

MODULE 2

8086 INSTRUCTION SET & ASSEMBLER DIRECTIVES

ADDRESSING MODES OF 8086

- Indicate the way of locating data or operands
- Describe the type of operands
- The different ways in which a source operand is denoted in an instruction is known as addressing modes.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

1

Categorization of instructions based on Flow of instruction execution

- **Sequential control transfer instructions**
 - Transfer control to next instruction immediately after it
eg: Arithmetic , logical, data transfer, processor control instructions
- **Control transfer instructions**
 - Transfer control to some predefined address/ address specified in the instruction
eg: INT(Interrupt) , CALL, RET (Return from CALL), JUMP

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

2

ADDRESSING MODES FOR SEQUENTIAL CONTROL TRANSFER INSTRUCTIONS:

1. **Immediate** : Immediate data is a part of instruction

MOV AX , 0005H



Immediate data(8 bit or 16 bit size)

2. **Direct** : 16 bit memory address(**offset/displacement**) is directly specified in the instruction.

MOV AX , [5000H]

Here data resides in a memory location in the data segment

Effective address= offset address + segment address (content of DS)

10H*DS+5000H

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

3

- Example:

Given DS=1000H

Shifting a number 4 times is equivalent to multiplying it by 16D or 10H

DS:OFFSET ⇔ 1000H: 5000H

10H* DS ⇔ 10000

Offset ⇔ + 5000

15000H - Effective address

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

4

3. **Register** : Data is stored in register. All the registers **except IP** can be used.

Eg: MOV BX, AX

4. **Register Indirect**: offset of data is in either BX or SI or DI registers. The default segment is either DS or ES

Eg : MOV AX, [BX]

Here data is in DS whose offset address is in BX

Effective address= 10H * DS+[BX]

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

5

• Example :

Given DS=1000H and BX=2000H

$$\begin{array}{rcl}
 \text{DS:BX} & \Leftrightarrow & 1000\text{H}:2000\text{H} \\
 10\text{H} \cdot \text{DS} & \Leftrightarrow & 10000 \\
 [\text{BX}] & \Leftrightarrow & + 2000 \\
 & & \hline
 & & 12000\text{H} - \text{Effective address}
 \end{array}$$

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

6

5. Indexed : offset of the operand is stored in one of the index registers.

For SI (source index), default segment is DS

For DI(destination index), default segment is ES

Eg: MOV AX, [SI]

Effective address= $10H * DS + [SI]$

6. Register relative : Data is available by adding the displacement with the content of any one of the register BX, BP, SI and DI

Default segment is DS or ES

Eg: MOV AX, 50H [BX]

Effective address= $10H * DS + 50H + [BX]$

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

7

• Example:

MOV AX, 5000 [BX]

Given DS=1000H and BX=2000H

DS: [5000 + BX]

$10H * DS \Leftrightarrow 10000$

Offset $\Leftrightarrow + 5000$

[BX] $\Leftrightarrow + 2000$

17000H - Effective address

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

8

7. Based Indexed : effective address is sum of base register (BX or BP) and Index register (SI or DI)

Default segment register may be ES or DS

Eg: MOV AX , [BX] [SI]

Effective address= $10H \cdot DS + [BX] + [SI]$

Example:

Given DS=1000H, BX=2000H and SI=3000H

$$\begin{array}{rcl}
 \text{DS:} & [BX + SI] & \\
 10H \cdot DS & \Leftrightarrow & 10000 \\
 [BX] & \Leftrightarrow & + 2000 \\
 [SI] & \Leftrightarrow & + 3000 \\
 \hline
 & & 15000H - \text{Effective address}
 \end{array}$$

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

9

8. Relative Based Indexed : effective address is formed by adding displacement with the sum of content of any of base registers (BX or BP) and any one of the index registers

Eg: MOV AX, 50H [BX] [SI]

Effective address= $10H \cdot DS + 50H + [BX] + [SI]$

Example:

