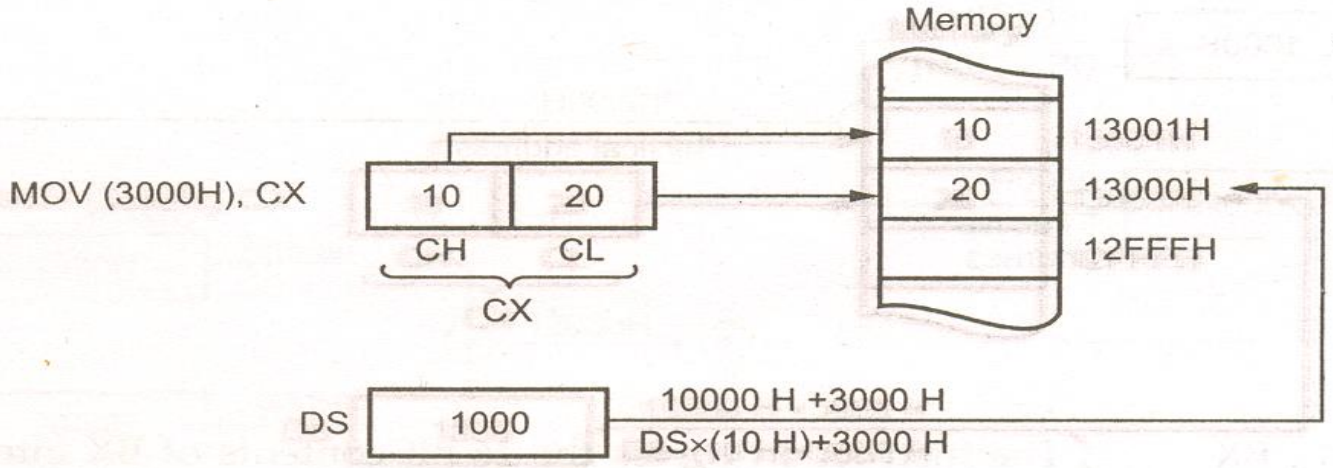
→Based Indexed Addressing

→String Addressing

Direct Addressing Mode

- EA= 8/16 bit displacement
- PA= Segment: EA
- PA= { CS, DS, ES, SS } : { Direct Address}
- By default it is DS register.





Note : 1. Assume DS = 1000

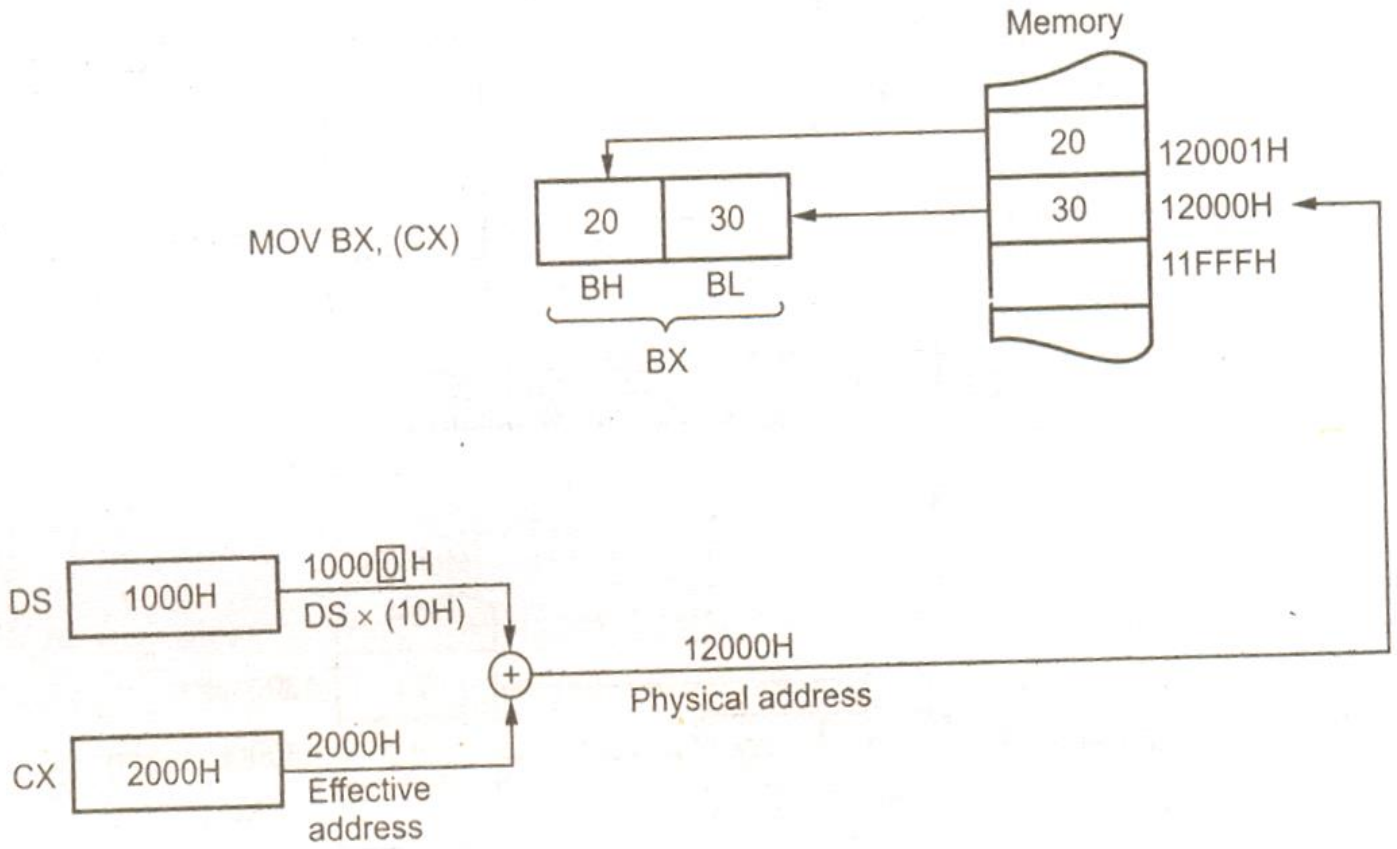
$\therefore$ Physical address = DS $\times$(10H)+ 3000H

= 1000[0] + 3000H = 13000H

2. Arrow indicates direction of data flow.

Register Indirect Addressing mode

- EA= {BX,BP,SI,DI}
- PA= Segment: EA
- PA= { CS, DS, ES, SS } : {BX,BP,SI,DI}
- By default it is DS register.


Memory
20
30
120001H
12000H
11FFFH
MOV BX, (CX)
20
30
BH
BL
BX
DS
1000H
10000H
DS × (10H)
12000H
Physical address
CX
2000H
2000H
Effective address

- The advantage of this type of addressing mode is, one instruction can operate on many different memory locations if the value in the base or index register is updated appropriately.

Based Addressing Mode

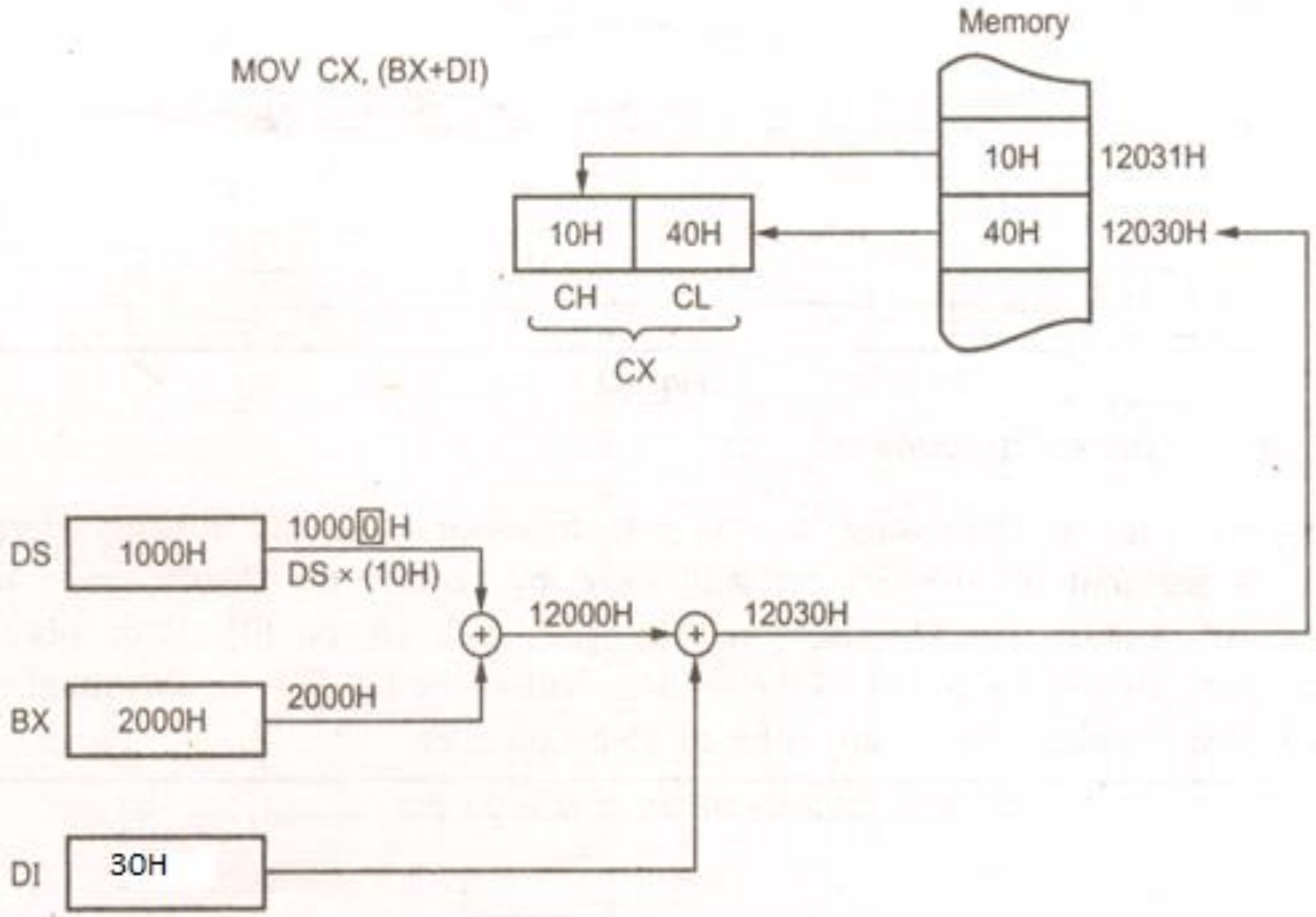
- EA= {BX,BP}
- PA= Segment: EA
- PA= $\left\{\begin{array}{l} CS \\ DS \\ ES \\ SS \end{array}\right\}$: {BX,BP}
- By default it is DS register.
- E.g: MOV AX,[BX]

