Pheonix

Processors and Controllers

Processors :-

→ 8085 is the first processor by intel.

→ 8086 - First processor implemented successfully by intel.

↓

16 bit microprocessor

→ 8085 - 8bit processor

→ 8085

↓

8086

↓

80186

↓

80286 → 80386 → 80486 → .... → i3 → i5 → i7

→ we use Binary in ALU (Arithmetic logic unit)

→ we use Hexadecimal in computer because

i) It has large Base i.e 16 to represent Maximum numbers

ii) We can convert Hexa decimal to Binary very easily.

→ Ex:- $16 \rightarrow 010000$ (in terms of Binary)

$16_H \rightarrow 00010110$ (Hexa in terms of Binary)

→ By using "8 4 2 1" code we convert Hexa into Binary

→ By using "8 4 2 1" code we write or convert any digit into Hexa

"8421"

Binary coded Decimal.

→ The highest number we can write "8421" code in Hexa is 15.

8+4+2+1 = 15.

Q) write 16 in terms of Hexa

Sol:- Split the sixteen 16

i.e.,

Split into two 1 6

↓ ↓

0001 0110

$16_H \rightarrow 0001\ 0110$

It is combination of 1 and 6

Q.) write 17 in terms of Hexa.

sol:- 17

↓

1 7

0001 0111

$17_H \rightarrow$ 0001 0111

Q) write 123 in terms of Hexa.

123

↓

1 2 3

0001 0010 0011

$123_H \rightarrow$ 0001 0010 0011

Q) write 3245 in terms of Hexa

3245

↓

3 2 4 5

0011 0010 0100 0101

$3245_H \rightarrow$ 0011 0010 0100 0101

Size of microprocessors

→ Bit

size of bit	Min	Max
(a (or) 1)	0H	1H

→ Nibble

4 bits

Min → max

0000H 1111H

0H → FH

→ Byte – 8 bits

Min → Max.

00000000H 11111111H

00H → FFH

→ Word – 16 bits

Min → max

0000000000000000H 1111111111111111H

0000H → FFFFH

ALU

→ Arithmatic Logic Unit.

→ Arithmatic Operations

ADD, SUB, MUL, DIV

→ logical operations

AND, OR, NOT, Inc.

→ ALU performs Arithmatic and logical operations.

→ Registors - We use registers to store some information.

Ex:- Result, output, values, etc...

8086 Microprocessor :-

→ There are 3 Buses.

→ Address Bus :- store the data at the address.
store the data's address.

→ Data Bus :- stores the data or values or inputs.

→ control Bus :- Control the flow of data i.e., the data is whether Read or write.

Features of 8086 (4 Marks)

→ It is a 16-bit processor.

→ Data Bus size is 16 bit. It supports maximum 16bit of data.

→ Address Bus size is 20 bits.

→ It consists 14 16-bit registers.

→ Memory capacity depends on Address Bus.

$= 2^{\text{Add.bus}} = 2^{20}$ - 1 Megabyte = 1MB.

Register organisation (4 Marks)

→ It is classified into categories.

1) General Purpose Registers.
2) segment Registers.
3) Point and Index Registers.
4) Flag Registers.

General Purpose Registers

→ It is further classified into 4 types.

1) Ax - size of Ax is 16 bit
2) Bx - size of Bx is 16 bit
3) Cx - size of Cx is 16 bit.
4) Dx - size of Dx is 16 bit.

→ Ax divided into AH and AL
↓ ↓
size 8bit size 8 bit

→ Bx divided into BH and BL
↓ ↓
size 8bit size 8 bit

→ Cx divided into CH and CL
↓
size 8bit size 8bit.

→ Dx divided into DH and DL

↓ size 8 bit ↓ size 8 bit

→ L → Lower H → Higher

→ Ax → AH, AL
Bx → BH, BL
cx → CH, CL
Dx → DH, DL

→ General purpose Registers are used to move or manipulate the data.

→ Ax can be used as accumulators also.
→ Bx can be used as Base Register.
→ cx can be used as counters, Loops.
→ Dx can be used as input/output port address

} Multi function.

→ a = 02H
b = 03H
c = a+b.

```
MOV   AL, 02H
MOV   BL, 03H
ADD   AL, BL
      END
```

} For addition.

MOV, ADD
instructions in Processor

MOV
Data transfer instruction.

ADD, SUB
Arithmetic instruction

→ For subtraction

```
MOV   AL, 02H
MOV   BL, 03H
SUB   AL, BL
      END
```

Q) write an 8 bit program to perform sixteen bit, addition and subtraction

Sol: Addition

```
MOV   Ax, 0004H
MOV   Bx, 0002H
ADD   Ax, Bx
      END
```

subtraction:

```
MOV  AX, 0004H
MOV  BX, 0002H
SUB  AX, BX
END
```

Segment Registers :-

→ segment Register are further classified into 4types.

1) CS → code Segment (CS)
2) SS → stack segment (SS)
3) ES → Extra segment (ES)
4) DS → Data segment (DS)

→ memory is divided into segments i.e., C-segment, S-segment, E-segment and D-segment.

→ C-drive only takes data.

→ Microprocessor takes segmented data.

→ code segment Registers are used to hold the starting address of segment Registers.

→ code gets stored in code segment.

→ Data gets stored in Data segment.

→ Extra data gets stored in Extra segment.

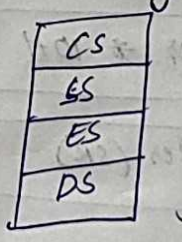
→ stack operations gets stored in stack segment.

→ SS is used to hold the starting address of the stack.

→ We use both segment and point and index registers to store the address.

→ CS - size is 16 bit
SS - size is 16 bit
ES - size is 16 bit
DS - size is 16 bit

memory.



→ 2^{16} = 64 kilo Bytes.
size 64 KB

→ The entire 1MB of memory can be segment into 4.

1. Code segment.
2. Stack segment.
3. Extra segment
4. Data segment.

→ The size of CS, ES, DS, SS registers are 16 bit.

→ CS, DS, ES, SS segment size is - 64 KB.

Point and Index Registers (P and I registers)

→ Point Registers.

i) Stack pointer (SP)

ii) Base pointer (BP)

iii) Instruction pointer (IP)

→ Index Registers :-

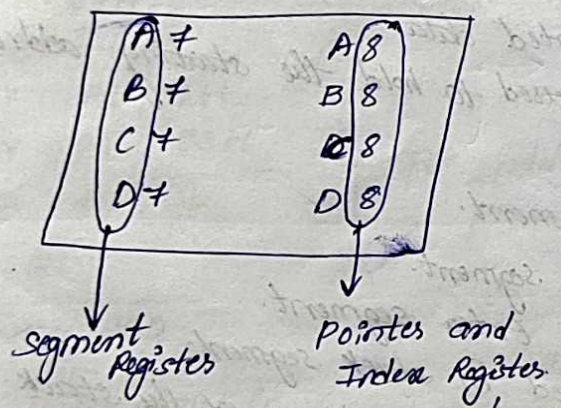
i) Source Index (SI)

ii) Destination Index (DI)

P and I Registers
↓
stores internal address

→ Segment Registers
↓
stores starting Address

Ex:- movie Hall



Generation of 20-bit Physical Address :-

→ Formulae :-

SR * 10H + P and I registers | OFF set code.