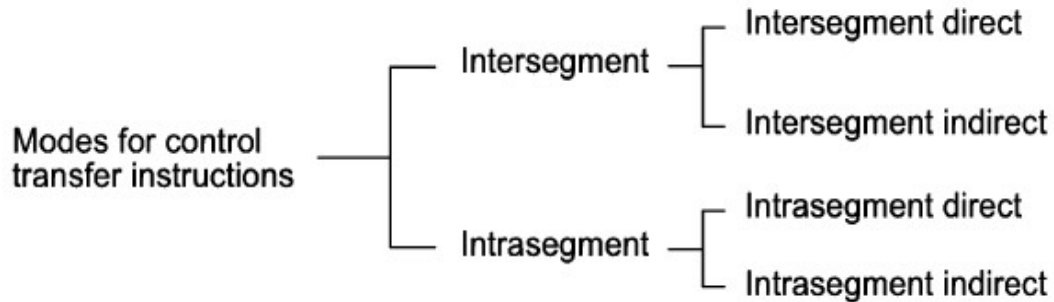
Given DS=1000H, BX=2000H and SI=3000H

$$\begin{array}{rcl}
 & \text{MOV AX, 5000 [BX] [SI]} & \\
 \text{DS:} & [BX + SI + 5000] & \\
 10H \cdot DS & \Leftrightarrow & 10000 \\
 [BX] & \Leftrightarrow & + 2000 \\
 [SI] & \Leftrightarrow & + 3000 \\
 \text{Offset} & \Leftrightarrow & + 5000 \\
 \hline
 & & 1A000 - \text{effective address}
 \end{array}$$

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

10

ADDRESSING MODES FOR CONTROL TRANSFER INSTRUCTIONS:



Intersegment: Destination location is in **different** segment.

Intrasegment: Destination location is in **same** segment.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

11

- **Intersegment Direct:**

- Destination is in different segment
- Provides branching from one code segment to another code segment
- CS and IP of destination address are specified directly in the instruction.

Example:

`JPM 5000H : 2000H;`

Jump to effective address 2000H in segment 5000H.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

12

- **Intersegment Indirect**

- Destination lies in different segment
- Destination location is passed to the instruction indirectly.

Example:

`JMP [2000H];`

Jump to an address in the other segment specified at effective address 2000H in DS

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

13

- **Intrasegment Direct:**

- Destination lies in same segment
- Displacement is computed using the content of the IP

- **Intrasegment Indirect:**

- Destination lies in same segment
- Destination location is passed to the instruction indirectly.
- Branch address is found as the content of a register
- Example

`JMP [BX];` Jump to effective address stored in BX.
`JMP [BX + 5000H]`

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

14

INSTRUCTION SET OF 8086

1. Data copy/ transfer instruction:

- Transfer data from source operand to destination operand
- Eg: store, move , load, exchange, I/O instructions

2. Arithmetic & Logical instructions

3. Branch instructions:

- Transfer control of execution to a specified address
- Eg: jump, interrupt, call, return instruction

4. Loop instructions:

- Implement loop structure
- Eg: LOOP, LOOPNZ, LOOPZ instructions

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

15

5. Machine control instructions:

- Control machine status
- Eg: NOP, HLT, WAIT, LOCK

6. Flag manipulation instructions:

- Affect flag registers
- Eg: CLD, STD, CLI, STI

7. Shift & Rotate instructions:

- For bitwise shifting or rotation in either side

8. String instructions:

- String manipulation operations
- Eg: load, move, scan, compare, store

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

16

1. DATA COPY /TRANSFER INSTRUCTIONS

1. MOV (move)

- Transfer data from one register/memory to another register/memory
- **Source** - general purpose /special purpose register or memory location
- **Destination** – register or memory location
- **Syntax** : **MOV destination, source**
- Eg: MOV AX, BX
- Direct loading of segment register with immediate data is not permitted

MOV DS, 5000H (not permitted)

correct procedure is

MOV AX, 5000H

MOV DS, AX

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

17

2. PUSH (push to stack)

- Push content of specified register on to the stack.
- Store a word (2 bytes) on to the stack

➤ **Syntax: PUSH source**

Eg: PUSH AX

PUSH DS

PUSH [5000H] – content of location 5000H and 5001H in DS are pushed on to the stack.

3. POP (pop from stack)

- Get a word from the stack to the provided location.