Indexed Addressing Mode

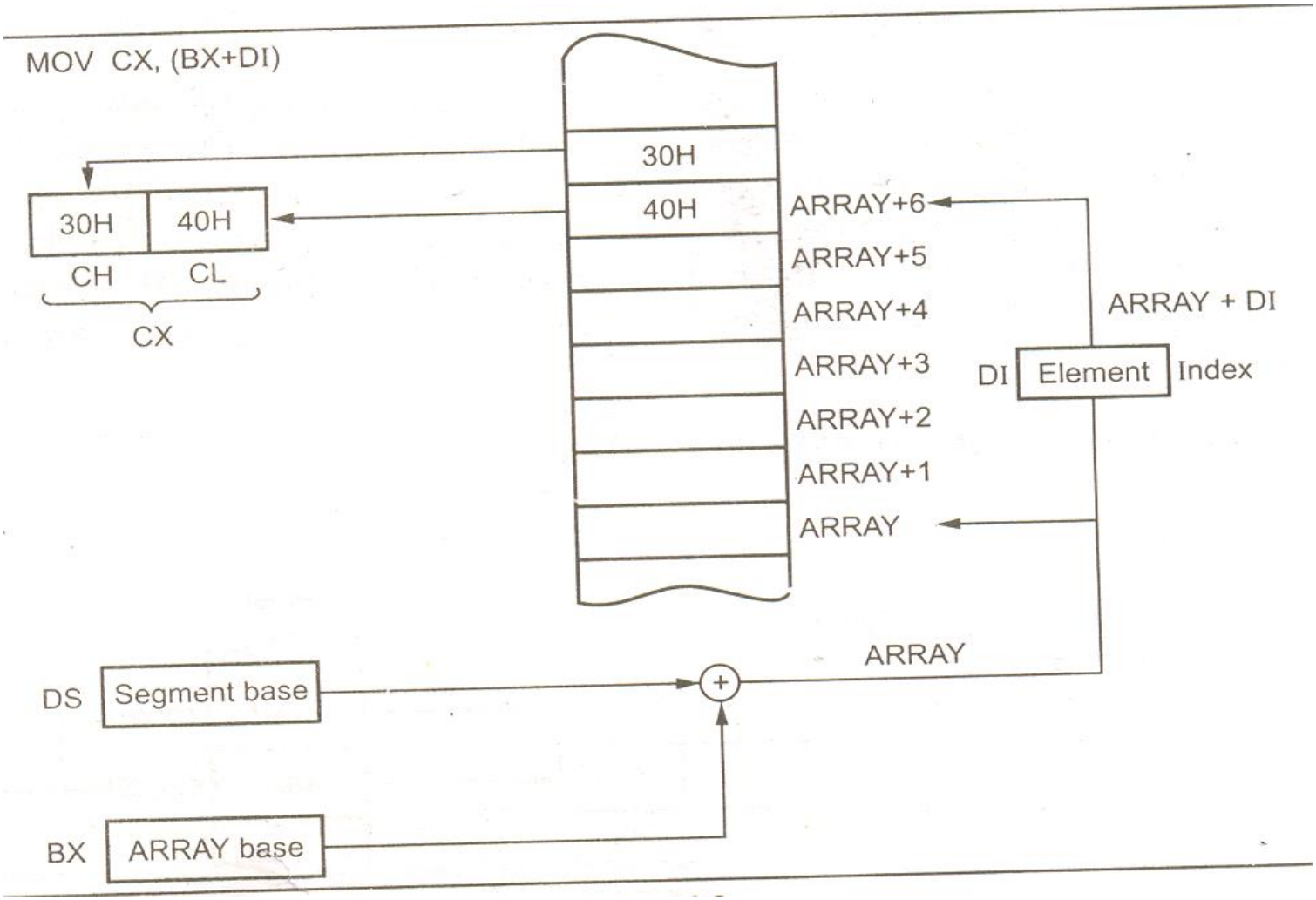
- EA= {SI,DI}
- PA= Segment: EA
- PA= $\left\{\begin{array}{l} CS \\ DS \\ ES \\ SS \end{array}\right\}$: {SI,DI}
- By default it is DS register.
- E.g: MOV [SI],AL

Based Indexed Addressing Mode

- It is similar to indirect addressing.
- This addressing uses one base reg. and one index reg. to indirectly address memory.
- The base reg. holds the beginning location of a memory array, while the index reg. holds the relative position of an element in the array
- BP addresses the memory data, the contents of SS, BP and Index reg. are used to generate PA

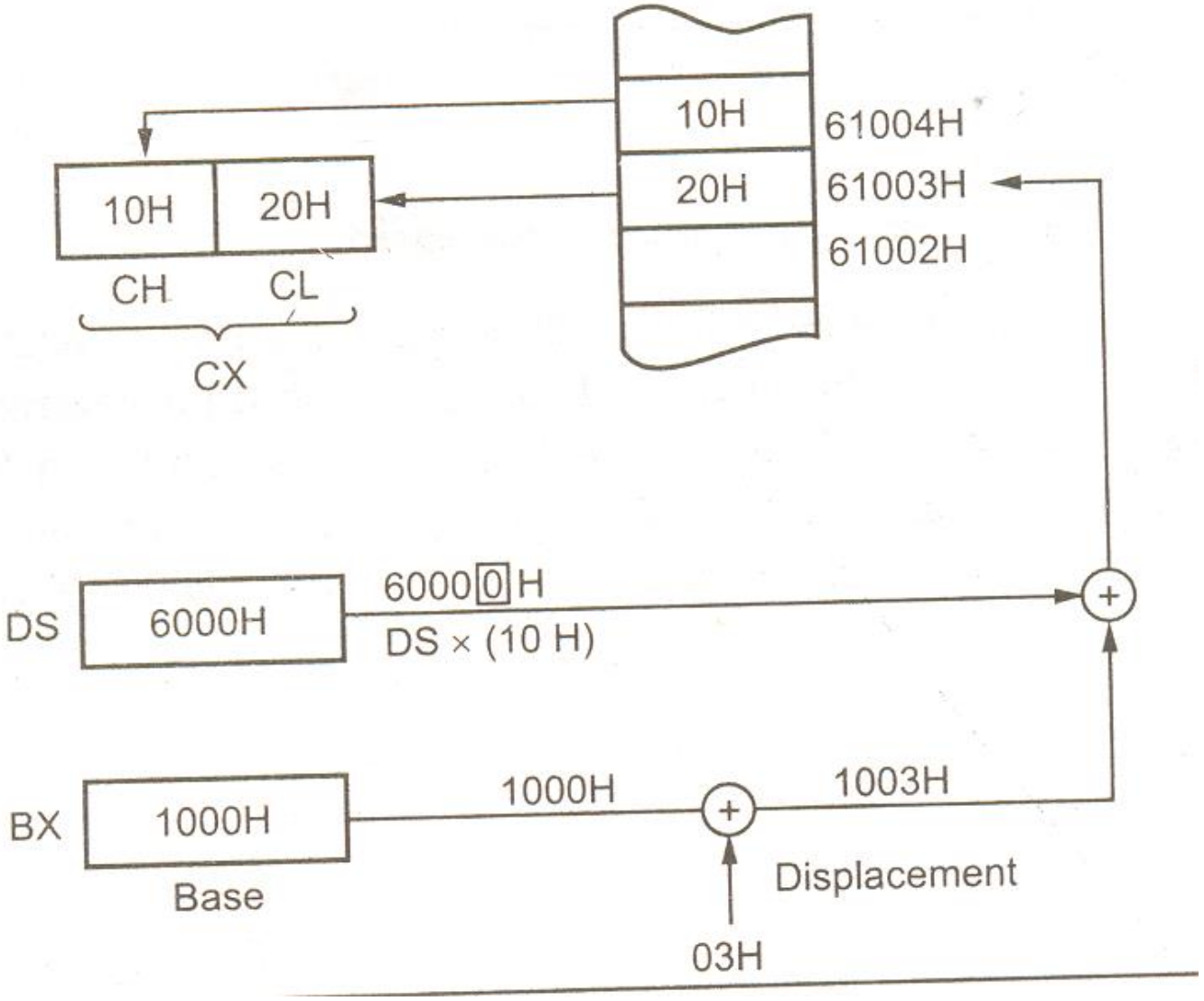

MOV CX, (BX+DI)
Memory
10H
12031H
40H
12030H
10H
40H
CH
CL
CX
DS
1000H
10000H
DS × (10H)
12000H
12030H
BX
2000H
2000H
DI
30H

