Module: 2

ADDRESSING MODES OF 8086

26/07

	Mode	Example	Register	Effective Address
1.	Immediate * Doesnot contains #	MOV AX, 0005H desti source MOV AL, 05 H 8 bit	—	—
2.	Direct	(Content of) → [ ] MOV AX, [5000H]	Seg. R → DS offset → add. ress	10H * DS + 5000 Shifting of DS
3.	Register	MOV BX, AX * (except IP)	—	—
4.	Register Indirect	MOV AX, [BX]	Seg R → DS/ES Offset → BX, SI, DI	Shift DS + address of 10H * DS + [BX] BX
5.	Indexed	MOV AX, [SI] operand in the one address in SI given	Default: DS/ES reg Offset: SI, DI	10H * DS + [SI]
6.	Register Relative	MOV Ax, 50H [BX] 8 bit displacement	De.reg: DS/ES offset: BX, BP, SI, DI.	displ/ value Relative 10H * DS + 50H + [BX]
7.	Based Indexed	MOV AX, [BX] [SI]	de.reg: DS/ES Base reg: BX/BP offset: SI/DI	10H * DS + [BX] + [SI
8.	Relative Based Indexed	MOV AX, 50H [BX] [SI]	d.reg: DS/ES B Reg: BX/BP offset: SI/DI	10H * DS + 50H + [BX] + [SI]
9.	Intra segment Direct Jump within the same seg.	Unconditional −128 to +127 JMP SHORTLABEL	CS/IP	
10.	Intra segment Indirect	JMP [BX] Indexed reg/ base		10H * CS + [BX] + [IP]

Control transfer / add. modes

Control transfer

11. Intersegment Direct	20bit address CS Seg. rc. JMP 5000H: 2000H off add IP	10H * CS + IP
12. Inter segment Indirect	JMP [2000H] →	This will jump is ~~JMP~~ to an address in other segment at eff. address 2000H in DS and it points to a memory block with 4 bytes. IP(UP), IP(LP) } 16b upper CS(UP), CS(LP) 16 bit

Instruction set of 8086

i) Data transfer instructions
ii) Arithmetic instructions
iii) Logical instructions
iv) Branching instructions
v) Machine control / Processor control
vi) Flag manipulation instructions
vii) Shift & rotate instructions
viii) String instructions
ix) Loop instructions it can come under branching

i) Flag manipulation instructions.

- CLC - Clear carry
- CMC - Complemen Carry
- STC - Set Carry
- CLD - Clear direction flag
- STD - Set ,,
- CLI - Clear interrupt flag
- STI - Set ,,

ii) Machine control

- WAIT - WAIT is enable TEST is checked.
- HLT -
- LOCK -
- ESC -
- NOP - No operation, to give delay.

• WAIT : WAIT when executed hold operation of processor with current status till logic level on $\overline{TEST}$ Pin goes low.

• HLT : Processor will enter into HLT state & 2 ways remove from HLT state.

1. interrupt
2. Reset.

• LOCK : LOCK prefix appears with another instructions when executed, best access is not allowed for another master till low prefix instruction is completely executed.

• ESC : when executed, free the bus for an external master.

• NOP : Processor performs nothing, for 4 clk cycles other than incrementing the IP.

iii) Logical instructions

AND

→ Result will be in the destination (except for TEST)
→ Atleast one of the operand must be register or memory address.
→ Both operands cannot be memory ~~@~~locations or immediate data. → ~~X~~ AND [5000H] [2000]

d s

• AND AND AX, 0008H
AND AX, BX
AND [5000H], DX

• OR

• NOT NOT AX
NOT [5000H]

• XOR

• TEST TEST AX, BX TEST is like compare instn
It will perform bitwise AND & change the flags. but not save it in destinat

TUTORIAL

H/W Contents of diff. registers are given. Form effective address for diff. address modes. Let the offset be 5000 H.