➤ **Syntax: POP destination**

Eg: POP AX

POP DS

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

18

4. XCHG (exchange)

- Exchange the contents of source and destination operands
- Exchange of data content of two memory locations is not permitted

Syntax: XCHG destination, source

Eg: XCHG [5000H], AX ;

XCHG BX ; (exchange data between AX and BX)

5. IN (input the port)

- Used for reading an input port
- AL and AX are destinations
- DX is the only register which is allowed to carry the port address
- Eg: IN AL, 0300H ; - read data from port address 0300H and store in AL
IN AX ; - read data from port whose address is in DX and store in AX

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

19

6. OUT (output to the port)

- Used for writing to an output port
- AL and AX are the source operands
- Address of output port may be specified in the instruction or in DX
- Eg: OUT 0300H, AL ; - send data available in AL to the port whose address is 0300H

7. XLAT (translate)

- Finding the code in code conversion problem
- Eg: translate the code of the key pressed to 7-segmented code

8. LEA (Load Effective Address)

- Load offset of an operand in specified register
- Used to save pointer(address) of a value
- Eg: LEA BX, ADR; - offset of label ADR will be transferred to register BX

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

20

9. PUSHF (push Flag to stack)

- Pushes the flag register on to the stack
- First the upper byte then the lower byte will push on to the stack
- SP is decremented by 2 , for each push operation

10. POPF (pop Flag from stack)

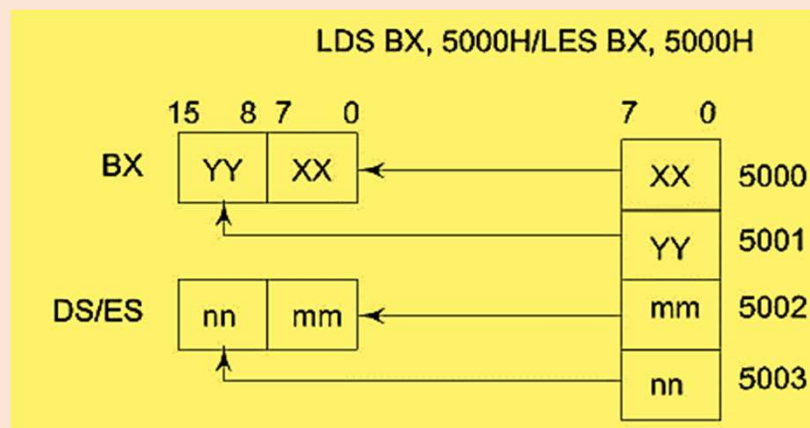
- Used to copy a word at the top of the stack to the flag register.
- SP is incremented by 2 ,for each pop operation

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

21

11. LDS/LES (Load pointer to DS/ES)

- It loads DS or ES register & destination register with the content of memory location(source)



Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

22

2. ARITHMETIC INSTRUCTIONS

1. ADD (add)

- Source and destination may be registers
- Memory to memory addition not possible
- **Syntax: ADD Destination, Source**

➤ **Eg:** **ADD AX, 0100H**
 ADD AX, BX
 ADD AX, [SI]
 ADD AX, [5000H]

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

23

2. ADC (Add with carry)

- Same as ADD instruction but add the carry flag
- **Syntax: Destination ← Source + Destination + CF**

Eg: **ADC 0100H**
 ADC AX, BX
 ADC AX, [SI]
 ADC AX, [5000H]

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

24

3. INC (Increment)

- Increment the content of the register or memory location by 1

➤ **Syntax: INC Destination**

- Eg: INC AX
INC [BX]
INC [5000H]

4. DEC (Decrement)

- Subtract 1 from the content of the specified register or memory location

➤ **Syntax: DEC Destination**

- Eg: DEC AX
DEC [5000H]

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

25

5. SUB (Subtract)

- Subtract source operand from the destination operand
- Result is stored in destination operand

➤ **Syntax: Destination ← Destination – Source**

- Eg: SUB AX, BX
SUB AX, [5000H]
SUB [5000H], 0100

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

26

6. SBB (Subtract with borrow)

➤ **Syntax:** **Destination** ← **Destination – Source – CF**

➤ **Eg:** SBB AX, BX

SBB AX, [5000H]

SBB [5000H], 0100

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

27

7. CMP (Compare)

➤ Compare the source operand with a destination operand

➤ **Source** - register, immediate data or memory location

➤ **Destination** – register or memory location

➤ For comparison, it subtract the source operand from destination operand (**Destination - Source**) but does not store the result.