Locating array data using based indexed Addressing



Register Relative Addressing mode

- It is similar to based-indexed addressing
- Displacement(8 bit/ 16 bit) should be added to the reg. within [ ].
- EA= { BX, BP, SI, DI } + {8 bit / 16 bit displacement}
- PA= segment: EA
- E.g: MOV CX,[BX+0003H]


10H
20H
CH
CL
CX
10H
20H
61004H
61003H
61002H
DS
6000H
6000[0]H
DS × (10 H)
BX
1000H
Base
1000H
1003H
Displacement
03H

- Displacement can be subtracted from the register. MOV AL,[DI-2]
- Displacement can be offset address appended to the front of the [ ] :MOV AL,OFF_ADD[DI+4]

Relative based addressing

- EA={BX, BP}+{8/16 bit displacement}
- PA={DS,SS}:{BX, BP}+{8/16 bit displacement}
- E.g: MOV CX,START[BX]

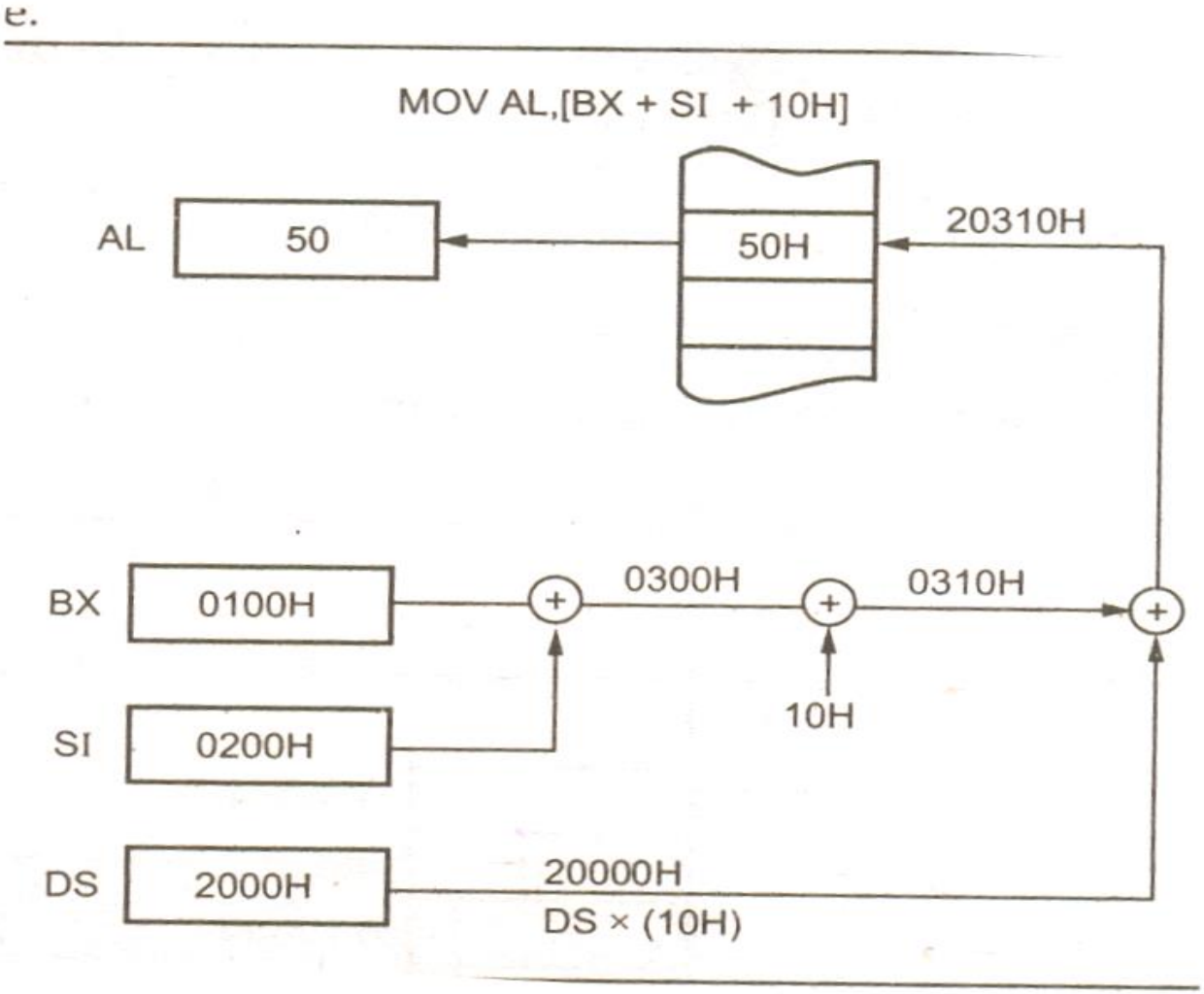
Relative Indexed addressing

- EA={SI,DI}+{8/16 bit displacement}
- PA={DS,SS}:{SI,DI}+{8/16 bit displacement}
- E.g: MOV CX,COUNT[DI]

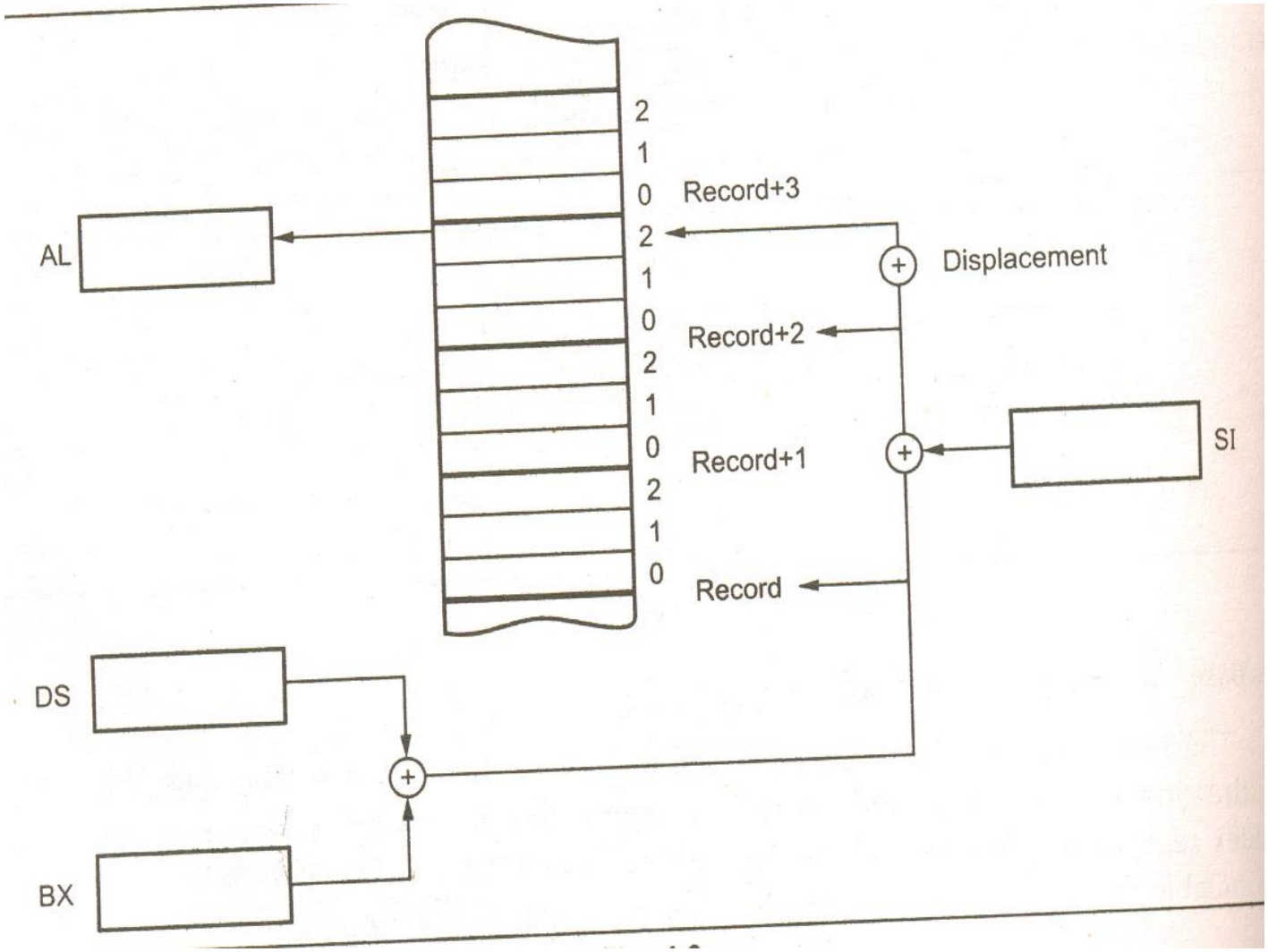
Relative Based – Indexed Addressing Mode