Segment Register * 10H + P and I registers

→

Segment Register (SR)	P and I registers
CS	IP
SS	SP (In some cases BP)
ES	DI
DS	SI

→ We have to use specific P and I registers for specific Registers.

→ 20 B.L P.A for CS = CS * 10H + IP
20 B.L P.A for SS = SS * 10H + SP
20 B.L P.A for ES = ES * 10H + DI
20 B.L P.A for DS = DS * 10H + SI

Q) Calculate the 20 bit physical address from the following information.

CS = 1234H
IP = 1111H

sol:- 20 bit physical address = CS * 10H + IP.

1234 * 10H

1234H
+ 1111H
13451H

13451 H.

Q) Calculate the 20 bit physical Address from the following information.

SS = 2222H
SI = 1234H

sol:- NOT Possible because stack segment uses stack pointer for calculating 20 bit physical address.

Q) Calculate the 20-bit physical address from the following information.

DS = 3475H, SI = 3435H

sol:- 20 bit physical address = DS * 10H + SI

3475H * 10H

34750H
+ 3435H
34885H

= 37885H.

→ Brackets i.e., [ ] → indicates Memory.

Q) Write an 8-bit program by using memory and assigning registors. for addition program.

sol:- → SI 2100 → 02H
SIH 2101 → 03H

```
→ MOV  SI, 2100H
  MOV  AL, [SI]
  INC  SI
  MOV  BL, [SI]
  Add  AL, BL
  End
```

→ with memory by assigning registors.

→ MOV AL, 02H
MOV BL, 03H
ADD AL, BL
End.
} without memory

8) write an 8-bit program by using memory for subtraction program.

Sol:- → SI 2100 → 02H
SI_H 2101 → 03H

→ MOV SI, 2100H
MOV AL, [SI]
Inc SI
MOV BL, [SI]
SUB AL, BL
End.
} → with memory by assigning registors

→ MOV AL, [2100 H]
MOV BL, [2101 H]
SUB AL, BL
} → simple by using memory without assigning registors.

→ In the above program the result/output is stored in AL.

Q. Write an 8-bit addition program by using memory and assigning registors and storing the output data in other 8-bit registor.

Sol: MOV DI, 3100H
MOV SI, 2100H
MOV AL, [SI]
Inc SI
MOV BL, [SI]
ADD AL, BL
MOV [DI], AL
end.

9. write an 16-bit addition program by using memory.

Sol:- 16-bit

1234_H 3311_H

AH AL BH BL

→

2100	34H
2101	12H
2102	11H
2103	33H

```
→ MOV SI, 2100H
  MOV AX, [SI]
  Inc SI
  Inc SI
  MOV SI, 2102H
  MOV BX, [SI]
  ADD AX, BX
  end
```

(or)

```
MOV SI, 2100H
MOV AX, [SI]
Add SI, 02
MOV SI, 2102H
MOV BX, [SI]
Add AX, BX
end
```

→ Giving the AX into AH, AL

```
MOV SI, 2100H
MOV AL, [SI]
Inc SI
MOV AH, [SI]
Inc SI
MOV BL, [SI]
Inc SI
MOV BH, [SI]
Inc SI
Add AX, BX
end.
```

Write an 16-bit subtraction program by using memory.

Sol:-

```
MOV SI, 2100H
MOV AX, [SI]
Inc SI
Inc SI
MOV SI, 2102H
MOV BX, [SI]
SUB AX, BX
end.
```

Logical Instructions (4 marks)

→ AND
OR
NOT
XOR

A	B	AND	OR	XOR	NOT
0	0	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	0
1	1	1	1	1	0

Q. Write an 8 bit program on AND program/operation.

A:-

```
MOV AL, 02H
MOV BL, 03H
AND AL, BL
End
```

Q. Write an 8 bit program on OR program/operation.

```
MOV AL, 02H
MOV BL, 03H
OR AL, BL
end.
```

Q. write an 8 bit program on NOT operation

Sol:
```
MOV AL,02H
NOT AL
End
```

Q. write an 8-bit program on NAND operation

NAND → AND + NOT

```
MOV AL,02H
MOV BL,03H
AND AL,BL
NOT AL
end.
```

Q. write a 8 bit program on And operation using memory

Sol:

SI 2100H → 02H

SI$_H$ 2101H → 03H

```
MOV SI,2100H
MOV AL,[SI]
Inc SI
MOV BL,[SI]
AND AL,BL
end.
```

10/6/24

Q. Write a 8 bit program of multiplication

Sol:

MUL BL * AL

↓ ↓

Given operand — compulsory operand

→ Any given operand in multiplication should be get multiplied with (or) by "AL" operand only.

→ Ex:- MUL CL

↓

MUL CL * AL

Ex:- MUL AL

↓

MUL AL * AL

→ As it is multiplication 2 "8" bit multiplication results in "16" bit.

So the result automatically gets stored in Accumulator i.e "Ax"

→ Here "AL" is default operand and by default the result gets stored in "Ax"

→ If MUL AH
↓
MUL AH * AL
} → In this case also by default it is AL

9. 16-bit Multiplication

→ In this 16-bit Multiplication the default multiplication operand is Ax.

→ Result is stored in Ax and Dx.

→ Lower unit gets stored in Ax and higher unit in Dx.

→ ex:- MUL Bx
↓
Bx * Ax

8-Bit Division

→ DIV BL — $\frac{Ax}{BL}$

Ax → is default divident

→ Any divisor operand will divide default Ax dividents

→ By default AL stores Quotient.

→ By default Quotient will be stored in AL.

→ By default Remainder will be stored in AH

→ ex:- DIV BL - $\frac{Ax}{BL}$

BL) Ax (AL
(AH)

16-Bit Division

DIV Bx - $\frac{Dx.Ax}{Bx}$

Bx) Dx Ax (Ax
(Dx)

→ By default remainder gets stored in DX.

→ By default Quotient gets stored in AX.

→ DX AX → is default divident

→ Any divisor operand will / should divide DX AX divident

Q. Write an 8-bit Multiplication program

Ans: → MUL BL
↳ BL × AL
↳ Result (AX)

→ MOV BL, 02H
MOV AL, 03H
MUL BL
End

→ 0006 stored in AX.

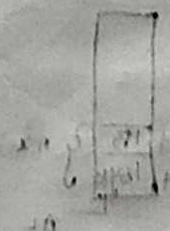
Q. write an 16-bit multiplication program.

