Department of Electronics & Communication Engg.

Course :Microprocessor-15EC42. **Sem.: 4th (2017-18)**

Course Coordinator:

Prof. P. V. Patil

MODULE I

INTRODUCTION TO MICROPROCESSOR

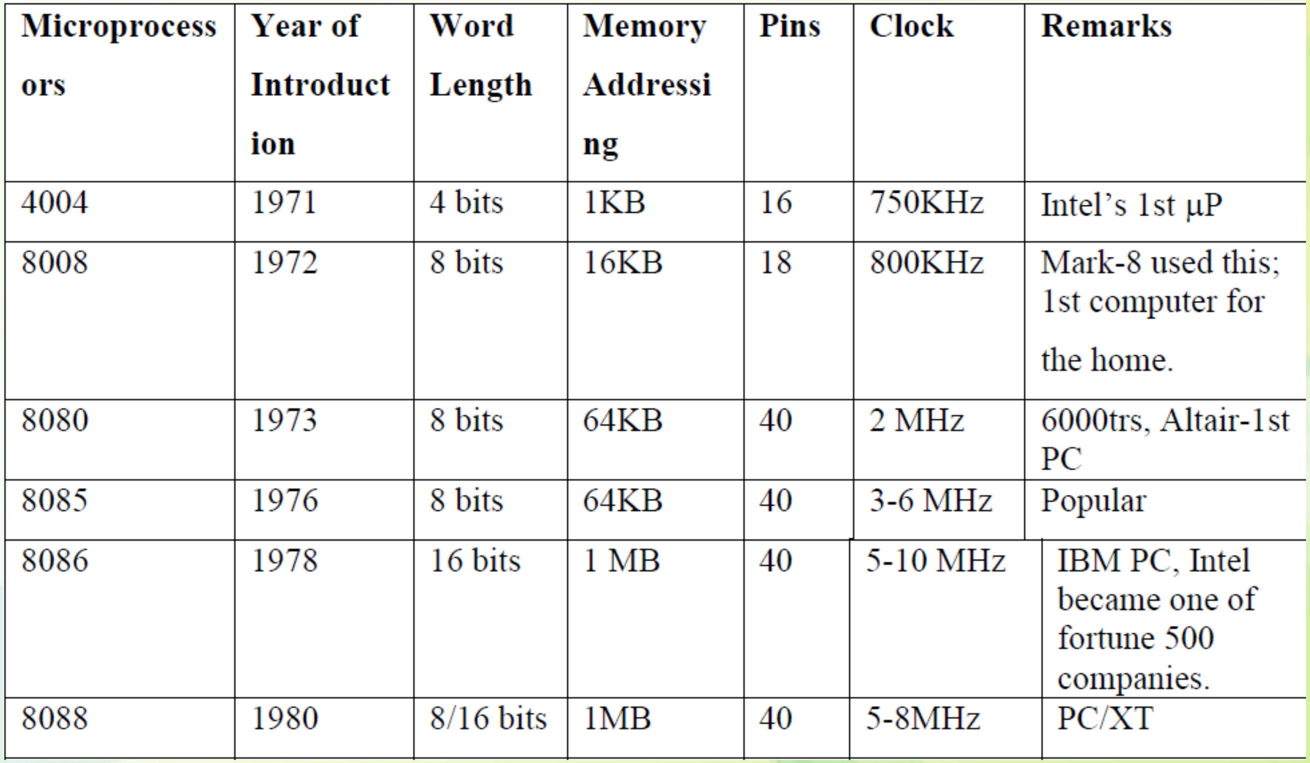
HISTORICAL BACKGROUND

- 80X86, Pentium, Pentium Pro, Pentium III, Pentium 4, and Core2 microprocessors.
- **Mechanical Age**
  - **Abacus(**first mechanical calculator**)**
  - Mechanical Calculator(Gear & wheel enabled)(1642 Blaise Pascal)
  - **Analytical Engine(**Steam-powered mechanical computer**)(stores 1000 -20 decimal numbers)**
- **The Electrical Age**
  - electric motor
  - **Bomar Brain(**handheld electronic calculator 1970**)**

History Of MP

- The first processor was a 4 bit processor and was called 4004
- The microprocessor was invented in the year 1971 in the Intel labs
- Intel, Motorola, Zylog Corporation, Fairchild and National (Hitachi, Japan) are the microprocessor manufacturers.

Microprocessor revolution



Microprocessors	Year of Introduction	Word Length	Memory Addressing	Pins	Clock	Remarks
4004	1971	4 bits	1KB	16	750KHz	Intel’s 1st µP
8008	1972	8 bits	16KB	18	800KHz	Mark-8 used this; 1st computer for the home.
8080	1973	8 bits	64KB	40	2 MHz	6000trs, Altair-1st PC
8085	1976	8 bits	64KB	40	3-6 MHz	Popular
8086	1978	16 bits	1 MB	40	5-10 MHz	IBM PC, Intel became one of fortune 500 companies.
8088	1980	8/16 bits	1MB	40	5-8MHz	PC/XT

Microprocessor

“Single Chip Integrated Circuit Computer Architecture”

- What is Microprocessor
- What is Microcontroller
- Difference Between MP & MC
- Applications Of MP

Difference between MP and MC

- **What is a Microprocessor?**
- is an electronic computers central processing unit (CPU) made from miniaturized transistors and other circuit elements on a single semiconductor integrated circuit (IC).
- It performs arithmetic, logic and control operations. It contains a control unit, an arithmetic & logic unit, registers and links to store data and connect to peripherals.
- **What is a Microcontroller?**

Dedicated to performing one task. Integrates the memory and other features of a microprocessor.

About M.Processor

- CPU is also known as microprocessor
- It includes (CPU, ALU, Registers)
- Control logic of almost all digital devices (fuel industry, automobile application)
- In its life period it has to handle many different tasks and programs given to it.
- may not handle a real time task at all times

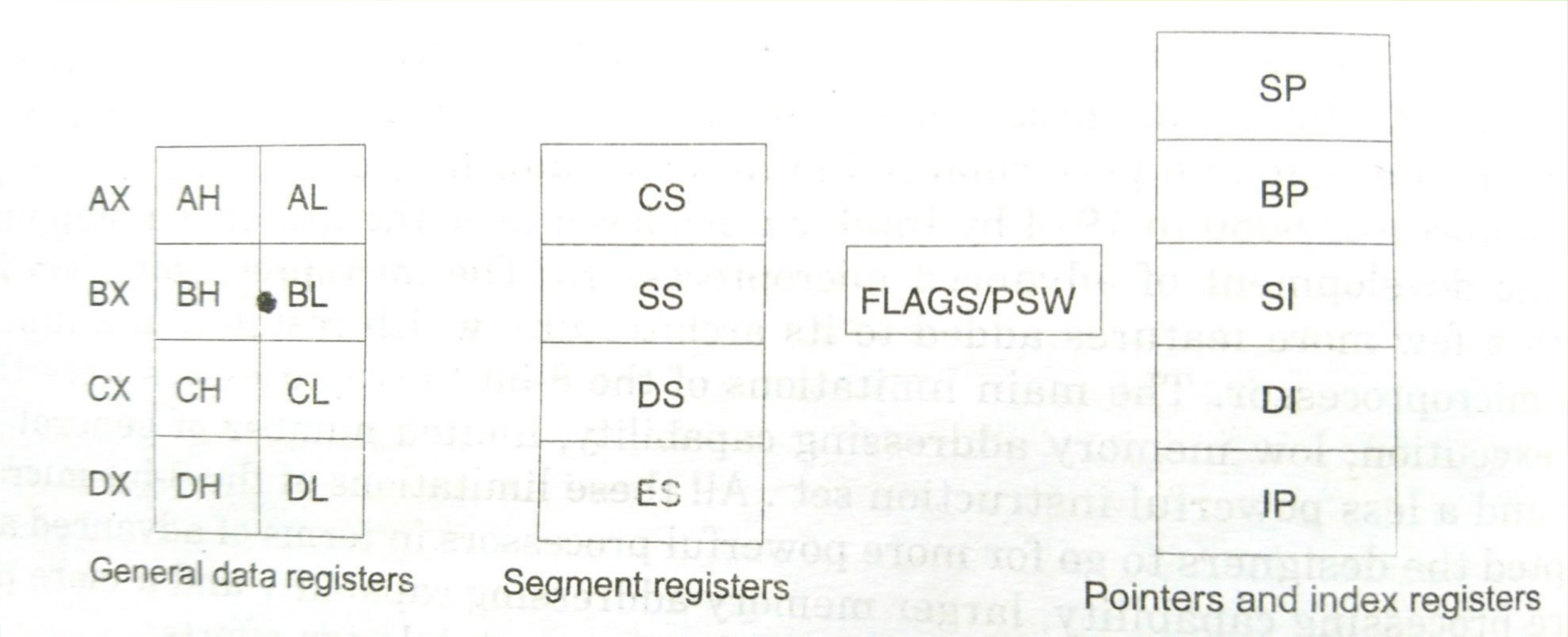
About M.controller

- Designed specifically for specific tasks
- self-sufficient and cost-effective
- microcontroller is part of an embedded system
- controls the operation of a machine using fixed programs stored in ROM.
- micro controllers are from 8051 family or PIC family or any other

Applications of microprocessors

1. computer applications
2. Control application (MC,embedded controllers)
3. Communication (DSP processors,Cell phones)

Register Organization of 8086



General data registers

Segment registers

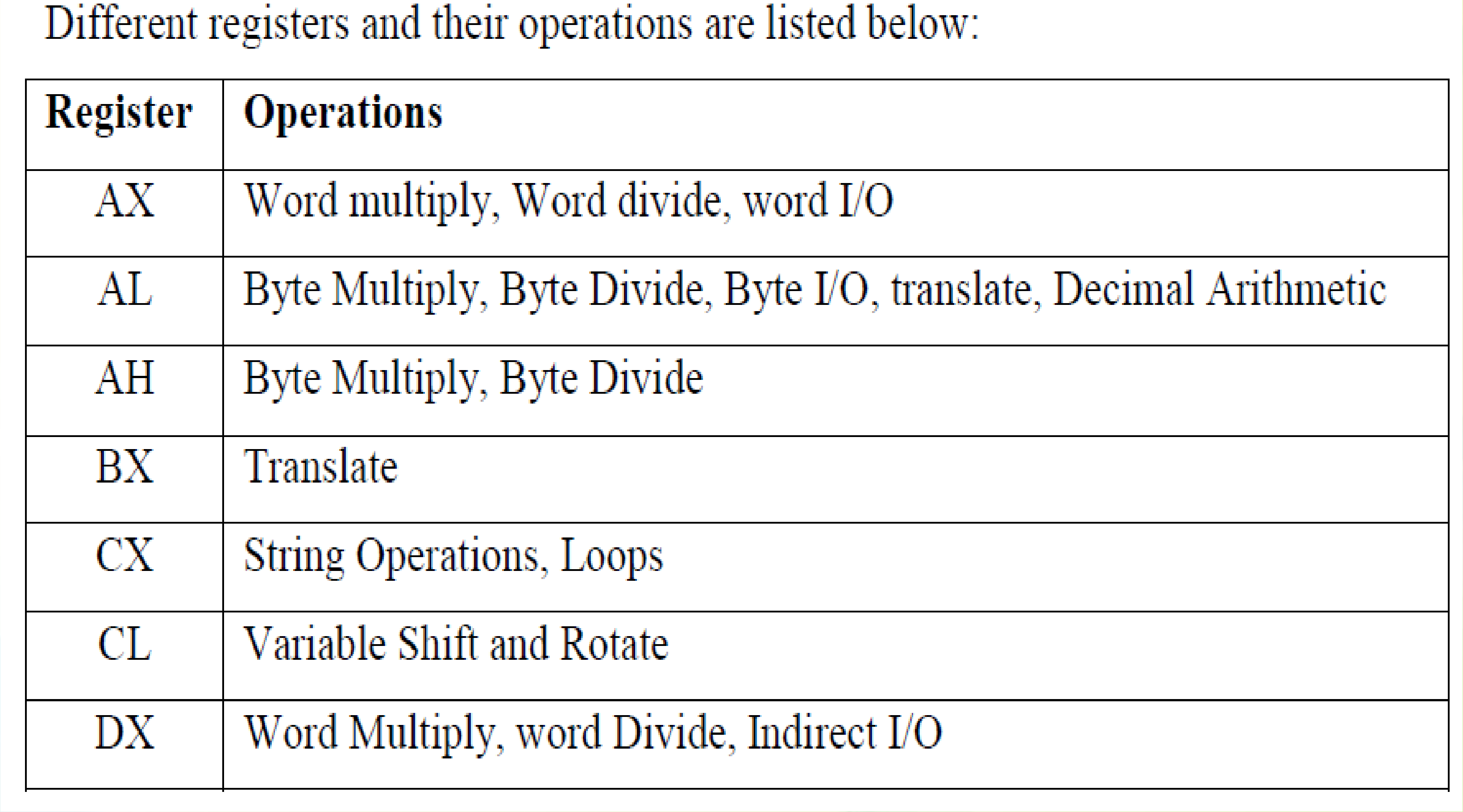
Pointers and index registers

General Registers

Data Registers file
AX - the Accumulator
BX - the Base Register
CX - the Count Register
DX - the Data Register

Pointer & Index Registers file
SP - the Stack Pointer
BP - the Base Pointer
SI - the Source Index Register
DI - the Destination Register

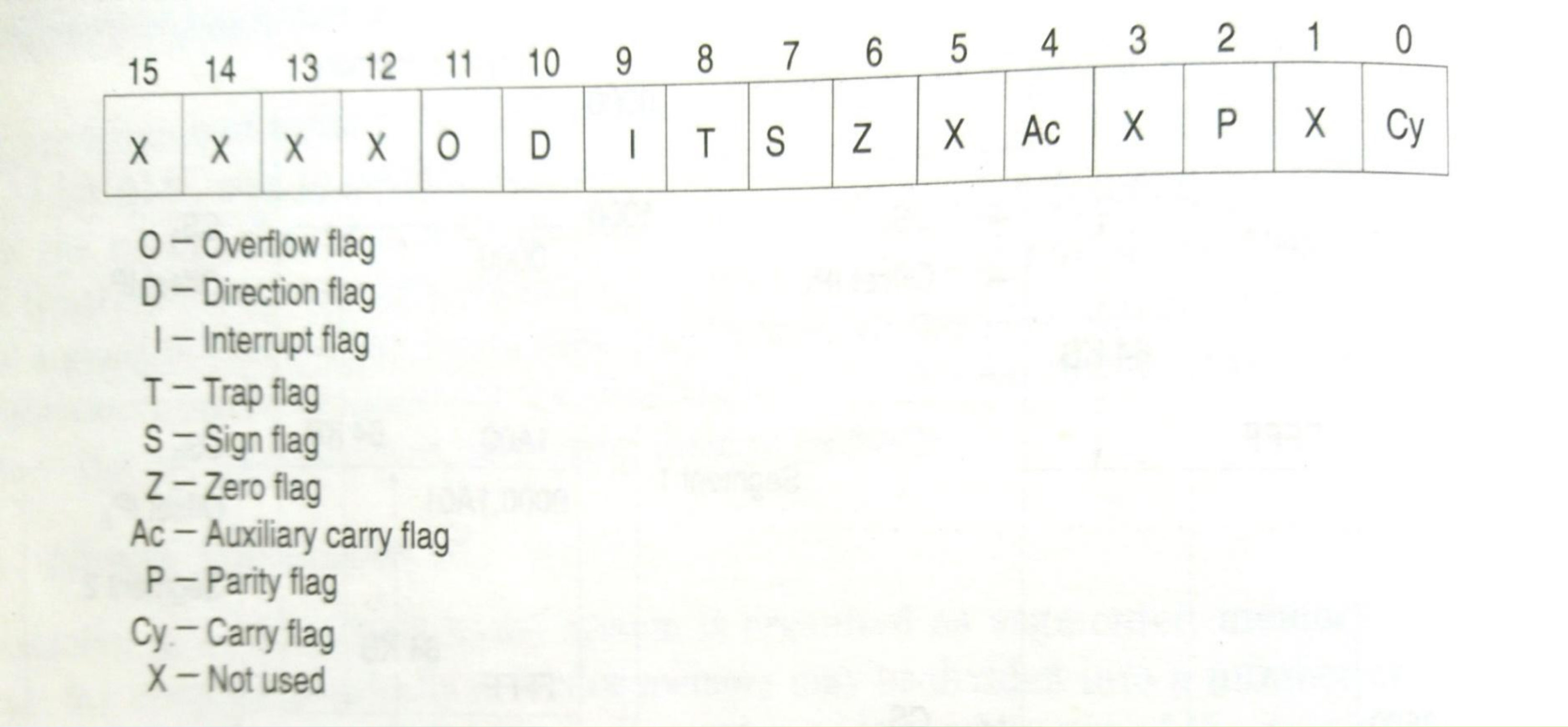
Registers and their operations



Different registers and their operations are listed below:

Register	Operations
AX	Word multiply, Word divide, word I/O
AL	Byte Multiply, Byte Divide, Byte I/O, translate, Decimal Arithmetic
AH	Byte Multiply, Byte Divide
BX	Translate
CX	String Operations, Loops
CL	Variable Shift and Rotate
DX	Word Multiply, word Divide, Indirect I/O

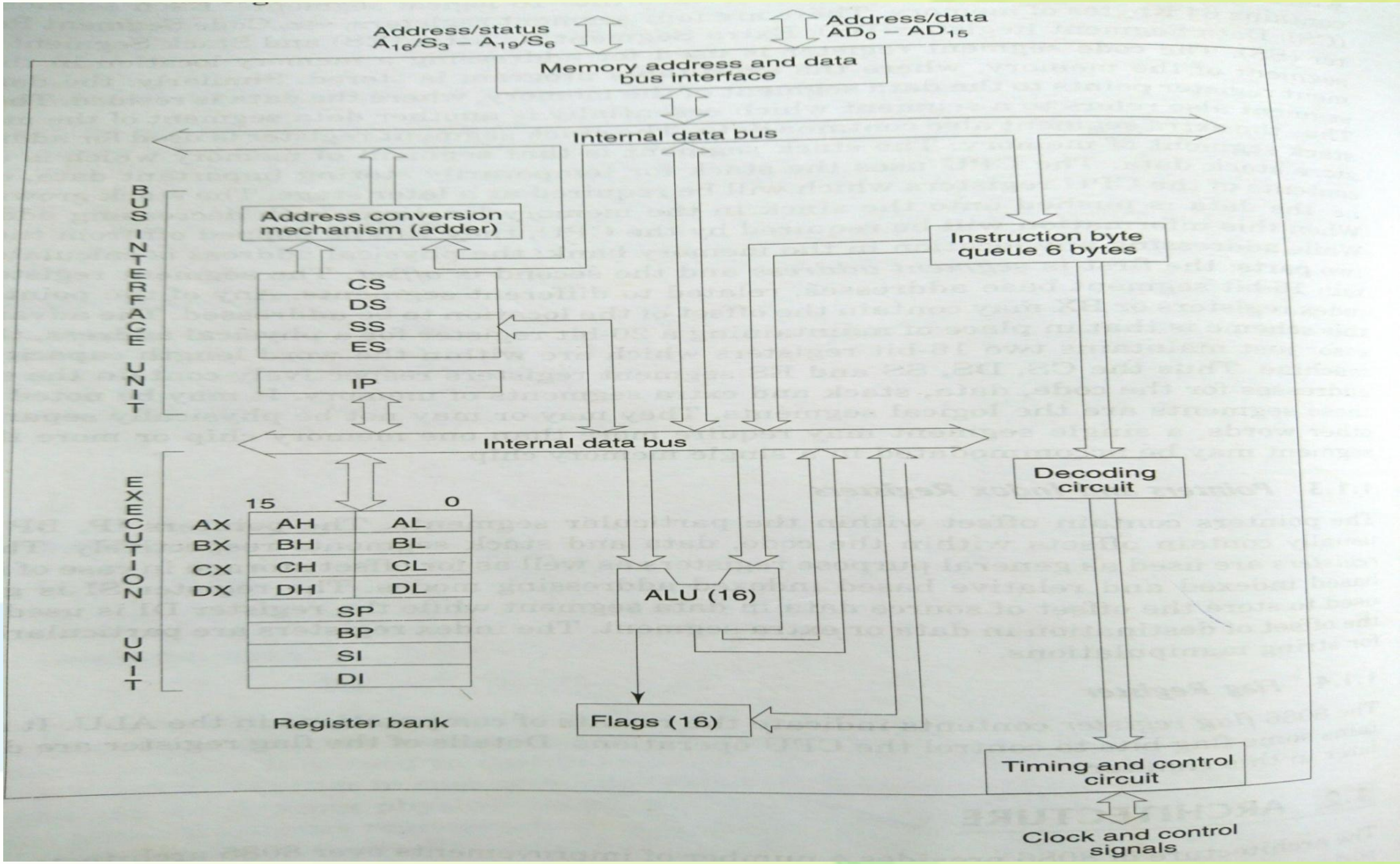
Flag Register 0f 8086



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	O	D	I	T	S	Z	X	Ac	X	P	X	Cy

O – Overflow flag
D – Direction flag
I – Interrupt flag
T – Trap flag
S – Sign flag
Z – Zero flag
Ac – Auxiliary carry flag
P – Parity flag
Cy. – Carry flag
X – Not used

CPU Architecture


Address/status
$A_{16}/S_3 - A_{19}/S_6$
Address/data
$AD_0 - AD_{15}$
Memory address and data bus interface
Internal data bus
BUS INTERFACE UNIT
Address conversion mechanism (adder)
Instruction byte queue 6 bytes
CS
DS
SS
ES
IP
Internal data bus
EXECUTION UNIT
Decoding circuit
ALU (16)
Flags (16)
Timing and control circuit
Clock and control signals
Register bank

	15	0
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL
	SP	
	BP	
	SI	
	DI	

8086 Features

- 16-bit Arithmetic Logic Unit
- 16-bit data bus
- 20-bit address bus - 2^{20} = 1,048,576 = 1 meg
- Even location byte-((A0-A7)-Lower half),
- Odd location byte-((A8-A15)-higher half)

Block diagram of 8086 can be subdivided into two parts

- 1. Bus Interface Unit
- 2. Execution Unit

1. Bus Interface Unit

- consists of segment registers, adder to generate 20 bit address and instruction prefetch queue.
- instruction and data bytes are fetched from memory and they fill a First In First Out 6 byte queue

2.Execution Unit:

- consists of scratch pad registers such as 16-bit AX, BX, CX and DX
- and pointers like SP (Stack Pointer), BP (Base Pointer) and
- finally index registers such as source index and destination index registers
- The 16-bit scratch pad registers can be split into two 8-bit registers ex(AX-AH & AL)

Working of EU & BIU

- The EU and BIU work asynchronously.
- Any External from m/m or i/o needed to the execution EU informs it to BIU and BIU will perform that operation.
- BIU performs its operations and hand over's the Bus service for EU Even though requests comes from the EU unit

Memory Segmentation

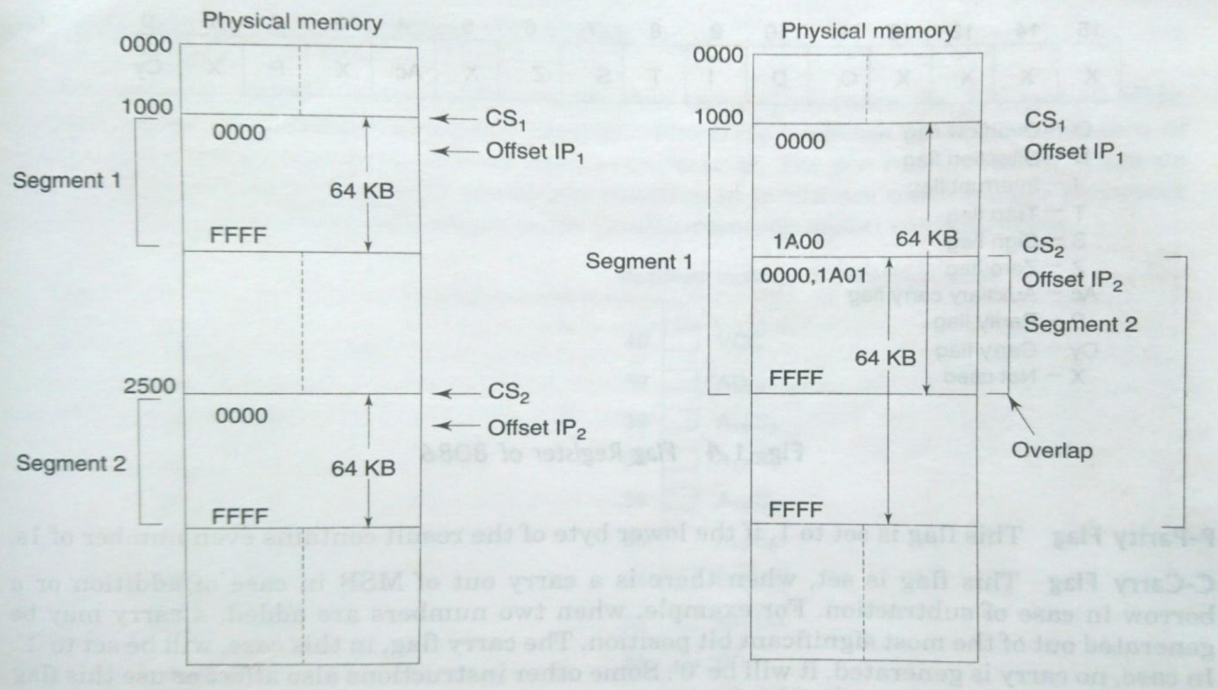
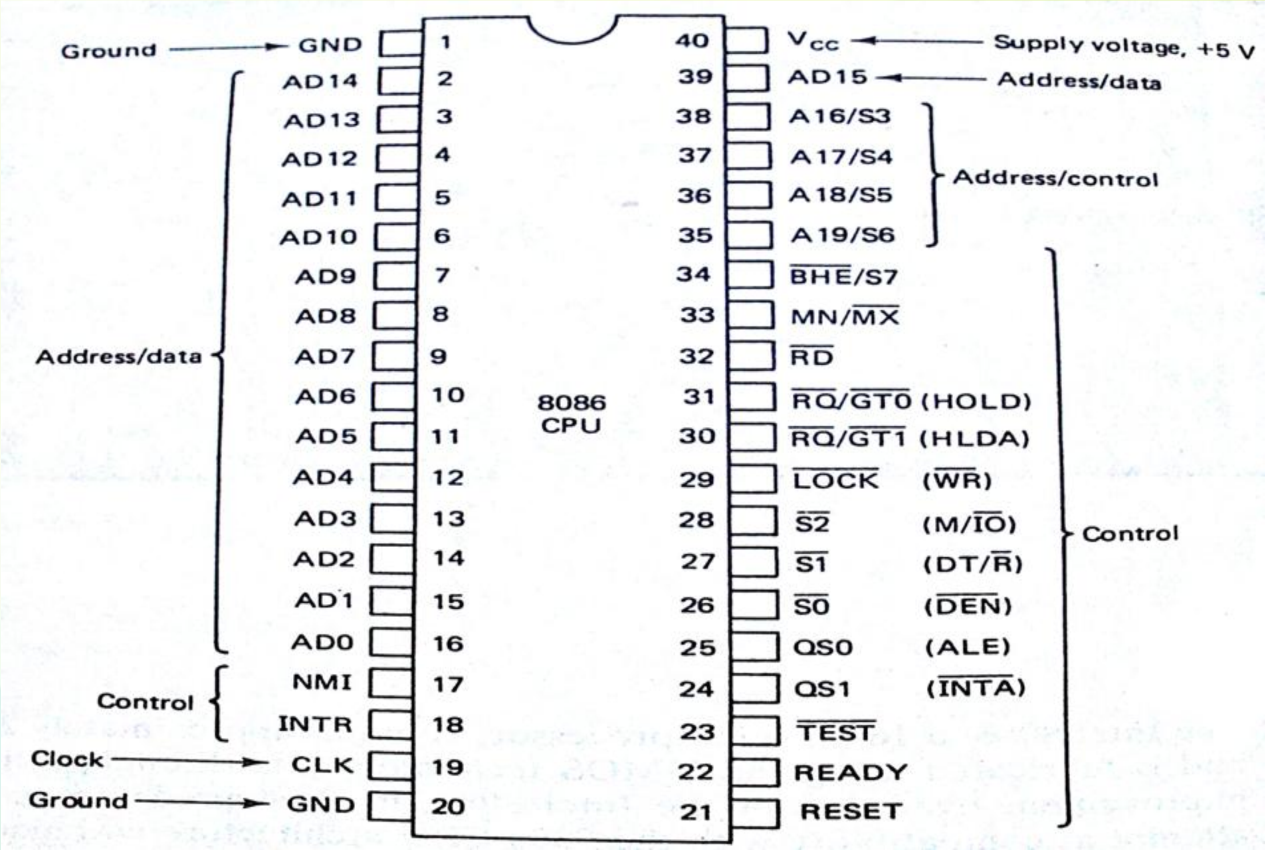