- It is similar to based – indexed addressing mode, but it adds displacement, besides using a base register and an indexed register to generate a PA.
- This addressing mode is suitable to address data within the two dimensional array.
- EA={BX,BP}+{SI,DI}+{8/16 bit displacement}
- PA={DS,SS}:{BX,BP}+{SI,DI}+{8/16 bit displacement}

Addressing data with relative based indexed



Addressing arrays with relative based indexed



String Addressing modes

- MOVS BYTE;

DF=0;[DS]=3000H,[SI]=0600H,[ES]=5000H,

[DI]=0400H,[30600H]=38H AND [50400H]=45H

After execution;

[50400H]=38H [SI]=0601 AND [DI]=0401

Addressing modes for I/O

- Memory mapped I/o
- I/o mapped I/o:

E.g: OUT 05H,AL

IN AL,80H

8 / 16 bit I/O transfer must take place via AL and Ax respectively.

Implied Addressing Mode

- XLAT, CMA,STC,STD

- Overflow and under flow of stack?

INSTRUCTION SET OF 8086.

INSTRUCTION SET OF 8086/8088.

- Data copy/transfer instructions
- Arithmetic & logical instructions
- Branch instructions
- Loop instructions
- Machine control instructions
- Flag manipulation instructions
- Shift & rotate instructions
- String instructions

Rules For Execution Of an Instruction

- Source & Destination can not be memory locations

  e.g: MOV[1000],[1200]

- Destination can not be immediate no.

  e.g:MOV 592Fh,BX

- The source& destination must be of type byte or word.

e.g: MOV AX,BX

- It can not copy value of one segment reg. to another segment reg.(copy to general reg. first)

e.g: MOV DS,CS {Not allowed}

- It can not copy immediate value to segment reg.

e.g: MOV CS,5487H

- It can not set the value of CS & IP reg.

DATA TRANSFER INSTRUCTIONS

- MOV Instruction to transfer byte or word
- PUSH/POP instruction
- Load Effective Address instruction
- String Data transfer instruction
- Miscellaneous data transfer instruction

MOV INSTRUCTION

- Mnemonic: MOV Destination,Source.

  MOV operand1, operand2.

- Operation: Destination ← Source(Word/Byte)

- Valid source & Destination operands are:

Memory-Accumulator,	Accumulator-Memory,
Register-Register,	Register-Memory,
Memory-Register,	Register-Immediate,
Memory-Immediate,	SegReg-Reg16,
SegReg-mem16,	Reg16-SegReg,
Memory-SegReg	

MOV MEMORY,ACCUMULATOR

- Operation: Memory ← Accumulator
- E.g.: MOV [SI],AL

  Copies the contents of AL reg. to memory location whose offset in DS is stored in SI reg.
- Flags: No flags are affected.
- Addressing Mode: Register Indirect

MOV ACCUMULATOR,MEMORY

- Operation: Accumulator ← Memory
- E.g.: MOV AX,TEMP_RESULT

  The contents of memory location TEMP_RESULT will be transferred to AL reg., then the IP will be incremented by 1 & contents of mem. location after TEMP_RESULT will be copied to AH reg.
- Flags: No Flags are affected.
- Addressing Mode: Register Direct

MOV Destination Register, Source Register

- Operation: Destination Reg. ← Source Register
- E.g.: MOV AX,BX

The LSB of BX i.e. BL is copied to AL & the MSB of BX i.e. BH is copied to AH.

- Flags: No Flags are affected.
- Addressing Mode: Register

MOV Register, Memory

- Operation: Register ← Memory
- E.g.: MOV CX,Count[DI]

  It copies the contents of 2 mem. Locations to the CX reg. contents of 1st location are copied to CL & contents of the next location to CH reg.

  EA=Offset of Count + Offset in [DI]
- Flags: No Flags are affected
- Addressing Mode: Indexed/Register Relative

MOV Memory, Register

- Operation: Memory ← Register
- E.g.: MOV Count[DI], CX

  It copies the contents of CX to 2 mem. Locations. contents of CL are copied to 1st location & contents of CH to the next location
  EA=Offset of Count + Offset in [DI]
- Flags: No Flags are affected
- Addressing Mode: Indexed/Register Relative

MOV Register, Immediate

- Operation: Register ← Immediate data
- E.g.: MOV CL,02H

  It copies the Immediate data 02H to the Destination register CL.
- Flags: No Flags are affected
- Addressing Mode: Immediate

MOV Memory, Immediate

- Operation: Memory ← Immediate data
- E.g.: MOV Count[DI], 2DH

  EA=Offset of Count + Offset in [DI]
- Flags: No Flags are affected
- Addressing Mode: Immediate

MOV Seg. Reg., Reg-16

- Operation: Seg Reg. ← 16 bit Reg.
- E.g. : MOV DS, AX

1. It copies the 16 bit reg. contents AX to the destination seg reg. DS
2. This can not be used to copy the contents of CS reg & only used for 16 bit transfer.

- Flags: No Flags are affected.
- Addressing Mode: Register

MOV Seg. Reg., Mem-16

- Operation: Seg Reg. ← 16 bit Mem.
- E.g. : MOV DS, [SI]

This can not be used to copy the contents of CS reg & only used for 16 bit transfer.

- Flags: No Flags are affected.
- Addressing Mode: Indexed

MOV Reg-16, Seg. Reg.

- Operation: 16 bit Reg. ← Seg Reg.
- E.g. : MOV AX, DS

  It copies the contents of seg reg DS to the destination reg AX
- Flags: No Flags are affected.
- Addressing Mode: Register

MOV Memory,Seg. Reg.

- Operation: Memory ← Seg Reg.
- E.g. : MOV Count,DS

  It copies word from data seg. Reg. to 2 mem. locations, whose offset of displacement in DS represented by Count
- Flags: No Flags are affected.
- Addressing Mode: Register direct

PUSH

- Used to load data to Stack Memory.
- Decrements SP by 2 & copies a word from source to the location in the stack where SP points.
- The Source must be a word (16 bit).

1. MSB data byte →SS mem location addressed by SP-1.
2. LSB data byte →SS mem location addressed by SP-2.

- E.g.: PUSH CX
- Flags: No Flags are affected.
- Addressing Mode: Register

POP

- Used to read data from Stack Memory.
- Increments SP by 2 & copies a word from the location in the stack where SP points to the destination.
- The destination can be a general purpose reg., a seg. Reg. or a mem. location
  1. MSB data byte ← SS mem location addressed by SP.
  2. LSB data byte ←SS mem location addressed by SP+1.
- E.g.: POP CX
- Flags: No Flags are affected.
- Addressing Mode: Register

Initializing the Stack

- Method 1:

```
ASSUME CS:CODE,DS:DATA,SS:STACK
|
|
 STACK SEGMENT
 S_DATA DB 100 DB(?)
 STACK ENDS
```

- Method 2:

```
.Stack [size]
.Stack 100
```

XCHG : EXCHANGE

- Operation: XCHG Destination, Source

  Destination ←→ Source
- It exchanges the contents of a reg. with the contents of another reg. or the contents of the reg. with the contents of a memory locations
- It can not exchange the contents of two mem. locations.
- The Source & Destination both must be words or bytes. The Seg. Reg. can not be used in these instructions.
- E.g.: 1.XCHG BX,CX

  2. XCHG AL,SUM[BX];{E.A= SUM + [BX]}
- Flags: No Flags are affected.
- Addressing mode: Implied

XLAT : TRANSLATE

- Translate Byte in AL.
- Replaces a byte in AL with byte from a lookup table in memory.
- BX reg. store the offset of the starting address of lookup table & AL reg. stores the byte no. from the lookup table. This instruction copies byte from address pointed by [BX+AL] back in to AL.
- E.g: AL=20H, [25276]=30H,[25277]=90H

  After Execution of XLAT, AL= 30H(Contents of lookup table are transferred in to Al reg).
- Flags: No Flags are affected.
- Addressing mode: Implied

LEA : LOAD EFFECTIVE ADDRESS

- Mnemonic: LEA Register,Source
- Operation: Register ← Source
- This instruction determines the offset of the variable or mem. Location named as the source & loads this address in the specified 16 bit reg.
- Normally the offset is loaded into index reg. Or base pointer reg. Such as SI,DI,BX,BP.
- E.g. :

1. LEA CX,TOTAL; Load CX with offset of TOTAL in DS.
2. LEA BP,SS:STACK_TOP
3. LEA AX,[BX][DI];load AX with EA=[BX]+[DI]

- Flags: No Flags are affected.
- Addressing mode: Register Direct

LDS : LOAD POINTER TO DS

- Mnemonic:LDS register,source
- Operation: Register←Source;(DS)←(Source+2)
- Load reg & DS with words from memory.
- It copies a word from 2 mem. locations into the reg. Specified in the instruction.It then copies a word from the next 2 mem. locations in to the DS reg.
- The source can not be a register.
- E.g.: LDS CX,[391AH] ;It copies the contents of mem. at displacement of 391AH & 391BH to CX. Then it copies the contents at displacement of 391CH & 391DH in DS.
- Flags: No Flags are affected.
- Addressing mode: Register Direct