Sol: → MOV BX, 0002H
MOV AX, 0003H
MUL BX
End

Q. write an 8-bit Division program.

Sol: $\frac{AX}{BL}$ BL) AX (AL
(AH)

→ MOV BL, 02H
MOV AX, 0016H
DIV BL
End



→ AL = 8 BL = 02H AL = 0BH,
AH = 0 AX = 0016H AH = 00H

Q. write an 16-Bit Division program

Sol:- DX AX / Bx

Bx) DX AX | AX

(DX)

→
```
MOV  BX, 0002H
MOV  DX, 0000H
MOV  AX, 0004H
DIV  BX
end
```

→ 00000004 / 0002

→ Ex:- 2222H

Data Address

→ Point and Index Registers → OFFSet

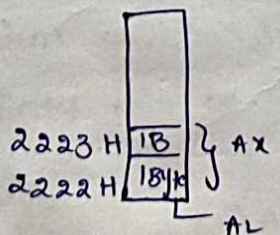
General purpose Register	Point	Index
↓	↓	↓
AX	SP	SI
BX	BP	DI
CX	IP	
DX		

→ Ex:- MOV AX, 2222H → Stores data

Ex:- MOV SI, 2222H → we can store information in it.

↓

Storing Address



2) write an ALP (Assembly Language Program) to perform 16 bit Arithmetic Operations by using 8086 Microprocessor.

Sol:
```
ADD → ADD AX, BX  - 4 memory locations
SUB → SUB AX, BX  - 4 memory locations
MUL → MUL BX      - 4 memory locations
DIV → DIV BX      - 6 memory locations
```

→ AX, 1234H
BX, 1211H

→ 2222H	34H
2223H	12H
2224H	11H
2225H	12H

```
→ MOV SI, 2222H
  MOV AX, [SI]
  Inc SI
  Inc SI
  MOV BX, [SI]
  ADD AX, BX
  RET
```

Q) write an ALP to perform 16 Bit subtraction operation by using memory.

sol:
```
MOV SI, 2222H
MOV AX, [SI]
Inc SI
Inc SI
MOV BX, [SI]
SUB AX, BX
RET
```

Q) write an ALP program to perform 16-bit Multiplication operation by using memory.

sol:
```
MOV SI, 2222H
MOV BX, [SI]
Inc SI
Inc SI
MOV AX, [SI]
MUL BX
RET
```

Q) write an ALP program 16-bit Division operation by using memory.

sol:
```
MOV SI, 2222H
MOV BX, [SI]
ADD SI, 02
MOV AX, [SI]
ADD SI, 02
MOV DX, [SI]
DIV BX
RET
```

8) Write an ALP program to perform the 16-bit logical operations.

Sol:

	A	B	Add	OR	XOR	NOTA
AND	0	0	0	0	1	1
OR	0	1	0	1	0	1
XOR	1	0	0	1	0	0
NOT	1	1	1	1	1	0

AX = 1234
BX = 1111

	0001	0010	0011	0100	
	0001	0001	0001	0001	
AND	0001	0000	0001	0000	(1010)
OR	0001	0011	0011	0100	(1334)
XOR	1111	1100	1101	1010	(FCDA)

AND →
```
MOV SI, 2222H
MOV AX, [SI]
INC SI
INC SI
MOV BX, [SI]
AND AX, BX
RET
```

OR →
```
MOV SI, 2244H
MOV AX, [SI]
INC SI
INC SI
MOV BX, [SI]
OR AX, BX
RET
```

XOR →
```
MOV SI, 2244H
MOV AX, [SI]
INC SI
INC SI
MOV BX, [SI]
XOR AX, BX
RET
```

NOT →
```
MOV SI, 2244H
MOV AX, [SI]
NOT AX
RET
```

9) Write an ALP program to perform 5 factorial (5!) using 8086

Sol: For looping we use CL Registers.

```
→ MOV CL, 05H
  MOV AL, 01H
UP MUL CL
   DEC CL
   JNZ UP
   RET
```

CL: 05 → 04 → 03 → 02 → 01 → 00

→ JNZ - Jump Non zero
If the above register value is zero then terminate the program.
If the above register value is not zero then it goes to up where we mentioned up before the register.

***
a) write an ALP program to find the positive and negative numbers in an array.
b) write an ALP program to find the even and odd numbers in an array.

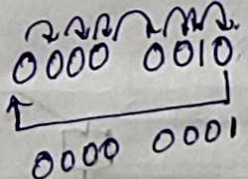
Shift and Rotate Instructions :-

Rotate Instruction :

→ There are 4 Rotate Instructions
ROR (Rotate Right)
ROL (Rotate Left)
RCR (Rotate Right with Carry)
RCL (Rotate Left with Carry)