Fig. 1.3 (a) *Non-overlapping Segments*

Fig. 1.3 (b) *Overlapping Segments*

Signal Description of 8086


Ground
GND 1
Address/data
AD14 2
AD13 3
AD12 4
AD11 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD1 15
AD0 16
Control
NMI 17
INTR 18
Clock
CLK 19
Ground
GND 20
8086 CPU
40 V_{CC}
Supply voltage, +5 V
39 AD15
Address/data
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
Address/control
34 $\overline{\text{BHE}}$/S7
33 MN/$\overline{\text{MX}}$
32 $\overline{\text{RD}}$
31 $\overline{\text{RQ}}$/$\overline{\text{GT0}}$ (HOLD)
30 $\overline{\text{RQ}}$/$\overline{\text{GT1}}$ (HLDA)
29 $\overline{\text{LOCK}}$ ($\overline{\text{WR}}$)
28 $\overline{\text{S2}}$ (M/$\overline{\text{IO}}$)
27 $\overline{\text{S1}}$ (DT/$\overline{\text{R}}$)
26 $\overline{\text{S0}}$ ($\overline{\text{DEN}}$)
25 QS0 (ALE)
24 QS1 ($\overline{\text{INTA}}$)
23 $\overline{\text{TEST}}$
22 READY
21 RESET
Control

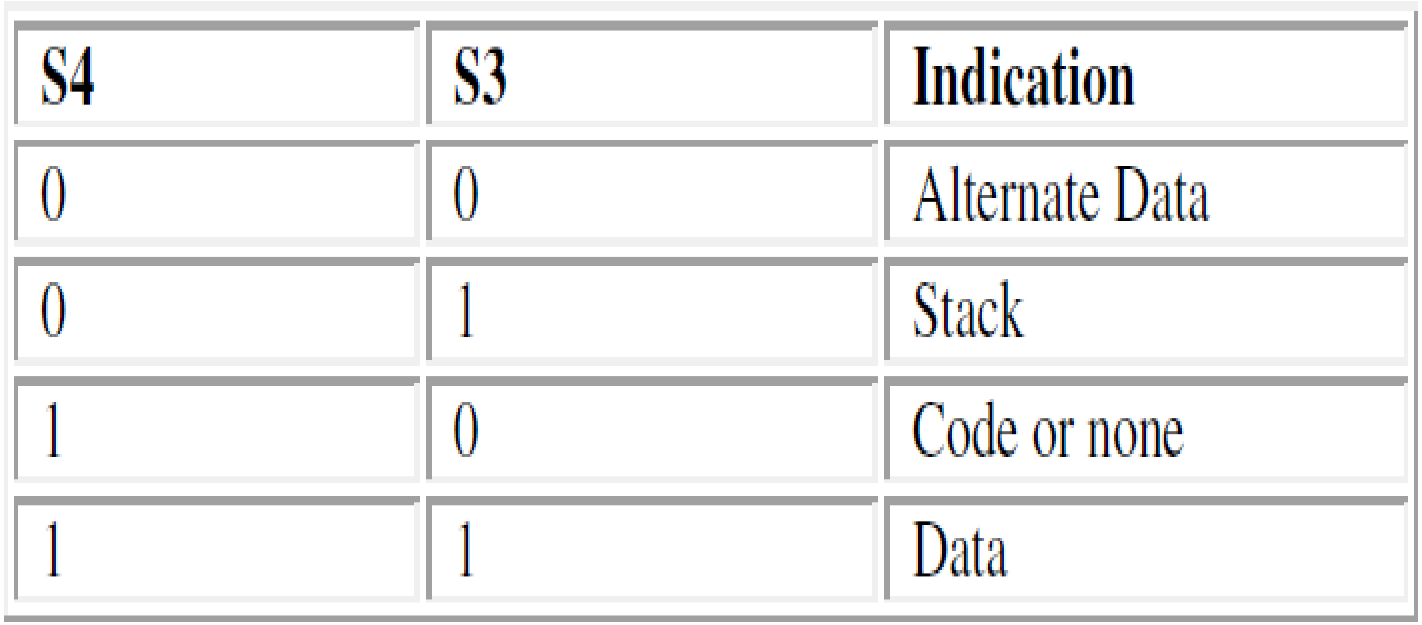
Pin Description

- 8086 Microprocessor is a 16-bit CPU available in 3 clock rates(5, 8 ,10MHz) packaged in a 40 DIP IC.
- Pins work in 2 modes {Minimum(single processor mode) & Maximum(multi processor mode)}
- Signals can be characterized in 3 parts

1. special function in min mode
2. special function in max mode
3. common function in both min and max mode.

Pins for both Min mode and Max Mode

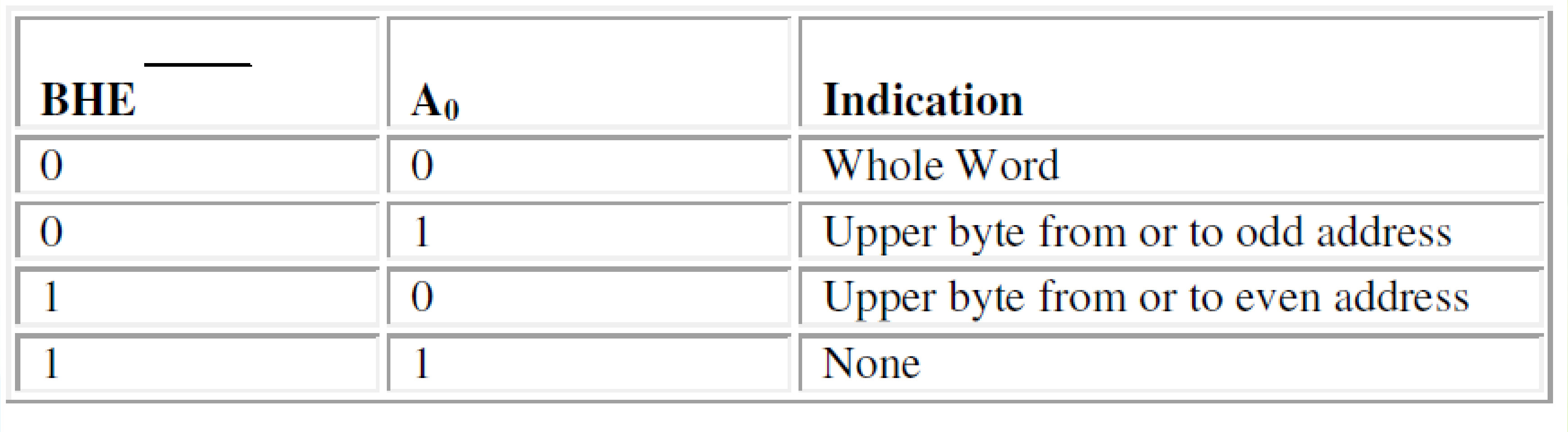
- **AD15-AD0-** Bidirectional multiplexed memory I/O address(T1) and data lines(T2,T3,Tw,T4), these pins will be in tri-state during Local Bus and Interrupt ACK.
- **(A16/S3-A19/S6) -**Time multiplexed address(T1) and status lines(T2,T3,Tw,T4) ALE is used to demultiplex the Address and status.
- S5- Interrupt enable flag bit.
- S6- always low



S4	S3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

Table 1.1 Bus High Enable/Status

BHE*/S7-Bus High Enable/Status- Transfer of data over higher order bus(D15-D8)(T1). (0-odd addr m/m (Upper bank), 1-Even addr m/m(Lower bank)).{U-od-0,L-Ev-1}



$\overline{\text{BHE}}$	A_0	**Indication**
0	0	Whole Word
0	1	Upper byte from or to odd address
1	0	Upper byte from or to even address
1	1	None

- **RD*-Read--**
- memory or I/O read operation
- RD* is active low and shows the state for T2,T3, TW of any read cycle
- **READY—**
- Signal is active high
- acknowledgement from the slow devices or memory

INTR-Interrupt Request

- This is a level triggered input
- Activated at Last clock cycle of each instruction to determine the availability of the request.
- **TEST-** idle state-1, continue execution-0
- The “wait” state examines the Test pin.
- on leading edge of clock states changes

- **NMI-Non-maskable Interrupt---**
- edge-triggered input
- not maskable internally by software

- **RESET—**
- terminate the current activity and start execution from FFFF0H.
- Active high input
- must be active for at least four clock cycles.

- **CLK—**
- provides the basic timing for processor operation
- Its an asymmetric square wave with 33% duty cycle
- Range is 5MHz to 10MHz.
- **VCC : +5V power supply for the operation**

- **MN/MX*---**
- weather the processor is in min mode or max mode.

- Min-1
- Max-0

minimum mode operation of 8086.

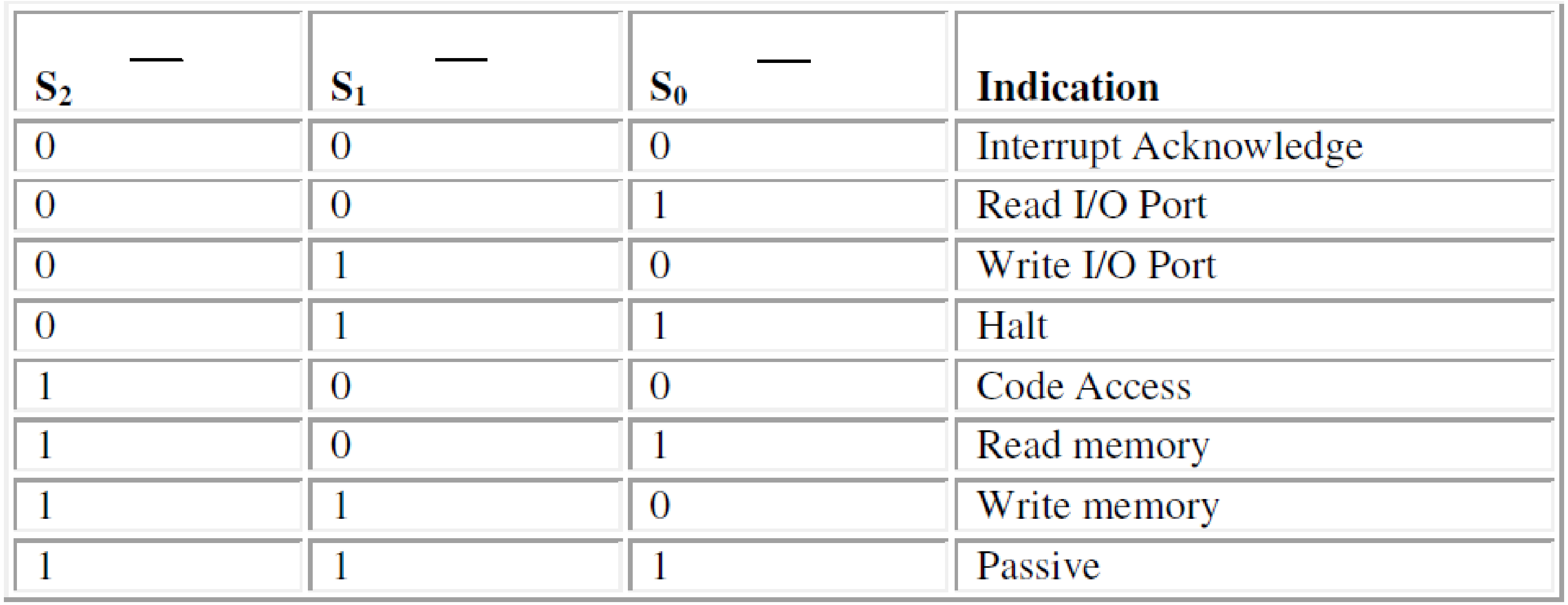
- **M/IO* -Memory/IO**
- m/m-1, i/o-0
- equivalent to S2 in maximum mode

- **INTA* -Interrupt Acknowledge**
- processor has accepted the interrupt when its 0.

- **ALE-Address latch Enable—**
- output signal indicates the availability of the valid address on the address/data lines.
- active high and is never tristated.
- **DT /R* -Data Transmit/Receive—**
- direction of data flow
- **Transmission-1**
- **Reception-0**

- **DEN*-Data Enable—**
- This signal indicates the availability of valid data over the address/data lines.
- **HOLD, HLDA-Hold/Hold Acknowledge—**
- another master is requesting the bus access
- When **HOLD is 1, HLDA is made 1**
- When **HOLD is 0, HLDA is made 0**

- **S2, S1, S0 -Status Lines**
- status lines reflect the type of operation, being carried out by the processor



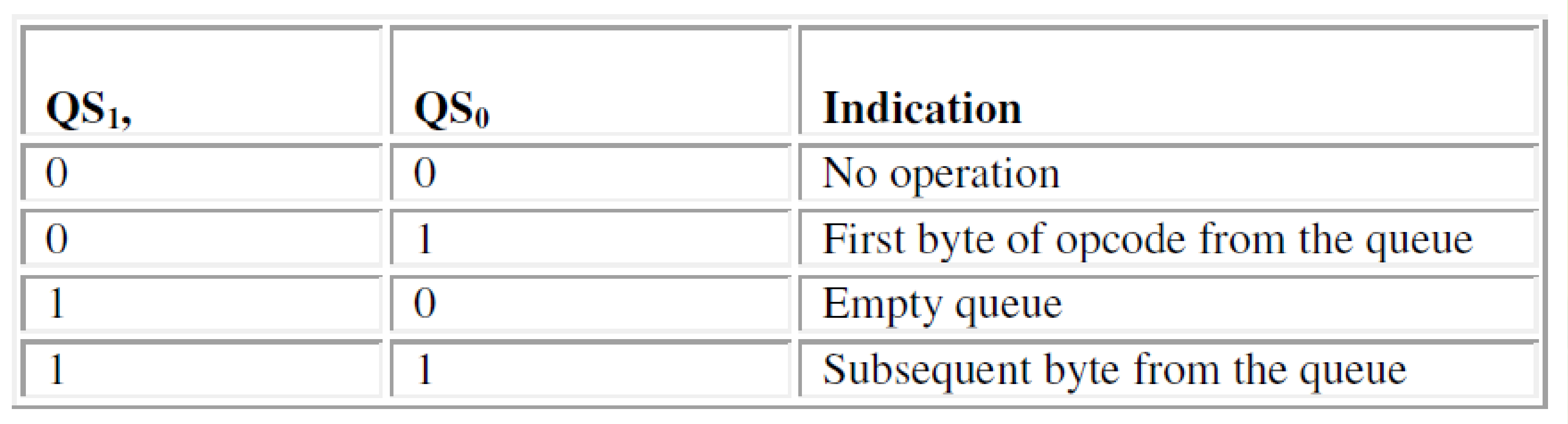
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	**Indication**
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

Table 1.3

LOCK*--

- other system bus masters will be prevented from gaining the system bus,
- When LOCK is ‘0’ it is activated using LOCK prefix instruction.
- During critical instruction execution this instruction activated.

- **QS1, QS0-Queue Status**
- status of the **code prefetch queue** can be observed with these lines



QS_1,	**QS_0**	**Indication**
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

- **RQ*/GT0*, RQ*/GT1*-ReQuest/Grant**
- used by other local bus masters, in maximum mode, to force the processor to release the local bus at the end of the processor's current bus cycle.
- RQ*/GT0*-Highest priority

Addressing Modes

- The way in which the operand is specified is called as the AM(Addressing modes)
- 2 Categories are provided
- 1.Data   2.Branch

- 1.Immediate 9.Intrasegment Direct
- 2.Direct 10.Intrasegment Indirect
- 3.Register 11.Intersegment Direct
- 4.Register Indirect 12. Intersegment Indirect
- 5.Indexed
- 6.Register Relative
- 7.Based Indexed

- **1.Immediate AM**
- Immediate data is part of instruction
- Ex- MOV AX,1234h
- **2. Direct AM**
- 16bit address is directly specified in the instruction
- Ex-MOV AX,[1234h]

3.Register AM

-Data's are stored in the register

- All the register except IP is used for storage.
- Ex- MOV AX,BX

4.Register Indirect AM

-Registers are used to hold the Address of the data in indirect way using Offset register(BX,SI,DI)

-Default segment is DS or ES

- **5 Indexed AM**

-Offset is stored on the index register

-DS,ES are default segments for SI & DI

-Ex-MOV AX,[SI]

6.Register Relative AM

-EA is obtained by adding 8 or 16bit relative value with content of any register(BX,BP,SI,DI)

- **Ex-**MOV AX,50h[BX]

- **7.Based Indexed AM**

- EA is calculated by adding content of base register(BX or BP) to the content of Index register (SI or DI). Default segments are(ES/DS)
- **EX-** MOV AX,[BX][SI]

- **8 Relative Based Indexed**

- EA is formed by adding 8 or 16bit displacement with the sum of contents of base register(BX or BP) & any of the Index register(SI or DI).
- **Ex-** MOV AX,50h[BX][SI]

- **9 Intrasegment Direct mode**

- The address for which control has to be transferred lies in the segment & address appears as immediate value.
- EA= 16/8b Displacement+ IP content

Ex- Short jump or long jump instructions

- **10 Intrasegment Indirect mode**
- **-** Displacement address where the control has to be transferred will be found in some register

-11 Intersegment Direct

- The address for which control has to be transferred lies in some other segment & the address specified directly as part of instruction.

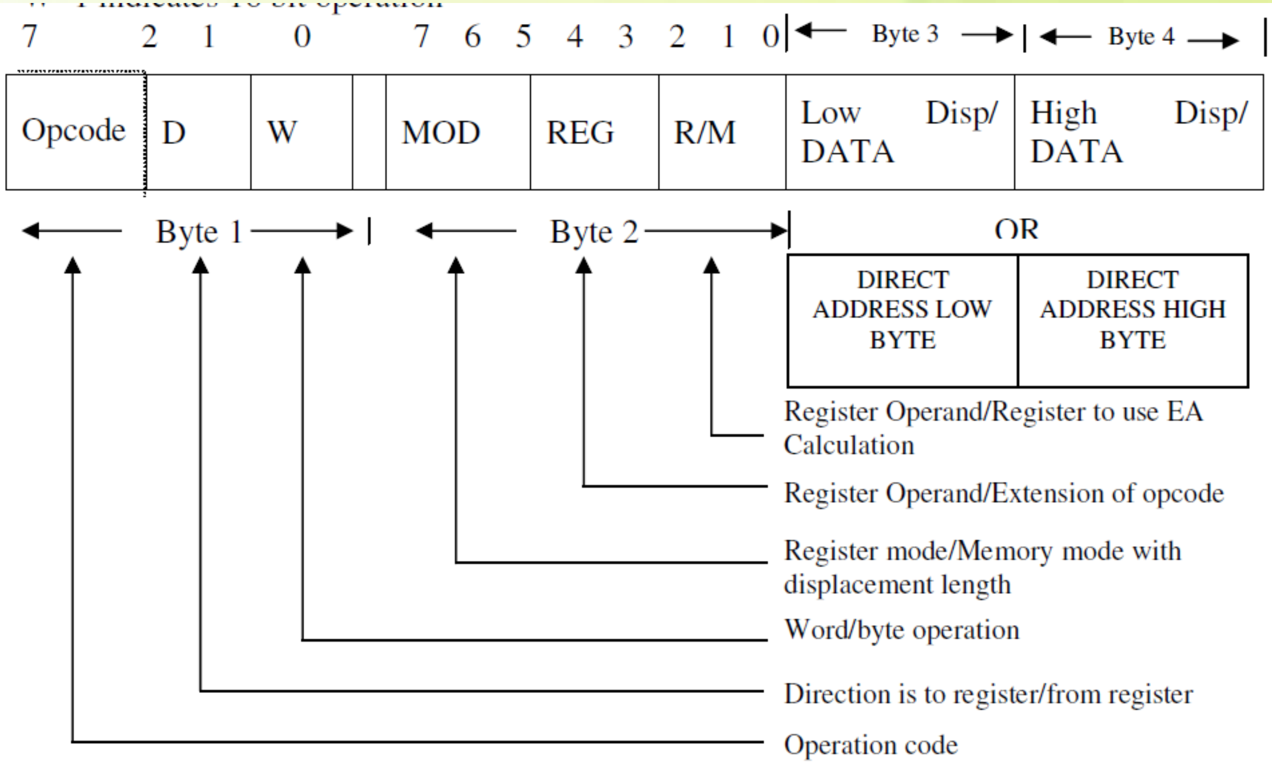
- **12 Intersegment Indirect**

-The address to which control has to be passed will be stored on the memory location from there it will be read to CS & IP

Machine Language Instruction Format :

- The instructions in 8086 varies from 1 to 6 bytes.
- 1. **Opecode field**-type of operation by the processor.
- 2. **Operand field**- Consists of source/destination registers/addresses.

W = 1 indicates 16 bit operation



Register Operand/Register to use EA Calculation

Register Operand/Extension of opcode

Register mode/Memory mode with displacement length

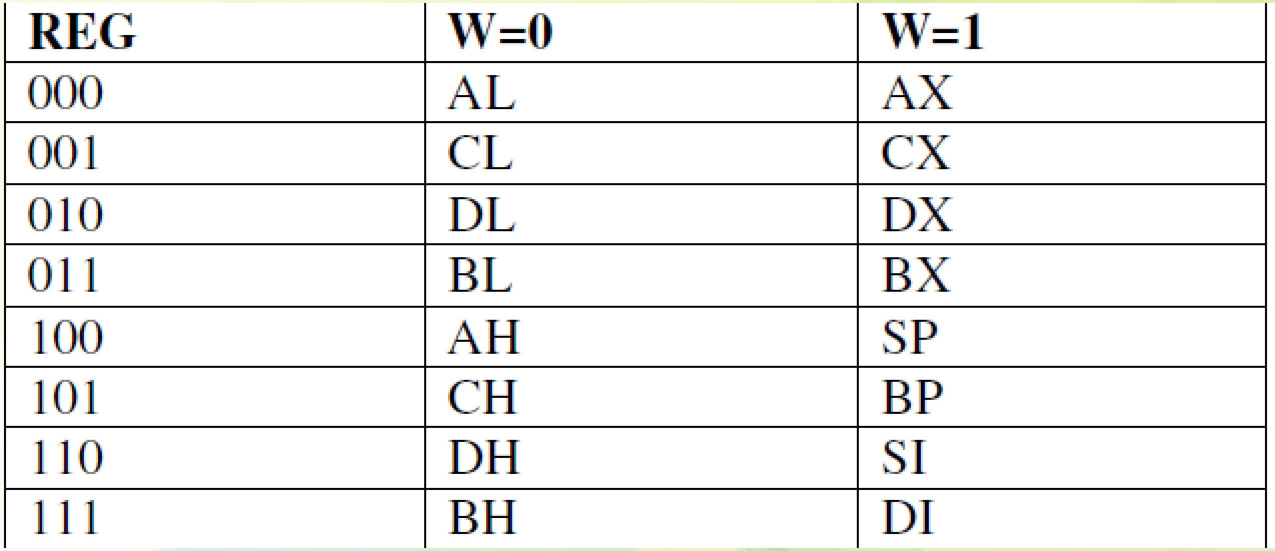
Word/byte operation

Direction is to register/from register

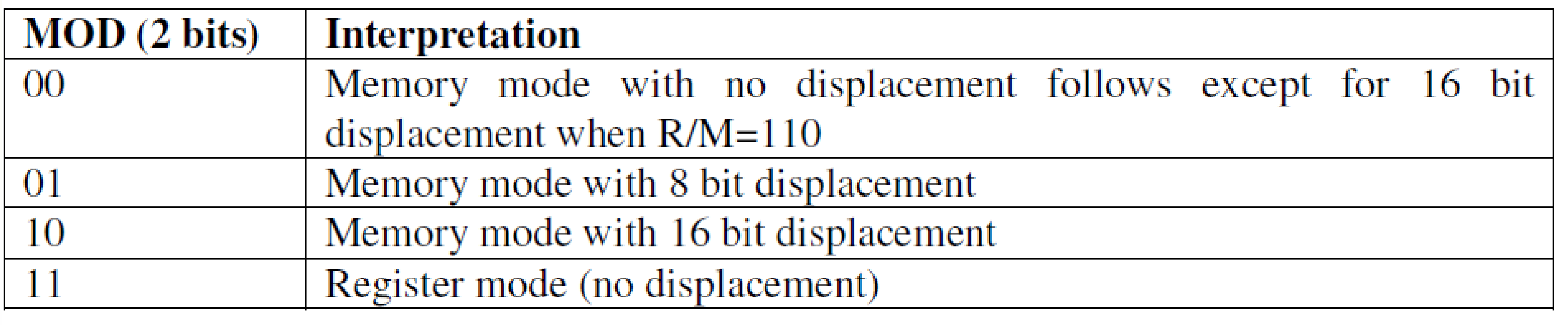
Operation code

- Register Direct bit (D) occupies one bit. It defines whether the register source or destination operand.
- D=1 register operand is the destination operand.
- D=0 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

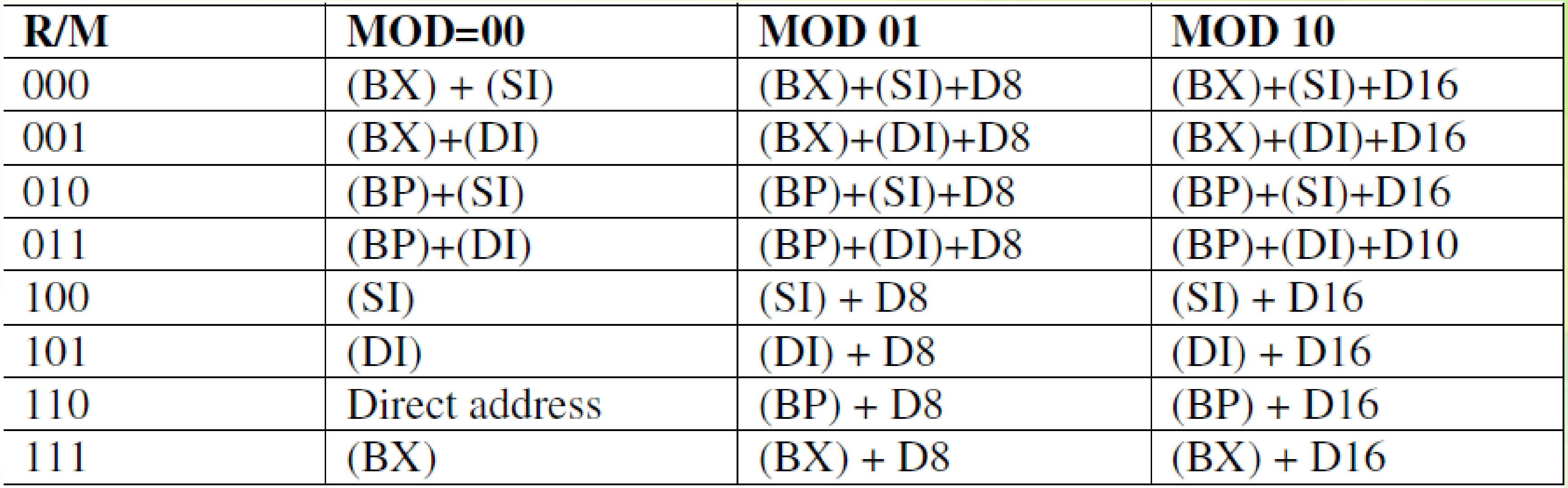


REG	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI



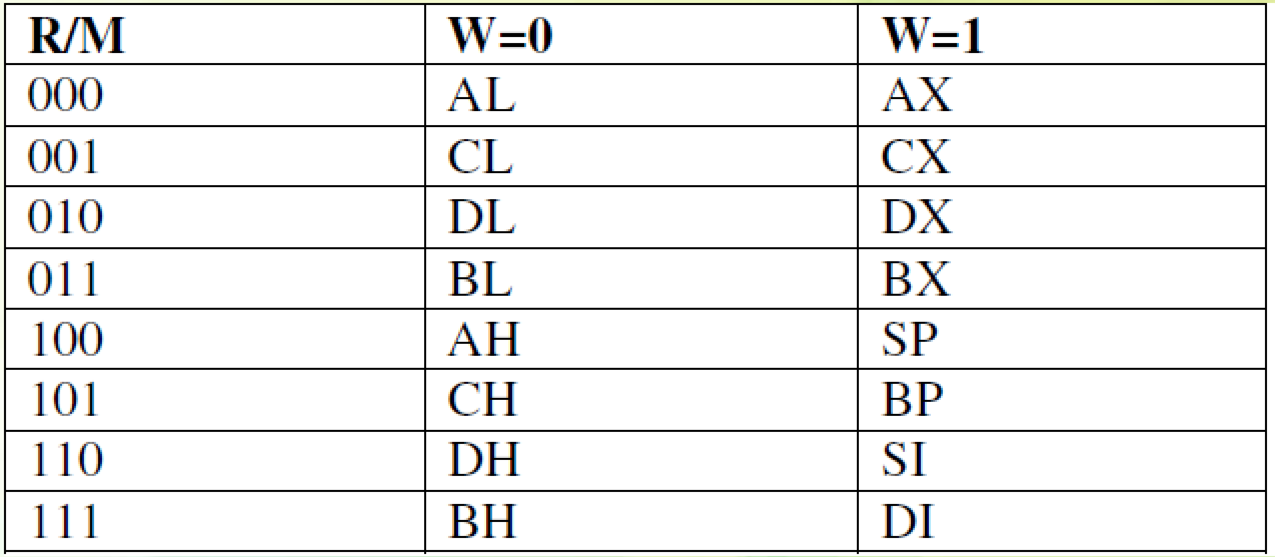
MOD (2 bits)	Interpretation
00	Memory mode with no displacement follows except for 16 bit displacement when R/M=110
01	Memory mode with 8 bit displacement
10	Memory mode with 16 bit displacement
11	Register mode (no displacement)

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)+D10
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

Mode 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

EX:MOV CH, BL This instruction transfers 8 bit content of BL

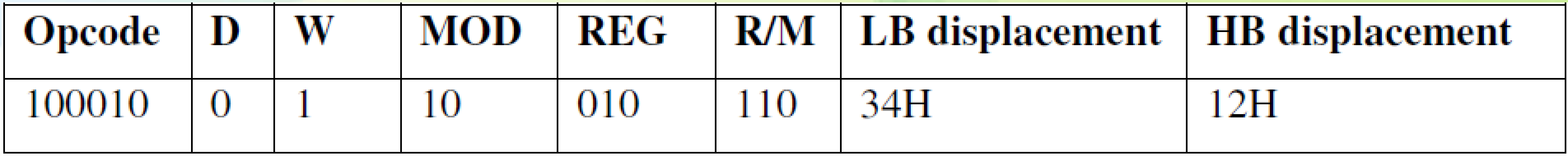
- 6 bit Opcode for this instruction is 100010
- *D*=0 indicates BL is a source operand
- D=1 indicates CH is a destn operand
- **W**=0 indicates 8 bit data transfer
- MOD=11 reg to reg transfer.
- REG=***101*** for CH from table.
- R/M = **011**
- machine code for MOV CH, BL is

10001000 11 **011** ***101***

- **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.
- 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 0010 1011 0001 1101
- 2 B 1 D
- **2B1D16**

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

- Opecode-100010
- D-0 cause DX register is source
- W-1
- MOD-10
- Reg-010
- R/M-110
- 1000 1001 1001 0110 (34 12)
- 89963412



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

Code for MOV DS : 2345 [BP], DX

- Opcode-1000 10
- D-0
- W-1
- MOD-10
- Reg-010
- R/M-110
- displacement = 2345 H.
- SOP byte is 001 SR 110

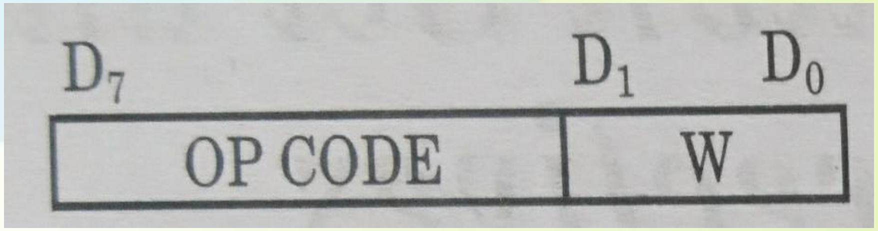
6 General Machine Language Instruction formats.