LES : LOAD POINTER TO ES

- Mnemonic:LES register,source
- Operation: Register←Source;(ES)←(Source+2)
- Load reg & ES with words from memory.
- It copies a word from 2 mem. locations into the reg. Specified in the instruction.It then copies a word from the next 2 mem. locations in to the ES reg.
- The source can not be a register.
- E.g.: LES CX,[3483H] ;It copies the contents of mem. at displacement of 3483H in DS to CL & 3484H to CH. Then it copies the contents at displacement of 3485H & 3486H in DS to ES reg.
- Flags: No Flags are affected.
- Addressing mode: Register Direct

IN

- Mnemonic: IN Accumulator, Port Address
- Operation: AX/AL ← Contents of Port
- It will copy data from a port to the accumulator.
- If an 8 bit port is read,the data will go in to AL.
- If an 16 bit port is read, the data will go to AX.
- IN instruction can be executed in 2 different addressing modes:

1. Direct: 8 bit address of the port is a part of instruction

   E.g: IN AL,80H; IN AX,95H

1. Indirect: The address of the port is referred from DX reg. The port address can be any no.between 0000H to FFFFH.

   E.g: MOV DX,30F8H; load 16bit address of port in DX

   IN AL,DX; copy a byte from 8 bit port 30F8H to AL

   IN AX,DX;copy a word from 16 bit port 30F8H to AX

- Flags: No Flags are affected.

OUT

- Operation: Contents of AX/AL → Port address.
- It will copy data from AL or AX to the specified port.
- OUT instruction can be executed in 2 different addressing modes:

1. Direct: 8 bit address of the port is a part of instruction

   E.g: OUT 80H,AL

2. Indirect: The address of the port is referred from DX reg. The port address can be any no.between 0000H to FFFFH.

   E.g: MOV DX,30F8H; load 16bit address of port in DX

   OUT DX,AL

   OUT DX,AX

- Flags: No Flags are affected.

FLAG INSTRUCTIONS (FLAG TRANSFER)

- LAHF : Load AH from lower byte of flag
- SAHF : Store AH to lower byte of flag
- PUSHF : Push flags to stack
- POPF : Pop flags from stack

LAHF:Load AH from lower byte of flag register

- Operation: AH← Lower byte of Flag register.
- Flags: No flags are affected.
- Addressing mode: Implied
- Before Execution: AH=A0H, AL=03H
- Flag Registers:

U U U U OF DF IF TF SF ZF U AF U PF U CF

0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0

\----------\ \----------\

MSB=0CH LSB=C6H →AH

- After Execution: AH=C6H, AL=03H
- Contents of LSB of flag register are copied to AH reg.

SAHF:Store AH to lower byte of flag register

- Operation: AH→Lower byte of Flag register.
- Flags: OF,DF,IF,TF are not affected.
- Addressing mode: Implied
- Before Execution: AH=A0H, AL=03H
- After Execution: AH=A0H, AL=03H
- Flag Registers:

U U U U OF DF IF TF SF ZF U AF U PF U CF

0 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0

\----------\ \----------\

MSB=0CH LSB=A0H ←AH

- Contents of AH are copied to LSB of flag register

PUSHF :PUSH FLAGS TO STACK

- It puts flag register contents on the stack.
- Whenever this instruction is executed, the MSB of flag reg. moves into stack segment mem. location addressed by SP-1.{ MSB→SP-1 }
- The LSB of flag reg. moves into the stack segment mem. location addressed by Sp-2.{ LSB→SP-2 }
- SP points to location SP-2.
- Flags: No flags are affected.
- Addressing mode: Register

POPF

- It removes the word from top of the stack to the flag reg.
- Whenever this instruction is executed, the byte from the SS mem location addressed by SP moves into the MSB of flag reg.{ SP→ MSB}
- The byte from SS mem location addressed by SP+1 moves into the LSB of the flag reg.{SP+1→LSB}
- Flags: All flags are affected.
- Addressing mode: Register

Segment Override Prefix

Instruction	Segment Accessed	Default Segment
MOV BX,DS:[BP]	DATA	STACK
MOV CX,ES:[BP]	EXTRA	STACK
MOV DX,SS:[DI]	STACK	DATA
MOV AX,ES:[SI]	EXTRA	DATA
MOV BX,CS:TABLE	CODE	DATA

ARITHMETIC INSTRUCTIONS

ARITHMETIC INSTRUCTIONS

- Arithmetic Data formats :

Arithmetic operations may be performed on four types of numbers:

1. Unsigned binary.
2. Signed binary.
3. Unsigned packed decimal.
4. Unsigned unpacked decimal.

Unsigned binary.

- These nos. may be either 8 or 16 bits long.
- All bits are considered in determining a number's magnitude.
- The value range of an 8 bit unsigned binary no. is 0-255.
- The value range of an 16 bit unsigned binary no. is 0-65,535.
- Addition,Subtraction,Multiplication & Division operations are available for unsigned binary numbers.

Signed Binary.

- These nos. may be either 8 or 16 bits long.
- MSB = 0, no. is Positive
- MSB = 1, no. is Negative
- -Ve nos. are represented in 2's complement notation.
- The range of nos. for 8 bits are –128 to +127.
- The range of nos. for 16 bits are –32,768 to +32,767.The value 0 has +Ve sign.
- Multiplication & Division operations are provided for signed binary nos.
- Addition & Subtraction operations are performed with the unsigned binary instructions.

Packed Decimal Numbers.

- These nos. are stored as unsigned byte quantities.
- The byte is treated as having one decimal digit in each half byte(nibble), the digit in the high order byte is the most significant hex values 0-9 are valid in each half byte, & the range is 0-99.
- Addition & subtractions are performed in 2 steps:

1. An unsigned binary instruction is used to produce an intermediate result in reg AL(AF- is set reset)
2. An adjustment operation is performed which changes the intermediate value in which changes the intermediate value in AL to a final correct packed decimal result.

MUL & DIV adjustments are not available .

Unpacked Decimal Numbers.

- These nos. are stored as unsigned byte quantities, magnitude is determined from the low order half byte(nibble).
- Hex values 0-9 are valid data are interpreted as decimal nos. The high order half byte(nibble) must be 0 for multiplication & division.
- Arithmetic on unpacked decimal nos. is performed in 2 steps:

1. The unsigned binary addition/subtraction & multiplication is used to produce an intermediate value in AL reg.
2. After that an adjustment instruction then changes the value in AL to a final correct unpacked decimal no.

- Unpacked decimal nos. are similar to ASCII character representation to the digits 0-9. The high order half byte of an ASCII numerals is always 3H.
- ASCII i/p→ unpacked BCD → binary→ → unpacked BCD → ASCII o/p

ADD: Add byte or Word.

- Mnemonic: ADD Destination, Source.
- Operation: Destination← Destination+Source
- Source & Destination can not be memory locations & both have to be of same type.
- The contents of segment regs. Can not be added.
- Destination & Source may be:

  Reg-Reg, Reg-Mem, Mem-Reg,

  Reg-Immediate, Mem-Immediate,

  Accumulator-immediate

ADD Register,Register

- Operation: Reg← Reg+Reg.
- E.g : CL=04H, BL=07H(Before Execution)

  ADD BL,CL

  BL=0BH, CL=04H(After Execution)
- Flags: All flags are affected.
- Addressing Mode: Register.

ADD Register,Memory

- Operation: Reg← Reg+Contents of memory.
- E.g : ADD AX,[2048]

  This instruction adds the data at mem. Location whose offset in DS are [2048] & [2049] with data in AX reg. The result is stored in AX reg.
- Flags: All flags are affected.
- Addressing Mode: Register.

ADD Memory,Register

- Operation: Mem← Contents of memory+ Reg.
- E.g : ADD [2048],AX

  // [2048]← [2048]+AL//

  //[2049]← [2049]+AH//
- Flags: All flags are affected.
- Addressing Mode: Register direct.

ADD Reg,Immediate/ADD Accumulator,Immediate

- Operation: Reg← Reg + immediate data.
- E.g: ADD AL,74H

  //{AL←AL+74H}//
- Flags: All flags are affected.
- Addressing Mode: Immediate .

ADD Memory,Immediate data

- Operation: Memory ← Memory+Immediate data.
- E.g: ADD COST,75H

  COST← COST +75H

  //{COST= memory location}//
- Flags: All flags are affected.
- Addressing Mode: Immediate .

ADC: Add with Carry

- Operation: destination← destination+source+Carry.
- Addition of two mem location along with carry is not possible.
- E.g:

  AL=20H,BL=30H,CY=01H(Before execution)

  ADC AL,BL //{AL← AL+BL+CY}//

  AL=51H,BL=30H,CY=01H(After execution)
- Flags: All flags are affected.
- Addressing Mode: Register Immediate .