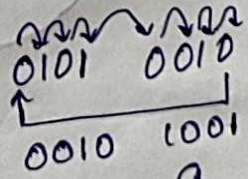
Rotate Right (ROR) :-

→ MOV AL, 02H
ROR AL, 01H



=> 1
01H

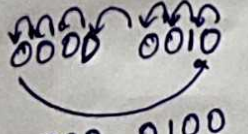
→ MOV AL, 52H
ROR AL, 01H



=> 29
29H

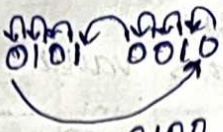
Rotate Left (ROL) :-

→ MOV AL, 02H
ROL AL, 01H



=> 4
04H

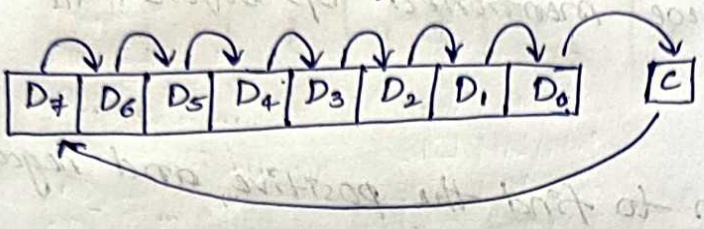
→ MOV AL, 52H
ROL AL, 01H



1010 0100 ⇒ A4
A4H

Rotate Right with carry (RCR) :-

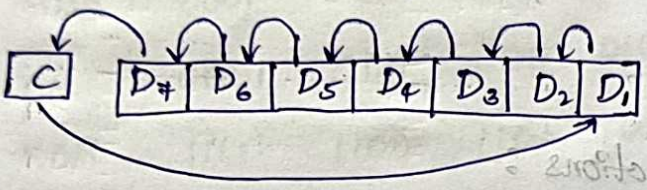
→



C = 1 (or) 0

Rotate left with carry (RCL) :-

→

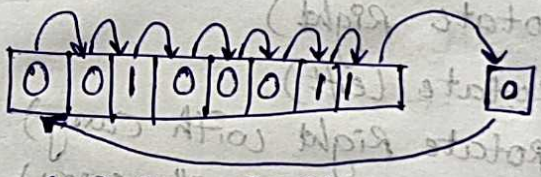


C = 1 (or) 0

→ CLRC → clear the carry

a) **RCR**

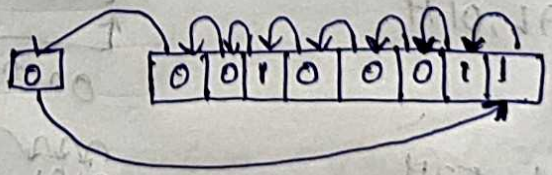
```
CLRC
MOV AL, 23H
RCR, AL, 01H
```



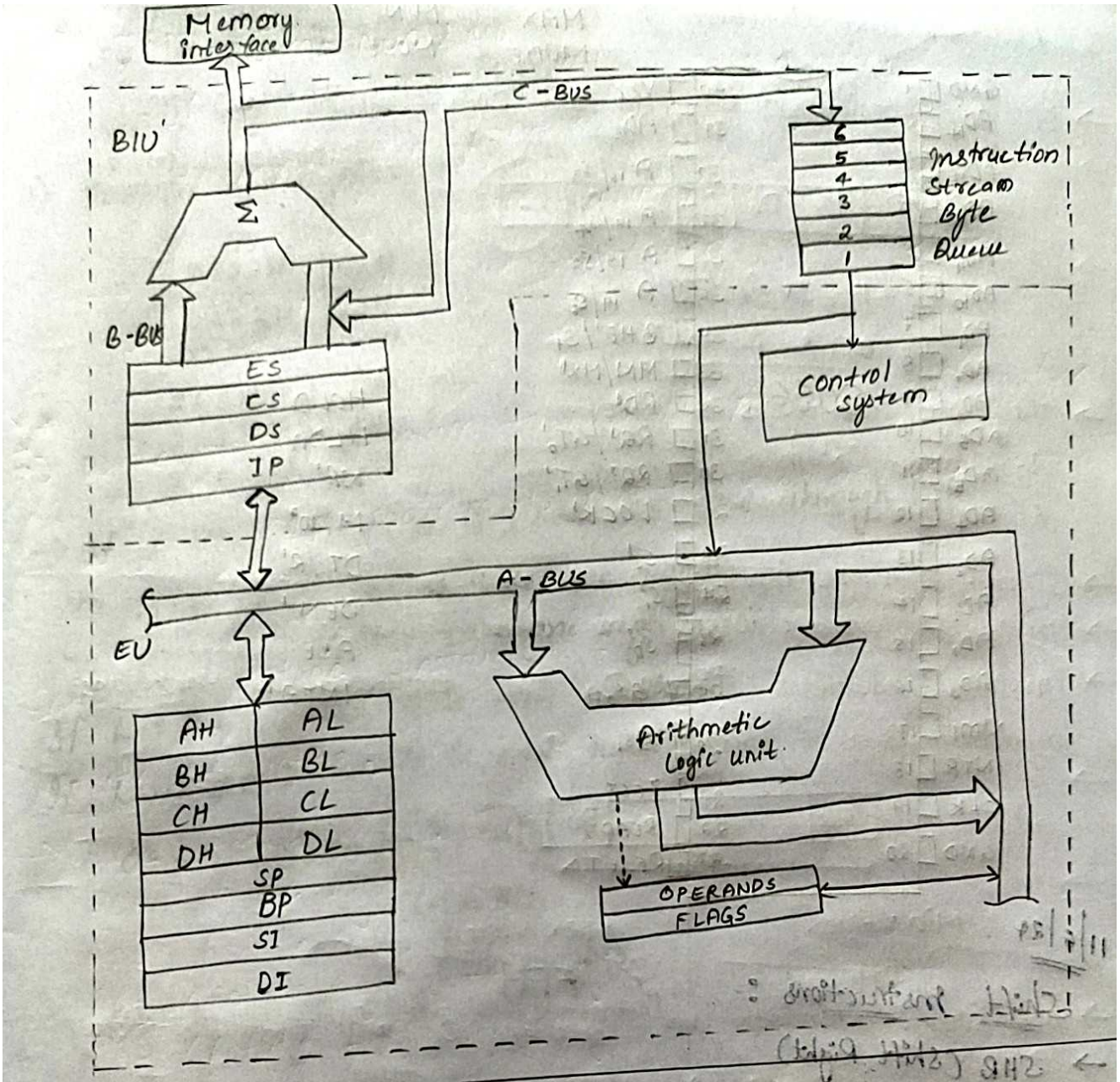
0 0 01 0001 ⇒ 11H
1 1

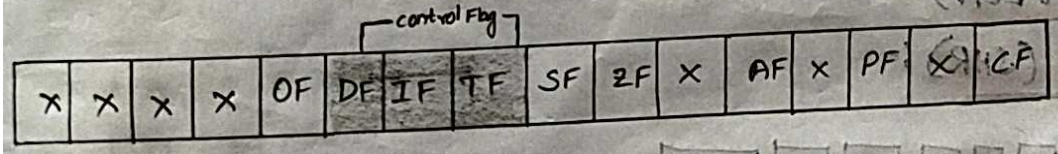
a) **RCL**

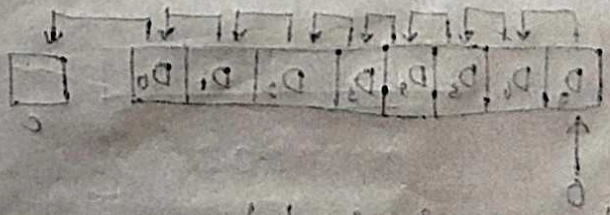
```
CLRC
MOV AL, 23H
RCL AL, 01H
```

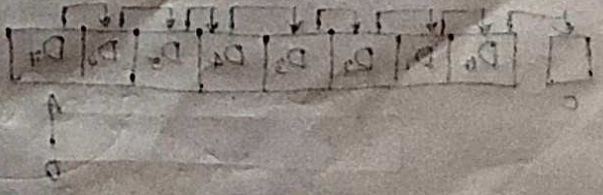


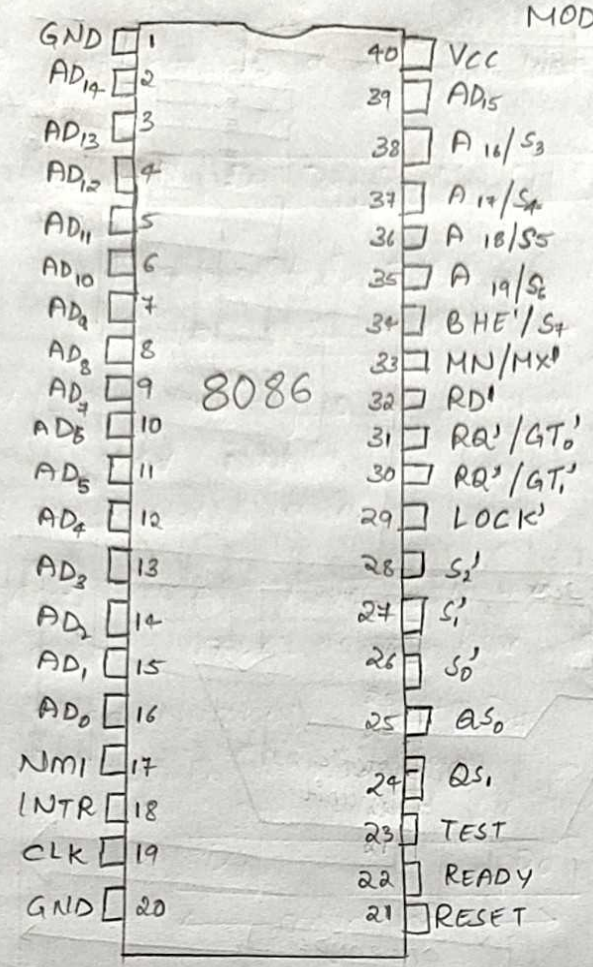
0100 0110 ⇒ 46H
4 6


Memory interface
C-BUS
BIU
Σ
6
5
4
3
2
1
Instruction Stream Byte Queue
B-BUS
ES
CS
DS
IP
Control system
A-BUS
EU
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
Arithmetic Logic unit
OPERANDS
FLAGS


control Flag
X
X
X
X
OF
DF
IF
TF
SF
ZF
X
AF
X
PF
X
CF







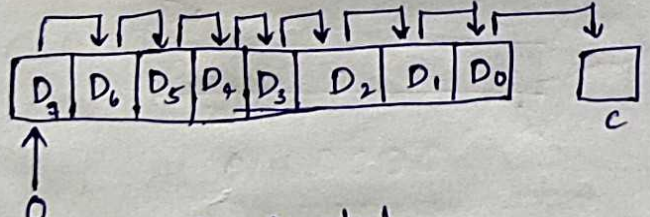
11/6/24

Shift Instructions :-

→ SHR (Shift Right)