[AX] - 1000 H [BX] = 2000 H [SI] = 3000 H
[DI] - 4000 H [BP] = 5000 H [SP] = 6000 H
[CS] - 0000 H [DS] = 1000 H [SS] = 2000 H.
[IP] = 7000 H

(i) MOV AX, [5000H] — Direct

EA = 10H * DS + 5000H = 00015000
= 15000 H

(ii) MOV AX, [BX] — Indirect register

EA = 10 H * DS + BX
= 00012000 = 12000 H.

(iii) MOV AX, 5000 [BX] — Relative

EA = 10H * DS + BX + 5000
= 00012000 + 5000 = 17000 H

(iv) MOV AX, [BX], [SI] — based indexed

EA = 10H * DS + BX + SI
= 00012000 + 3000 = 15000 H

(v) MOV AX, 5000 [BX] [SI] — Relative based indexed

EA = 10H * DS + BX + SI + 5000
= 00012000 + 3000 + 5000
= 15000 + 5000
=

iv) Shift & Rotate instructions

27/09 Friday

- SHL/SAL
- SHR
- SAR
- ROR
- ROL
- RCR
- RCL

• SHL/SAL : Shift logical/Arithmetic left

operand	1010	1100	1010	0101
shl result(1)	0101	1001	0100	1010 ← inserted
shl result(2)	0101	0010	1001	0100 ← inserted.

* Both 3.

• SHR : Shift logical right

operand	1010	1100	1010	0101 → CF
shr(1)	0101 (inserted)	0110	0101	0010 → 1 / 0
shr(2)	00100 (inserted)	1011	0010	1001

Count

MSB=0 • SAR : Shift Arithmetic right

operand	1010	1100	1010	0101 → 1 / 0
MSB=1 sar(1) :	1101 (inserted)	0110	0101	0010
sar(2) :	1110	1011	0010	1001

v) String Manipulation

- REP
- MOVSB / MOVSW
- CMPS
- SCAS
- LODS
- STOS

- **REP** – prefix – Repeat

If REP will come as prefix of any statement / string instruction, it will be repeat until CX = 0 → counter. CX is automatically

- **MOVSB** – Move string byte
  MOVSW – Move string words.

Move string of bytes or words stored in memory locations to another locations.
* In each move, CX is decrementing.

- **CMPS** – Compare string Byte to word Byte:

comparing whether the strings are same or not.

- **SCAS** – Scan string Byte or string word:

Searching a string

- **LODS** : Load String Byte or String word

• MOVSB : Mov can not give directly address to DS, ES

```
MOV AX, 5000H      → direct address
MOV DS, AX         (Source)
MOV AX, 6000H
MOV ES, AX         (destination)
MOV CX, OFFH       Move length of string to CX
MOV SI, 1000H
MOV DI, 2000H
CLD                Clear Direction flag (Auto increment mode)
REP MOVSB          CX = FF → FD → FC → FB . . . . . 0
```

• CMPS

```
MOV AX, SEG1
MOV DS, AX
MOV AX, SEG2
MOV ES, AX
MOV SI, OFFSET STRING1
MOV DI, OFFSET STRING2
MOV CX, 010H
CLD
REPE CMPSW
```

SI & DI will be updated & CX is decrementing they are ↳ continue until Non equal

• SCAS

```
MOV AX, SEG1
MOV ES, AX
MOV DI, OFFSET     → Starting point
MOV CX, 010H
MOV AX, WORD
CLD
REPNE SCASW        It is stop when equal
```

vi) Looping Instructions

- LOOP
- JCXZ
- LOOPZ / LOOPE → zero flag → equal
- LOOPNZ / LOOPNE

Set of statements that will repeat until CX=0

```
MOV CX,0005
MOV BX,0FFFFH
Label: MOV AX,CODE1
       OR  BX,AX
       AND DX,AX
       LOOP label
```

Loop works until CX=0

- JCXZ - Jump if CX=0
  - * ~~Conditional~~

It depends on cx (general purpose register)

01/10/19

vii) Data copy / Transfer Instructions

- MOV - Transfer data from one register to another

Eg: Load DS with 5000H

MOV DS,5000H ✗

- Source & destinations cannot be memory locations

- PUSH: Push to stack

PUSH AX

↳ 16 bit

1st PUSH higher byte into AX

PUSH lower byte into AL

• POP : Pop from stack

1st lower byte into AL
and higher byte into AX

• XLAT : It is used to byte conversion /translation.
• byte to a code in a table.