INC:Increment

- Mnemonic: INC Destination
- Operation: Destination← Destination+1
- The operand may be a byte or a word & is treated as an unsigned binary no.
- The destination operand may be a reg. or mem location.
- Carry flag is not affected.
- If contents of 8 bit reg are FFH & 16 bit reg are FFFFH, after INC instruction, contents of reg will be 0 without affecting CF.
- Seg reg can not be incremented by this instruction.
- E.g: INC CX; INC AL
- Flags: All flags are affected except carry flag.
- Addressing Mode:Implied.

AAA:ASCII Adjust After Addition

- Mnemonic: AAA
- Operation: If lower nibble of AL>9 or AF=1;

  AL=AL+6 ; AH=AH+1

  AF=1; CF=1

  Else : AF=0; CF=0

  In both cases clear the higher nibble of AL
- Addressing mode: Implied
- Before execution:

  AX=35, BX=39,CF=0
- After execution:

  AH=01, AL=04,CF=1

E.g: AL=0011 0101(ASCII 5)

BL=0011 1001(ASCII 9)

ADD AL,BL

```
 0011 0101   (AL)
+0011 1001   (BL)
 ------------------
 0110 1110  (Invalid BCD)
+       0110
 ------------------
 0000 0100
```

AAA

(CF=1)

(01H)→AH

(04H)→AL(Valid BCD)

DAA : Decimal Adjust Accumulator

- Mnemonic: DAA
- Operation:

If lower nibble of AL>9 or AF=1 then;

AL=AL+06H, AF=1

If AL>9FH or CF=1 then;

AL=AL+60H , CF=1

- Flags: It changes AF,CF,PF,ZF.The OF is undersigned.
- Addressing Mode: Implied

- E.g :

AL=59H,BL=34H(BEFORE)

ADD AL,BL

(AL) 0101 1001

(BL) +0011 0100

1000 1101(Invalid BCD)

\+ 0110

1001 0011 (93H-BCD)

AL=93H,BL=34H

SUB: Subtract Byte or Word.

- Mnemonic: SUB Destination,Source.
- Operation: Destination← Destination-Source.
- Source & Destination can not be memory locations & both have to be of same type.
- Destination & Source may be:

  Reg-Reg, Reg-Mem, Mem-Reg,

  Reg-Immediate, Mem-Immediate,

  Accumulator-immediate

SUB Register,Register

- Operation: Reg← Reg-Reg.
- E.g : CL=04H, BL=07H(Before Execution)

  SUB BL,CL

  BL=03H, CL=04H(After Execution)
- Flags: All flags are affected.
- Addressing Mode: Register.

SUB Register,Memory

- Operation: Reg← Reg-Contents of memory.
- E.g : SUB AX,[2048]

  This instruction subtracts the data at mem. Location whose offset in DS are [2048] & [2049] with data in AX reg. The result is stored in AX reg.
- Flags: All flags are affected.
- Addressing Mode: Register.

SUB Memory,Register

- Operation: Mem← Contents of memory- Reg.
- E.g : SUB [2048],AX

  // [2048]← [2048]-AL//

  //[2049]← [2049]-AH//
- Flags: All flags are affected.
- Addressing Mode: Register direct

SUB Reg,Immediate/SUB Accumulator/Immediate

- Operation: Reg← Reg - immediate data.
- E.g: SUB AL,74H

  //{AL←AL+74H}//
- Flags: All flags are affected.
- Addressing Mode: Immediate .

SUB Memory,Immediate data

- Operation: Memory ← Memory-Immediate data.
- E.g: SUB COST,75H

  COST← COST -75H

  //{COST= memory location}//
- Flags: All flags are affected.
- Addressing Mode: Immediate .

SBB: Subtract with Borrow

- Operation: destination← destination-source-Carry.
- Subtraction of two mem location along with borrow is not possible.
- Source & Destination can be a signed or unsigned binary nos.
- E.g:

  AL=34H,BL=24H,CY=01H(Before execution)

  SBB AL,BL //{AL← AL-BL-CY}//

  AL=0FH,BL=24H,CY=01H(After execution)
- Flags: SF is not affected.
- Addressing Mode: Register Immediate .

DEC:Decrement

- Mnemonic: DEC Destination
- Operation: Destination← Destination-1
- The operand may be a byte or a word & is treated as an unsigned binary no.
- The destination operand may be a reg. or mem location.
- Carry flag is not affected.
- If contents of 8 bit reg are 00H & 16 bit reg are 0000H, after DEC instruction, contents of reg will be FFH & FFFFH with no CF(Borrow).
- E.g: DEC CX; DEC AL
- Flags: All flags are affected except carry flag.
- Addressing Mode:Register.

AAS:ASCII Adjust After Subtraction

- Mnemonic: AAS
- Operation: If lower nibble of AL>9 or AF=1;

  AL=AL-6 ; AH=AH-1

  AF=1; CF=1

  Else : AF=0; CF=0

  In both cases clear the higher nibble of AL
- Addressing mode: Implied
- Before execution:

  AL=39, BL=35,CF=0
- After execution:

  AH=00, AL=04,CF=0

E.g: AL=0011 1001(ASCII 9)

BL=0011 0101(ASCII 5)

SUB AL,BL

0011 1001 (AL)

- 0011 0101 (BL)

0000 0100 (BCD=04)

AAS

(CF=0)

(00H)→AH

(04H)→AL(Valid BCD)

DAS : Decimal Adjust After Subtraction

- Mnemonic: DAS
- Operation:

If lower nibble of AL>9 or AF=1 then;

AL=AL-06H, AF=1

If AL>9FH or CF=1 then;

AL=AL-60H , CF=1

- Flags: It changes AF,CF,PF,ZF.The OF is undersigned.
- Addressing Mode: Implied

- E.g :

AL=86H,BL=57H(BEFORE)

SUB AL,BL

(AL) 1000 0110

(BL) -0101 0111

0010 1111(Invalid BCD)

\- 0110

0010 1001 (29H-BCD)

AL=29H,BL=57H

NEG: Negate Byte or Word

- Mnemonic:NEG Destination
- Operation: This replaces the no. in destination with 2's complement of that no.
- The destination can be a reg. or mem location.
- It is useful for changing the sign of a signed word or byte.
- E.g: AX=00A3H=0000 0000 1010 0011

  NEG AX; AX← 2's complement of no. in AX

  i.e. AX=1111 1111 0101 1101=FF5DH
- Flags: All flags are affected.
- Addressing mode: Register

CMP: Compare Byte or Word

- Mnemonic: CMP Destination, Source.
- Operation: Destination(D)-Source(S).
- The Destination & Source remain unchanged, only flags are updated.

Compare	CF	ZF	SF	
S > D	1	0	1	Subtraction reqd. borrow(CF=1)
S = D	0	1	0	Result of subtraction is ‘0’
S < D	0	0	0	No borrow required(CF=0)

- Flags: AF,OF,SF,ZF,PF & CF are updated according to result.
- Addressing mode: Register

MUL: Multiply Byte or Word Unsigned.

- Mnemonic: MUL Source {source =reg/mem.location}
- Operation:

When operand is a byte; AX←AL*Operand.

(MSB→AH; LSB→AL)

When operand is a word; (DX:AX)←AX*operand

(MSB→DX; LSB→AX) **(mul.asm)**

- It can not be used to multiply immediate data.
- Flags: CF=0; OF =0; AF,PF,SF,ZF are undefined.
- E.g: AX=0009H, CX=0054H, DX=FFFFH(initial value)

  MUL CX; {0009H*0054H=_____(result)}

  DX←____H; AX←_____(result)
- Addressing mode: Register.

MUL: Multiply Byte or Word Unsigned.

- Mnemonic: MUL Source {source =reg/mem.location}
- Operation:

When operand is a byte; AX←AL*Operand.

(MSB→AH; LSB→AL)

When operand is a word; (DX:AX)←AX*operand

(MSB→DX; LSB→AX)

- It can not be used to multiply immediate data.
- Flags: CF=0; OF =0; AF,PF,SF,ZF are undefined.
- E.g: AX=0009H, CX=0054H, DX=FFFFH(initial value)

  MUL CX; {0009H*0054H=02F4H(result)}

  DX←0000H; AX←02F4H(result)
- Addressing mode: Register.

IMUL: Multiply Byte or Word Signed.

- Mnemonic: IMUL Source {source =reg/mem.location}

Operation: Byte operand; AX←AL*Operand.

(MSB→AH; LSB→AL)

Word operand; (DX:AX)←AX*operand

(MSB→DX; LSB→AX)

Concept of CBW.

- Flags: AF,PF,SF,ZF are undefined.
- E.g: AL=$(69)_{10}$ =_____H

  BL=$(14)_{10}$=____H

IMUL BL;

DX←_____H; AX←_____H (MSB =0;+ve result)

SF=0; CF=OF=1

- Addressing mode: Register.

IMUL: Multiply Byte or Word Signed.

- Mnemonic: IMUL Source {source =reg/mem.location}

Operation: Byte operand; AX←AL*Operand.

(MSB→AH; LSB→AL)

Word operand; (DX:AX)←AX*operand

(MSB→DX; LSB→AX)

Concept of CBW.

- Flags: AF,PF,SF,ZF are undefined.
- E.g: AL=$(69)_{10}$ = 45H

  BL=$(14)_{10}$= 0EH

IMUL BL;

DX←0000H; AX←03CEH (MSB =0;+ve result)

SF=0; CF=OF=1

- Addressing mode: Register.