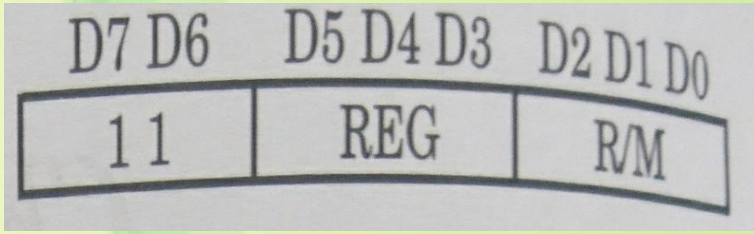
1. **One Byte Instruction-**

One byte long instruction, & data is available in the instruction itself. Least 3 bit of opcode maybe used specify for register else all 8bits are implied.

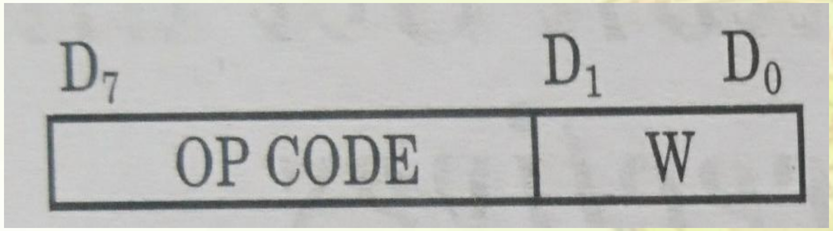
2. Register to Register- 2 byte long instruction

1st byte opcode & width ,2nd byte Reg & R/M field

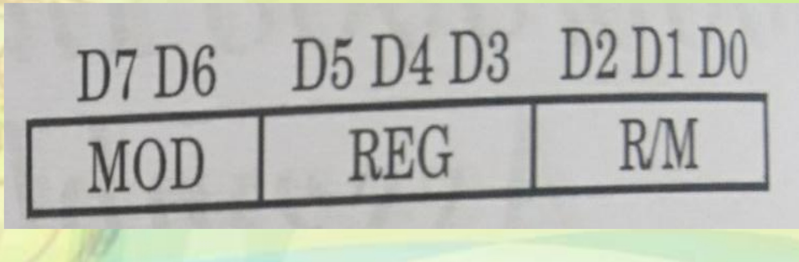




3. **Register to/from memory with no displacement-** 2 byte long

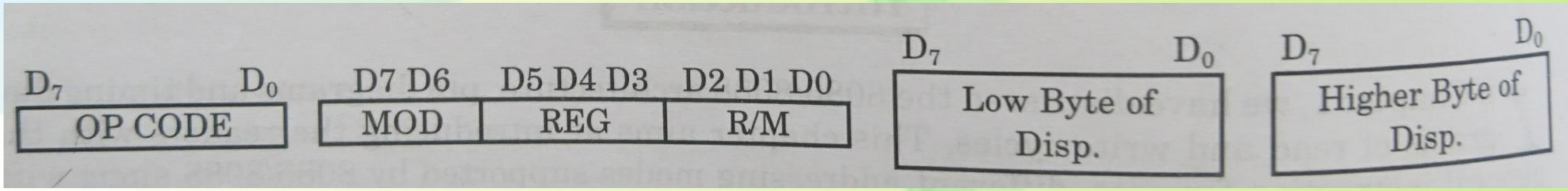


D_7 D_1	D_0
OP CODE	W



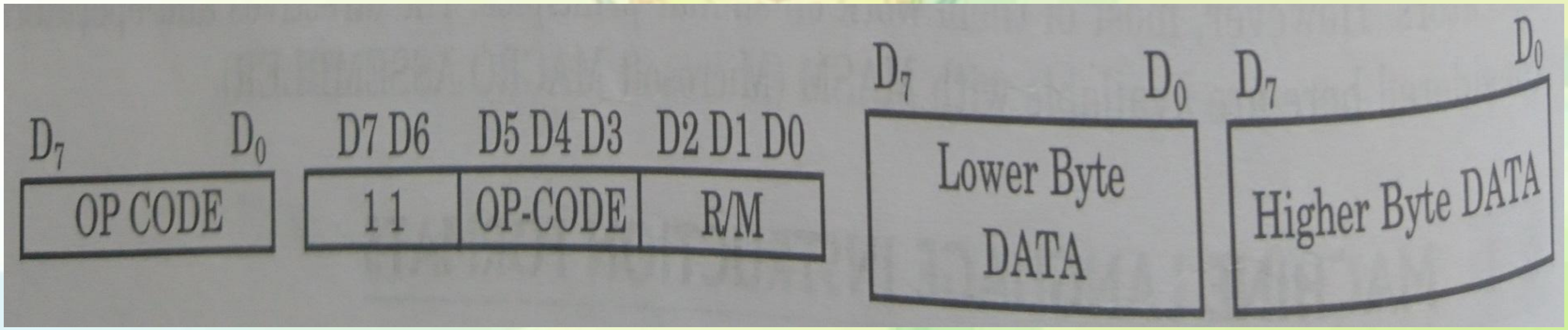
D7 D6	D5 D4 D3	D2 D1 D0
MOD	REG	R/M

4. **Register to/from memory with displacement-** 1 or 2 byte additional for displacement.

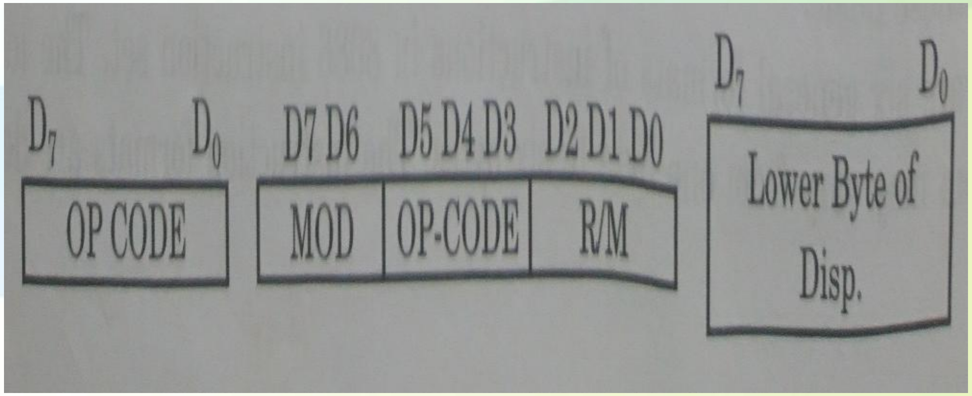


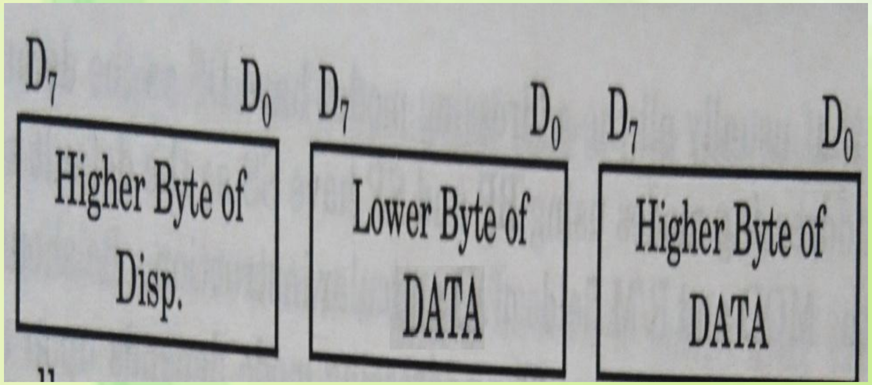
D_7 D_0	D7 D6	D5 D4 D3	D2 D1 D0	D_7 D_0	D_7 D_0
OP CODE	MOD	REG	R/M	Low Byte of Disp.	Higher Byte of Disp.

5.Immediate Operand to register- 1 or 2 byte additional bytes meant for data.



- **6.Immediate Operand to memory with 16bit Displacement-** 2 bytes contain OPCODE,MOD & R/M field remaining 4 bytes contains 2 byte displacement 2 byte data.





The 8086 instructions are categorized into the following main types.

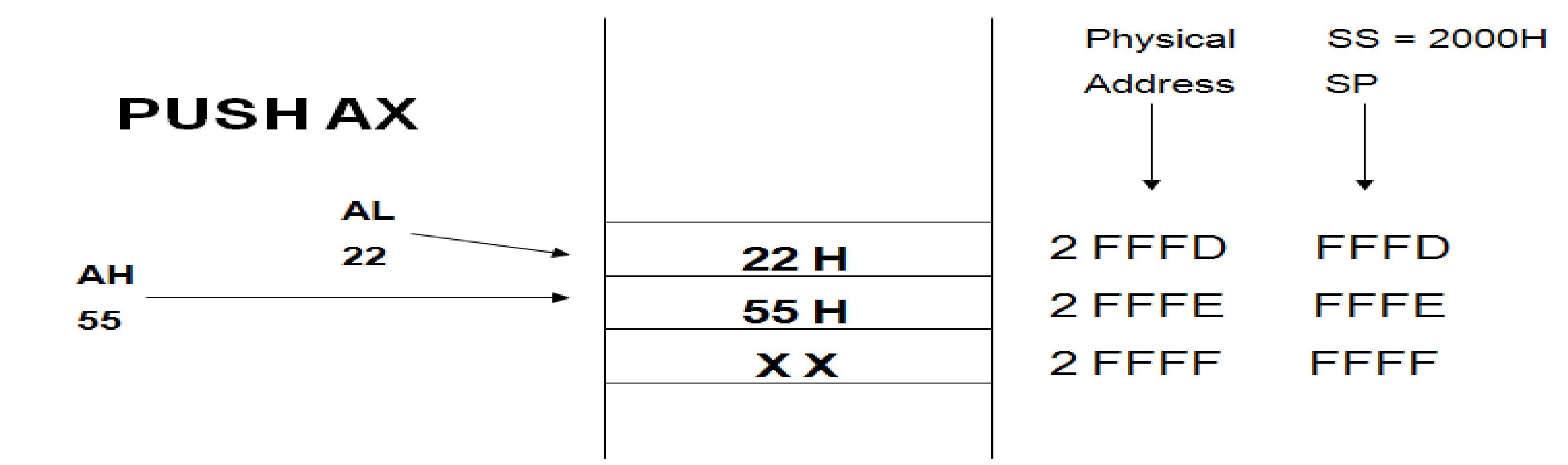
- i. Data Copy / Transfer Instructions
- ii. Arithmetic and Logical Instructions
- iii. Branch Instructions
- iv. Loop Instructions
- v. Machine Control Instructions
- vi. Flag Manipulation Instructions
- vii. Shift and Rotate Instructions
- viii. String Instructions

- **i. Data Copy / Transfer Instructions**

1.MOV :data from some source to a destination

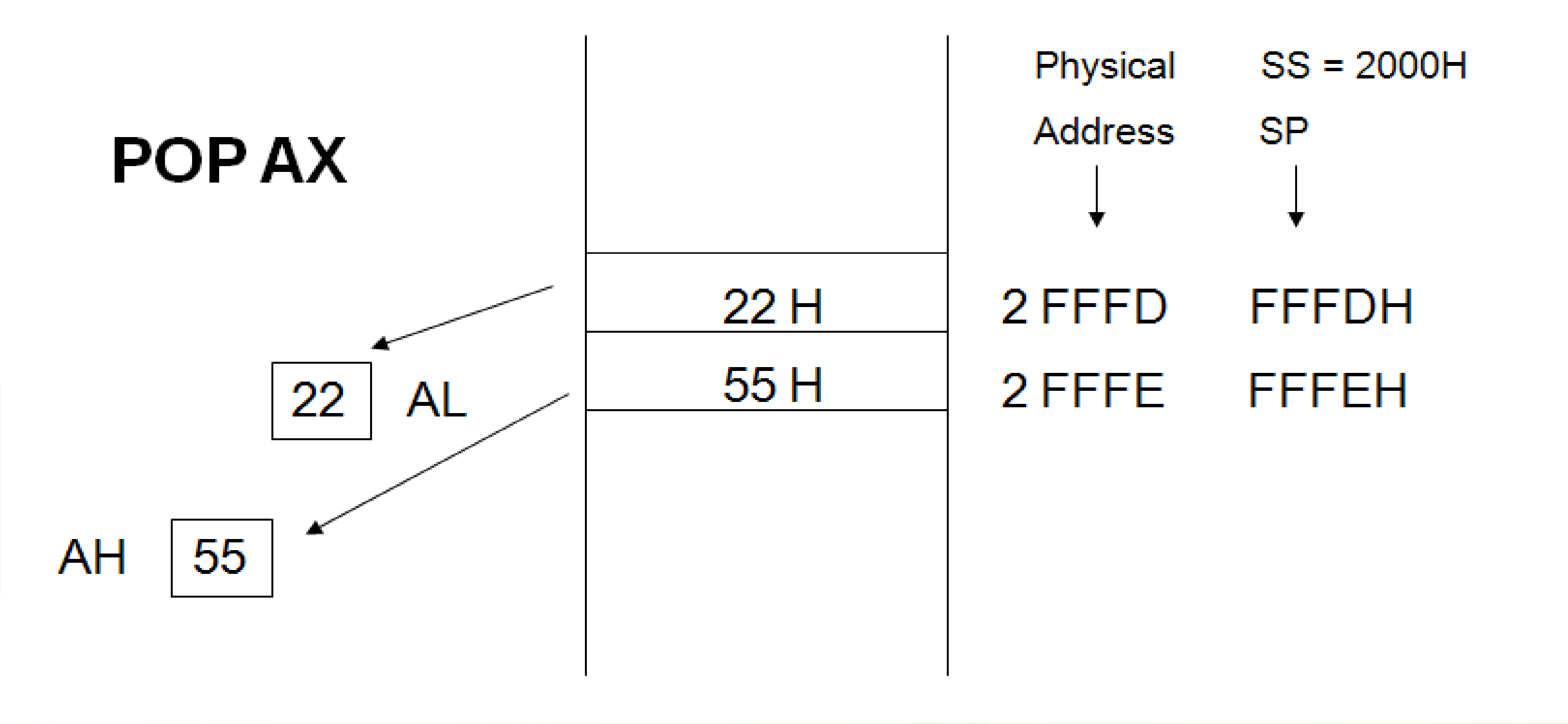
- EX-MOV AX,5000H,MOV AX,50H[BX]

2. PUSH: Push the contents of specified register to stack memory



3.POP : Pop from Stack

- stack pointer is incremented by 2
- POP AX



4.XCHG : Exchange byte or word

- exchange the contents of the specified source and destination operands
- Eg. XCHG [5000H], AX
- XCHG BX, AX

5.IN : Reading the content from address specified in the instruction

- Ex- IN AL,0300h;
- IN AX;// Read from default address stored on DX.

6.OUT: Copy a byte or word from accumulator specified port.

- Eg. OUT 03H, AL
- OUT DX, AX

7.XLAT :

- Translate byte using look-up table.

- Eg. LEA BX, TABLE1
- MOV AL, 04H
- XLAT
- Simple input and output port transfer Instructions:

8.LEA :

- Load effective address of operand in specified register.
- [reg] offset portion of address in DS
- Eg. LEA reg, offset

9.LDS: Load DS register and other specified register from memory(32bit).

- [reg] [mem]
- [DS] [mem + 2]
- Eg. LDS reg, mem

10 LES:

- Load ES register and other specified register from memory.
- [reg] [mem]
- [ES] [mem + 2]
- Eg. LES reg, mem

Flag transfer instructions:

11.LAHF: Load (copy to) AH with the lower byte of flag register.

- [AH] [Flags low byte]
- Eg. LAHF

12.SAHF:

- Store (copy) AH register to low byte of flag register.
- [Flags low byte] [AH]
- Eg. SAHF

13.PUSHF: Copy flag register to top of stack.

- [SP] [SP] – 2
- [[SP]] [Flags]
- Eg. PUSHF

14.POPF : Copy word at top of stack to flag register.

- [Flags] [[SP]]
- [SP] [SP] + 2

Arithmetic Instructions:

1.ADD :instruction adds contents of the source to destination.

- Eg. ADD AX, 0100H
- ADD 0100H

2.ADC : Add with Carry

3.SUB : Subtract

- subtracts the source &destn content
- Eg. SUB AX, 0100H

4.SBB : Subtract with Borrow

- Eg. SBB AX, 0100H
- SBB AX, BX

5.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]

6.DEC : Decrement

- The decrement instruction subtracts 1 from the contents of the specified register or memory location.
- Eg. DEC AX
- DEC [5000H]

7.CMP : Compare

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

8.AAA-ASCII Adjust After Addition

- Executed after ADD instruction, which adds two ASCII coded operands to give byte in AL
- EX.1--- If AL =58--- before AAA

  AL =08--- after AAA
- EX.2--- If AL =5A--- before AAA

  AL =00--- after AAA

  AH =01--- after AAA

  AX=0100h-- after AAA

9.AAS-ASCII Adjust After Subtraction

- Executed after SUB instruction, which subtract two ASCII coded operands to give byte in AL
- EX.1--- If AL =58--- before AAS

  AL =08--- after AAS
- EX.2--- If AX =050A--- before AAS

  AL =04--- after AAS(AL=AL-6)

  AH =05--- Before AAS

  AX=0404h-- after AAS

10.AAM-ASCII adjust for Multiplication

- Packed BCD of AL converted to Unpacked in AH &AL

  Ex. Suppose AL =5D after multiplication

  After AAM AH=06 and AL=03

11.AAD-ASCII adjust After Division

- Unpacked BCD of AH & AL converted into Packed BCD in AL

- Suppose AH =05 & AL=06 before AAD
- After AAD AL=38H(equivalent hex of 56d)

11.NEG : Negate

- forms 2’s complement of the specified destination
- Eg. NEG AL
- AL = 0011 0101 35H Replace number in AL with its 2’s complement
- AL = 1100 1011 = CBH

12.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)

13.IMUL :Signed Multiplication

- 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]

14.CBW : Convert Signed Byte to Word

- 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

15.CWD : Convert Signed Word to Double 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. 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

16.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.
- 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

17.IDIV : Signed division

2. Control Transfer Or Branching Instructions

- 1. Unconditional Control Transfer(Branch) Instruction

- 2. Conditional Control Transfer(Branch) Instruction

1.CALL : Unconditional Call

Instruction is used to call sub program from a main program.

- Address of procedure may be specified directly or indirectly.
- Two types of procedure depending upon whether it is available in the same segment or in another segment.

i. Near CALL i.e., ±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.

2.RET: Return from the Procedure.

- At the end of the procedure, the RET instruction must be 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.

3.INT N: Interrupt Type N.

- 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

4.INTO: Interrupt on Overflow

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

5.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.

6.LOOP Unconditionally

- This instruction executes the part of the program from the Label or address
- LOOP instruction CX indicates number of times the loop has to be performed

7.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.
- 2 types

1.Far JMP(CS,DS,ES,SS) (Inter segment)

2.Near JMP(+/- 32K) (Intra segment)

3. Short JMP(127- -128)(Intra segment)

2.Conditional Branch Instructions

- **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=0 and CX=0.

Conditional JMP instruction based on more than one flag.

Name/Alt	Meaning	Flag setting
JL/JNGE	Jump less than/not greater than or =	(SF xor OF) = 1
JNL/JGE	Jump not less than/greater than or =	(SF xor OF) = 0
JG/JNLE	Jump greater than/not less than or =	((SF xor OF) or ZF) = 0
JNG/JLE	Jump not greater than/ less than or =	((SF xor OF) or ZF) = 1
JB/JNAE	Jump below/not above or equal	CF = 1
JNB/JAE	Jump not below/above or equal	CF = 0
JA/JNBE	Jump above/not below or equal	(CF or ZF) = 0
JNA/JBE	Jump not above/ below or equal	(CF or ZF) = 1

MODULE 2
Instructions Sets

- **Logical Instructions**

1. AND

EX- AND AX,BX

2. OR

EX- OR AX,0098h

3. NOT

EX- NOT AX

4. XOR

EX- XOR AX,BX

5.TEST-Logical compare

-It is bit by bit AND of operands result will be 1 or 0

Ex- TEST AX,BX

- **6 SHL/SAL: Shift Logical/Arithmetical**
- shift the operand to the left by number of positions specified through carry & replace its bits by 0.
- **Ex.1- SAL AL,01**
- Before SAL/SHL AL=10101011 CF=0
- After SAL/SHL AL= 01010110 CF=1

6 SHR: Shift Logically Right

- **Ex.1- SHR AL,01**
- Before SHR AL=10101011 CF=0
- After SHR AL= 01010101 CF=1

7 SAR: Shift Arithmetically Right

- **Ex.1- SAR AL,01**
- Before SAR AL=10101011 CF=0
- After SAR AL= 11010101 CF=1

8. ROR:Rotate Right without Carry

Ex- ROR BL,01

Before ROR BL= 00001111 CF= not affected

After   ROR BL= 10000111 CF= not affected

9. ROL:Rotate Left without Carry

Ex- ROL BL,01

10. RCR:Rotate Right Through Carry

Ex- RCR BL,01

Before RCR BL= 00001111 CF=0

After   RCR BL= 00011110 CF=1

11. RCL:Rotate Left Through Carry

Ex- RCL BL,01

String Manipulation Instructions

1.REP-Repeat Instruction Prefix-Repeatedly Executes the mentioned instruction until CX becomes. Types-REPE,REPZ,REPNE,REPNZ

2.MOVSB/MOVSW-Move String Byte or Word-

Moves a Byte or word from source m/m (SI+(10*DS) to destination location(DI+(10*ES)

& DF will access in ascending or in descending order

EX- REP MOVSB

3.CMPS:Compare 2 strings

EX-REPE CMPSW(compare two strings held by SI & DI)

4.SCAS:Scan String byte or string word

Scans the ES:DI location by the string stored in AX register ,length is specified in CX register controls direction.

EX- REPNE SCASB

5.LODS:Load String Byte or Word

-Lods the content of AL/AX by the data specified by DS:SI, SI is modified depending on DF

- **EX-LODSB**

- **6.STOS: Store String Byte or String Word:**

- It will store the AL/AX content to location specified by ES:DI, DF will be 0 or 1
- **EX-STOSB,STOSW**

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(E).

1.HLT – Halt the process- NO F &E

- HLT state is revoked when INTR is activated or NMI pin is asserted, or a RESET signal on RESET pin.

2.WAIT –Wait for Test input. (NO PROCESSING) ,

- The processor stay in this condition until TEST pin is asserted.(TEST*=0→wait state)Or INTR ,or NMI is activated

3 NOP – No operation

- for Delay generation, NO E, only F & D
- It takes 3 clock cycles for execution
- Doesn’t affect any flags.

4. ESC – Escape to external device

- Passing the instruction to the co-processor which shares the address and data bus.
- Represented by 6 bit code
- Most of the time ESC is treated as NOP instruction.

5. LOCK – lock instruction prefix.

- It is 1 byte prefix,Does not affect any flag
- Used during accessing the data between register and memory
- LOCK XCHG sem,AL

Assembler Directives and Operators.

1.DB: Define Byte,

2.DW: Define Word

3. DQ: Define Quad word(4 words)

4. DT: Define Ten Bytes

5. DD: Define Double Word

Ex1. LIST DB 0lH

Ex2. LIST DW 0001H

Ex3. MESSAGE DW 'GOOD MORNING‘

Ex4. WDATA DW 5 DUP (6666H)

EX5. Array DB 100 DUP(0,2,3 DUP(1,2),0,3)

- Array DB 2 DUP(0,1,2,?)
- Mes DB ‘HELLO’
- Mes DB ‘H’,’E’,’L’,’L’,’O’
- Array DB 100 DUP(0,2,3 DUP(1,2),0,3)
- Parameter DW p1

  DW p2

  DW p3

Offset of P1,P2,P3 are stored.

ASSUME: Assume Logical Segment Name

- Used to inform the names of the different logical segments in the design environment.

Ex:1.ASSUME CS : CODE

2. ASSUME DS : DATA

- **END: END of Program**
- Marks the end of an assembly language program.
- Ignores all the lines after this instruction.
- **ENDS: END of Segment**
- This directive marks the end of a logical segment

DATA SEGMENT

DATA ENDS

- **ENDP: END of Procedure**

EVEN: Align on Even Memory Address

- Directive updates the location counter to the next even address if the current location counter contents are not even, if even then continues with the address
- EVEN
- PROCEDURE ROOT
- .
- .
- ROOT ENDP

- **EQU: Equate**
- The directive EQU is used to assign a label with a value or a symbol
- Example
- LABEL EQU 0500H
- ADDITION EQU ADD

EXTRN: External and PUBLIC

- EXTRN informs the assembler that the names, procedures and labels declared after this directive have already been defined

- The other module, where the names, procedures and labels actually appear, they must be declared public, using the PUBLIC directive

Example

```
MODULE1 SEGMENT
 PUBLIC FACTORIAL FAR
Module1 ENDS
MODULE2 SEGMENT
   EXTRN FACTORIAL FAR
MODULE2 ENDS
```

GROUP: Group the Related segment

- The directive is used to form logical groups of segments with similar purpose or type
- Linker/Loader take care that group declared segments or operands must lie within a 64Kbyte memory segment
- PROGRAM GROUP CODE, DATA, STACK
- ASSUME CS: PROGRAM, DS: PROGRAM, SS: PROGRAM

- **LABEL: used to assign the data segment with data type**
- **DATA-LAST LABEL BYTE FAR**
- **LENGTH:length of sstring**
- **NAME: Logical Name of a Module**
- Used to declare the module names.