• XCHG : Exchange the contents

• LEA : Load effective Address.

• LDS/LES :

LDS BX, 5000 H.

Load the content in the 5000 ~~is loaded~~ into BX and also load into DS.

~~LDS~~ LES BX, 5000H

Load the content into BX & ES.

• IN : Input the port

Used for reading an i/p port

• OUT : output to port

used for write to an o/p port.

• LAHF : Load AH from Lower Byte of flag

• SAHF : Store AH to lower Byte of flag

• PUSHF : Push flags to stack

• POPF : Pop flags from stack.

viii) Arithmetic Instructions

- ADD : Add
- ADC : Add with carry
- SUB : Subtract
- SBB : Subtract with borrow.
- ~~CMP : Compare~~
- INC : Increment
- DEC : Decrement
- Cmp : Compare

  Destination – Source.

- when both operands equal = Zero flag set.
  If source > destination = Carry flag set
  If source < destination = Carry flag set.

- AAA : ASCII Adjust After Addition
- AAS :
- AAM :
- AAD :
- DAA : Decimal Adjust Accumulator
- DAS :
- NEG : Negate.
- MUL : Unsigned multiplication byte / Word
- IMUL : Signed multiplication.
- DIV : Unsigned division.
- IDIV : Signed division
- CBW : Convert signed byte to word.
- CWD : Convert signed word to double word.

ix) Branching Instructions

Unconditional	Conditional
- CALL	- JNC
- RET	- JC
- INT N	- JP
- INTO	- JG
- IRET	
- LOOP	

Unconditional

- CALL
  - ↳ NEAR CALL (IP) — only need to push same location
  - ↳ FAR CALL (CS, IP) of previous
    - ↳ push both CS & IP.

- RET - Return from subroutine
- INT N - Interrupt followed by interrupt no.

8086 PROGRAMS

① Program for addition of 2 numbers

Assembler directive

```
ASSUME CS:CODE, DS:DATA

DATA SEGMENT
OPR1 DW 1234H
OPR2 DW 0002H
RESULT DW 01 DUP(?)      ; Not sure about the value
RESULT DW ?
DATA ENDS

CODE SEGMENT
START: MOV AX, DATA
       MOV DS, AX
       MOV AX, OPR1
       MOV BX, OPR2
       CLC
       ADD AX, BX
       MOV DI, OFFSET RESULT
       MOV [DI], AX
DOS    MOV AH, 4CH      ; terminate
Interrupt
       INT 21H          ; go immed. & refer the value of AH
CODE ENDS (1st Comd)
END START
```

currently point to RESULT

Set DH value

content of DI = address of RESULT

Define Byte

Variable / Array declaration

Use of DB, DW, DOP

DB → Define Word

4/10

Syntax : Varname type value

Variable declaration

```
eg:  A          DB   9
     MESSAGE    DB   'HELLOWORLD'
     VAR        DW   1122H
     STR1       DB   "HELLOWORLD","$"
     STR2       DB   10,13,"HELLOWORLD","$"
```

it assign 1 byte for each value

16 bit value

indicate end of the string

New line / code of carriage return

Array declaration

		A[0] A[1] A[2] A[3]
A	DB	01h, 02h, 03h, 04h
B	DW	1111h, 2222h

(Static declaration we already declared)

DUP: duplicate

~~DB(3)DUP~~ → parameter

DB 3 DUP(7) ; X DUP 7,7,7

It makes 3 copies of 7.

declare a array with Empty value / Array

RES DW ?