➤ Both operands equal – Zero flag is set

➤ Source operand > Destination operand – Carry flag is set

➤ Otherwise carry flag is reset

➤ **Eg:** CMP BX, 0100H

CMP BX, CX

CMP BX, [SI]

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

28

8. MUL (Unsigned multiplication)

- Multiplies unsigned byte/word with the content of AL/AX
- unsigned byte/word may be in general purpose register or memory location
- **MSB** of result is stored in **DX** & **LSB** is stored in **AX**
- Eg: MUL CX { (DX)(AX) ← AX * CX }

9. IMUL (Signed multiplication)

- Multiplies signed byte/word in source operand with the content of AL/AX
- Source can be general purpose register , index register or base register
- Eg : IMUL BH

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

29

10. DIV (Unsigned Division)

- Divides unsigned word / double word by 16 bit/8 bit operand
- **Quotient** will be stored in **AL**
- **Reminder** will be stored in **AH**

11. IDIV (Signed Division)

- Same as DIV instruction but signed
- Result will also be signed number

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

30

- AAA – ASCII ADJUST AFTER ADDITION
- AAS – ASCII ADJUST AFTER SUBTRACTION
- AAM – ASCII ADJUST FOR MULTIPLICATION
- AAD – ASCII ADJUST FOR DIVISION
- DAA – DECIMAL ADJUST ACCUMULATOR
- DAS –DECIMAL ADJUST AFTER SUBTRACTION
- NEG – NEGATE (To find 2's complement)

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

31

3. LOGICAL INSTRUCTIONS

1. AND (Logical AND)

- ANDs the source operand to the destination operand
- Result stored in destination operand
- **Source operand** : immediate, register or memory location
- **Destination operand**: register or memory location
- Eg: AND AX, 0008H

If content of AX is 3F0FH, Then

0 0 1 1	1 1 1 1	0 0 0 0	1 1 1 1	= 3F0F H [AX]
↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	AND
0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	= 0008 H
0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	= 0008 H [AX]

The result 0008H will be in AX.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

32

2. OR (Logical OR)

➤ Carries out the OR operation

➤ Eg: OR AX, 0098H

If the content of AX is 3F0FH , then

0 0 1 1	1 1 1 1	0 0 0 0	1 1 1 1	= 3F0F H
↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	OR
0 0 0 0	0 0 0 0	1 0 0 1	1 0 0 0	= 0098 H
0 0 1 1	1 1 1 1	1 0 0 1	1 1 1 1	= 3F9F H

Thus the result 3F9FH will be stored in the AX register.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

33

3. NOT (Logical Invert)

➤ Complement the content of an operand register/ memory location.

➤ Eg: NOT AX

If the content of AX is 200FH, then

AX	=	0 0 1 0	0 0 0 0	0 0 0 0	1 1 1 1
invert		↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓
		1 1 0 1	1 1 1 1	1 1 1 1	0 0 0 0

The result DFF0 will be stored in AX

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

34

4. XOR (Logical Exclusive OR)

- When two **inputs are different** , XOR gives **high output**
- When two **inputs are same** , XOR gives **low output**

➤ **Eg:** XOR AX, 0098H

If the content of AX is 3F0FH, then

AX = 3F0FH =	0 0 1 1	1 1 1 1	0 0 0 0	1 1 1 1
XOR	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓
0098H =	0 0 0 0	0 0 0 0	1 0 0 1	1 0 0 0
AX = Result =	0 0 1 1	1 1 1 1	1 0 0 1	0 1 1 1
= 3F97H				

Result is stored in AX

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

35

5. TEST (Logical Compare instruction)

- Performs bit by bit logical AND operation
- If both the operands are 1, result is set to 1 else 0
- The result is not available for further use
- Affected flags are CF, SF, ZF and PF