OFFSET: Offset of a Label

Directive replaces the string 'OFFSET LABEL' by the computed 16bit displacement

- Example:

```
CODE SEGMENT
MOV SI, OFFSET LIST
CODE ENDS
DATA SEGMENT
LIST DB 10H
DATA ENDS
```

ORG: Origin

- The ORG directive directs the assembler to start the memory allotment for the particular segment
- By default the location counter is initialised to 0000
- If ORG 0200H --- Initialized with 0200 location, It can be also used with the data segment.

PROC

- Calling a procedure
- NEAR (<64K)& FAR(>64K) PROCs

- Example
- RESULT PROC NEAR
- ROUTINE PROC FAR

PTR: Pointer

- PTR operator is used to specify the data type -byte or word
- Example:
- MOV AL, BYTE PTR [SI]
- INC BYTE PTR [BX]
- MOV BX, WORD PTR [2000H]

SEG: Segment of a Label

- The SEG operator is used to decide the segment address of the label, variable, or procedure and substitutes

- MOV AX, SEG ARRAY
- ------ Moves segment address of ARRAY into AX

GLOBAL

- The variable declared here can be used by any module in the program

- ROUTINE PROC GLOBAL

SHORT

- displacement is within -128 to +127 bytes

- JMP SHORT LABEL

TYPE

- for byte type, the data type is 1.
- word type variable, the data type is 2,
- for double word type, it is 4
- EX-

Suppose, the STRING is a word array

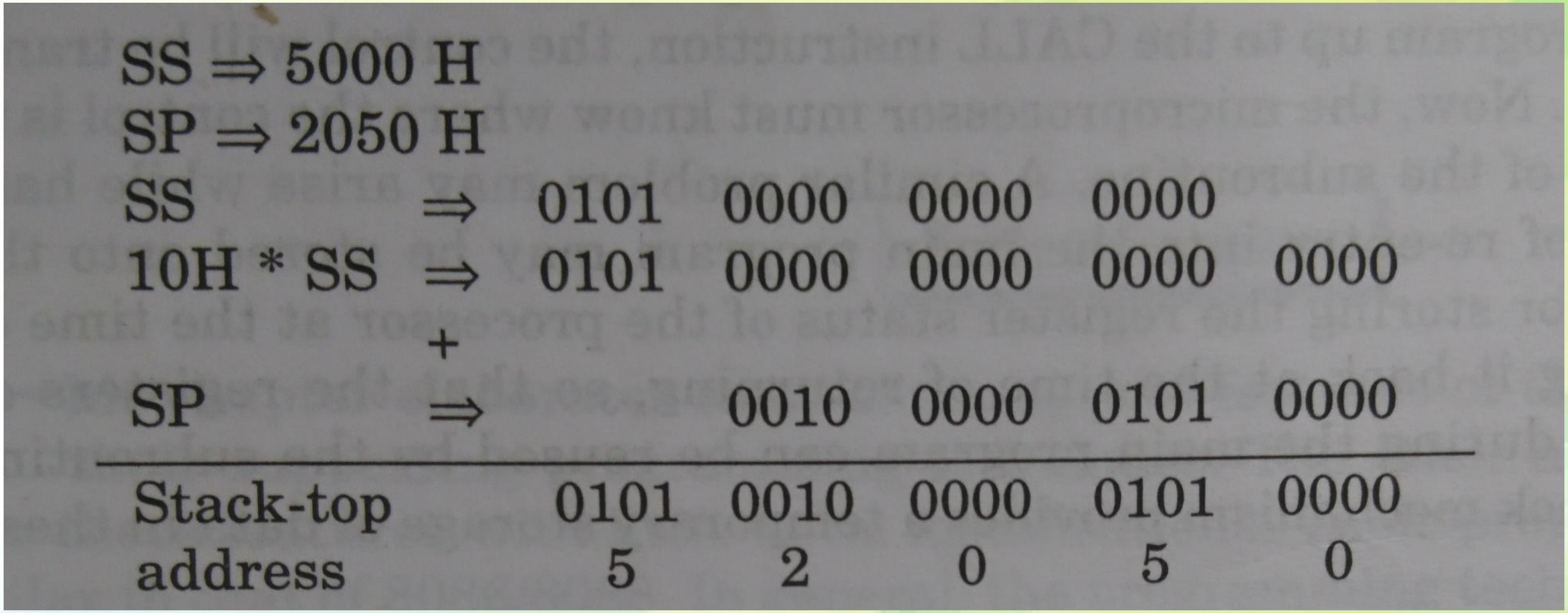
MOV AX, TYPE STRING

the value 0002H in AX.

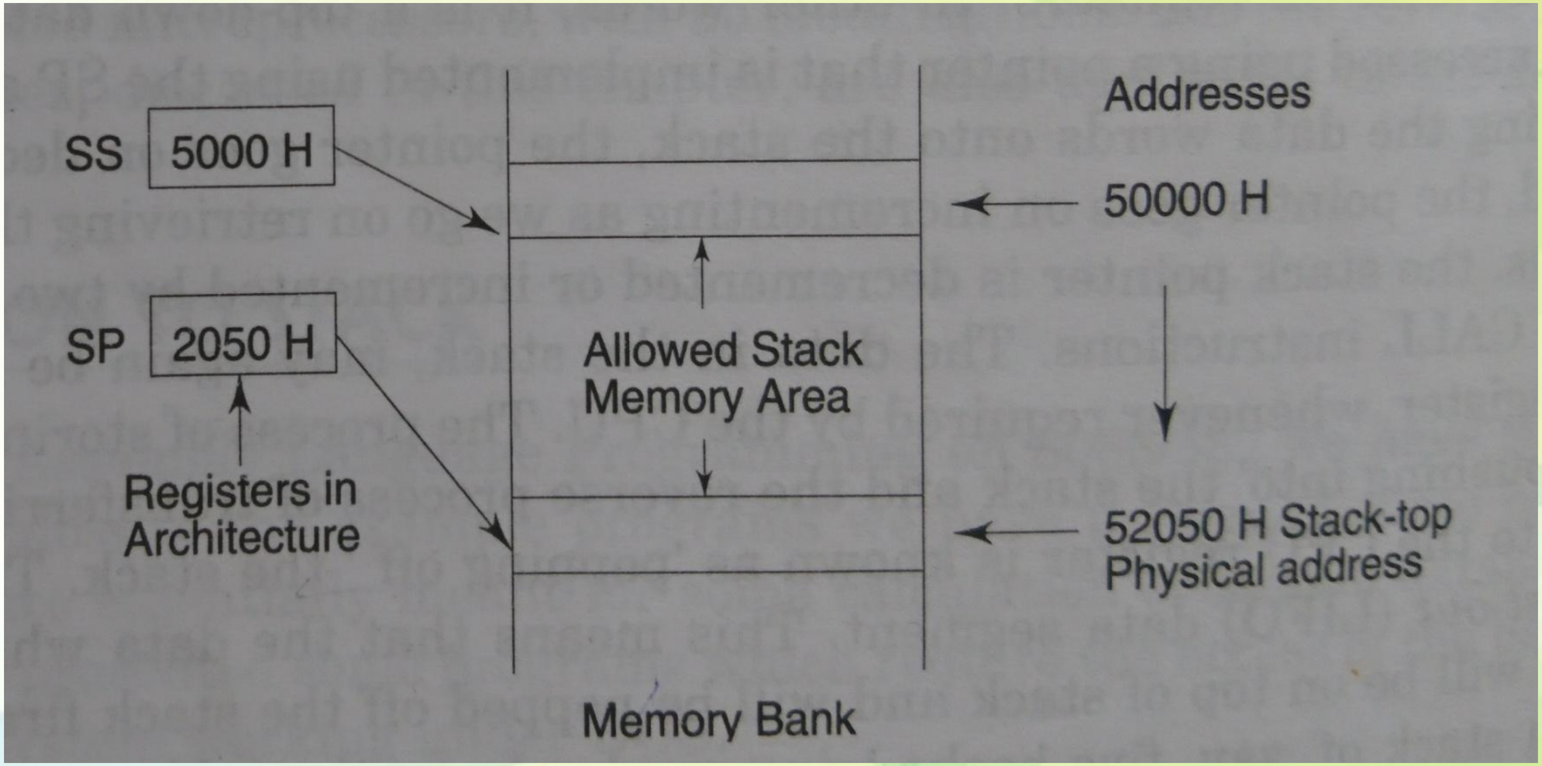
Module 3
Stack & Interrupts

- Stack is temporary memory storage used to store intermediate values, It uses SP & SS registers.
- Stack is operated using PUSH & POP

STACK STRUCTURE OF 8086/88



```
SS ⇒ 5000 H
SP ⇒ 2050 H
SS          ⇒   0101   0000   0000   0000
10H * SS    ⇒   0101   0000   0000   0000   0000
            +
SP          ⇒          0010   0000   0101   0000
-----------------------------------------------------
Stack-top       0101   0010   0000   0101   0000
address           5      2      0      5      0
```


Addresses
SS
5000 H
50000 H
SP
2050 H
Allowed Stack
Memory Area
Registers in
Architecture
52050 H Stack-top
Physical address
Memory Bank

Effect of PUSH and POP SP

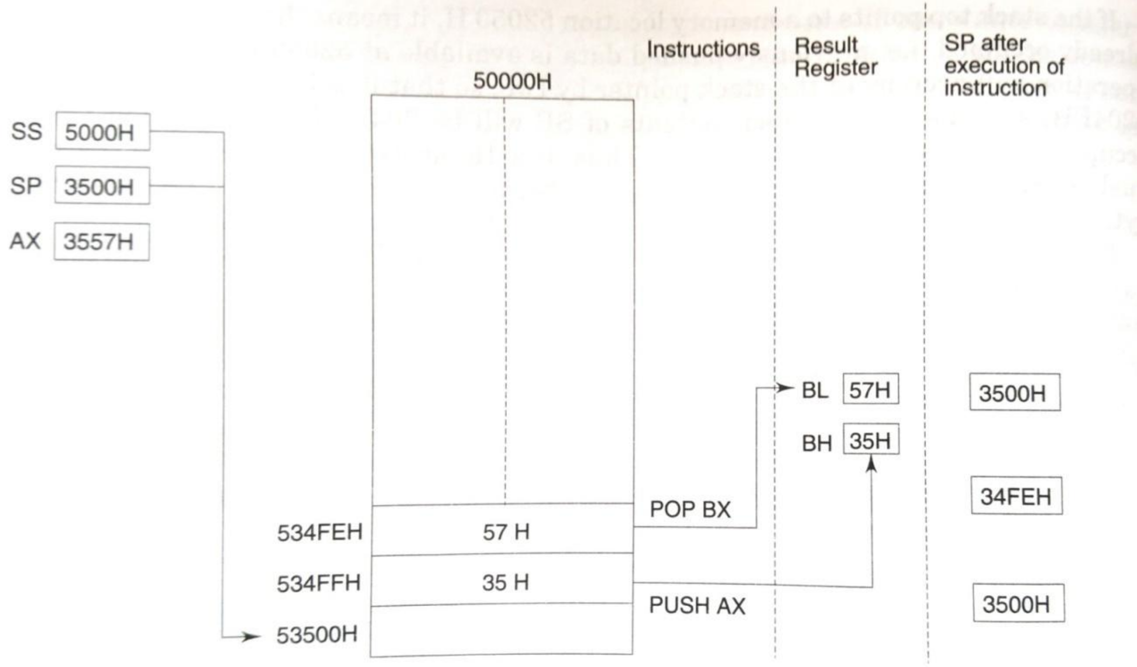


Fig. 4.3 *Effect of PUSH and POP on SP*

Interrupt & Interrupt Service Routines

- Interrupt-It breaks normal sequence of execution.
- Interrupt Service Routine(ISR): It is the program written for interrupt.
- Control is transferred to main program after interrupt execution.
- INTR & NMI
- 255 types of INTR (00 to FFh)
- Interrupt handler handles if more than one interrupt comes at a time

Interrupt cycle of 8086

- 2 types interrupts

1. External Interrupts- Generated by external devices.
2. Internal Interrupts- generated internally by INTR instructions,devide by zero, overflow.

- When NMI,TRAP,devide by zero,INTR occurs then INTA occurs in response.
- ‘IF’ is set when interrupt occurs, if ‘IF’ is not set then interrupt is ignored.
- Current content of IP & CS stored on stack and the new content of IP & CS is loaded with new address of ISR which is obtained by IVT.
- The content of Stack are reloaded into IP& CS when IRET instruction obtained in ISR.
- 1 interrupt requires 4 bytes (2 B CS & 2 B IP)Total 256 interrupts requires 1024 bytes(0000:03FFh)

Interrupt Response sequence

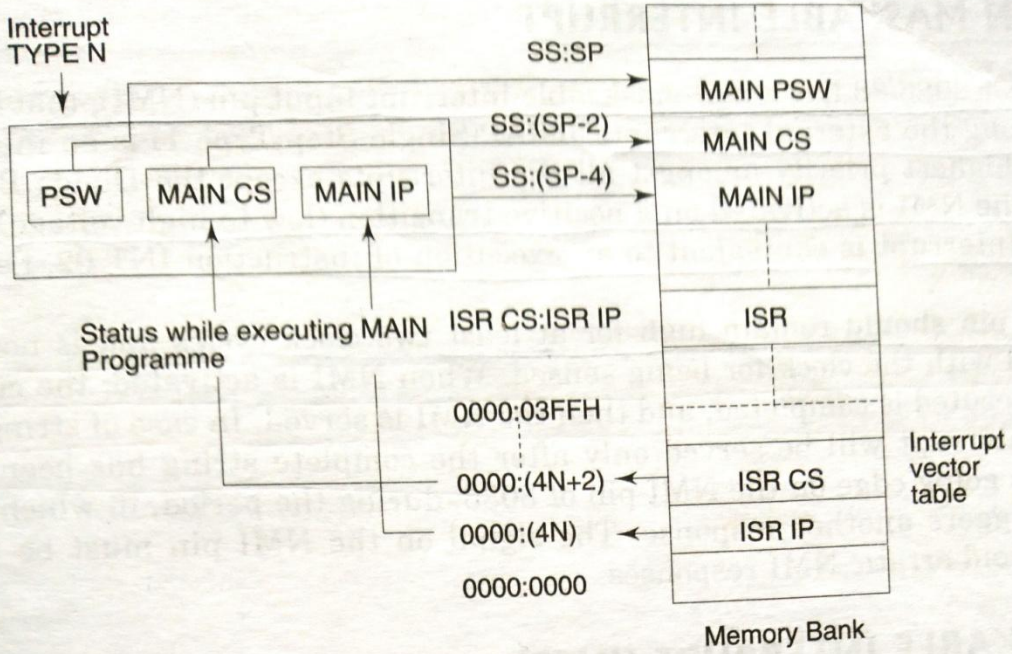
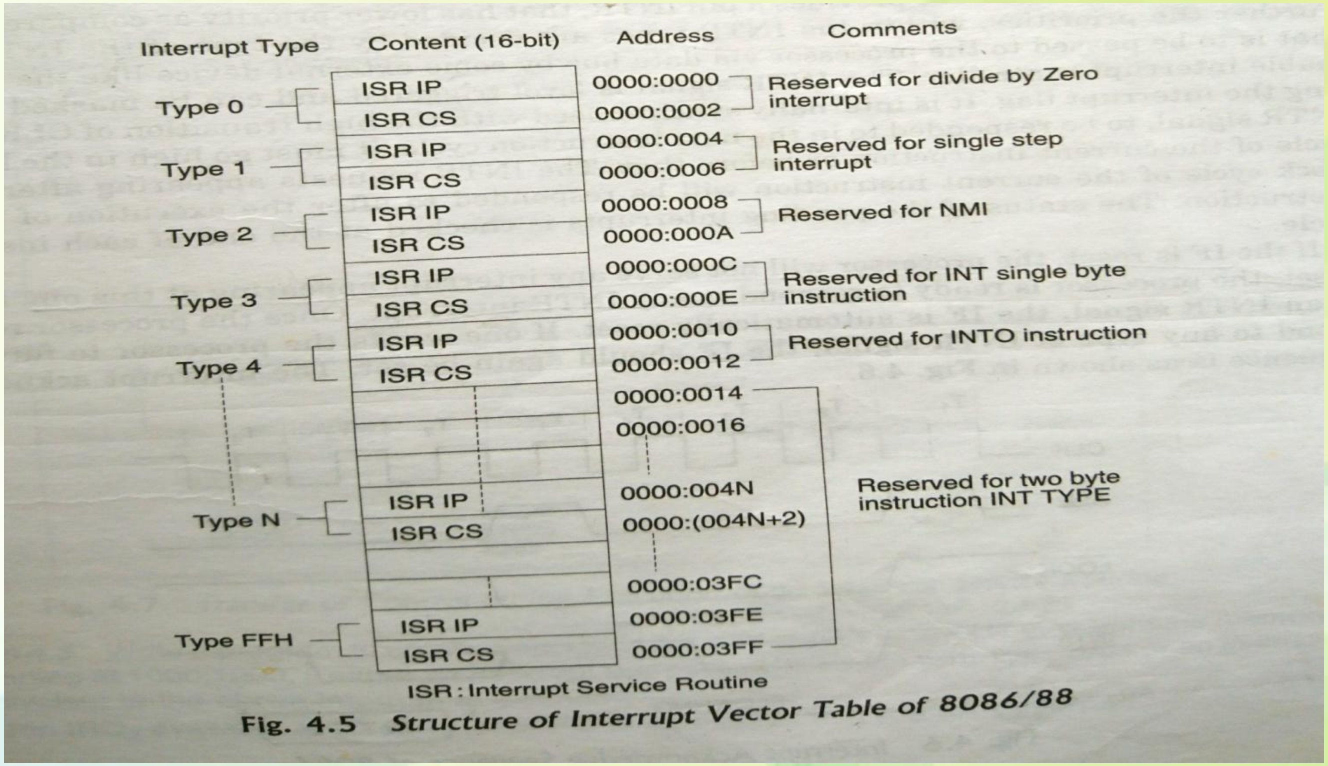


Fig. 4.4 *Interrupt Responce Sequence*

Structure of IVT



Interrupt Type	Content (16-bit)	Address	Comments
Type 0	ISR IP	0000:0000	Reserved for divide by Zero interrupt
	ISR CS	0000:0002	
Type 1	ISR IP	0000:0004	Reserved for single step interrupt
	ISR CS	0000:0006	
Type 2	ISR IP	0000:0008	Reserved for NMI
	ISR CS	0000:000A	
Type 3	ISR IP	0000:000C	Reserved for INT single byte instruction
	ISR CS	0000:000E	
Type 4	ISR IP	0000:0010	Reserved for INTO instruction
	ISR CS	0000:0012	
		0000:0014	Reserved for two byte instruction INT TYPE
		0000:0016	
Type N	ISR IP	0000:004N	
	ISR CS	0000:(004N+2)	
		0000:03FC	
Type FFH	ISR IP	0000:03FE	
	ISR CS	0000:03FF	

ISR : Interrupt Service Routine

Fig. 4.5 *Structure of Interrupt Vector Table of 8086/88*

MACROS

- Macro is a segment of code that needs to be written only once.
- whose basic structure can be repeated several times
- no memory is saved
- no linkage is required
- the assembler replaces the call with the macro code

- **macro expansion** Insertion of the macro code by the assembler for a macro call is referred to as a macro expansion
- The macro definition is constructed as follows

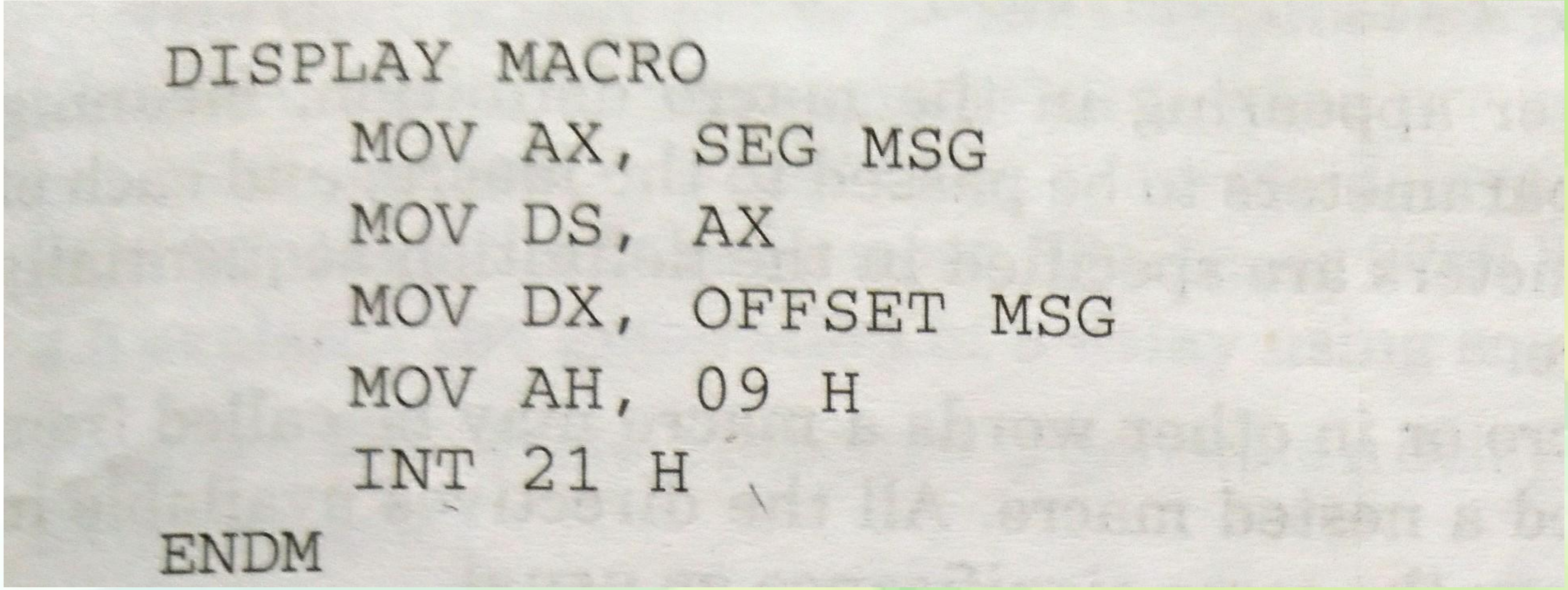
%*DEFINE(Macro name(Dummy parameter list))

(

Prototype code

)

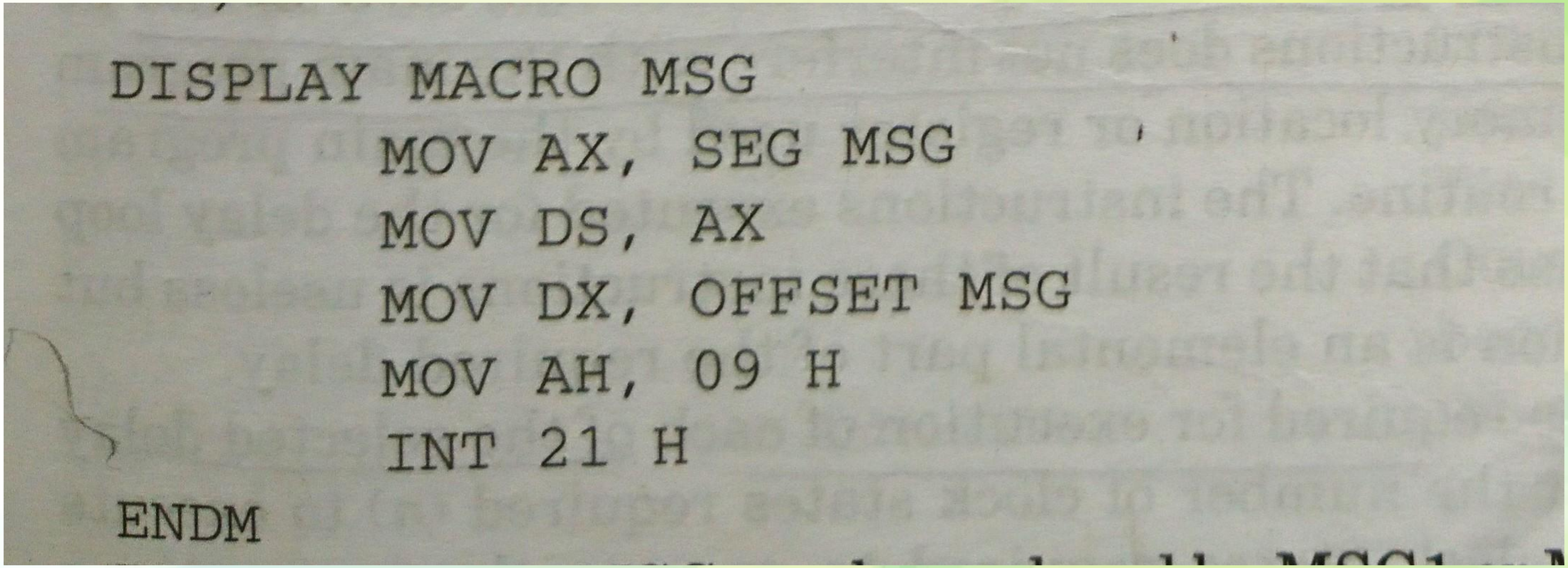
- Macro name has to begin with a letter and can contain letters, numbers and underscore characters



```
DISPLAY MACRO
    MOV AX, SEG MSG
    MOV DS, AX
    MOV DX, OFFSET MSG
    MOV AH, 09 H
    INT 21 H
ENDM
```

Passing Parameter to MACROS

-Replacing MSG by MSG1&MSG2



```
DISPLAY MACRO MSG
        MOV AX, SEG MSG
        MOV DS, AX
        MOV DX, OFFSET MSG
        MOV AH, 09 H
        INT 21 H
ENDM
```

NON Maskable Interrupt

- NMI is non maskable,It has highest priority in External Interrupts(activated from 0 to 1 transation held for 2 clk cycles), level 2 type of interrupt.
- TRAP is at Highest priority in internal interrupts

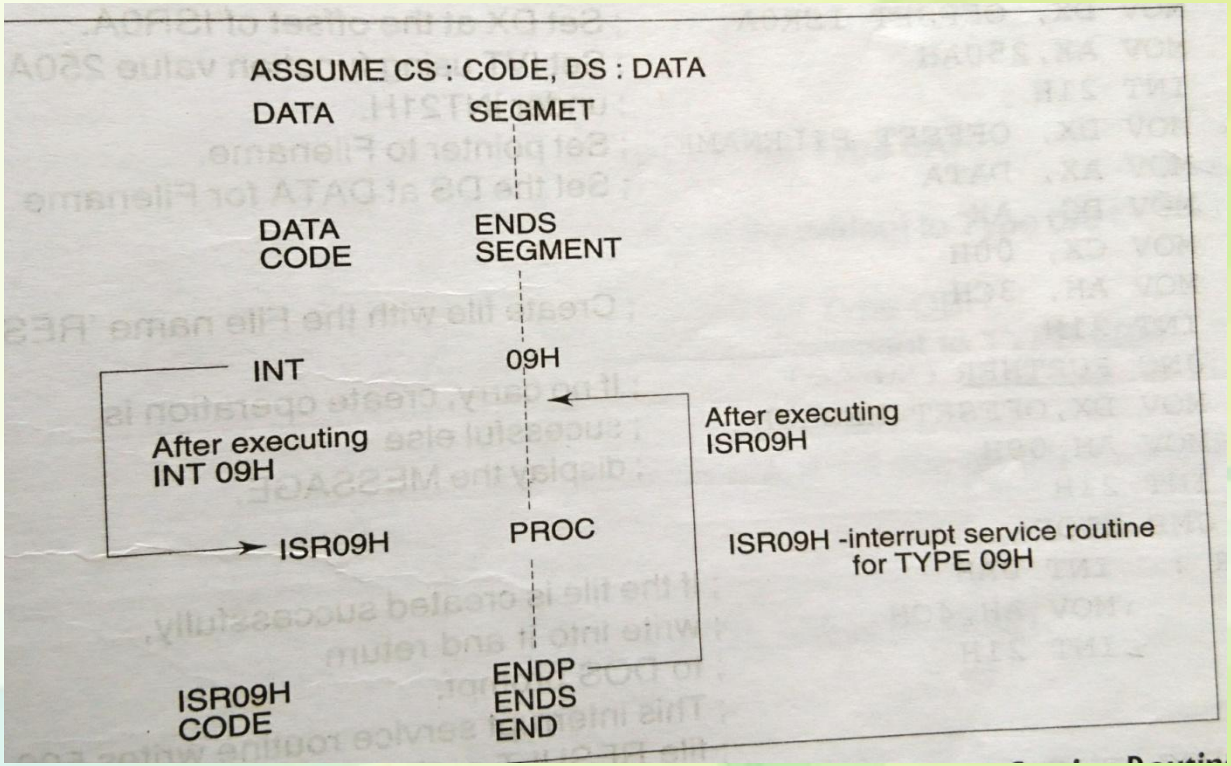
Maskable Interrupt(INTR)

- INTR will be at lowest priority compared to NMI
- INTR is level triggered, & can be masked by resetting “IF” flag.
- Based on “IF” flag INTR is executed.