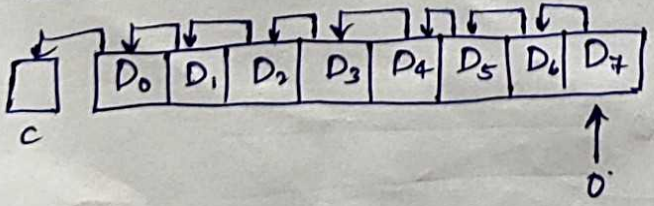
→ SHL (Shift Left)

Shift Right (SHR) :-



→ New zero is inserted.

Shift left (SHL) :-



∴ New zero is inserted

MOV AL, 56H
MOV AL, 01H

$56 \rightarrow 0101\ 0110$ (C)

$0010\ 1011$ (C) $\rightarrow$ 2BH

2 B

Even Odd

a) Positive and negative with an array?

2222H	02H
2223H	03H
2224H	04H
2225H	09H
2226H	06H
2227H	07H

→ If a number is even (or) odd we find through LSB.

To store even numbers take one register.

To store odd number take one register.

If L.S.B = 0 ; then it is even numbers.

If L.S.B ≠ 0 ; then it is odd numbers.

$02 \rightarrow 0000\ 0010 \rightarrow 0$ (C)

L.S.B Bit

↓

Even numbers.

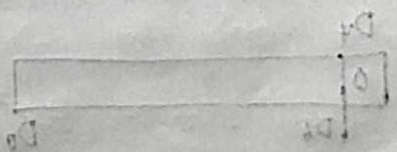
$03 \rightarrow 0000\ 0011 \rightarrow 1$ (C)

L.S.B Bit

↓

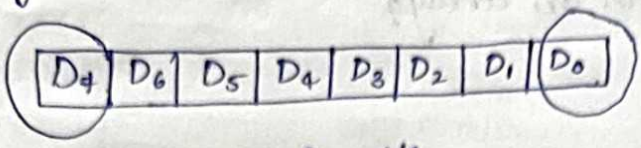
Odd numbers

```
MOV BX, 0000H
MOV DX, 0000H
MOV CL, 06H
MOV SI, 2222H
UP: MOV AL, [SI]
    SHR AL, 01H
    JC  GO
    Inc BX
    JMP Down
GO: Inc SI
Down: Dec CL
    JNZ UP
    RET
```



Q) Positive and Negative in an array?

→ If you consider a bit the last Digit is consider as sign



$D_7 = 1$; It is -ve

$D_7 = 0$; It is +ve

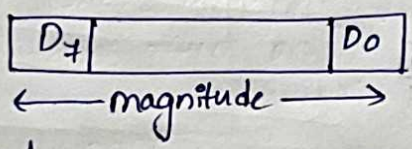
→ They are two types of Numbers.

1. Signed Number.
2. unsigned Number.

→ Signed Numbers have both +ve and -ve indications.

→ unsigned Numbers donot have sign indications.

→ unsigned Numbers D_0 to D_7 are considered as Magnitude because there is no Sign.



Min value 00H

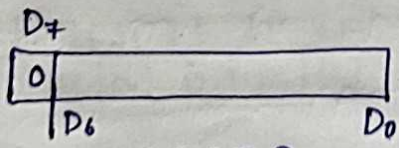
MAX value FFH

→ For signed Numbers we consider D_7 as sign indication.

$D_7 = 1 \rightarrow$ -ve numbers

$D_0 = 0 \rightarrow$ +ve number

→ positive numbers :-

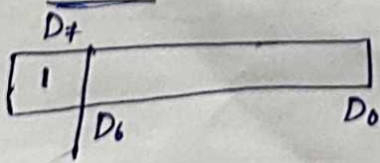


0000000

1111111

Min value 00H ⟶ Max value 7FH } → this is positive numbers.

→ Negative numbers :-



0 00 0000

1 11 1111

Min value 80H ⟶ max value FFH } These are Negative Numbers

→ For signed numbers if it is b/w 00H → 7FH it is positive number, but if it is b/w 80H → FFH it is negative number.

→ positive and Negative Numbers :-

```
    MOV  BX, 0000H
    MOV  DX, 0000H
    MOV  CL, 06H
    MOV  SI, 2222H
UP: MOV  AL, [SI]
    SHL  AL, 01H
    JC   GO
    Inc  BX
    JMP  Down
GO: Inc  DX
Down: Inc SI
    Dec  CL
    JNZ  UP
    RET
```

Flag Register :-

→ In Flag Register. Flag only 9-bits are only used
7-bits are not used

→ Further these 9-bits are classified into 2.
i- status flag - 6
ii- control flag - 3
} 6+3=9

→ (control flag are Direction Flag, Interrupt Flag, Trap flag (DF, IF, TF)

→ status Flag are Overflow Flag, sign Flag, zero Flag, Auxillary Flag, carry Flag (OF, SF, ZF, AF, CF).

→ If there is a carry forward from the position D_7 to outside then carry flag is "1" (CF = 1)

→ If there is no carry forwarded from D_7 to outside then carry flag = 0. (CF = 0)

→ Ex:

```
  1000 0000   -80
  1000 0000   -80
 ------------
  0000 0000
```

carry flag = 1, CF = 1

1 ↲ carry Forwarded from D_7

```
  0010 0000   -20
  0010 0000   -20
 ------------
  0100 0000
```

carry flag = 0, CF = 0

No carry from D_7.

Parity flag :- (PF)

→ Even no. of one's then parity flag = 1
odd no. of one's then parity flag = 0

→ Ex:-

```
  0111 0001
  0000 0000
 ------------
  0111 0001
```

No. of one's = 4
Parity flag = 1, PF = 1

→ Ex:-

```
  0011 0001
  0000 0000
 ------------
  0011 0001
```

No. of one's = 3
Parity flag = 0, PF = 0

→ Applicable only for Arithmetic Operations.

Axuillary carry Flag :- (AF)

→ If there is a carry forward from D_3 to D_4 then Axuillary carry flag is 1. (AF = 1)

→ If there is a no carry forward from D_3 to D_4 then Axuillary carry flag is 0. (AF = 0).