Eg: TEST AX, BX

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

36

6. SHL/ SAL (Shift Logical / Arithmetic Left)

- Shift the operand bit by bit to the left and insert 0 in LSB
- The count is either 1 or specified by register CL

BIT POSITIONS	CF	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
OPERAND			1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1
SHL RESULT 1st	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1	0	0
SHL RESULT 2nd	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1	0	0	0

7. SHR (Shift Logical Right)

BIT POSITIONS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CF
OPERAND	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1	
Count = 1	0	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1
Count = 2	0	0	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0

8. SAR (Shift Arithmetic Right)

BIT POSITIONS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CF
OPERAND	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1	
Count = 1	1	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1
Count = 2	1	1	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0

9. ROR (Rotate Right without carry)

- The Least Significant bit is pushed in to the carry flag and simultaneously it is transferred in to the MSB position

BIT POSITIONS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CF
OPERAND	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1	x
Count = 1	1	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1
Count = 2	0	1	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0

➤ 10 . ROL (Rotate Left without carry)

- The Most Significant bit is pushed in to the carry flag as well as LSB position

BIT POSITIONS	CF	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPERAND		1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1
SHL RESULT 1st	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1	1
SHL RESULT 2nd	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1	1	0

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE,
MALAPPURAM

39

11. RCR (Rotate Right through carry)

- Carry flag is pushed in to the MSB of the operand and the LSB is pushed in to carry flag.

BIT POSITIONS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CF
OPERAND	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1	0 (arbitrary)
Count = 1	0	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1

➤ RCL (Rotate Left through carry)

BIT POSITIONS	CF	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPERAND	(arbitrary)	0	1	0	0	1	1	1	0	1	1	0	1	1	0	1	0
Count = 1	1	0	0	1	1	1	0	1	1	0	1	1	0	1	0	1	0

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE,
MALAPPURAM

40

4. FLAG MANIPULATION INSTRUCTIONS

- CLC – CLEAR CARRY
- CLD – CLEAR DIRECTION
- CLI – CLEAR INTERRUPT
- STC – SET CARRY
- STD – SET DIRECTION
- STI – SET INTERRUPT
- CMC – COMPLEMENT CARRY

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

41

5. PROCESSOR CONTROL INSTRUCTIONS

- WAIT – Wait for TEST input pin to go Low
- HLT – Halt the processor
- NOP – No operation
- ESC – Escape to external devices
- Lock – Bus lock instruction

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

42

6. STRING MANIPULATION INSTRUCTIONS

- ❖ A string is a series of data bytes or words available in memory at consecutive locations
- ❖ To refer a string 2 parameters are needed
 1. start/ end address of string
 2. Length of string (stored as a count in CX register)

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

43

1. REP (Repeat)

- Used as a prefix to other instruction
- **Repeat** the given instruction **till CX != 0**
- When CX become zero , execution proceeds to the next instruction in sequence
- **REPE/REPZ** : Repeat operation while Equal/Zero
- **REPNE/REPNZ** : Repeat operation while Not Equal/ Not Zero

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

44

2. MOVSB/MOVSW (Move String Byte/ Move String Word)

- Move a string of byte/word from DS:SI (source) to ES:DI (destination)
- Starting address of source string is $10H * DS + [SI]$
- Starting address of destination string is $10H * ES + [DI]$

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

45

3. CMPS (Compare string)

- Used to compare the strings
- Length of string must be stored in CX register
- If both the byte/word string are equal, Zero flag is set.
- **CMPSB** – Compare String Byte
- **CMPSW** – Compare String Word

4. SCAS (Scan string)

- Used to scan a string
- **SCASB** – scan string byte
- **SCASW** – Scan string word

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

46

5. LODS (Load String)

➤ It loads the AL/AX register by the content of a string pointed to by DS : SI register pair.