Interrupt Programming

- Before writing program for interrupt the IVT has to set by either internally or externally.
- ISR will be written in same manner for s/

Interrupt Programming



```
ASSUME CS : CODE, DS : DATA
DATA        SEGMET

DATA        ENDS
CODE        SEGMENT

INT         09H

ISR09H      PROC

            ENDP
ISR09H      ENDS
CODE        END
```

After executing INT 09H

After executing ISR09H

ISR09H -interrupt service routine for TYPE 09H

Nested Interrupts

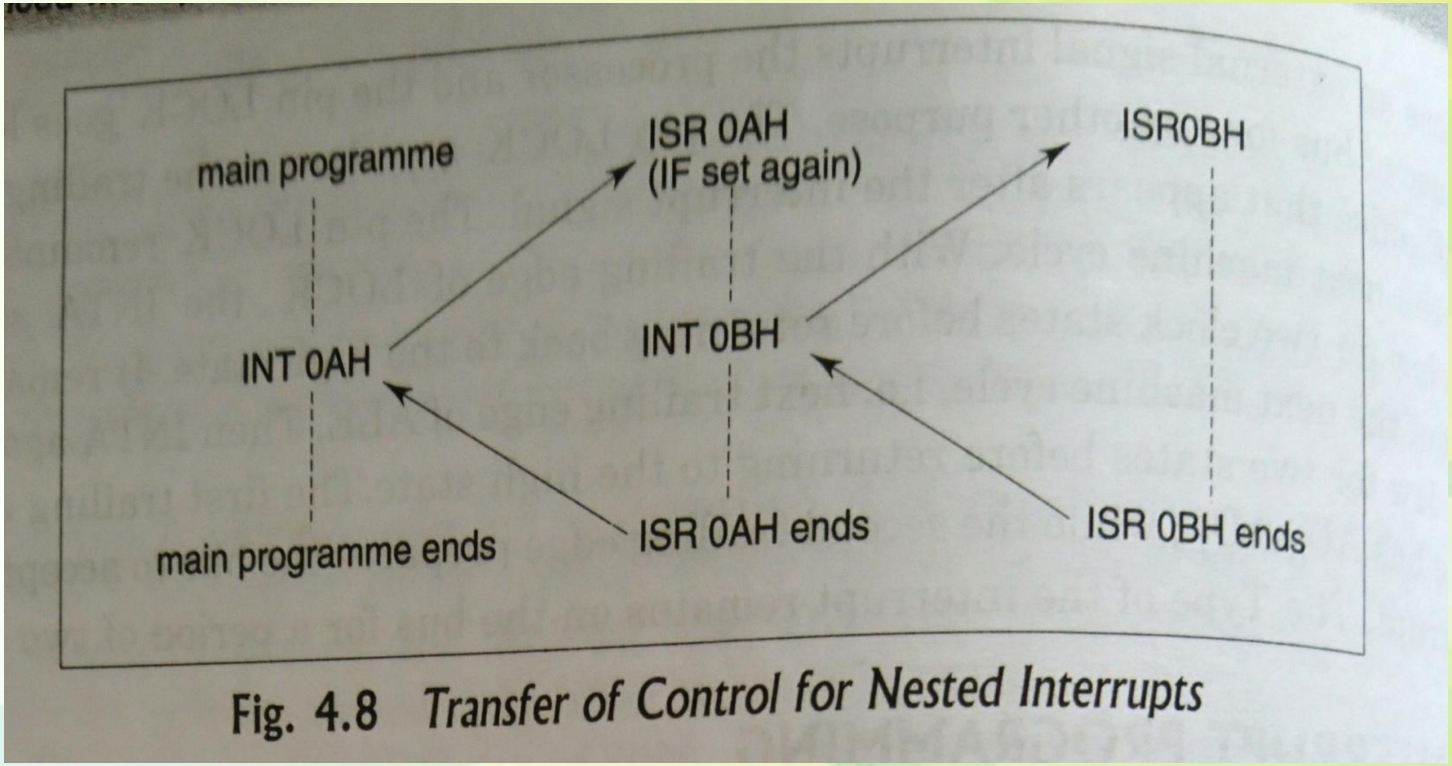
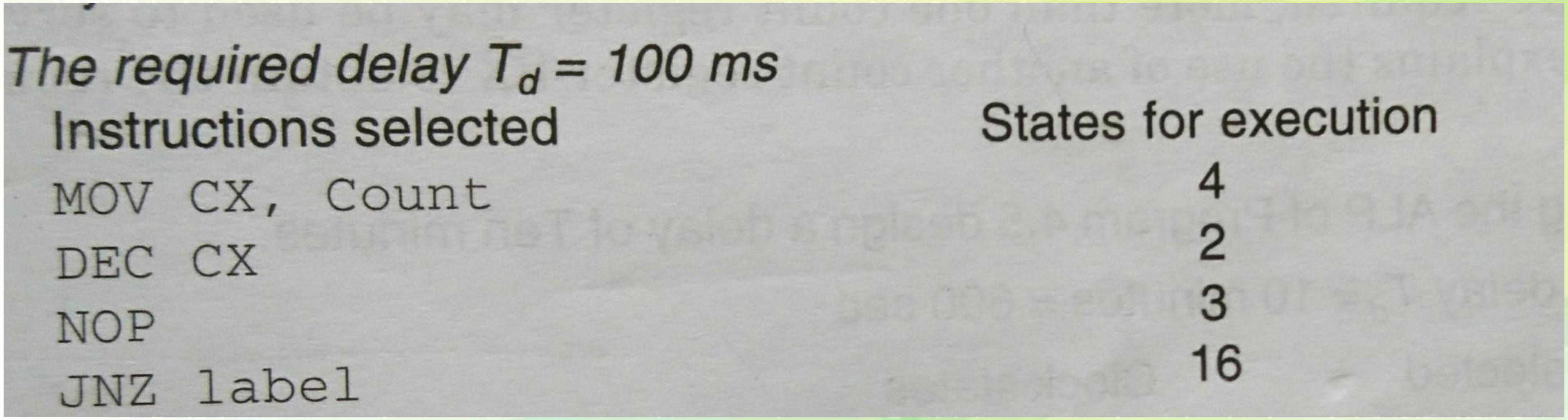


Fig. 4.8 *Transfer of Control for Nested Interrupts*

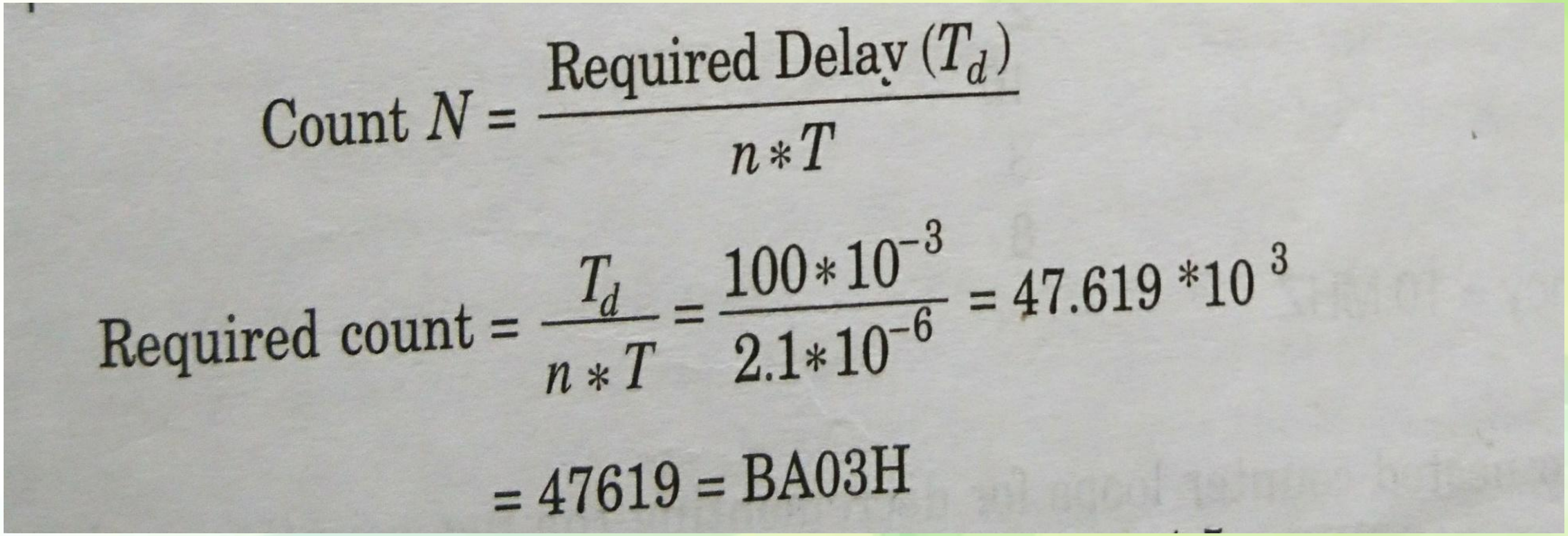
Write a program to generate a delay of 100ms using 8086 system that runs on 100Mhz frequency



The required delay T_d = 100 ms

Instructions selected	States for execution
`MOV CX, Count`	4
`DEC CX`	2
`NOP`	3
`JNZ label`	16

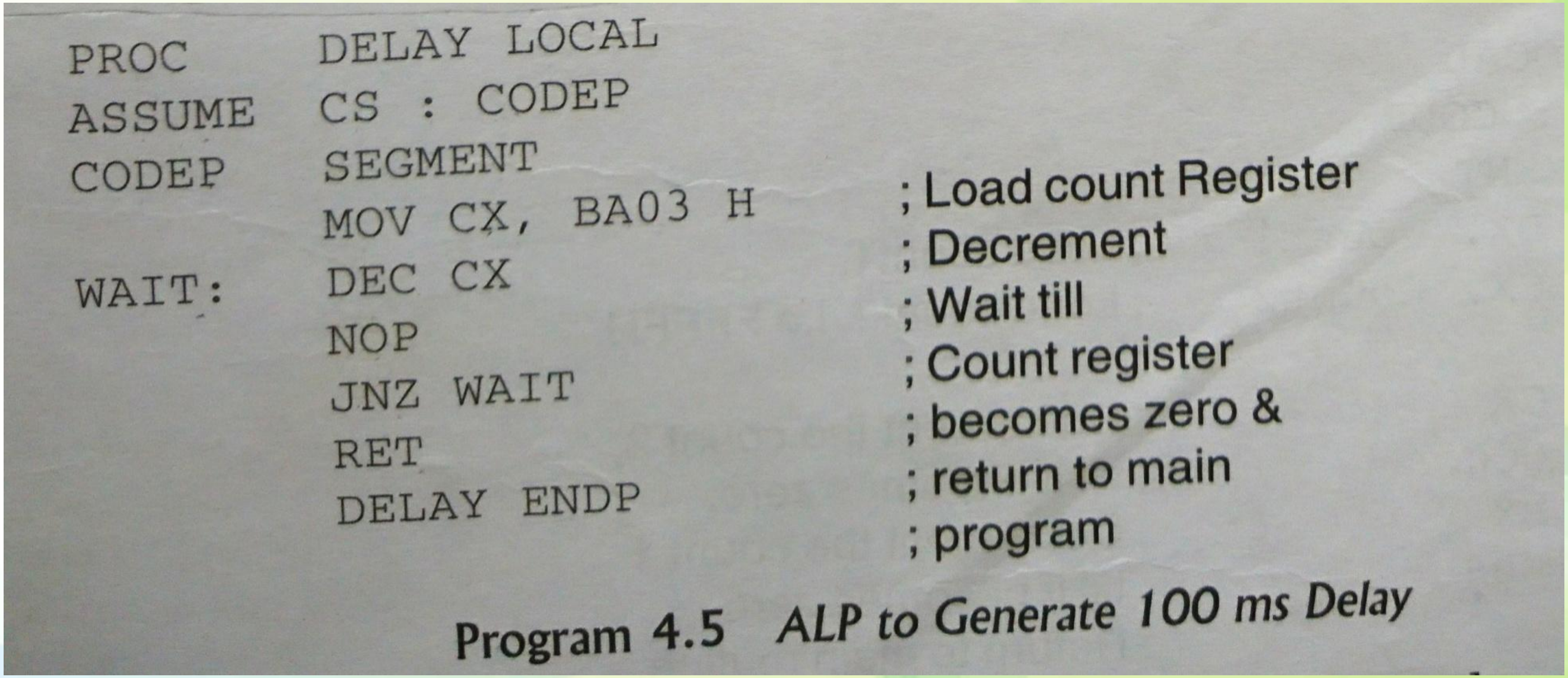
- Number of clk for execution of the loop once=2+3+16=21
- Time required for execution of loop once=21*0.1micro sec=2.1micro sec



$$\text{Count } N = \frac{\text{Required Delay } (T_d)}{n*T}$$

$$\text{Required count} = \frac{T_d}{n*T} = \frac{100*10^{-3}}{2.1*10^{-6}} = 47.619*10^{3}$$

$$= 47619 = \text{BA03H}$$



```
PROC      DELAY LOCAL
ASSUME    CS : CODEP
CODEP     SEGMENT
          MOV CX, BA03 H        ; Load count Register
WAIT:     DEC CX                ; Decrement
          NOP                   ; Wait till
          JNZ WAIT              ; Count register
          RET                   ; becomes zero &
          DELAY ENDP            ; return to main
                                ; program
```

Program 4.5 *ALP to Generate 100 ms Delay*

- Td(exact)=

0.1*4+4(2+3)*47619*0.1+16*47618*0.1+8*0.1

=0.4+23809.5+76188.8+0.4+0.8

=100ms

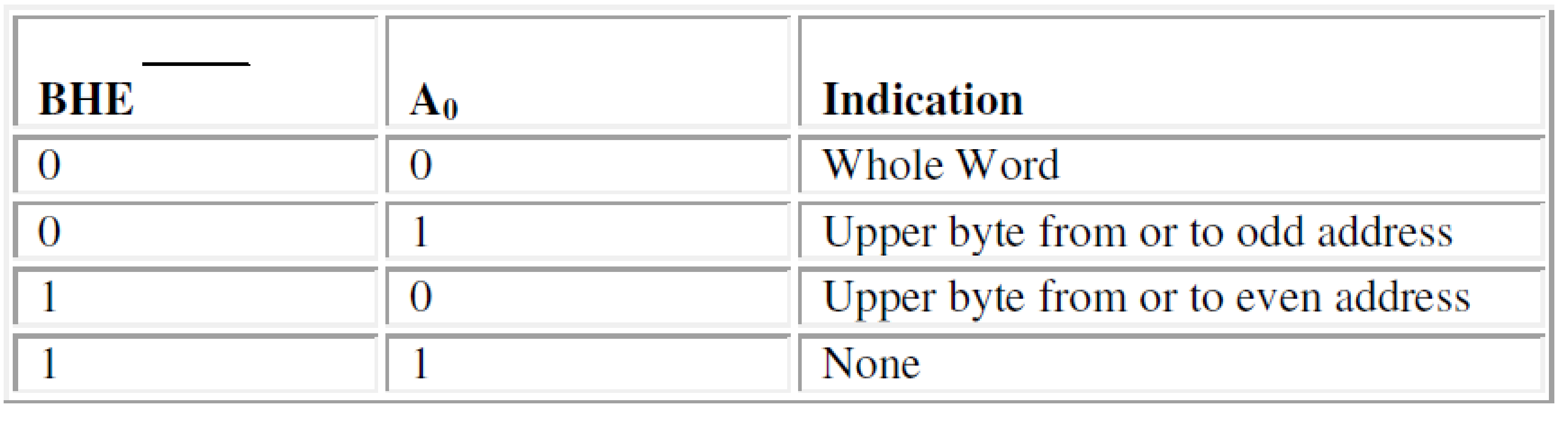
MODULE 4
8086 BUS CONFIGURATION & TIMINGS

- Physical memory Organization
- General Bus operation cycle
- I/O addressing capability
- Special processor activities
- Minimum mode 8086 system and Timing diagrams
- Maximum Mode 8086 system and Timing diagrams.

Physical memory Organization

- 1 MB of m/m organized as ODD & EVEN Bank
- Addressed in parallel way
- Even address is transferred through D0-D7
- Odd address is transferred through D8-D15
- BHE & A0 lines are meant for Even or Odd addr line selection

BHE*/S7-Bus High Enable/Status- Transfer of data over higher order bus(D15-D8)(T1). (0-odd addr m/m (Upper bank), 1-Even addr m/m(Lower bank)).{U-od-0,L-Ev-1}



$\overline{\text{BHE}}$	A_0	Indication
0	0	Whole Word
0	1	Upper byte from or to odd address
1	0	Upper byte from or to even address
1	1	None

- When Data is fetched 3 possibilities

1.both bytes may be data operands

2. both bytes may contain opcode bits.

3. one byte may be opcode other may be operand

- Opcodes & operands are identified by internal decoder circuit.
- Timing & control unit derives all the signals which are required for the execution of instruction.
- It is always preferred to store the data from even m/m location(Even m/m-1cycle,Odd m/m-(2 cycle(to lower & upper byte)))

- Locations FFFF0h TO FFFFFh reserved for JUMP & I/O Processor Initializations.
- Locations 00000h TO 003FFh (1KB)reserved for IVT.

General Bus operation in Maximum mode

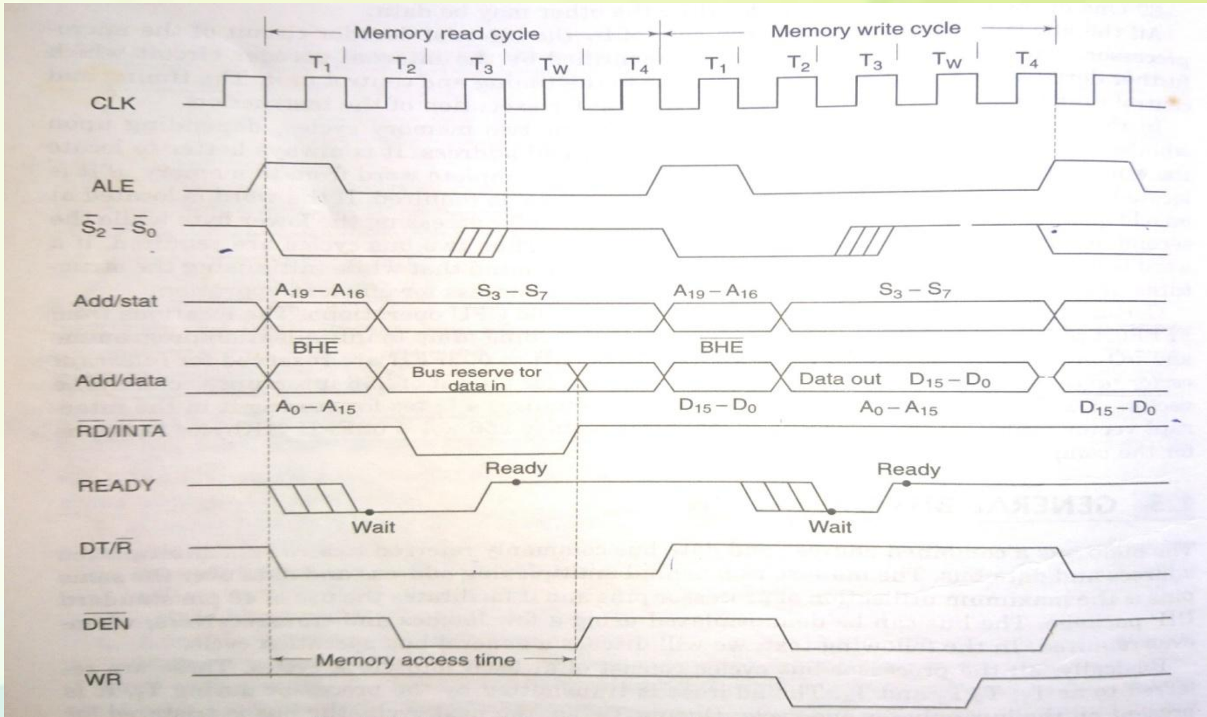


Fig. 1.7 *General Bus Operation Cycle in Maximum Mode*

- There are 4 clock cycles,T1,T2,T3,&T4
- Address is transferred during T1
- During T2 it is in Tristate mode
- Data transfer takes place during T3 & T4
- Tw state is inserted to cope up with slower device.
- ALE is activated by processor or the bus controller based upon weather it is Min Or Max mode.
- In Maximum mode s0,s1,s2 are used to indicate type of operation.
- S3 to s7(T2 state) multiplexed with higher order address bus (BHE)(T1 state)

I/O Addressing Capability

- I/O Address appears on the line A0-A15 for one clock cycle T1

- 16 bit register DX is used as I/O address pointer

Special Processor Activities

1.Processor Reset & Initialization:-

- When 1 is applied it will be there in Reset mode until 0 is applied(Positive Edge of Signal)
- It will be there for minimum 10 cycles.
- During this Period all the internal reg content are set to 0000H except CS(F000h)&IP(FFF0h).
- To activate Reset signal must be there for at least 4 clk cycles.
- RESET cannot applied just after power on

2.HALT:-

- It enters into Halt state
- To enter halt state in Min mode It activates ALE signal.
- To enter halt state in Max mode It indicates on S2,S1,S0 lines and then ALE will be asserted
- Only INTR & RESET will make processor to come out of HLT state.

3.TEST & Synchronization with external Signals

- TEST will make processor to go into WAIT state.
- To activate TEST clk pulse must be activate for at least 5 cycles.

Minimum mode 8086 system and Timings

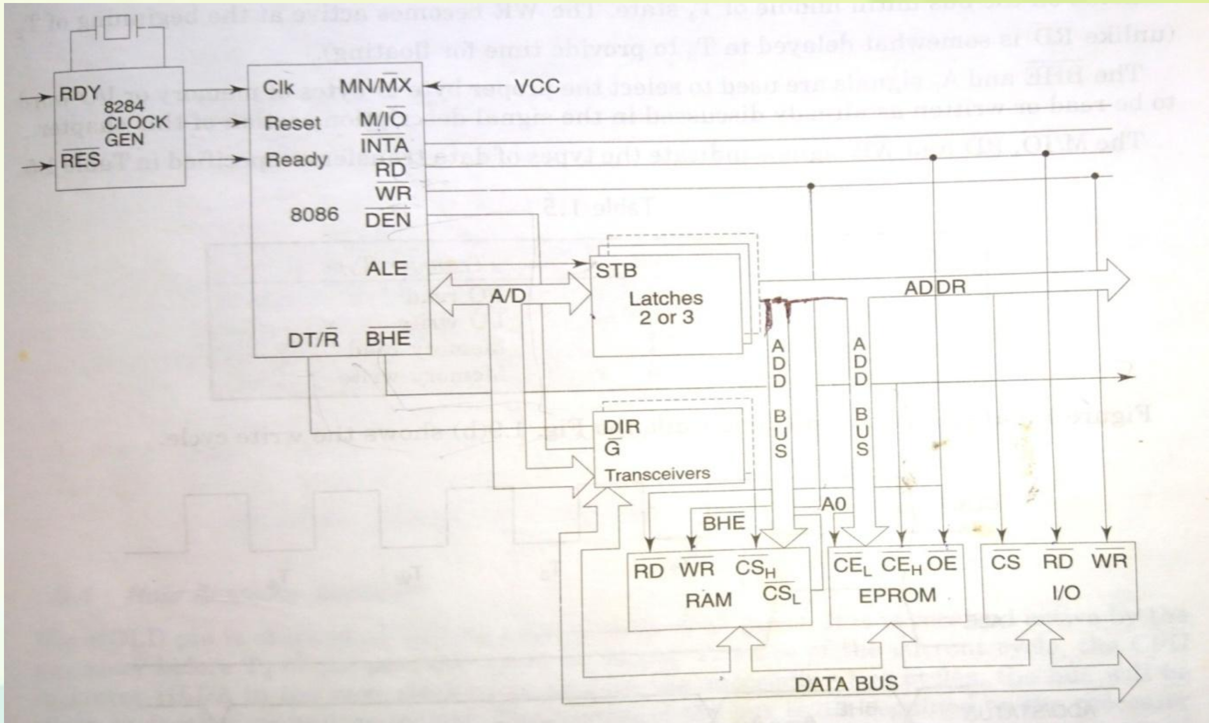


Fig. 1.8 *Minimum Mode 8086 System*

Types of Data Transfer

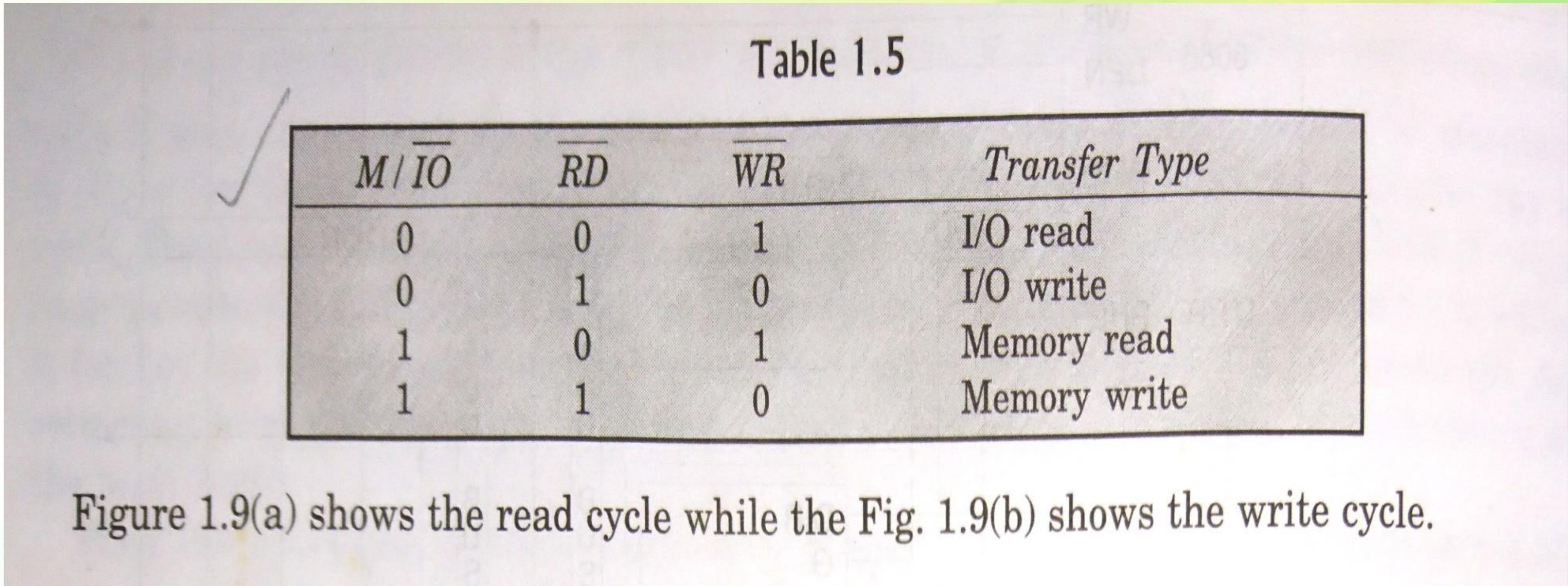


Table 1.5

$M/\overline{IO}$	$\overline{RD}$	$\overline{WR}$	*Transfer Type*
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

Figure 1.9(a) shows the read cycle while the Fig. 1.9(b) shows the write cycle.