AAM: Integer multiply byte or word ASCII Adjust for multiplication.

- Mnemonic: AAM
- Operation: AH← AL/10; AL←remainder
- It is used to adjust the product of 2 unpacked BCD digits in AX.
- E.g: AL= 0000 0101= unpacked BCD 5

  BL= 0000 1001=unpacked BCD 9 **(aam.asm)**

MUL BL: AL*BL=_____H=_____10

AAM;

AX=_____H unpacked BCD for _____

- Flags: It updates PF,SF,ZF.
- Addressing mode: Implied.

AAM: Integer multiply byte or word ASCII Adjust for multiplication.

- Mnemonic: AAM
- Operation: AH← AL/10; AL←remainder
- It is used to adjust the product of 2 unpacked BCD digits in AX.
- E.g: AL= 0000 0101= unpacked BCD 5

  BL= 0000 1001=unpacked BCD 9

MUL BL: AL*BL=002DH=45_{10}

AAM;

AX=0405H unpacked BCD for 45

- Flags: It updates PF,SF,ZF.
- Addressing mode: Implied.

DIV: Divide Byte or Word unsigned.

- Mnemonic: DIV Source
- Operation:

1. Byte Operand: AL←AX/Operand(Quotient)

AH←Remainder(Modulus)

2. Word Operand: AX←(DX:AX)/Operand(Quotient)

DX←Remainder(Modulus)

If quotient is too large to fit in Al then 8086 will execute a TYPE 0 Interrupt/Divide overflow error.

(div.asm & div1.asm)

→E.g: AX=37D7H=_____decimal,BH=97H=___ decimal

DIV BH; AX/BX, AL← quotient=___H=____ decimal

AH←Remainder=____H=_____ decimal

- Flags: All flags are not affected.

- Addressing mode: Register

DIV: Divide Byte or Word unsigned.

- Mnemonic: DIV Source
- Operation:

1. Byte Operand: AL←AX/Operand(Quotient)
   AH←Remainder(Modulus)
2. Word Operand: AX←(DX:AX)/Operand(Quotient)
   DX←Remainder(Modulus)

If quotient is too large to fit in Al then 8086 will execute a TYPE 0 Interrupt.

→E.g: AX=37D7H=14,295decimal,BH=97H=151decimal

DIV BH; AX/BX, AL← quotient=5EH=94decimal
AH←Remainder=65H=101decimal

- Flags: All flags are not affected.
- Addressing mode: Register

IDIV: Divide Byte or Word signed

- Mnemonic: DIV Source
- Operation:

1. Byte Operand: AL←AX/Operand(Quotient)
   AH←Remainder(Modulus)
2. Word Operand: AX←(DS:AX)/Operand(Quotient)
   DX←Remainder(Modulus)

- E.g: AX=03ABH, BL=00D3H

IDIV BL; AX/BX, AL← quotient=____H
AH←Remainder=____H

- Flags: All flags are undefined.
- Addressing mode: Register

IDIV: Divide Byte or Word signed

- Mnemonic: DIV Source
- Operation:

1. Byte Operand: AL←AX/Operand(Quotient)
   AH←Remainder(Modulus)
2. Word Operand: AX←(DS:AX)/Operand(Quotient)
   DX←Remainder(Modulus)

- E.g: AX=03ABH, BL=00D3H

IDIV BL; AX/BX, AL← quotient=ECH
AH←Remainder=27H

- Flags: All flags are undefined.
- Addressing mode: Register

AAD: ASCII Adjust for Division

- Mnemonic: AAD
- Operation:It converts 2 unpacked BCD digits in AH & AL to the equivalent binary no. in AL.
- After division:

  AL→unpacked BCD quotient

  AH→unpacked BCD remainder
- Flags: PF,SF,ZF are affected

  AF,CF,OF are undefined after AAD.
- Addressing mode: Implied

- E.g:

AX=0607H Unpacked BCD for 67 decimal

CL=09H

AAD;adjust to binary

AH*10=06*10=60_{10}=3CH

AH*10+AL= $60_{10}+7_{10}$= 43H

i.e. AL=43H, AH=00H
AX=0043H(new)

DIV CL;Divide AX by unpacked BCD in CL

Quotient =07H→AL,

Remainder=04H→AH

CBW:Convert Byte to Word

- Mnemonic: CBW
- Operation: If MSB of AL=1 then AH=FFH else AH=0
- This operation must be done before a signed byte in AL can be divided by another signed byte with IDIV instruction
- E.g: AX=00ACH

CBW; convert signed byte in AL to signed word in AX. i.e.

AX= 1111 1111 1010 1100=-163 decimal

Before execution:

0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0

/ AH // AL /

After execution:

1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 0

/ AH // AL /

- Flags: No flags are affected.
- Addressing mode: Implied

CWD:Convert Word to Double Word

- Mnemonic: CWD
- Operation: If MSB of AX=1 then DX=FFFFH else DX=0
- This operation must be done before a signed word in AX can be divided by another signed word with IDIV instruction
- E.g: DX=4AA3H AX=00ACH

Before execution:

0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 1 0 1 0 1 0 1 0 0 0 1 1

/ AH // AL / / DH // DL /

After execution:

0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

/ AH // AL / / DH // DL /

- Flags: No flags are affected.
- Addressing mode: Implied

Bit Manipulation Instructions

BIT MANIPULATION INSTRUCTIONS

- Logical: NOT, AND, OR, XOR,TEST.
- Shift: SHL/SAL, SHR, SAR.
- Rotate: ROR, ROL, RCL, RCR.

LOGICAL INSTRUCTIONS

- OF=CF=0 & AF= undefined after execution of instruction.
- The SF,ZF & PF are always posted to reflect the result of the operation & can be tested by conditional jump.
- NOT has no effect on flags.
- Here destination may be a Word or Byte. It may be a register or a memory location specified by one of the addressing mode.

NOT: NOT Byte or Word.

- Mnemonic: NOT Destination.
- Operation: Destination←(Destination)’
- It finds 1’s complement of a no.
- E.g:

  AX= 0000 0001 0010 0011(before execution)

  NOT AX

  AX= 1111 1110 1101 1100(after execution)
- Flags: No flags are affected.
- Addressing mode: Register

AND: AND Byte or Word

- Mnemonic: AND Destination,Source.
- Operation: ANDs each bit in a source byte or word with a destination byte or word.
- E.g: AND AL,BL

  Before Execution:

  AL=1001 0011= 93H; BL=0111 0101 =75H

  After Execution:

  AL= 0001 0001= 11H; BL=0111 0101 =75H
- Flags: CF=OF= 0(after execution)

  SF,PF,ZF are updated & AF is undefined.
- Addressing mode: Register

OR: OR Byte or Word

- Mnemonic: OR Destination,Source.
- Operation: ORs each bit in a source byte or word with a destination byte or word.
- E.g: OR AL,BL

  Before Execution:

  AL=1001 0011= 93H; BL=0111 0101 =75H

  After Execution:

  AL= 1111 0111= F7H; BL=0111 0101 =75H
- Flags: CF=OF= 0(after execution)

  SF,PF,ZF are updated & AF is undefined.
- Addressing mode: Register

XOR: XOR Byte or Word

- Mnemonic: XOR Destination,Source.
- Operation: XORs each bit in a source byte or word with a destination byte or word.
- E.g: XOR AL,BL

  Before Execution:

  AL=1001 0011= 93H; BL=0111 0101 =75H

  After Execution:

  AL= 1110 0110= E6H; BL=0111 0101 =75H
- Flags: CF=OF= 0(after execution)

  SF,PF,ZF are updated & AF is undefined.
- Addressing mode: Register

TEST:Test Byte or Word.

- Mnemonic: TEST Destination, Source.
- Operation: Destination← Destination AND Source.
- It is used to set flags before conditional Jump.
- E.g: TEST AL,75H

  AND immediate no. 75H with AL.
- Flags: CF=OF=0;

  PF,SF,ZF will be affected to show result of ANDing

  PF=SF=0, ZF=1
- Addressing mode: Immediate

SHIFT INSTRUCTIONS

- Arithmetic Shift: SAL, SAR
- Logical shift: SHL,SHR
- AF=undefined.
- PF,SF,ZF updated normally.

SHL/SAL:Shift logical/Arithmetic left byte or word.

- Mnemonic: SAL/SHL destination,count
- Operation:CF←MSB←LSB←0.
- Count is specified in CX register.
- E.g: (1) SHL AX,01H

Before execution:

AX= 0001 0010 0011 0100 CF=1

After Execution:

AX= 0010 0100 0110 1000 CF=0

(2) SAL AX,CX {AX=ACA5H; CX=02H}

- Flags: All flags are affected.
- Addressing mode: Immediate.

SHR: Shift Logical right byte or word.

- Mnemonic: SHR Destination,Count
- Operation: 0→MSB→LSB→CF
- E.g: (1)SHR AX,CX; AX=1234H, CX=02H

Before execution:

AX= 0001 0010 0011 0100 ,CF=1

After execution:AX= 0000 0100 1000 1101 ,CF=0

(2) SHR AX,02H {AX=ACA5H}

- Flags: All flags are affected.
- Addressing mode: Register.

SAR: Shift Arithmetic Right byte or word.

- Mnemonic: SAR Destination, Count
- Operation: MSB→ MSB→LSB→CF.