→ Ex:-

```
  0000 1000
  0000 1000
 ------------
  0001 0000
```

carry forward from D_3 to D_4

∴ AF = 1

→ Ex:-

```
  0010 0000
  0010 0000
-------------
  0100 0000
```

No carry forward from D_3 to D_4

Auxillary carry flag = 0

∴ AF = 0.

→ Applicable only for Arithmetic operations.

18/6/24 Zero Flag (ZF):-

→ If the result of any arithmetic operations contains all zero's then zero flag = 1 (ZF=1).

→ If the result of any arithmetic operations doesnot contain all zero's then zero Flag = 0 (ZF=0).

→ Ex:-

```
  1000 0000
  1000 0000
-------------
  0000 0000
```

All zero's in result zero Flag = 1 (ZF=1)

→ Ex:-

```
  1000 0000
  0000 0000
-------------
  1000 0000
```

All ~~zeros~~ are not zero's in result zero flag = 0 (ZF=1)

Sign Flag (SF) :-

→ D_7 is 1 then it is negative number.

→ D_7 is 0 then it is positive number.

→ Ex:-

```
  1000 0000
  0000 0000
-------------
  1000 0000
```

↓

1 => so it is negative number.

→ Ex:-

```
  0010 0000
  0010 0000
-------------
  0100 0000
```

↓

0 => so it is positive number.

Overflow flag (OF) :-

→ If the result exceeds the byte :
Then overflow = 1 (OF = 1)

→ Overflow flag used at the / considered at the signed Arith-metic operations.

→ It is set to "1", when result is exceeded the limit.

→ Max in unsigned Arithmetic Operations overflow is zero.

Q) Show the status of flag after execution of the following instruction.

```
MOV AL, 23H
MOV BL, 70H
ADD AL, BL
RET
```

Sol:-

```
  0010 0011
  0111 0000
-------------
  1001 0011
-------------
```

C.F = 0
P.F = 1
A.F = 0
Z.F = 0
S.F = 1
O.F = 1

→ For overflow flag if the limit exceeds overflow = 1.

→ If we take positive numbers and the results is Negative then also overflows flag = 1.

Control flags :-

→ These flags are controlled by users

→ Programmer can control set or Reset the control flags
i) Trap flag.
ii) Direction flag.
iii) Interrupt flag.

→ To set them "ST → 1".

→ To clear them "CL → 0".

→ After ST or CL we have mention the flag's 1st letter to for better understanding for computer.

→ Ex for carry flag :- STC → 1
CLC → 0

→ After ST or CL we have mention the flag's 1st letter to mention the particular flag.

Interrupt flag (IF):-

→ Interrupt means disturb.
→ If a program is getting runned in 8086 processor if it gets disturbed by other devices then interrupt flag=0 (IF=0).
→ If any peripheral devices interrupt the processor then IF=0.
→ If there is no disturbance then IF=1.
→ To stop the disturbance we can give "CLI" instruction.
→ I.F=0 → CLI.
I.F=1 → STI.

Direction flag:-

→ Direction flag used at the string operations.
→ We write Inc and Dec in some operations, To automatically do the/ perform the Inc or Dec we can use this Direction flag.
→ only used in string operations.
→ CLD → Automatically increments SI by 1 and increments DI by 1.
→ STD → Automatically decrements SI by1 and decrements DI by 1.

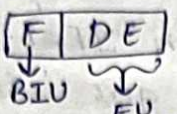
Trap flag:-

→ If the program gets executed step by step then Trap flag=1 (TF=1).
→ If the program gets executed directly i.e., not step by step then Trap flag = 0. (TF=0).

Architecture of 8086:-

→ F D E → Execute
↓ ↳ Decode
Fetch
→ Any instruction gets executed by these FDE only.
→ Fetch - Fetching data from the memory.
It might be data/ instructions.
→ Decode - Fetched information is decoded into machine language
→ Execute - The decoded data is executed in ALU.
→ Two or three instructions gets executed at same time is pipelining.
→ pipelining is first implemented in 8086 processor.

→ 8086 Architecture is divided into 2 units.
  i) Bus Interface unit (BIU)
  ii) Execution unit (EU)
→ Fetching is done by 'BI' unit.
→ Decoding and execution are done by "E" unit.



→ 2 ⇒ This calculates the 20 Bit physical Address which is combination of segment Registers and P & I Registers.
→ As it is sum of segment Registers and P and I Registers it is denoted by Σ.
→ size of Queue is 6-Bytes.
→ Queue fetches 6-Bytes of Information and stores in it.
→ First code gets Fetched so it's Pointer is IP so it is placed with segment Registers.
→ At the time of exchanging information operands are used.

Ex:- BL, 03
AL, 02
XCHG AL, BL

→ The information gets overwright in BL. That information gets stored temporarily in operands.
→ BIU
- Fetch the instructions from the memory.
- Fetch the instructions from the I/o Devices (Input/output)
- It consists of 6-Bytes of Queue with First In First out (FIFO)
- It consists of 20-Bit physical Address calculation Part.
- Fetched instructions are stored in 6-Bytes of Queue.

→ EU
- Decode the instruction from "BI" unit.
- It execute the instructions.
- The status of execution instruction are stored into flags.
- It contains GPIO Register and point and Index Registers.
  ↓
  General purpose

Pin Diagram of 8086

Addressing Modes of 8086

1. Immediate Addressing Mode.
2. Register Addressing Mode.
3. Direct Addressing Mode.
4. Register Indirect Addressing Mode
5. Base Addressing Mode
6. Base Index Addressing Mode
7. Relative Base Index Addressing Mode

→ way of Accessing Data from Memory is called Addressing Mode
→ way of Accessing Data from Sources is termed as Addressing Mode
→ The sources may be a value, a register, a memory.
→ Destination is not changed only source is changed (MOV D,S)

Immediate Addressing Mode :-

→ Ex :- MOV AL, 02H
→ The value gets stored immediately. so it is called Immediate Addressing Mode.
→ If the source is a value then it is Immediate Addressing Mode.

Register Addressing Mode :-

→ Both operands are registers i.e., both source and destination are registers
→ If the sources is registers then it is Register Addressing Mode.
→ Ex:- MOV AL, BL

Direct Addressing Mode :-

→ Ex:- MOV AL, [2100H]
→ If the memory is directly visible in brackets then it is Direct Addressing Mode

Register Indirect Addressing Mode :-

→ The memory is pointed to register. That register is visible in brackets Then it is Register Indirect Addressing Mode.
→ Ex:- MOV AL, [SI]

Base Addressing Mode :-

→ If source is Base Register, then it is Base Addressing Mode
→ It is mentioned in Brackets.
→ Ex:- MOV AL, [BX]

Base Index Addressing Mode :-

→ If source is Base Register + Index Register i.e., Source Index (SI) or Destination Index (DI), then it is Base Index Addressing Mode. Ex:- MOV AL, [BX+SI]

Relative Base Index Addressing Mode :-

→ Ex:- MOV AL, [BX+SI] 04H

→ If the source is Base Register + Index Register + value then it is Relative Base Index Addressing Mode.

Q) Write an ALP program to perform Addition, Subtraction, Multiplication and Division of 8-bit data using Direct Addressing Mode.