Read Cycle timing diagram for Min mode

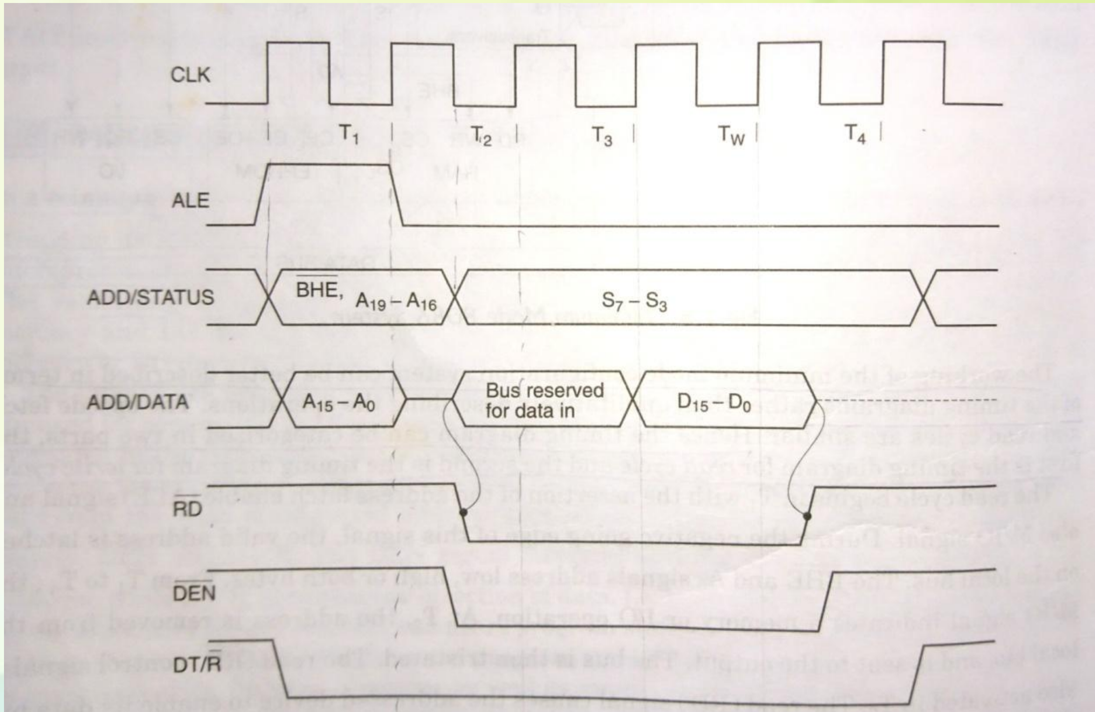


Fig. 1.9(a) *Read Cycle Timing Diagram for Minimum Mode*

Write cycle timing diagram for Min mode

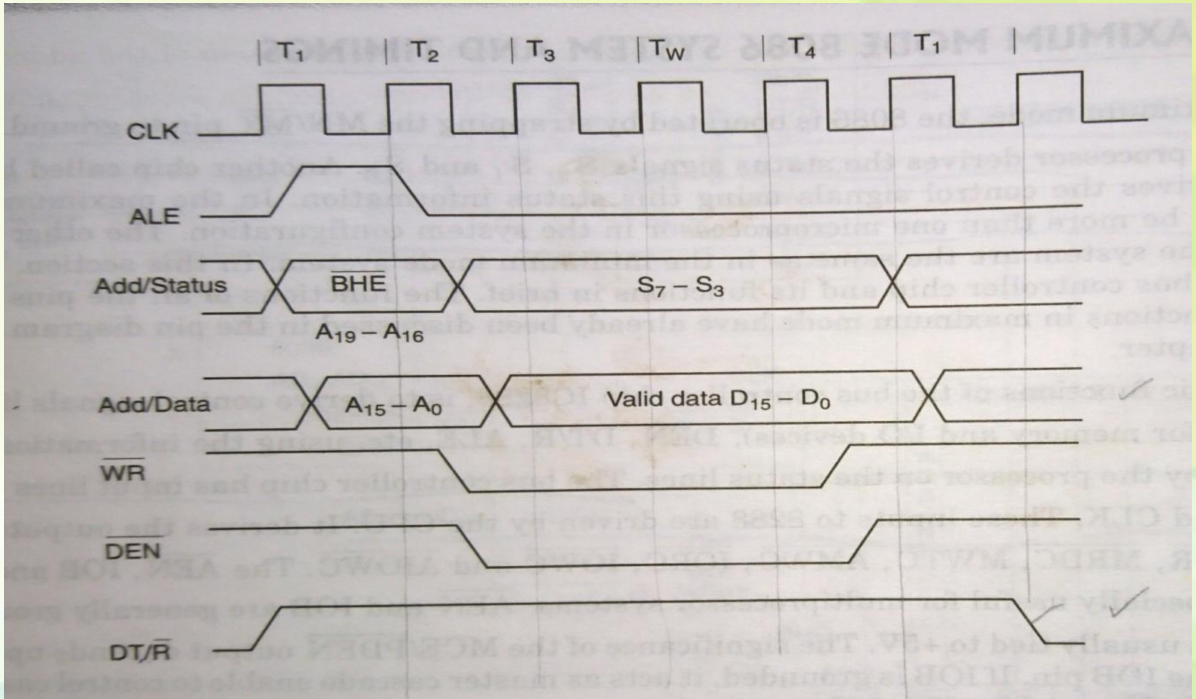
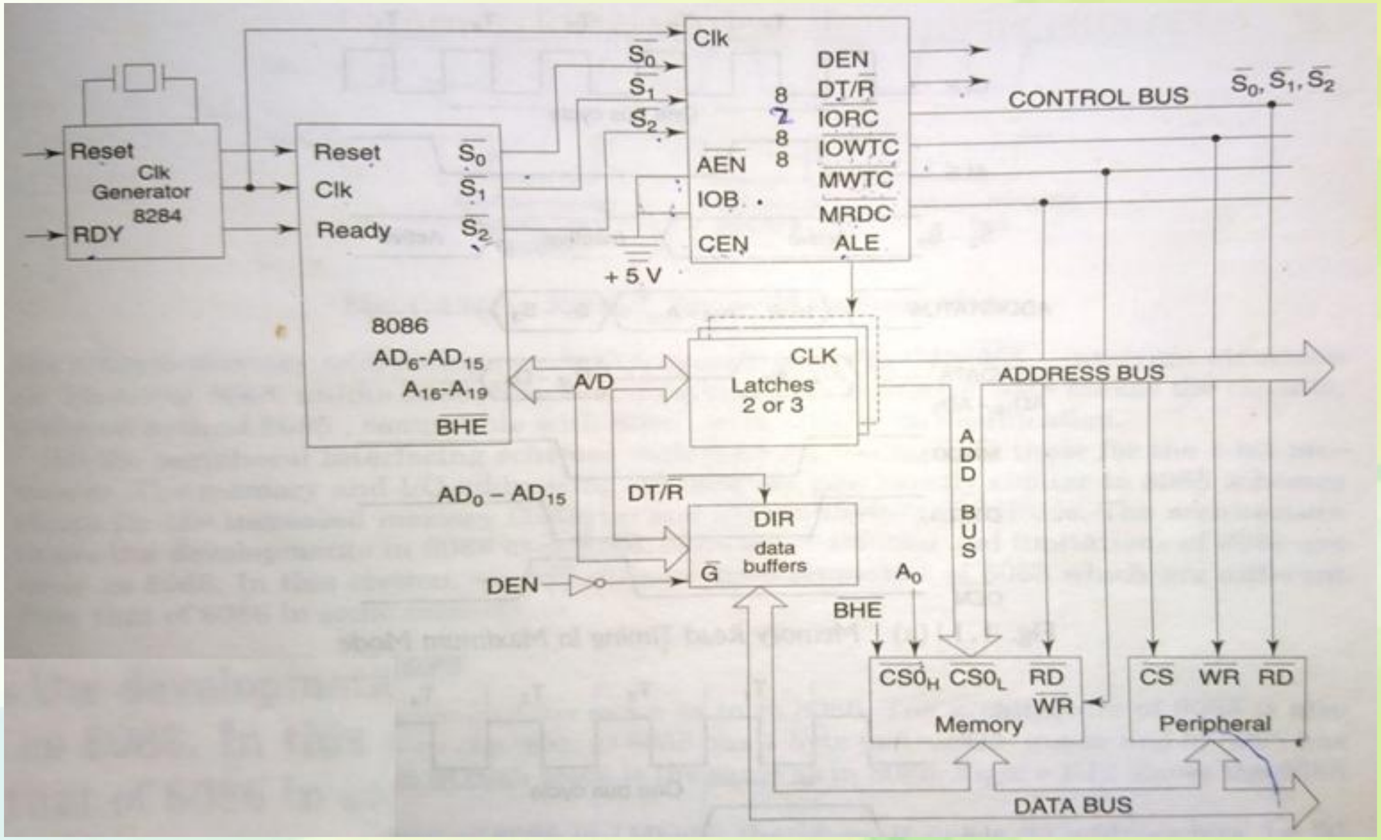


Fig. 1.9(b) *Write Cycle Timing Diagram for Minimum Operation*

Maximum mode 8086 system



Memory Read In Max mode

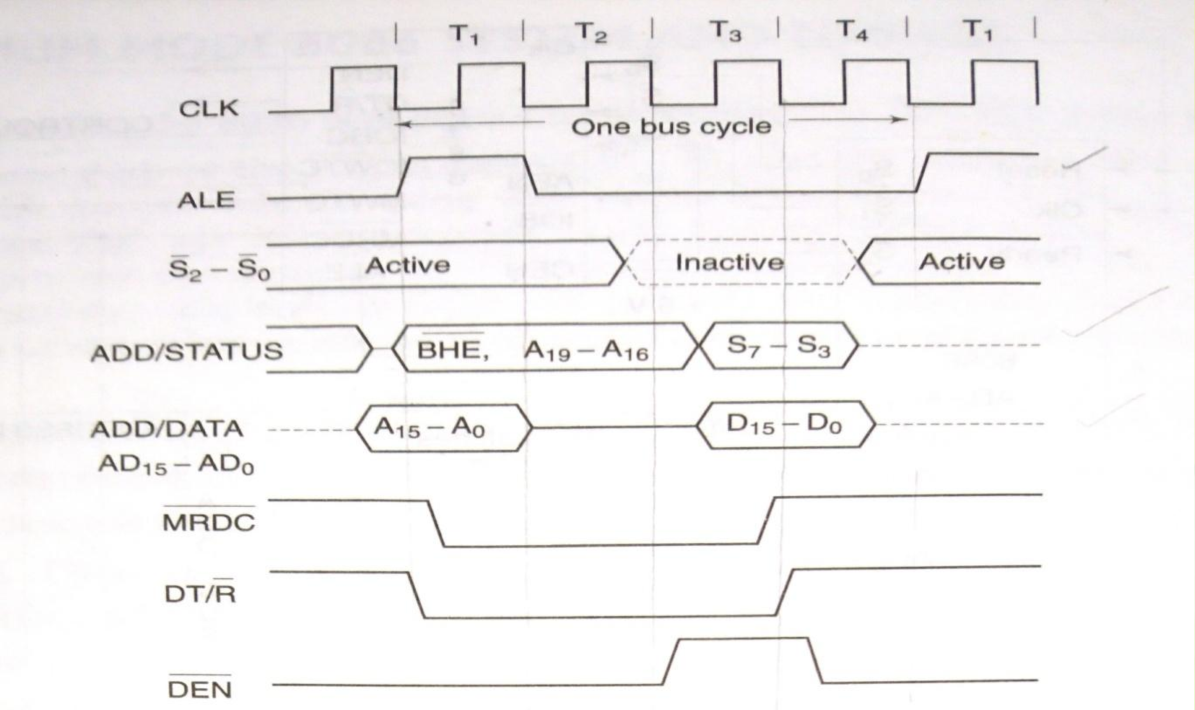


Fig. 1.11(a) *Memory Read Timing in Maximum Mode*

Memory Write In Max mode

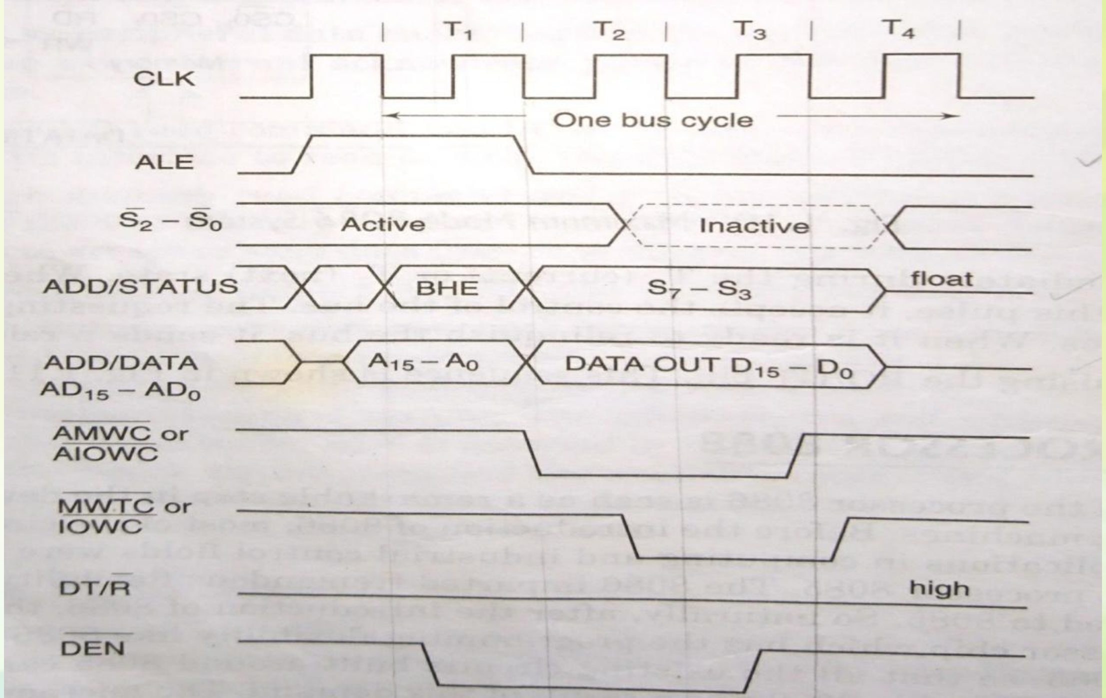


Fig 1.11(b) *Memory Write Timing in Maximum Mode*

Static RAM Interfacing

- 2 types of RAM

a. Static RAM b.Dynamic RAM

- Memories are arranged in two dimensional arrays(4K X 8-4096 bytes with each 8bit)
- 4K bytes location requires 12 address lines.

Procedure for static memory interfacing with 8086

1.Arrange m/m so as to obtain 16bit data bus width,Upper 8bit bank called ‘odd addr m/m bank’ and lower is called as ‘Even addr m/m bank’.

2.Connect available m/m addr lines of m/m RD’ & WR’ inputs to the corresponding processor control signals. Connect the 16 bit data bus of the m/m bank with that of the MP.

3.Remaining addr lines of the MP ,BHE & A0 are used for decoding the required chip select s/g for the odd & even m/m banks. The CS’ is arrived from decoding ckt.

Interface two 4k X 8 EPROMS & two 4K X 8 RAM chips with 8086 select suitable maps

- After reset IP & CS are initialized to form address FFFFOh hence this addr must lie in the EPROM
- Addr of RAM may be selected any where in the 1MB addr space of 8086

- For 8K bytes of EPROM need 13 addr lines

A0-A12

- A13-A19 are used for decoding to generate CS’ signal
- BHE’ will be low when transfer is from odd(higher) addr byte.
- If addr ,BHE’, & data lines availbe for interfacing

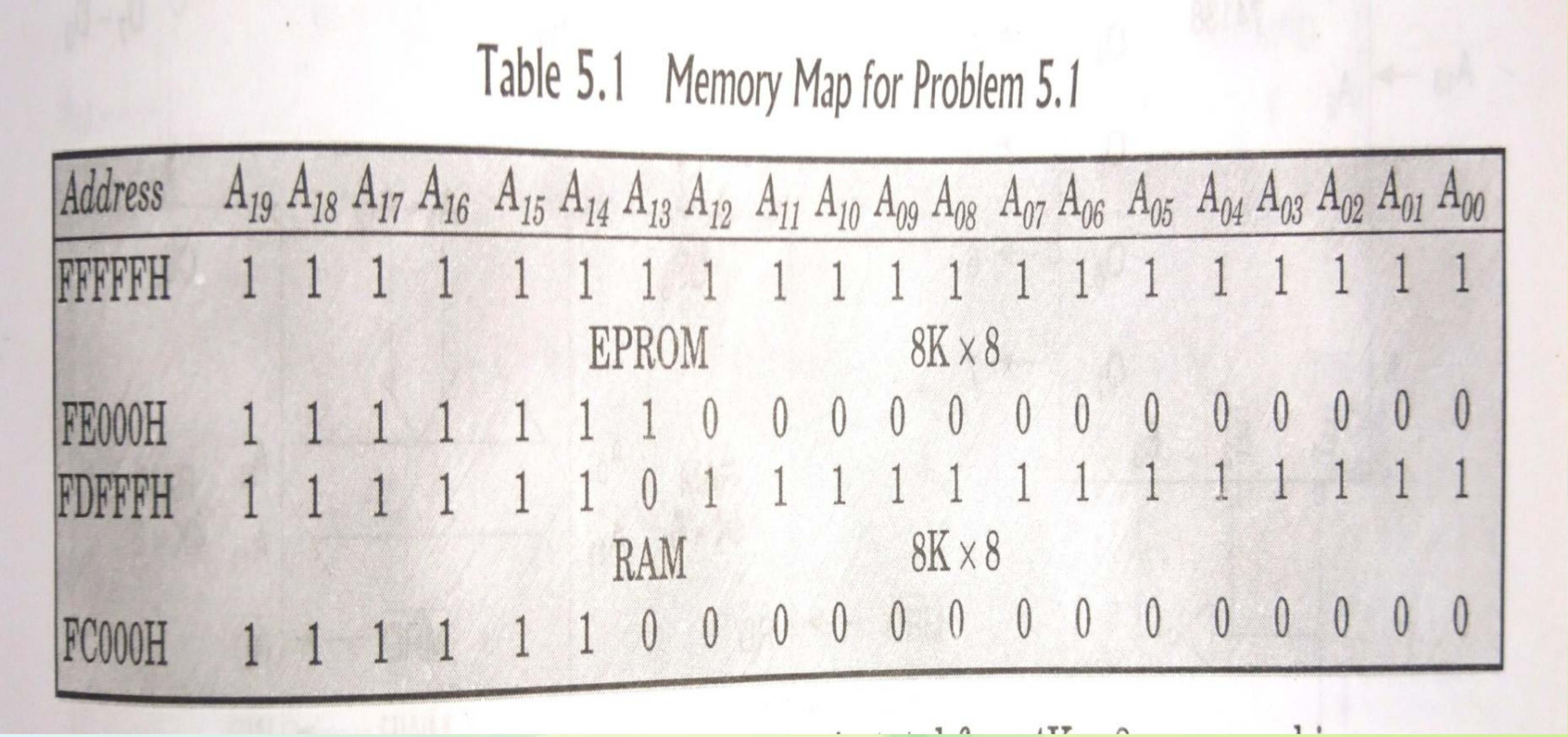


Table 5.1 Memory Map for Problem 5.1

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
							EPROM					8K × 8								
FE000H	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
FDFFFH	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
							RAM					8K × 8								
FC000H	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

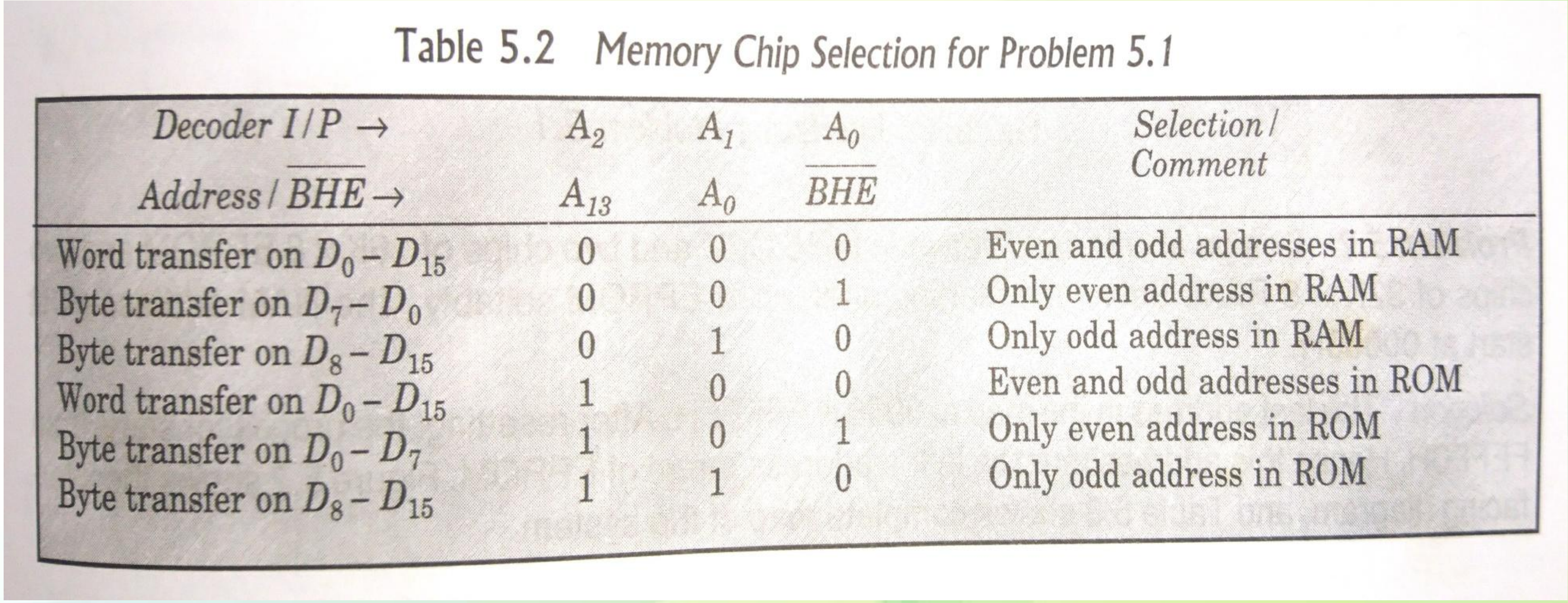


Table 5.2 Memory Chip Selection for Problem 5.1

Decoder I/P →	A_2	A_1	A_0	*Selection / Comment*
Address / $\overline{BHE}$ →	A_{13}	A_0	$\overline{BHE}$	
Word transfer on $D_0 - D_{15}$	0	0	0	Even and odd addresses in RAM
Byte transfer on $D_7 - D_0$	0	0	1	Only even address in RAM
Byte transfer on $D_8 - D_{15}$	0	1	0	Only odd address in RAM
Word transfer on $D_0 - D_{15}$	1	0	0	Even and odd addresses in ROM
Byte transfer on $D_0 - D_7$	1	0	1	Only even address in ROM
Byte transfer on $D_8 - D_{15}$	1	1	0	Only odd address in ROM

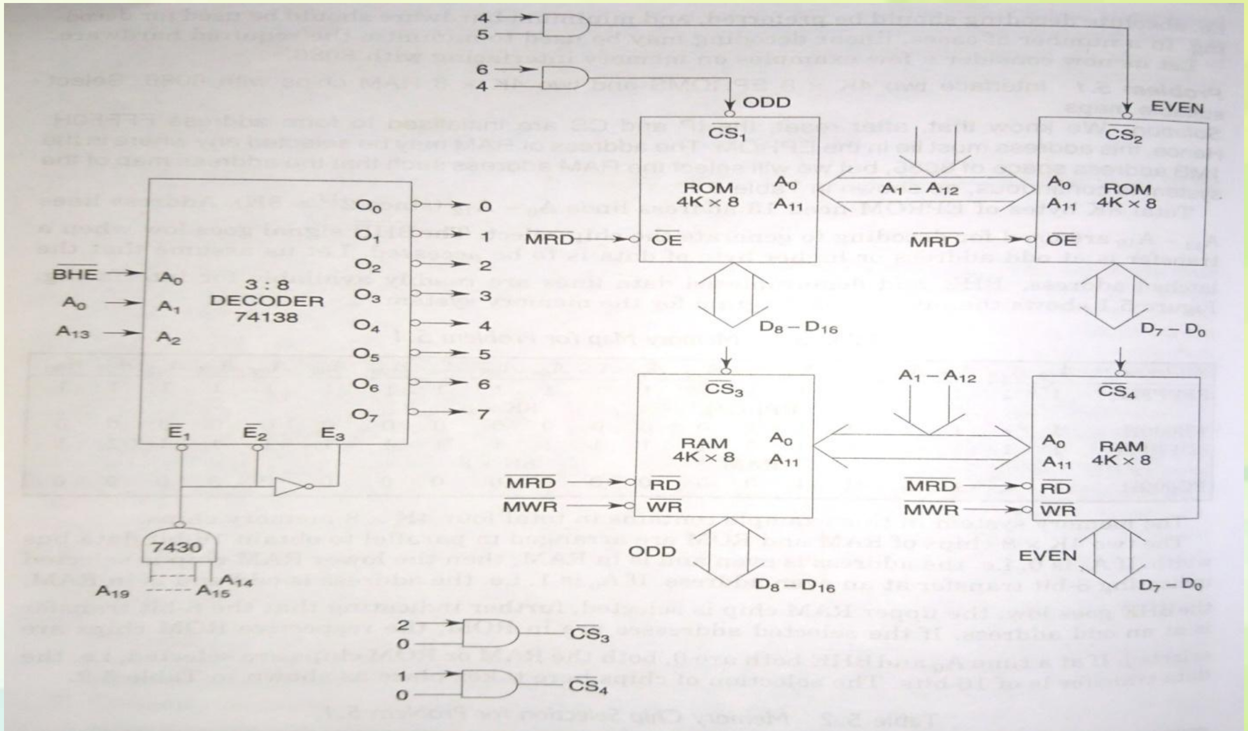


Fig. 5.1 *Interfacing Problem 5.1*

- Design an interface between 8086 CPU and two chips of 16K X 8 EPROMS and two chips of 32K X 8 RAM. Select the starting address of EPROM suitably.The RAM address must start at 00000H.