➤ **LODSB** – Load string byte

➤ **LODSW** – Load string word

6. STOS (Store string)

➤ It store the AL/AX register contents to a location in the string pointed by ES:DI register pair

➤ **STOSB** – Store String Byte

➤ **STOSW** - Store string word

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

47

ASSEMBLER DIRECTIVES

➤ An Assembler is a program used to convert an Assembly Language Program in to machine code

Assembler directives

➤ Assembler directives are statements that direct the assembler to do a task

➤ It control the organization of the program

➤ Provide necessary information to the assembler to understand ALPs

➤ It consist of 2 type of statements

1. **Instructions** → Translated to the machine code by the Assembler

2. **Directives** → Not translated to the machine code

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

48

I. Data declaration directives

➤ DB, DW, DQ and DT are data declaration directives

1.DB (Define Byte)

➤ Used to declare a byte or 2- byte variable

➤ It reserve a byte or bytes of memory locations in the available memory

Eg: **RANKS DB 01H, 02H, 03H, 04H**

➤ Assembler reserve 4 memory locations for an **array** named **RANKS** and initialize them with the above specified 4 values.

Eg: VALUE DB 50H

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

49

Eg: MESSAGE DB 'GOOD MORNING'

➤ Reserve the number of bytes of memory equal to the number of characters in the string.

➤ Name of the string is MESSAGE and initialize the location by the ASCII values of these characters

2. DW (Define Word)

➤ Used to declare a word type variable (a word = 16 bit)

Eg: **WORDS DW 1234H, 4567H, 78ABH, 045CH**

- Declare an array of 4 words and initialize them with above value.

- Array name is WORDS

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

50

➤ We can use DUP operator with DW directive

Eg: **WDATA DW 5 DUP (6666H)**

- This statement reserve 5 words, ie, 10 bytes of memory for a word label WDATA and initialize all the word location with 6666H

3. DQ (Define Quad word)

➤ Used to reserve 4 words (8 bytes) of memory for a specified variable and Initialize it with specified value

4. DT (Define Ten bytes)

➤ Declare a variable which is 10 bytes in length and initialize with a specified value

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

51

II. ASSUME

➤ This directive is used to name the logical segment

➤ 8086 works directly with 4 physical segment; ie,

Code segment

Data segment

Stack segment

Extra segment

➤ In assembly language , each segment is given a name.

Code segment may be given the name CODE

Data segment may be given the name DATA

➤ **ASSUME** statement is must at starting of each program.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

52

Eg: **ASSUME DS : DATA**

Data items related to the program are available in a logical segment DATA

Eg : **ASSUME CS: CODE**

Machine codes are available in a segment named CODE; and hence the CS register is to be loaded with the address allotted by the operating system for the label CODE

III. END (end of program)

- End of an assembly language program
- It should be the last statement in the file

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

53

IV. ENDP (end of procedure)

- It indicate end of a procedure(subroutines)
- A procedure has a name or label
- Syntax: **procedure_name ENDP**

Eg:

```
PROCEDURE STAR
:
STAR ENDP
```

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

54

V . ENDS (end of segment)

➤ End of logical segment

➤ Syntax: **segment_name ENDS**

```
ASSUME      CS : CODE, DS : DATA
CODE        SEGMENT
            :
CODE        ENDS
```

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

55

VI. SEGMENT

➤ To indicate the start of a logical segment

➤ Syntax: **segment_name SEGMENT**

➤ Segment may be assigned a type like **PUBLIC** or **GLOBAL**

➤ **PUBLIC** ➡ can be used by other modules of the program while linking

➤ **GLOBAL** ➡ can be accessed by any other modules

```
EXE.CODE SEGMENT GLOBAL; Start of Segment named EXE.CODE,
                        ; that can be accessed by any other module.
EXE.CODE ENDS          ; END of EXE.CODE logical segment.
```

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

56

VII. EQU (Equate)

- Used to give a name to some value or to a symbol.
- Each time the assembler finds the name in the program , it will replace the name with the value or symbol you given to that name.
- Eg: **LABEL EQU 0500H** ➔ Assigns the constant 0500H with the label LABEL

➤ Eg: **FACTOR EQU 03H**

You has to write this statement at the starting of your program. Later in the program you can use this as follows

ADD AL, FACTOR

When it codes this instruction, the assembler will code it as

ADD AL, 03H

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

57

VIII. PROC (procedure)

- Used to identify the start of a procedure
- The term **NEAR** or **FAR** is used to specify the type of the procedure
- **NEAR** ➔ Procedure is located with in same segment (ie, with in 64K)
- **FAR** ➔ Procedure is located in different segment
- Eg: **RESULT PROC NEAR**