  New MSB= Old MSB.
- E.g: SAR AX,02H

Before execution:

AX=1010 1100 1010 0101 , CF=0

After execution:

Ax=1110 1011 0010 1001 , CF=0

Flags: All flags are affected.

- Addressing mode: Immediate

ROTATE INSTRUCTIONS

- ROL
- ROR
- RCL
- RCR.

ROL: Rotate Left Byte or Word

- Mnemonic: ROL Destination,Count
- Operation: CF←MSB←LSB←MSB
- E.g(1)ROL AX,01H

Before execution:

AX=0001 0010 0110 0100, CF=0

After execution:

AX= 0010 0100 1100 1000, CF=0

(2) ROL AX,CX {AX=AF5DH; CX=02H}

- Flags: Only CF & OF are affected.
- Addressing mode: Immediate register

ROR: Rotate Right Byte or Word

- Mnemonic: ROR Destination, Count
- Operation: MSB→LSB→ CF, LSB→MSB
- E.g:ROR AX,CX

Before execution:

AX= 1010 1111 0101 1101,CX=02H,CF=0

After execution:

AX=0110 1011 1101 0111, CF=0

- Flags: Only CF & OF are affected.
- Addressing mode: Register

RCL: Rotate through Carry Left Byte or Word

- Mnemonic: RCL Destination, Count
- Operation: CF←MSB←LSB←CF
- E.g: (1)RCL AX,01H

Before execution:

AX= 0001 0010 0011 0100,CF=0

After execution:

AX=0010 0100 0110 1000,CF=0

(2) RCLAX,04H ;{AX=AF5DH}

- Flags: Only CF & OF are affected.
- Addressing mode: Immediate

RCR: Rotate through Carry Right Byte or Word

- Mnemonic: RCR Destination, Count
- Operation: CF→MSB→LSB→CF
- E.g: (1)RCR AX,04H

Before execution:

AX= 0001 0010 0011 0100,CF=0

After execution:

AX=0100 0001 0010 0011,CF=0

(2) RCR AX,01H {AX=AF5DH}

- Flags: Only CF & OF are affected.
- Addressing mode: Immediate

Processor Control Instructions

PROCESS CONTROL INSTRUCTIONS

- Flag Operations: STD,CLC,CMC,
  STD,CLD,STI, CLI
- No operation: NOP.
- External Synchronization:HLT,WAIT,ESC,
  LOCK

CLC: Clear Carry Flag

- Mnemonic: CLC
- Operation: CF←0
- Flags: Except carry flag no other flags are affected.
- Addressing mode: Implied

STC: Set Carry Flag

- Mnemonic: STC
- Operation: CF←1
- Flags: Except carry flag no other flags are affected.
- Addressing mode: Implied

CMC: Complement Carry Flag.

- Mnemonic: CMC
- Operation: CF←CF'(1's Complement)
- Flags: Except carry flag no other flags are affected.
- Addressing mode: Implied

CLD: Clear Direction Flag

- Mnemonic: CLD
- Operation: DF←0
- Flags: Except direction flag no other flags are affected.
- Addressing mode: Implied

STD: Set Direction Flag.

- Mnemonic: STD
- Operation: DF←1.
- Flags: Except direction flag no other flags are affected.
- Addressing mode: Implied

CLI: Clear Interrupt Flag

- Mnemonic: CLI
- Operation: IF←0
- If IF reset, the 8086 will not respond to an interrupt signal on its INTR I/P.
- This instruction has no effect on the NMI.
- Flags: Except interrupt flag no other flags are affected.
- Addressing mode: Implied

STI: Set Interrupt flag.

- Mnemonic: STI
- Operation: IF←1.
- This enables INTR interrupt of 8086.
- Flags: Except interrupt flag no other flags are affected.
- Addressing mode: Implied

NOP:No Operation

- Mnemonic: NOP
- Operation: Do Nothing
- It causes the CPU to do nothing.
- It uses 3 Clock cycles & Increments the IP to point to the next instruction
- It can be used to Increases the delay of a delay loop.
- Flags: No flags are affected.
- Addressing mode: Implied

- External Synchronization:HLT,WAIT,ESC, LOCK

String Instructions

String Instructions

1. REPE/REPZ
2. REPNE/REPNZ
3. MOVS
4. MOVSB/MOVSW
5. CMPS
6. SCAS
7. LODS
8. STOS

String Instruction Registers & Flags used

- SI → Index (offset) for source string
- DI → Index (offset) for destination string
- CX → Counter
- AL/AX → Scan value destination for LODS/
  Source for STOS
- DF → 0 = Incremented SI & DI
  1= Decremented SI & DI
- ZF → Scan/Compare terminator

MOVS: Move String Byte or Word

- Mnemonic: MOVS Dest_string, Src_string
  MOVSB (Byte), MOVSW(Word)
- BYTE→ ES:[DI]=DS:[SI]
  If DF=0; SI=SI+1, DI=DI+1; Else SI=SI-1, DI=DI-1
- WORD→ ES:[DI]=DS:[SI]
  If DF=0; SI=SI+2, DI=DI+2; Else SI=SI-2, DI=DI-2
- E.g:

```
LEA SI,SRC_STRING
LEA DI,DEST_STRING
CLD
MOVSB;    Move (DI)← (SI) byte transfer
```

- Flags: No flags are affected.
- Addressing mode: String

MPS: Compare string byte or string words

- Mnemonic: CMPS Dest_string, Src_string

  CMPB (Byte), CMPW(Word)

Byte→ DS:[SI]-ES:[DI], If DF=0, SI=SI+1, DI=DI+1 ; else SI=SI-1, DI=DI-1

Word→DS:[SI]-ES:[DI], If DF=0, SI=SI+2, DI=DI+2 ; else SI=SI-2, DI=DI-2

- E.g: LEA SI,SRC

  LEA DI,DEST

  CLD

  CMPS SRC,DEST

Flags: All flags are affected.

- Addressing mode: String

SCAS/SCASB/SCASW
Scan a string Byte or word

- Mnemonic:SCAS Dest_string/SCASB/SCASW
- Operation:

For SCASB; ES:[DI]-AL,DF=0, DI=DI+1

For SCASW; ES:[DI]-AX,DF=0, DI=DI+2

- E.g: Let BUFF← 0EH

```
LEA DI,BUFF
MOV AL,0DH
CLD
SCASB
```

- Addressing mode: String

LODS: Loads string byte in Al or string word in AX

- Mnemonic:LODS Src_string/LODSB/LODSW
- Operation:

LODSB; AL=DS:[SI],If DF=0, SI=SI+1 else SI=SI-1

LODSW; AX=DS:[SI],If DF=0, SI=SI+2 else SI=SI-2

- E.g: CLD

```
MOV SI,OFFSET Al
LODSB
```

- Flags: No flags are affected.
- Addressing mode: String

STOS/STOSB/STOSW:Store byte or word in string

- Mnemonic: STOS Dest_string/STOSB/STOSW
- Operation:

For STOSB; ES:[DI]=AL

If DF=0, DI=DI+1 else DI=DI-1

For STOSW; ES:[DI]=AX

If DF=0, DI=DI+2 else DI=DI-2

- E.g: MOV DI,OFFSET STR1

```
CLD
MOV AX,00H
STOSW
```

- Flags: No flags are affected.

- Addressing mode: String

REPE/REPZ
Repeat while Equal/Repeat while Zero

- Mnemonic: REPE/REPNZ
- Operation: Check_CX for REP

  If CX< >0 then CX=CX-1, If ZF=1 then go to check CX else exit from REPE cycle
- E.g: LEA DI,STR1

  MOV AL,05H

  CLD

  MOV CX,35H

  REPE SCASB;Repeat Scan string byte operation till CX!=0 & ZF=1

REPNE/REPNZ

Repeat while not Equal/Repeat while not Zero

- Mnemonic: REPE/REPNZ
- Operation: Check_CX for REP

  If CX< >0 then CX=CX-1, If ZF=0 then go back to check CX else exit from REPNE cycle
- E.g: LEA DI,STR1

  MOV AX,25H

  CLD

  MOV CX,35H

  REPNE SCASW;Repeat Scan string word operation till CX!=0 & ZF=0

PROGRAM TRANSFER GROUP/BRANCHING INSTRUCTIONS

- Transfer the flow of execution of the program to a new address specified in the instruction directly or indirectly.
- The CS & IP reg. get loaded with new values of CS & IP corresponding to the location where the flow of execution is going to be transferred.
- 8086/88 provides→

  (1) Unconditional CALL

  (2) JMP: Conditional, Unconditional

  (3) INT (Software Interrupt)

BRANCH ADDRESSING MODES

- Intrasegment Direct(Direct NEAR):

Effective Branch Address is the sum of an 8/16 bit displacement & the current contents of IP. Branching is within the segment.

- Intrasegment Indirect(Indirect NEAR):

1. The effective branch address is the content of a register or mem location specified by one of the addressing mode except immediate mode.
2. The contents of IP are replaced by effective branch address & this addressing mode is used only for unconditional branch instruction.

- Intersegment Direct (Direct FAR):

  It deals with branching out of the present code segment, hence microprocessor requires information of new segment & new offset.

- Intersegment Indirect (Indirect FAR):

  It deals with branching from one code segment to another segment.