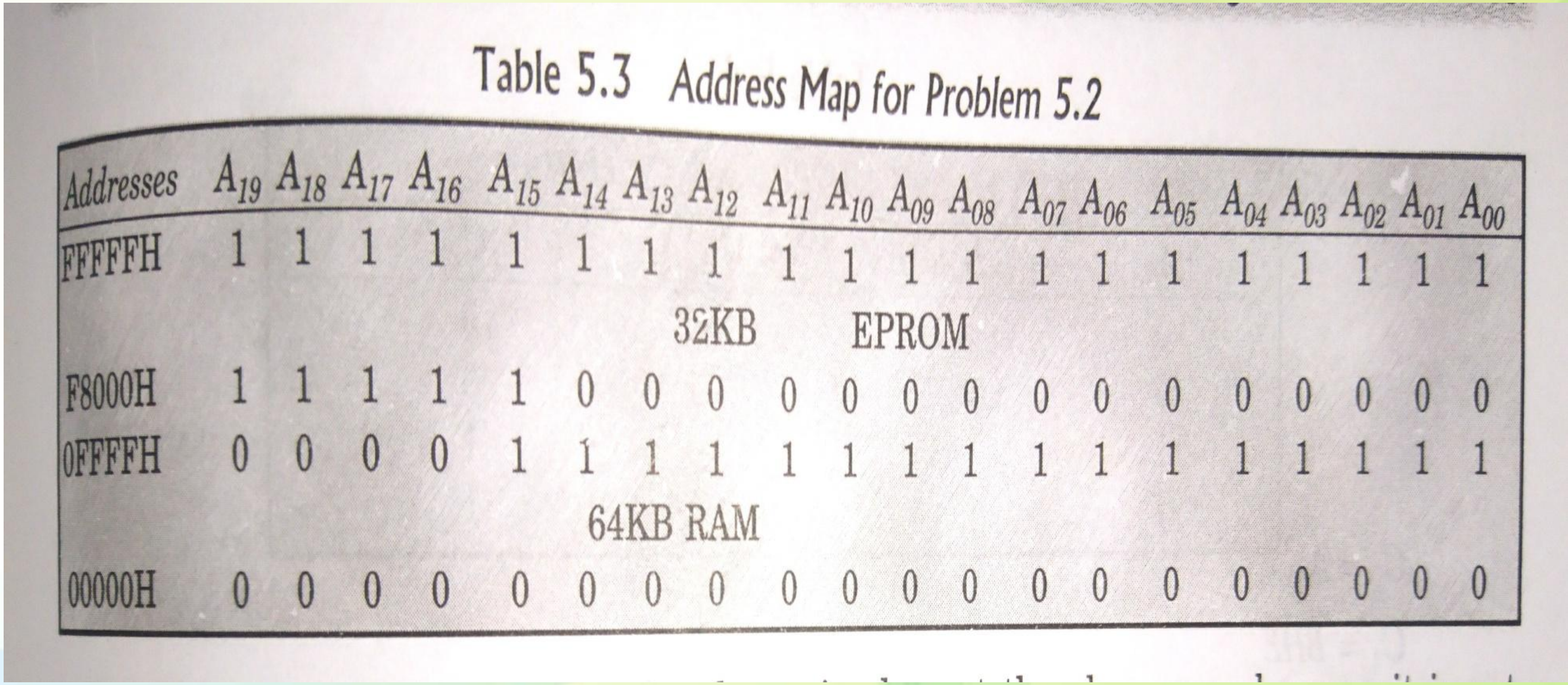


Table 5.3 Address Map for Problem 5.2

Addresses	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
								32KB			EPROM									
F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
							64KB RAM													
00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

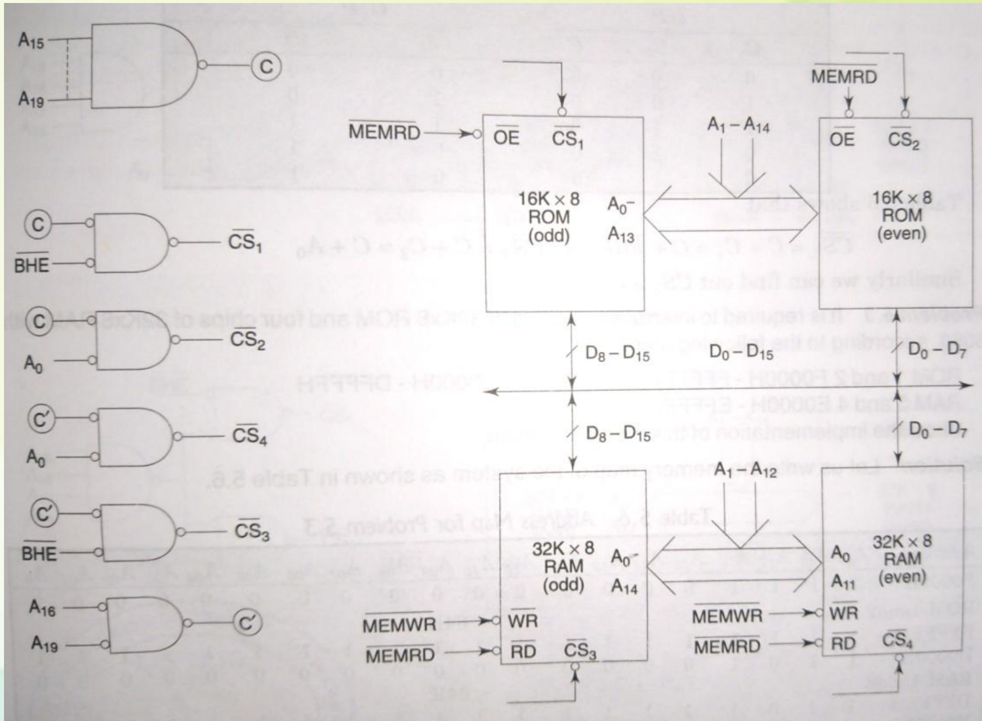


Fig. 5.2 *Interfacing Problem 5.2*

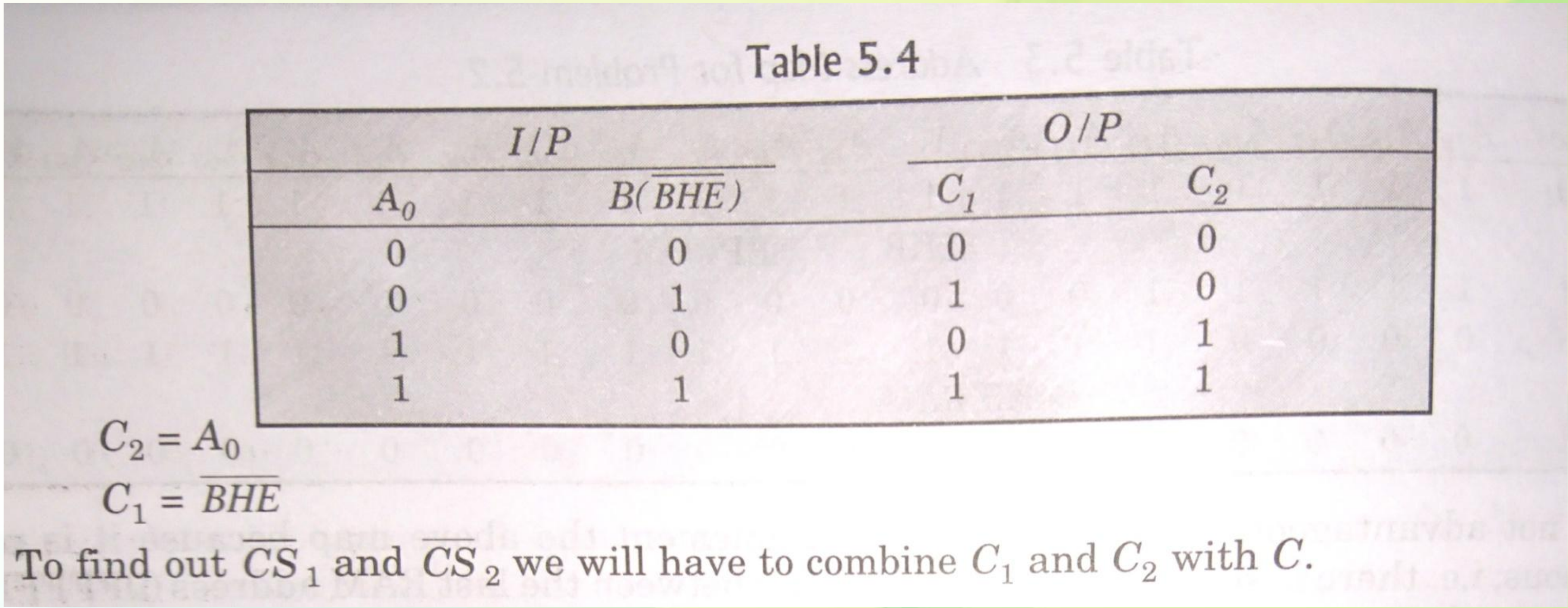


Table 5.4

I/P		O/P	
A_0	$B(\overline{BHE})$	C_1	C_2
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

$C_2 = A_0$

$C_1 = \overline{BHE}$

To find out $\overline{CS}_1$ and $\overline{CS}_2$ we will have to combine C_1 and C_2 with C.

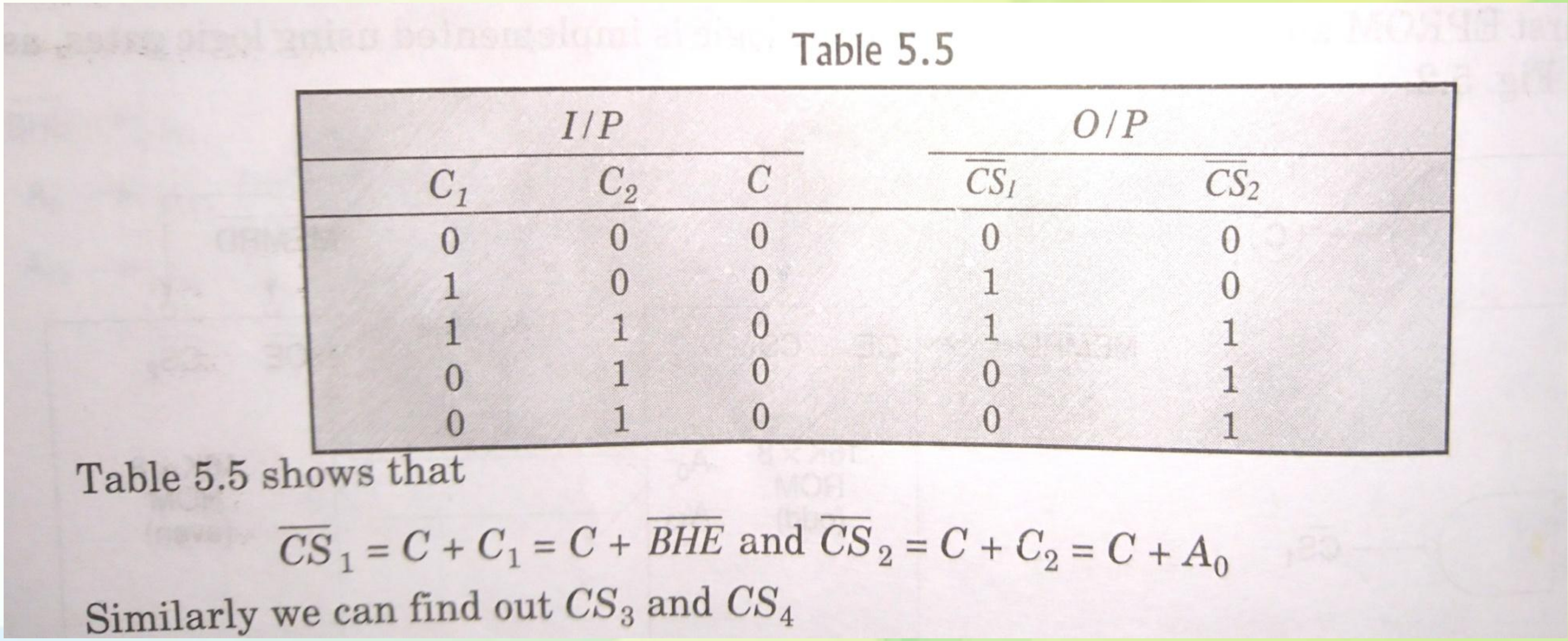


Table 5.5

I/P			O/P	
C_1	C_2	C	$\overline{CS_1}$	$\overline{CS_2}$
0	0	0	0	0
1	0	0	1	0
1	1	0	1	1
0	1	0	0	1
0	1	0	0	1

Table 5.5 shows that

$$\overline{CS}_1 = C + C_1 = C + \overline{BHE} \text{ and } \overline{CS}_2 = C + C_2 = C + A_0$$

Similarly we can find out CS_3 and CS_4

- It is required to interface two chips of 32K X 8 ROM and four chips of 32K X 8 RAM with 8086 according to the following map

ROM 1 and 2 F0000H-FFFFFH,

RAM 1 and 2 D0000H-DFFFFH,

RAM 3 and 4 E0000H-EFFFFH

Show the implementation of this m/m system.

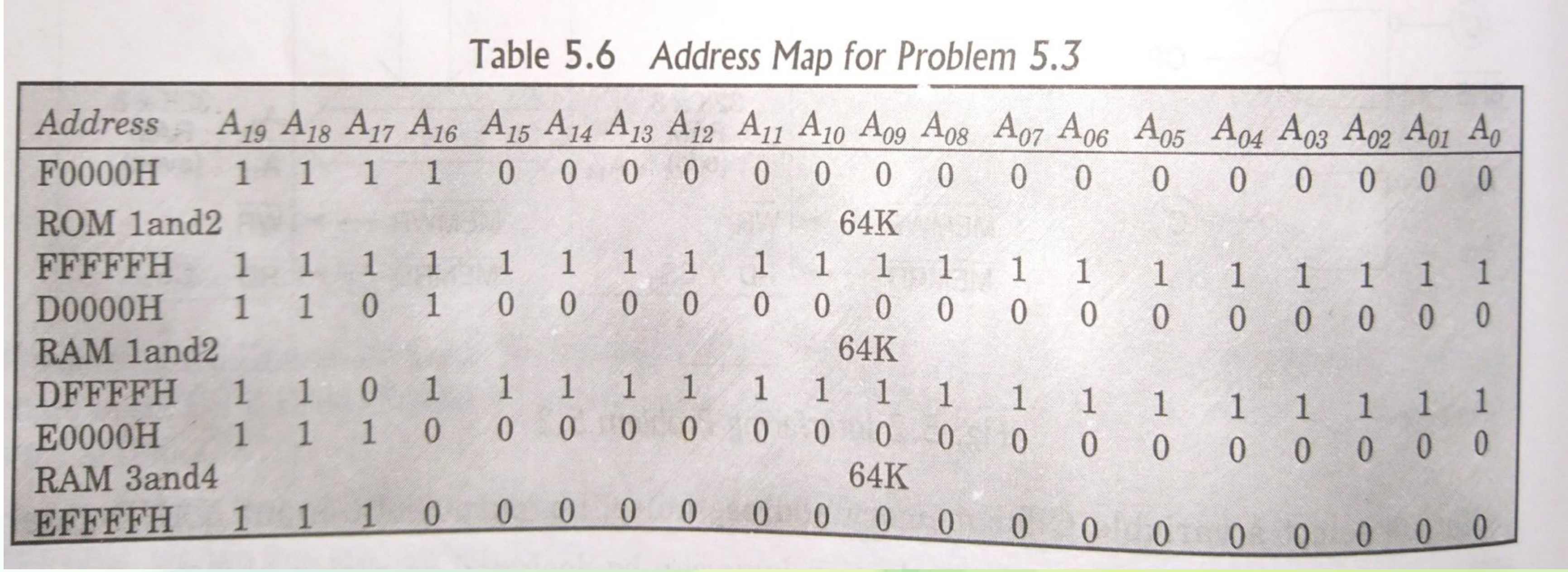


Table 5.6 *Address Map for Problem 5.3*

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{0}
F0000H	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ROM 1and2											64K									
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
D0000H	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RAM 1and2											64K									
DFFFFH	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
E0000H	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RAM 3and4											64K									
EFFFFH	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

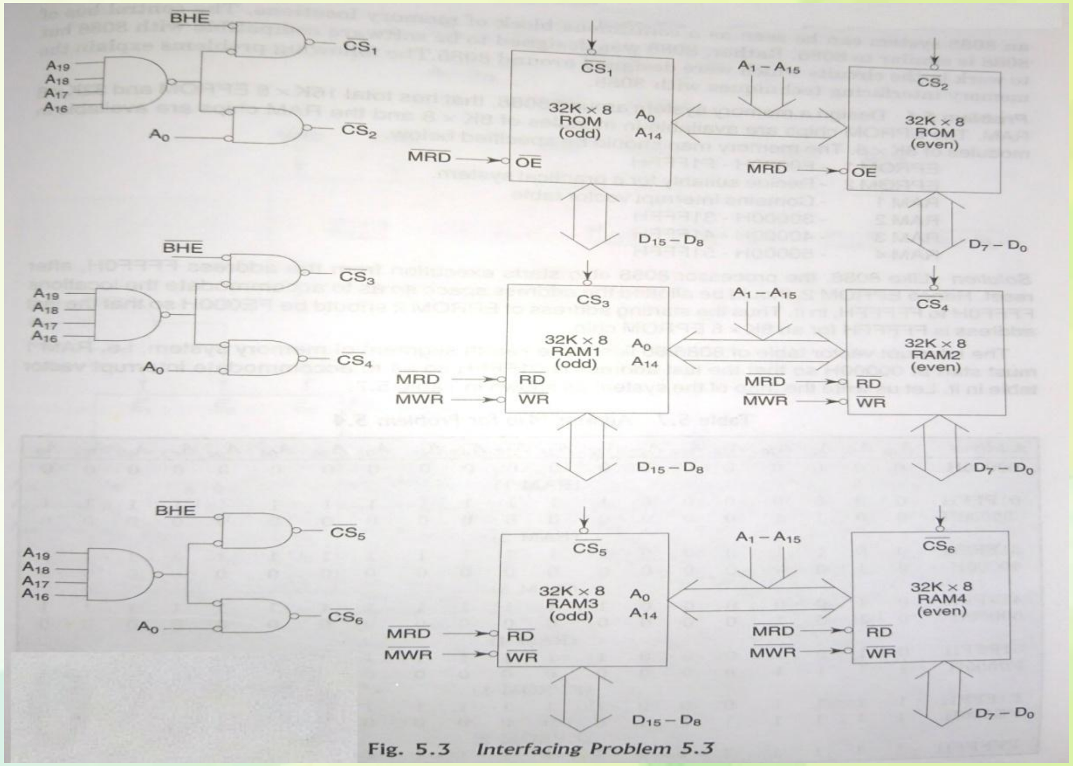


Fig. 5.3 *Interfacing Problem 5.3*

Interfacing IO Ports

- Microprocessor is interfaced to External I/O devices

Steps in Interfacing I/O Devices

1. Connect data bus of the mp s/m with data bus of the I/O port
2. Device address and decoding address to generate the cs is derived
3. Suitable control signals IORD /IOWR are used.

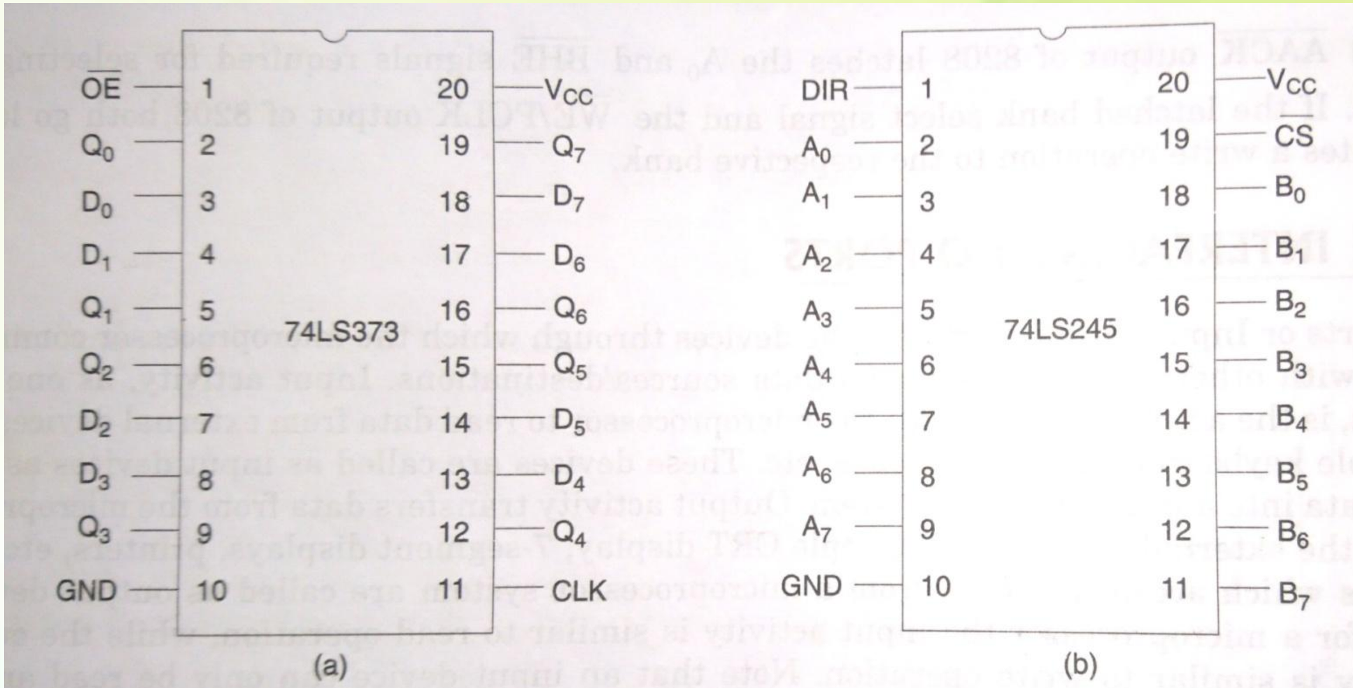


Fig. 5.11 *(a) Latch (O/P port) (b) Buffer (I/P port)*

Methods of Interfacing I/O Devices

- Two Methods

1. I/O Mapped –

- The devices are treated as distinct I/O devices only & addressed accordingly.
- All the available addressed may not be used.

(A0-A15 16 address lines/A0-A7 8 address lines)

- Uses IN & OUT instructions.
- Requires less Hardware for decoding
- IORD & IOWR signals are used.

2.Memory Mapped-

- The devices are treated as distinct Memory locations & addressed accordingly.
- MRDC & MRTC control signals are used.
- MOV,LEA kind of instrn are used.
- m/m operations are faster.
- They require complex H/W.

- **Problem**
- Interface an i/p port 74LS245 to read the status of switches SW1 to SW8. the switches when shorted i/p a ‘1’ else i/p a’0’ to the mp. Store the status in BL register. The address of the port is 0740H

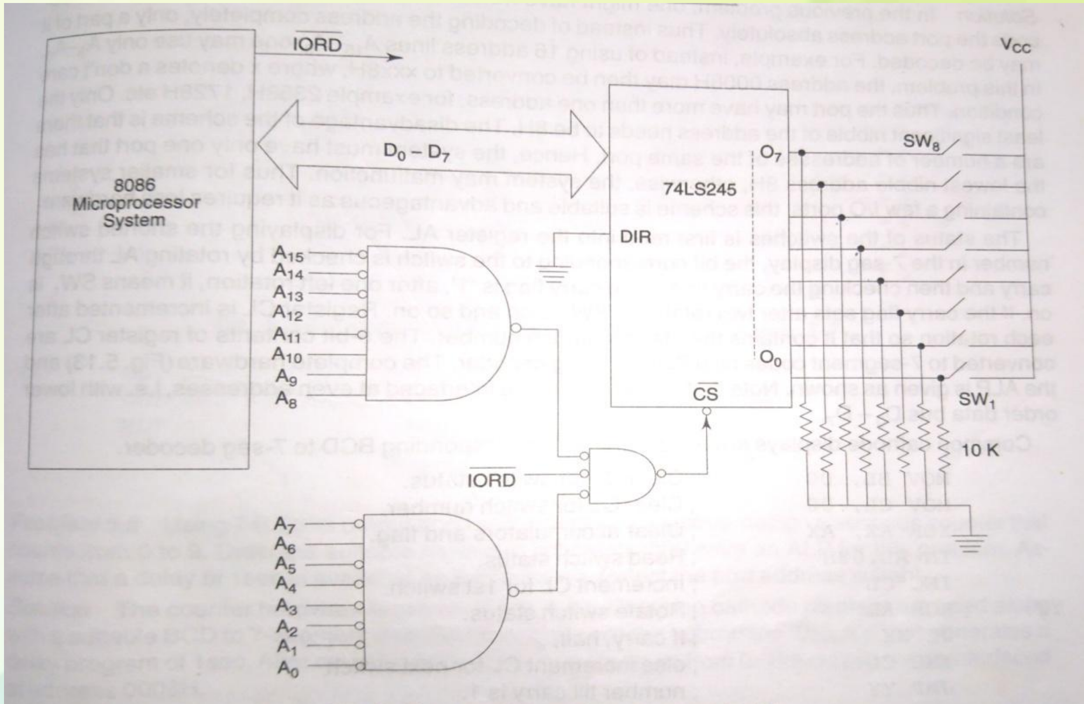


Fig. 5.12 *Interfacing Input Port 74LS245*

ALP is as follows

```
MOV BL,00h
MOV DX,0740h
IN AL,DX
MOV BL,AL
HLT
```

Module 5

Basic peripherals and interfacing with 8086

- Timer 8254 mode 0,1,2,3 and interfacing programmer for these modes
- INT 21h DOS function calls for handling keyboard & display
- Other architecture of 0808 & NDP 8087
- Von-neuman & harvard CPU architecture , CISC & RISC CPU architecture.

Interfacing ADC -0808,DAC-0800 using stepper motor

- ADC chips are 8 bit successive approximation converters, Successive approximation technique is the one of the fastest method to convert from ana;og to digital
- These converters internally have 3:8 analog multiplexers so that at a time 8 analog inputs can be connected to the chip.

Block diagram of ADC 0808/0809

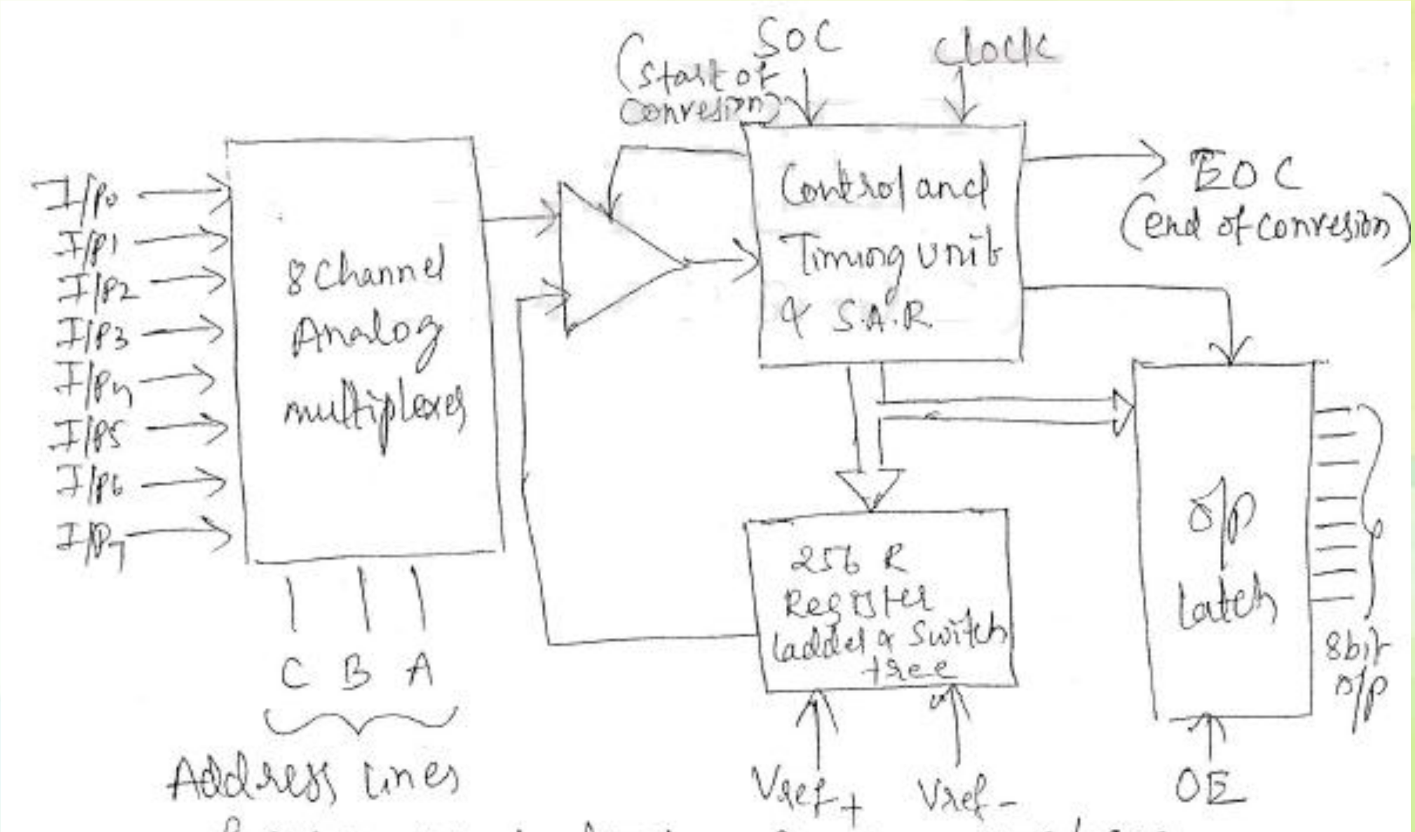
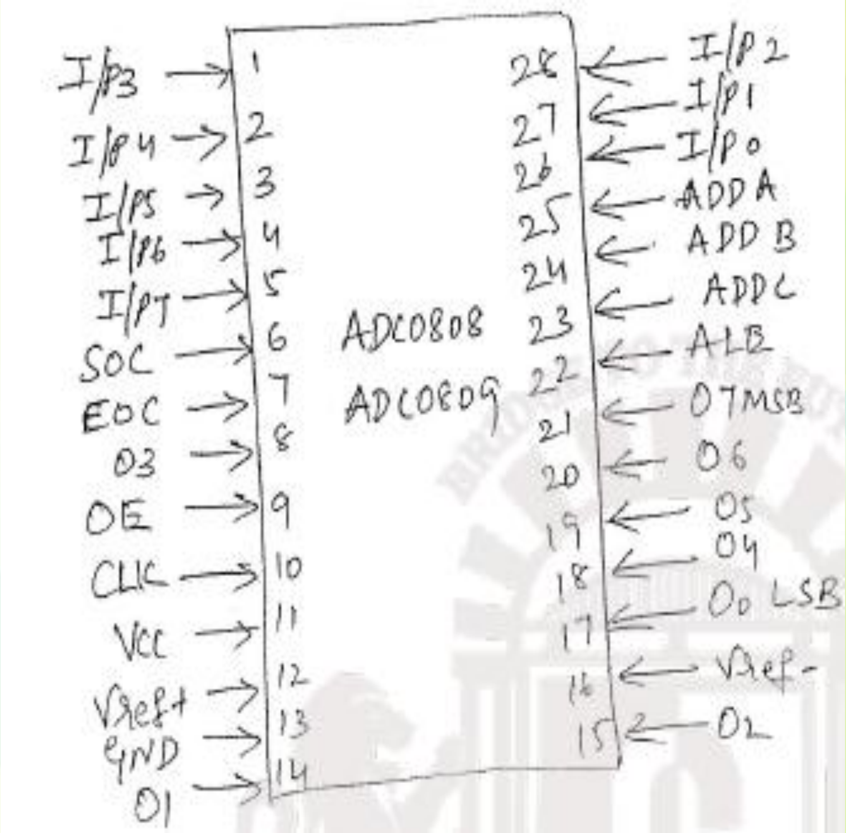


figure: Block diagram for ADC0808/0809.