VAR DB 10 DUP(?)

ASSEMBLER DIRECTIVES

DB	Define a bytes - 8 bit data
DW	Define a word - 16 bit
ASSUME	ASSUME CS:CODE, DS:DATA
END	
ENDP	End of Procedure
ENDS	End of segment
EQU	LABEL EQU 0500 H
OFFSET	MOV SI, OFFSET LIST
ORG	ORG 2000H
PROC	
PTR	BYTE PTR / ~~BYTE~~ WORD PTR
SEGMENT	
SEG	MOV AX, SEG ARRAY MOV DS, AX

to give the value of Array

② Write a pgm to move a string of data words from offset 2000H to offset 3000H, the length of string is 0FH.

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
    SOURCESTRT EQU 2000H
    DESSTRT    EQU 3000H
    COUNT      EQU 0FH
DATA ENDS
CODE SEGMENT
START : MOV AX, DATA
        MOV DS, AX
        MOV ES, AX
        MOV SI, SOURCESTRT
        MOV DI, DESTSTRT
        MOV CX, COUNT
        CLD
(CX=0) REP MOVSW
        MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

(3) Program to find out no. of even and odd

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
LIST DW 2357H,0A579H,0C322H,0C91EH,0C000H,0957H
COUNT EQU 006H
DATA ENDS
CODE SEGMENT
START: XOR BX,BX
       XOR DX,DX
       MOV AX,DATA
       MOV DS,AX
       MOV CL,COUNT
       MOV SI,OFFSET LIST
AGAIN: MOV AX,[SI]
       ROR AX,01
       JC ODD
       INC BX
       JMP NEXT
ODD:   INC DX
NEXT:  ADD SI,02
       DEC CL
       JNZ AGAIN
       MOV AH,4CH
       INT 21H
CODE ENDS
END START
```

Array of 6 elts

Values

DB or DW

0957H

150

→ 0, XOR of same values

step for clear the contents

→ counter for even

→ counter for odd

statically already exist

given no. rotate to right if the value no carry is 1 → odd, 0 → even.

if carry flag is set

(1 time rotate)

(it is word)

Use of subroutines, stacks and paumeter passing

Waile an ALP to find factorial of a given nu

ADDRESS	OPCODE	LABEL	MNEMONICS	COMMENTS
1000	BE 00 12		MOV SI, 1200H	
1003	8B 04		MOV AX, [SI]	
1005	50		PUSH AX	
1006	E8 05 10		CALL FACT 100EH	Sub routine call near Hall current add will be
1009	58		POP AX	
100A	46		ING SI	
100B	89 04		MOV [SI], AX	
100D	F4		HLT	
			PROCEDURE FACT NEAR	
100E	5B		POP BX	
100F	58		POP AX	
1010	8B C8		MOV CX, AX	
1012	49		DEC CX	
1013	F6 E1	L1:	MUL CL	CL*AL result in AX
1015	E2 FC		LOOP L1	
1017	50		PUSH AX	
1018	53		PUSH BX	
1019	C3		RET	Top (Stack)

Use of interrupt programming, macros

Write a masm pgm to check whether a given string is palindrome or not.

```
ASSUME CS:CODE,DS:DATA
DISP MACRO MESG
MOV AH,09H
LEA DX,MESG
INT 21H
ENDM
DATA SEGMENT
MSG DB 10,13,"ENTER THE STRING:$"
MSG1 DB 10,13,"STRING ENTERED IS:$"
MSG2 DB 10,13,"REVERSED STRING IS:$"
YES DB 10,13,"IS A PALINDROME :$"
NO DB 10,13,"NOT A PALINDROME:$"
STR DB 50,?,50 DUP("$")
REV DB 50,?,50 DUP("$")
DOL DB "$"
DATA ENDS
CODE SEGMENT
START: MOV AX,DATA ;initialize DS
       MOV DS,AX
       MOV ES,AX
       DISP MSG
       MOV AH,0AH ;Read string to STR
       LEA DX,STR
       INT 21H
       DISP MSG1
       DISP STR+2
       DISP MSG2
       XOR CX,CX
```