Sol:-

Addition - 8bits

```
MOV AL, [2100H]      ; 2100 - AL - 02H
MOV BL, [2101H]      ; 2101 - BL - 04H
ADD AL, BL
RET
```

Subtraction - 8bits

```
MOV AL, [2106H]      ; 2106 - AL - 04H
MOV BL, [2107H]      ; 2107 - BL - 02H
SUB AL, BL
RET
```

Multiplication - 8bits

```
MOV BL, [2103H]      ; 2103 - BL - 06H
MOV AL, [2104H]      ; 2104 - AL - 02H
MUL BL
RET
```

Division - 8bits

```
MOV BL, [2102H]
MOV AX, [2105H]
DIV BL
RET
```

***

Q) Write a program to divide a data byte located at offset "0800H" in "6500H" Segment to another data byte available at "0801H" in the same Segment and the result is stored at "0802H" in the same segment.

sol:-

Div → BL → $\frac{AX}{BL}$

```
MOV AX,00H
MOV AL,[0800H]
MOV BL,[0801H]
DIV BL
MOV [0802H],AX
RET
```

ADDITION →

```
MOV AL,[0800H]
MOV BL,[0801H]
ADD AL,BL
MOV [0802H], AL
RET
```

→ MUL BL → BL*AL

```
MOV BL,[0800H]
MOV AL,[0801H]
MUL BL
MOV [0802H],AX
RET
```

SUB →

```
MOV AL,[0800H]
MOV BL,[0801H]
SUB AL,BL
MOV [0802H],AL
RET
```

→ 2100	02H
2101	03H
⋮	
2109	05H

Using Direct Addressing Mode

```
MOV AL,[2100H]
MOV BL,[2109H]
ADD AL,BL
```

Using Relative Base Index

```
MOV SI,2100H
MOV SI,[SI]08H
```

Pin Diagram of 8086 :-

→ 33 pin – MN/MX'

MN – minimum Mode – 1

MX' – Maximum Mode – 0

MN/MX' ← 0 (If "0" default in pin)

0/1

33 → 0 → Max Mode

33 → 1 → Min Mode

33 pin of 8086 :- If 33 pin is 0 – Max Mode

If 33 pin is 1 – Min Mode

→ Among 40 pins 8 pins are different.

→ The different pins are 24 to 31.

→ Remaining pins are common for Min Mode and Max Mode.

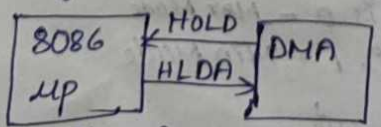
24 → 31 (Min)	24 → 31 (Max)
HOLD	RQ'/GT_0'
HLDA	RQ'/GT_1'
WR'	LOCK'
M/IO'	S_2'
DT/R'	S_1'
DEN	S_0'
ALE	QS0
INTA'	QS1

→ Address → A-20
Data → D-16
AD_0
⋮
AD_{15}
Address and Data Lines are 16.
Remaining are A_{16}, A_{17}, A_{18}, A_{19} are combined with signals i.e.,
A_{16}/S_3
A_{17}/S_4
A_{18}/S_5
A_{19}/S_6

→ Here we take higher amount of inputs to avoid any damage we use 8 GND pins.

→ ALE - Address Latch Enable (25 pin)
If ALE=1 then AD_0 to AD_{15} acts as address lines.
If ALE=0 then AD_0 to AD_{15} acts as Data Lines.
$AD_0 - AD_{15}$ acts as Address and Data Lines.
At a time they cannot act as address and Data. So to differentiate it we use ALE.

→ HOLD
HOLD Acknowledgment (HLDA)
To send larger amount of data to 8086 we use DMA (peripheral device)



→ WR (29 pin) - write pin
Read or write pin
RD
NR

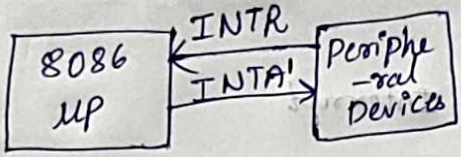
RD (32 Pin) - Read Pin.

→ M/IO' (28 Pin)
Input/data taken from Memory or Input pin
1 ⇒ Data taken from memory.
0 ⇒ Data taken from Input.

→ DT/R' (27 pins)
Data Transmit / Receives
Write operation ⇒ DT/R' = 1 (Transmits)
Read operation ⇒ DT/R' = 0 (Receive)

→ INTA' (24 Pin)
Interupt Acknowledgment.

→ INTR (18 Pin)
Interrupt Request



→ CLK (19 Pin)
In 8086 any information / instruction will gets passed by clock.
It supports of 5MHz, 8MHz, 10MHz } → clock frequency
Clock frequency of 8086 is 5MHz, 8MHz, 10MHz.

→ Reset (21 Pin)
If reset happens in 8086, then all registers are zero's except code segment.
CS = FFFFH

→ Test (23 Pin)
It genesates wait signals / weight Signals to synchronise the peripheral Devices.

→ Ready (22 pin)
Ready Signal is high (1) when Io/M is ready to send the data.
Ready Signal is low (0) when Io/M is not ready to send the data.

→ Types of Interrupts:
There are two types of Interrupts.

i) Maskable

ii) Non-maskable

Maskable - we change it either to 0 or 1

Non Maskable - we cannot change them.

Maskable interrupts → INTR

CLI → Clear the Interrupt } → Instructions of Maskable Int

STI → set the Interrupt

Non-Maskable interrupts are used at the Debugging process.

During debugging process or power failure cases we use NM

are enabled

NMI → Non-Maskable Interrupt.

→ DEN (26 pin)

Data Enable

If you want to send data through Data Bus, must be 1.

→ BHE (34 pins)

Bus High Enable

BHE pin is used at the Memory Banking Address

To enable BHE we have to use higher order Data bits.

→

ALE	$AD_0 - AD_{15}$	$A_{16}/S_3 - A_{19}/S_6$
0	$D_0 - D_{15}$	$S_3 - S_6$
1	$A_0 - A_{15}$	$A_{16} - A_{19}$

14/06/24

Operations	28 M/$\overline{IO}$	32 $\overline{RD}$	29 $\overline{WR}$	24 DT/$\overline{R}$	26 $\overline{DEN}$	
Memory Read	1	0	1	0	0	
Memory write	1	1	0	1	0	
I/O write	0	1	0	1	0	
I/O Read	0	0	1	0	0	

→ S_3 S_4 segment selection

0 0	ES
0 1	SS
1 0	CS
1 1	DS

→ S_5 is shows the state of Interrupt.

→ 1 → Already in Interrupt state.

0 → 8086 is not in Interrupt.

→ S_6, S_7 – Not used.

→

S_2	S_1	S_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

$\underbrace{S_2\ S_1\ S_0}$
Max Mode

→ Difference b/w Min Mode and MAX Mode

MIN Mode

i- 33 pin MN/$\overline{MX}$ is 1

ii- In Min Mode only single processor is consider

MAX Mode

i- 33 pin MN/$\overline{MX}$ is 0

ii- In Max Mode Multiple processor is consider

→ Lock :- To lock the remaining processor while performing multiple tasks with multiple processors.