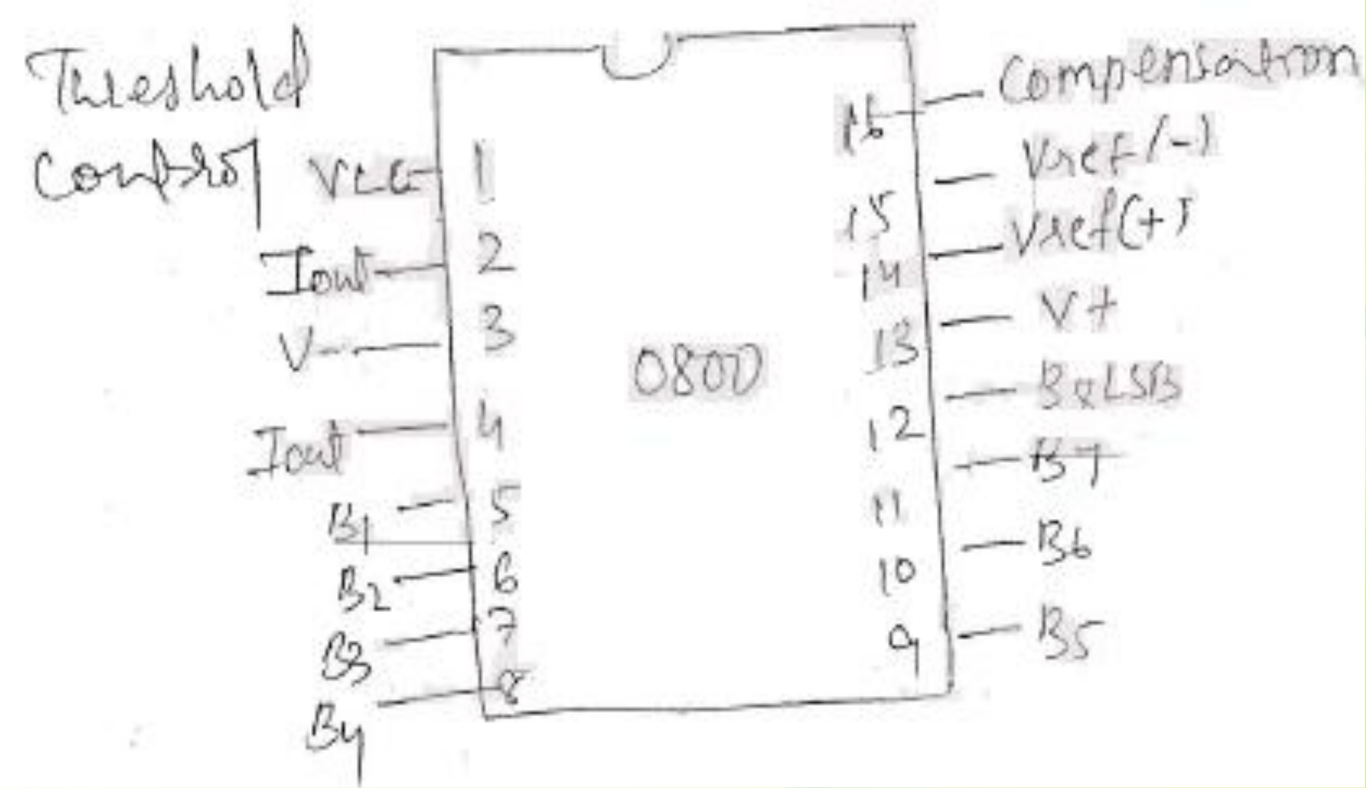
Pin Diagram of ADC 0808/0809



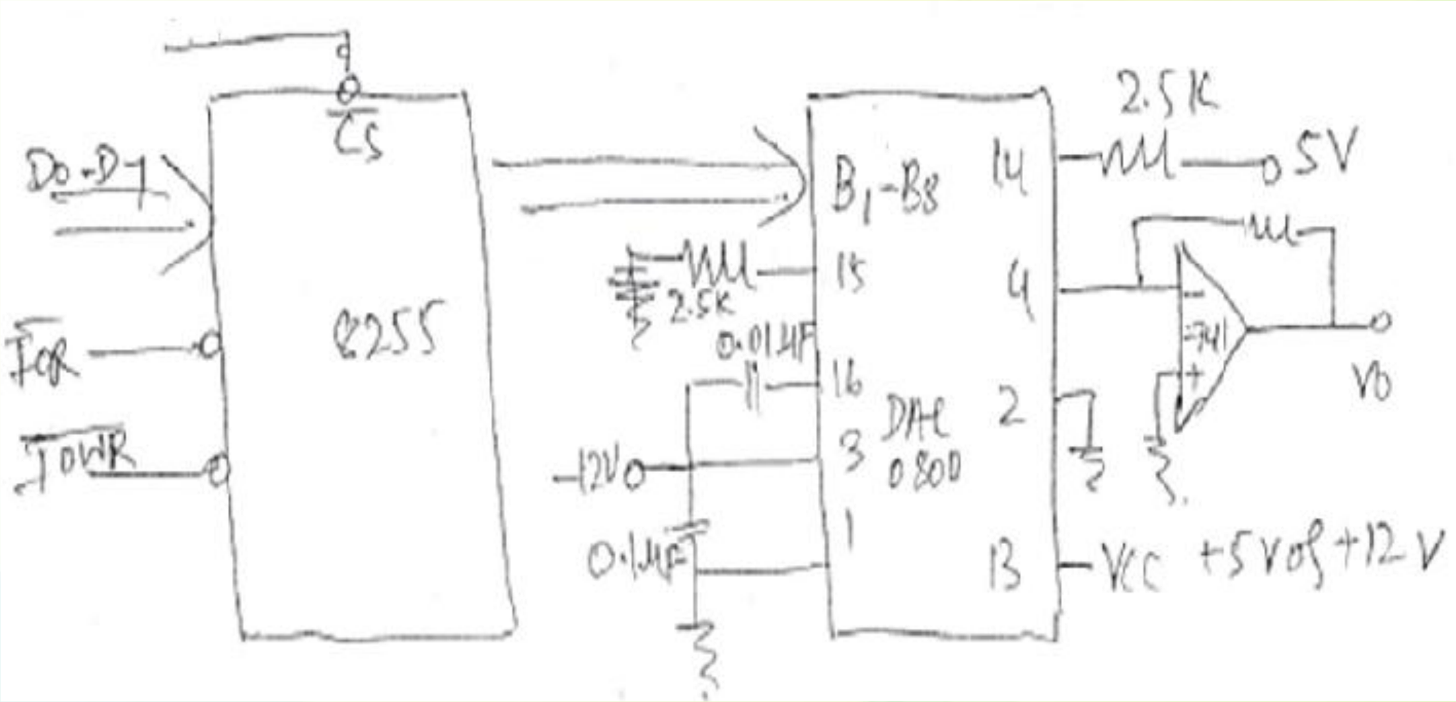
DAC 0800 8bit digital to analog converter

- It has a setting time around100ms and can operate on a range of power supply voltages that is from 4.5v to 18v usually the v+ is 5V or 12v. The V pin can be kept at a minimum of -12v

Pin Diagram of DAC 0800



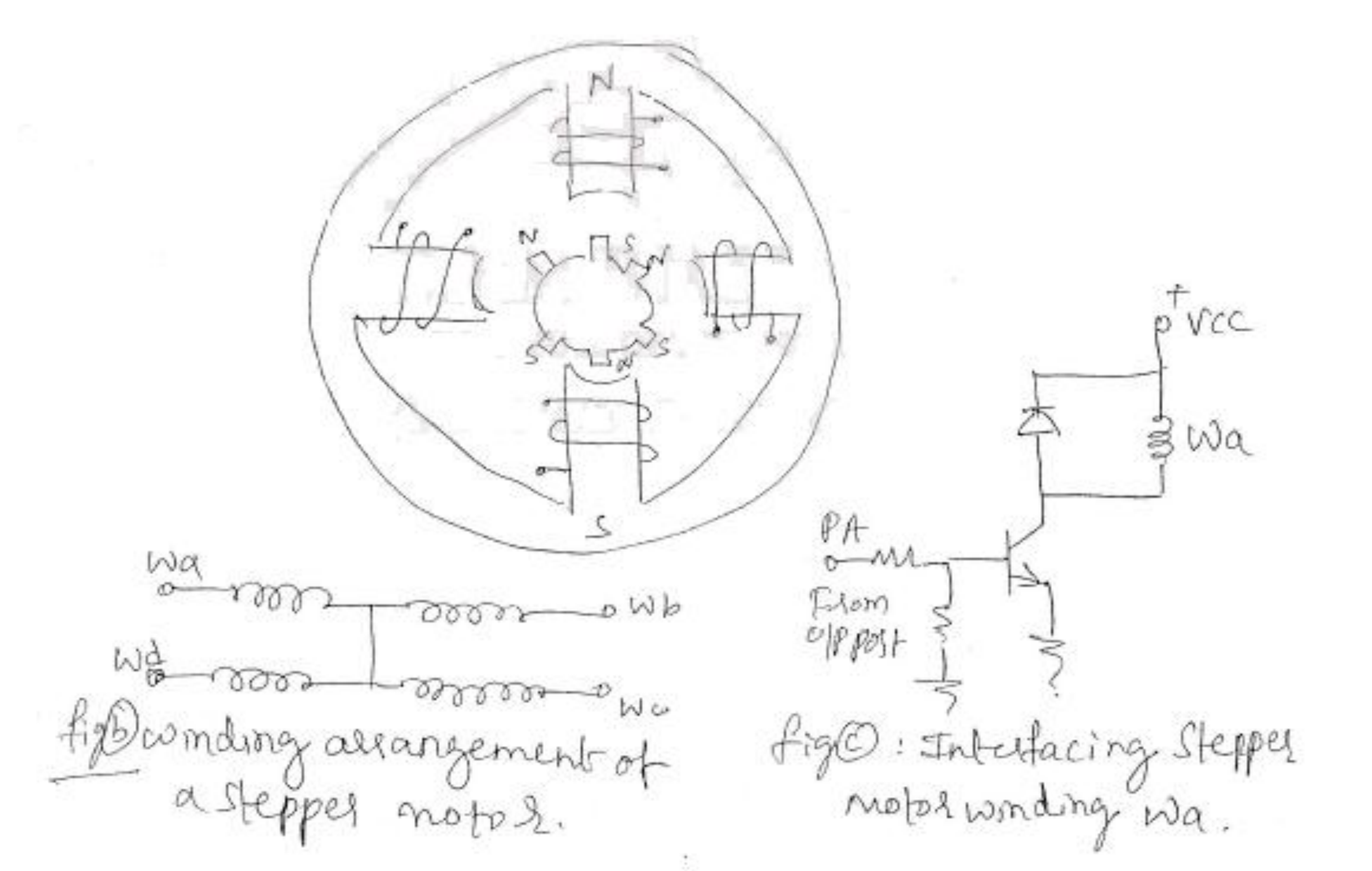
Interfacing of DAC 0800 with 8086



Stepper motor Interfacing

- Stepper motor is a device used to obtain an accurate position control of the rotating shafts. It employs rotation of its in terms of steps rather than continues rotation in case of AC or DC motors

Internals schematic of a four windings stepper motor



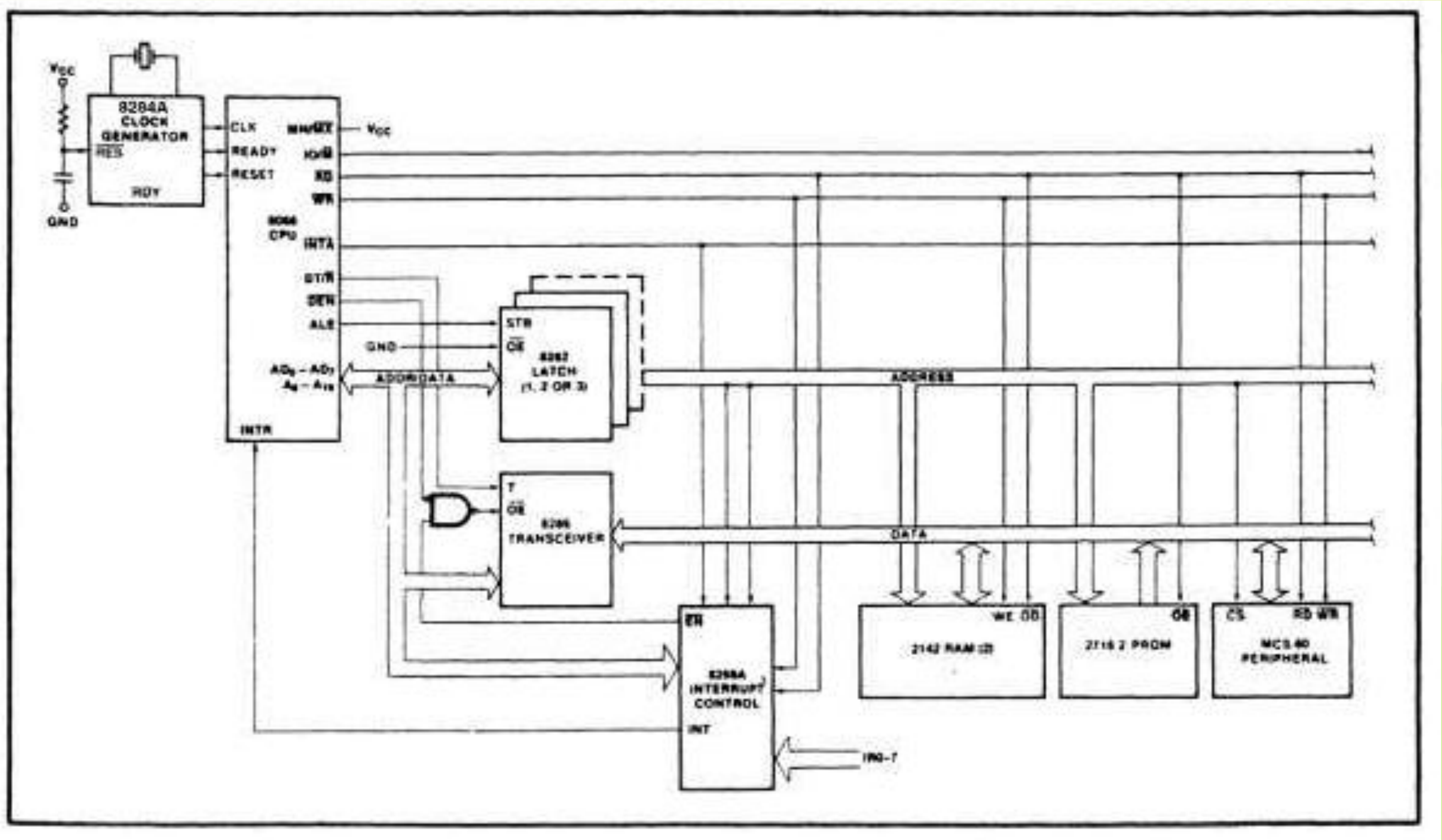
fig(b) winding arrangement of a stepper motor.

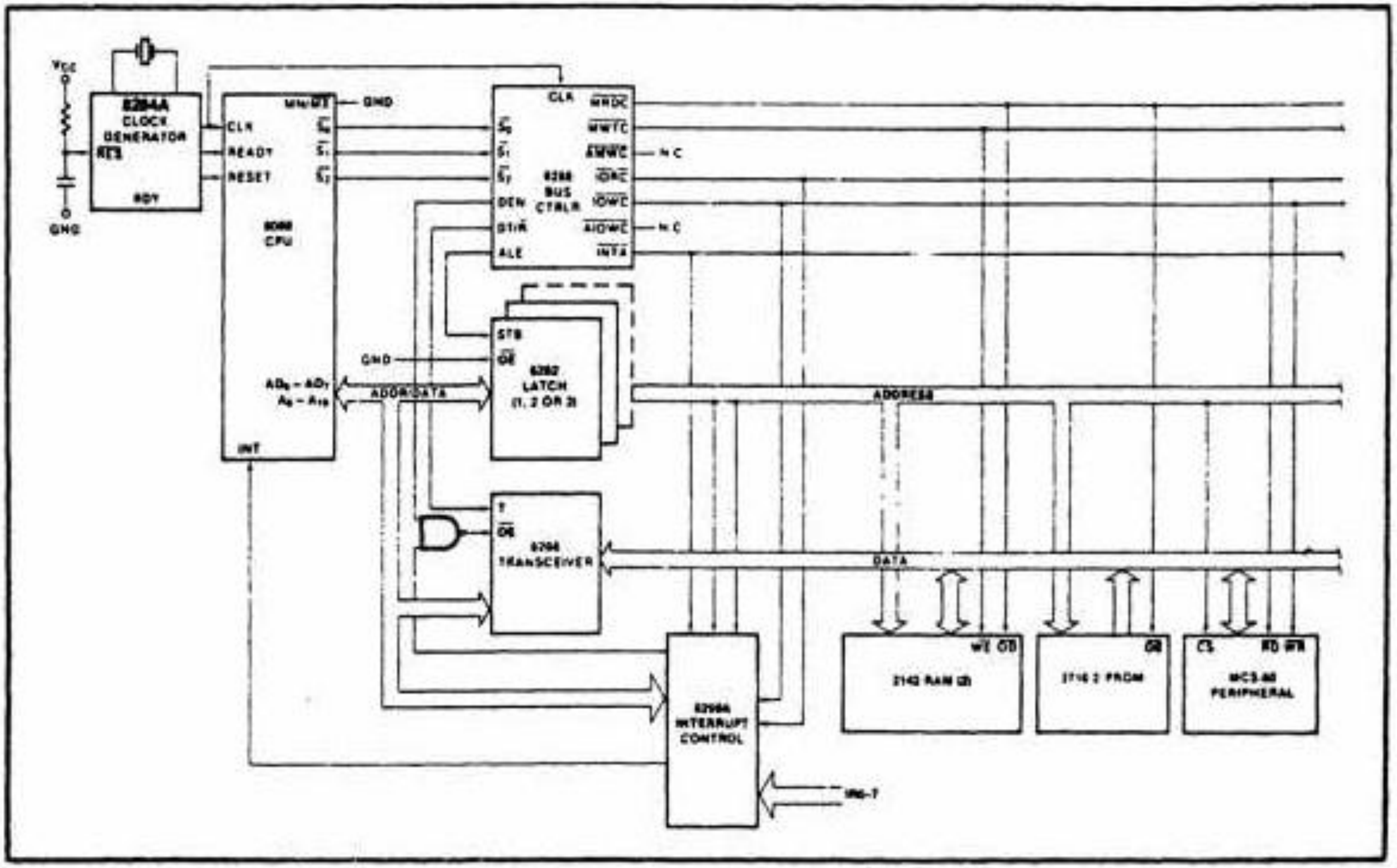
fig(c): Interfacing Stepper motor winding Wa.

DOS Functions

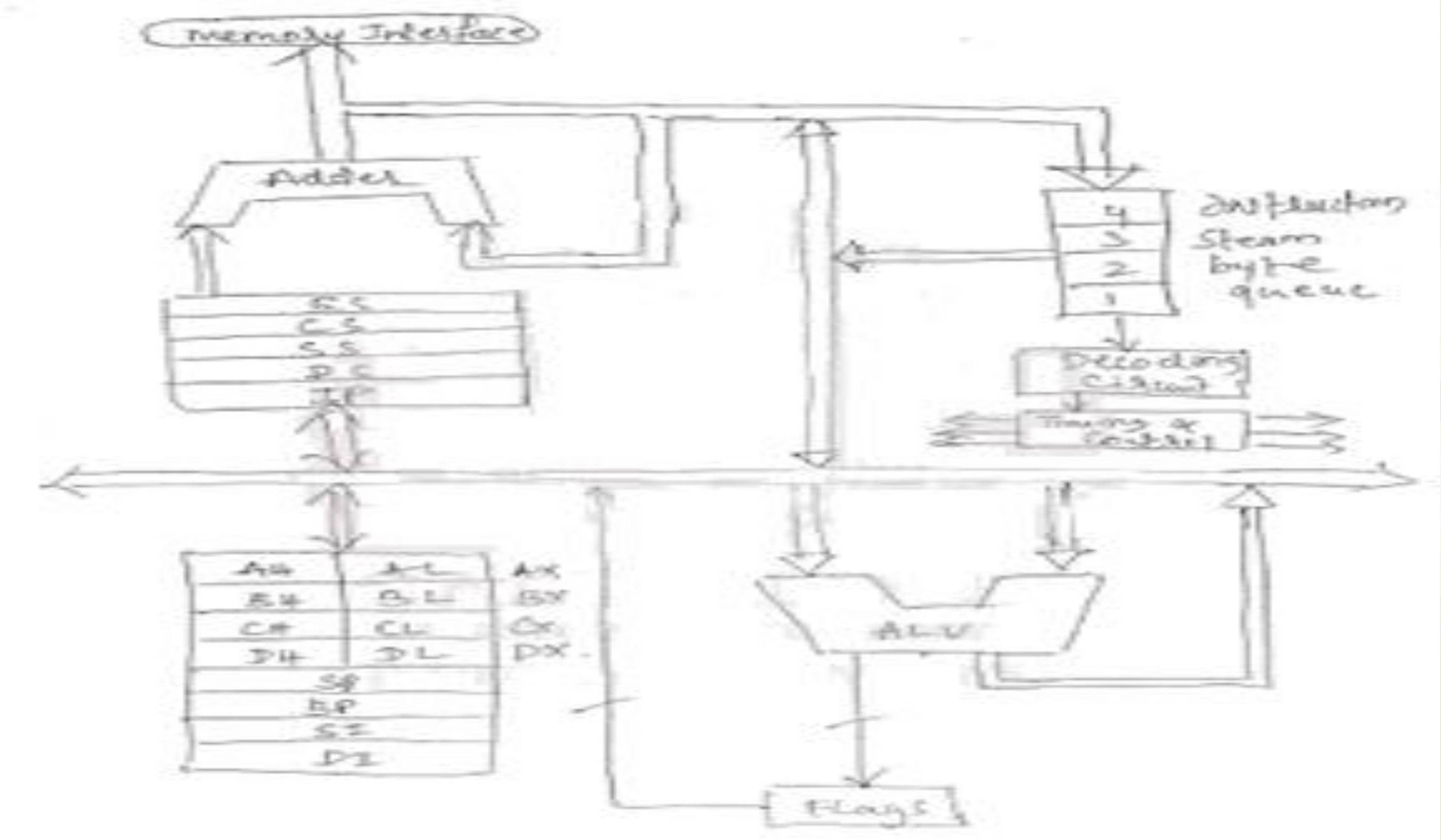
- DOS services are like Reading the Keyboard, Writing to Display, Disk access facilities etc
- All these services are accessed through INT 21H
- **Before invoking INT 21H**
- – Function Code is placed in the register AH.
- – Other relevant parameters if any are placed in appropriate registers

The 8088 Processor

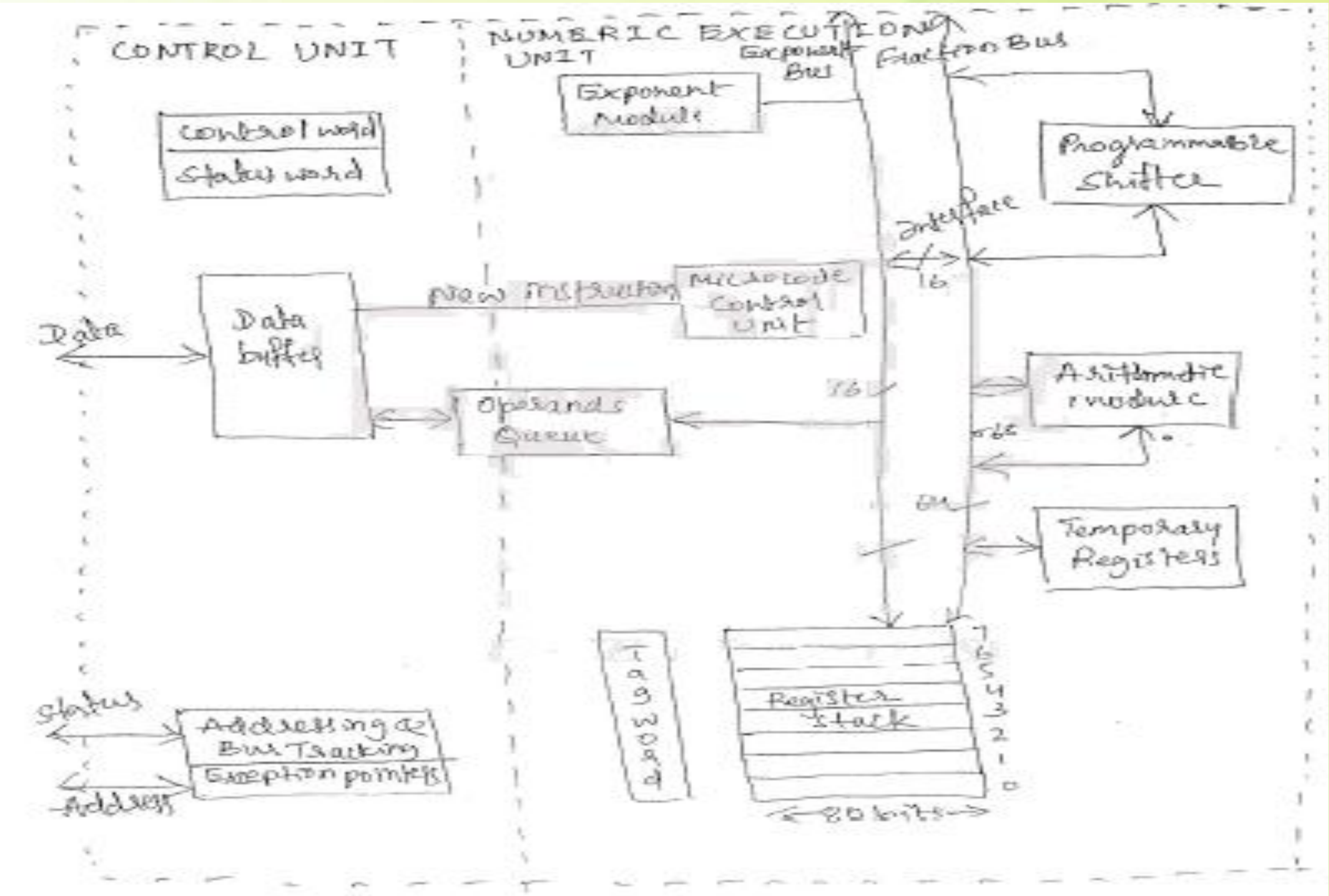



VCC
GND
8284A
CLOCK
GENERATOR
RES
RDY
CLK
READY
RESET
MN/MX
GND
S0
S1
S2
8088
CPU
AD0 – AD7
A8 – A19
INT
ADDR/DATA
CLK
MRDC
MWTC
AMWC
NC
IORC
IOWC
AIOWC
NC
INTA
8288
BUS
CTRLR
DEN
DT/R
ALE
STB
GND
OE
8282
LATCH
(1, 2 OR 3)
ADDRESS
T
OE
8286
TRANSCEIVER
DATA
8259A
INTERRUPT
CONTROL
IR0-7
WE OD
2142 RAM (2)
OE
2716-2 PROM
CS
RD WR
MCS-80
PERIPHERAL

Architecture of 8088



Architecture of 8087



Queries ...?