```
        LEA DI,REV+2 ; Find the length of
                                    string
        LEA SI,STR
        MOV CL,[SI+1]
        INC CL
        ADD SI,CX ;Reverse string and sto
                                    to Re
UP : MOV AL,[SI]
     MOV [DI],AL
     INC DI
     DEC SI
     LOOP UP
     DISP REV+2
     XOR CH,CH
     MOV CL,STR+1
     DEC CL
     LEA DI,REV+2 ;check for palindrom
     LEA SI,STR+2
     REPE CMPSB
     JNZ NOOT
     DISP YES
     JMP LAST
     NOOT:
NOOT: DISP NO
LAST: MOV AH,4CH ; Exit to DOS Prompt
      INT 21H

CODE ENDS
END START
```

MACROS

DEFINING A MACRO

(name of macro)
```
DISPLAY MACRO
    MOV AX,SEG MSG1 (Address of MSG1 kidakkunna data segment)
    MOV DS,AX
    MOV DX,OFFSET MSG1
    MOV AH,09H ;display string [before write comment put ;]
    INT 21H
ENDM
```

PASSING PARAMETERS TO MACRO

```
DISPLAY MACRO MSG1 ← parameter
        MOV AX,SEG MSG1
        MOV DS,AX
        MOV DX,OFFSET MSG1
        MOV AH,09H
        INT 21H
ENDM
    .
    .
    .
DISPLAY MSG1 (Call a macro) [macroname]
    .
    .
    .
DISPLAY MSG2
    .
    .
    .
MSG1 DB,0AH,0DH,"pgm terminated"
MSG2 DB,0AH,0DH,"Retry"
```

(Replace Display msg1 by above 5 lines in the display macro will be displayed)

Disadvantage: program length will be increased.

PARAMETER PASSING IN SUBROUTINE/PROCEDURE

1) Using global declared variable NUM DB 05H GLO
2) Using registers of CPU.
3) Using memory locations.
4) Using stack, eg: factorial pgm
5) Using PUBLIC, EXTERN

Program to print palindrome.

```
ASSUME CS:CODE,DS:DATA
DISP MACRO MSG
MOV AH,09H
LEA DX,MSG
INT 21H.
ENDM
DATA SEGMENT
S1 DB 10,13,"ENTER A STRING:$"        (carriage return)
S2 DB 10,13,"STRING PALINDROME:$"
S3 DB "$"
S4 DB 10,13,"STRING IS NOT PALINDROME"
STR DB 50H DUP(0)
REV DB 50H DUP(0)
CNT DW 00H
DATA ENDS
CODE SEGMENT
START: MOV AX,DATA
MOV DS,AX
MOV ES,AX
XOR AX,AX
XOR CX,CX
LEA SI,STR                            (effective Address)
DISP S1
L1 MOV AH,01H                         (character read from std keyboard)
```

```
INT 21H
MOV[SI],AL
INC SI
INC CX
CMP AL,0DH;
JNE L1
DEC CX
MOV CNT,CX
SUB S1,1
MOV AL[S3]
MOV[DI],AL
INC DI
DEC SI
LOOP L8
MOV AL,[S3]
MOV
LEA DI,REV
L2 :MOV AL,[SI]
MOV [DI],AL
INC DI
DEC SI
LOOP L2
MOV AL,[S3]
MOV [DI];AL
MOV Cx ,CNT
XOR SI,SI
XOR DI,DI
LEA SI ,STR
LEA DI ,REV
REPE CMPSB
CMP CX,0H
JNZ L4
DISP S2
JMP L5
L4 :DISP S4
L5 : MOV AH,4CH
CODE ENDS
END START
```