QS_1	QS_0	Queue
0	0	Ideal
0	1	1st byte is fetched
1	0	Queue is empty
1	1	

} Tell about status of Queue in 8086

RQ/GT_0

- To request and grant the buses.

RQ_1 = Request
GT = Grant

→ Timing diagram for minimum mode memory write.

→ We need
- $MN/\overline{MX}$ } will change based upon question
- $M/\overline{IO}$ } will change based upon question
- $\overline{WR}$ } will change based upon question

Minimum mode memory write

- ALE → conform
- $AD_0 - AD_{15}$ } → conform
- $A16/S_3 - A19/S_6$ } → conform
- DT/R → conform
- Ready → conform
- DEN → conform

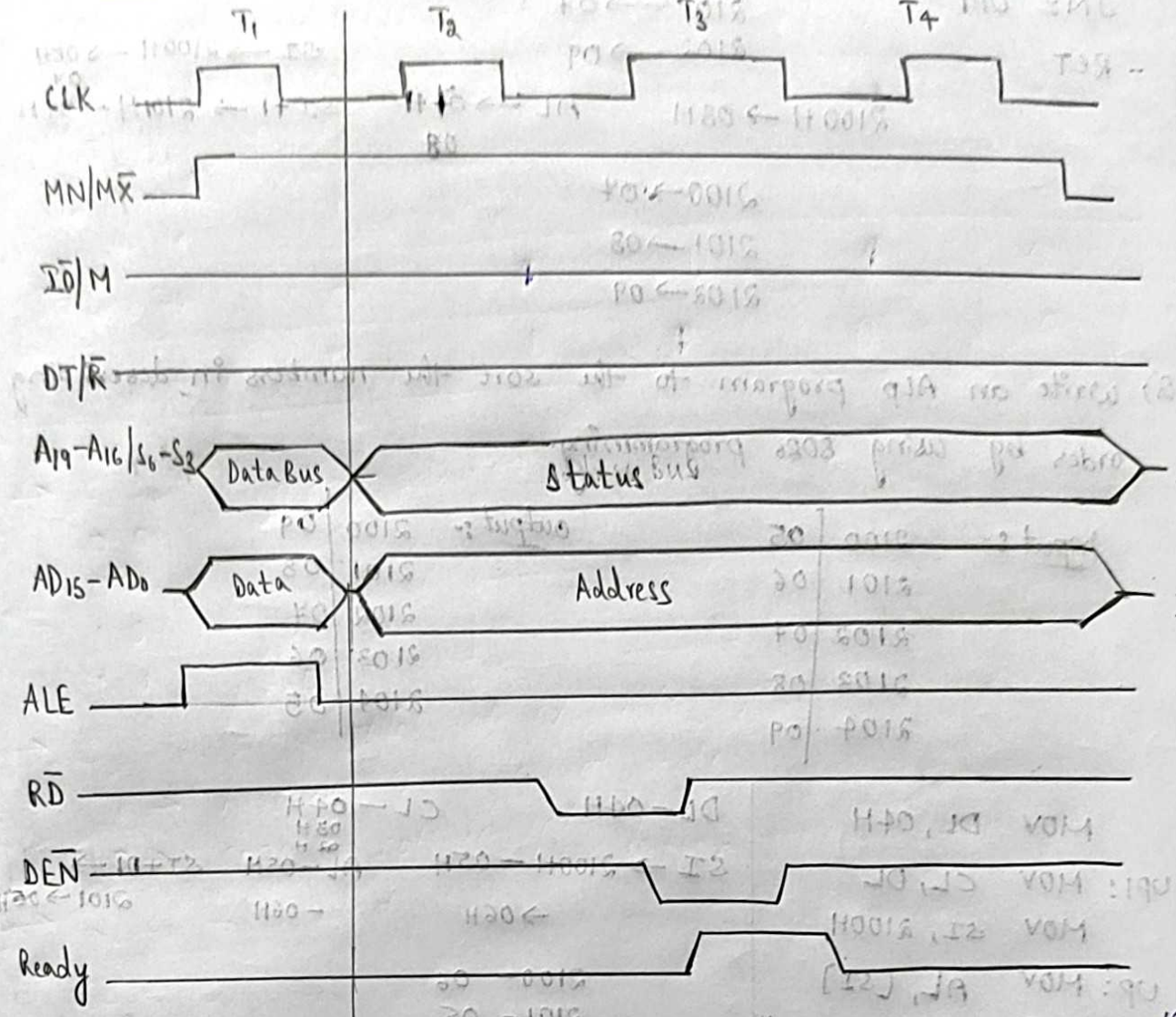
→ How the instruction gets executed in Architecture of 8086 timing diagram tells about this.

→ Any instruction needs 4 clock cycles to gets executed in 8086.

→ Fetching only occurs in T_1 state.

→ Decode and Execute occurs in 3 clock cycles i.e., T_2, T_3, T_4

→ Draw the Timing Diagram Minimum Mode I/o Read :-



→ Write an ALP program to sort the numbers in ascending order by using 8086 programming.

2100	09
2101	08
2102	07

```
    MOV DL, 02H
UP1: MOV CL, DL
    MOV SI, 2100H
UP: MOV AL, [SI]
    CMP AL, [SI+01]
    JB  GO
    XCHG AL, [SI+01]
    XCHG AL, [SI]
GO: INC SI
    DEC CL
    JNZ UP
    DEC DL
```

Output :-

2100	07
2101	08
2102	09

DL → 02 01 00 CL → 02 01 00

SI → 2100H → 09H 08 07

2101 — 09 07

AL → 09H 08 09

AL − SI+01

09 − 08

08 − 09

SI+1 → 2101H → 08 09

2100H → 08

2101H → 09

2102H → 07

AL → 09H 07H

SI+1 → 2102 − 07 09

```
JNZ UP1
RET
```

2100 → 08
2101 → 07
2102 → 09

SI → 2100H → 08H
SI+1 → 2101H → 07H

2100H → 08H

AL → 07H
08

2100 → 07
2101 → 08
2102 → 09

3) Write an ALP program to the sort the numbers in descending orders by using 8086 programming.

Input :-

2100	05
2101	06
2102	07
2103	08
2104	09

output :-

2100	09
2101	08
2102	07
2103	06
2104	05

```
    MOV  DL, 04H
UP1: MOV  CL, DL
     MOV  SI, 2100H
UP:  MOV  AL, [SI]
     CMP  AL, [SI+01]
     JNB  GO
     XCHG AL, [SI+01]
     XCHG AL, [SI]
GO:  Inc  SI
     Dec  CL
     JNZ  UP
     Dec  DL
     JNZ  UP1
     RET
```

DL − 04H
CL − 04H
03H
02H

SI → 2100H − 05H
→ 06H
AL − 05H
− 06H
SI+01 → 2101 → 05

2100 − 06
2101 − 05
2102 − 07
2103 − 08
2104 − 09

SI → 2101H − 05H
07H
AL − 05H
07H
SI+01 → 2102 − 07H

2100 − 06
2101 − 07
2102 − 05
2103 − 08
2104 − 09

SI → 2102 − 05
08H
AL − 05H
08H
SI+01 → 2103 − 08

2100 − 06
2101 − 07
2102 − 08
2103 − 05
2104 − 09