IX. ORG (origin)

- It changes the starting offset address of the data in the data segment
- Eg: The statement **ORG 2000H** tells the assembler to set the location counter to **2000H**
- ORG directive allows you to set the location counter to a desired value at any point in the program.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

58

X. EXTRN (external) & PUBLIC (public)

- **PUBLIC** directive is used along with **EXTRN** directive
- The directive **EXTRN** informs the assembler that the names, procedures and labels declared after this directive have already been defined in some other assembly language modules
- While in other module , the names, procedures and labels must be declared public using **PUBLIC** directive

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

59

Eg: If one wants to call a procedure **FACTORIAL** appearing in module 1 from module 2 , in module1, it must be declared public using the statement **PUBLIC FACTORIAL**, and in module2 it must be declared external using the statement **EXTRN FACTORIAL**

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

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

60

XI. GROUP (group the related segments)

➤ This directive forms a logical group of segments with similar purpose or type

➤ Eg: PROGRAM GROUP CODE, DATA, STACK

❖ Here CODE, DATA and STACK segment must lie within a 64Kbytes memory segment, that is named as PROGRAM

❖ For the ASSUME statement, one can use the label PROGRAM rather than CODE, DATA and STACK. ie,

ASSUME CS: PROGRAM DS: PROGRAM SS: PROGRAM

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

61

XII. PTR (pointer)

➤ Used to specify the data type byte or word

➤ If the prefix is BYTE, then the particular label, variable or memory operand is an 8-bit quantity.

➤ If word is the prefix, then it is 16-bit quantity

Eg: MOV AL, BYTE PTR [SI] ➡ Moves content of memory location addressed by SI (8 bit) to AL

Eg: MOV BX, WORD PTR [2000H] ➡ Moves 16 bit content of memory location 2000H to BX. ie, [2000H] to BL, [2001H] to BH

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

62

XIII. OFFSET (Offset of a Label)

- When the assembler comes across the OFFSET operator along with a label, It first compute the 16 – bit displacement of the label and replaces the string ' OFFSET LABEL ' by the computed displacement
- This operator is used with arrays , strings, labels and procedures to decide their offsets in their default segments.

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

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

63

Assembly Language Program to add two 16 –bit numbers

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
OPR1 DW 1234H           ; 1st operand
OPR2  DW 0002H           ; 2nd operand
RESULT DW 01 DUP(?)     ; A word of memory reserved for re-
                        ; sult
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA     ; Initialize data segment
      MOV DS, AX         ;
      MOV AX, OPR1        ; Take 1st operand in AX
      MOV BX, OPR2        ; Take 2nd operand in BX
      CLC                ; Clear previous carry if any
      ADD AX, BX          ; Add BX to AX
      MOV DI, OFFSET RESULT ; Take offset of RESULT in DI
      MOV [DI], AX        ; Store the result at memory address in DI
      MOV AH, 4CH         ; Return to DOS prompt
      INT 21H
CODE ENDS                ; CODE segment ends
END START                ; Program ends
```

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

64

- Some data may be required for the program. So DATA segment is needed
- CODE segment contains actual instruction . It is compulsory
- If stack facility is used we need STACK segment
- In the first line of program, ASSUME directive declares that the label CODE refers to the code segment & the label DATA refers to the data segment
- CODE = Logical name of code segment
- DATA = Logical name of data segment
- OPR1 is first operand & OPR2 is second operand
- RESULT DW 01H DUP (?) ➔ It reserve 01H words of memory for storing the result of the program and leaves it undefined due to the directive DUP (?)

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

65

- The label STARTS is the starting point of the execution sequence
- ASSUME directive just inform the assembler that the label CODE is used for code segment and the label DATA is used for data segment. It does not put the address of CODE in code segment register (CS) and address of DATA in data segment register (DS)
- The process of putting the actual segment address value in to the corresponding segment register is called **Segment register initialization**
- CS is automatically initialize by the loader. So we should initialize DS
- Ie, MOV AX, DATA
 MOV DS, AX

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

66

- The instruction **MOV DI, OFFSET RESULT** take offset of RESULT in DI. I.e, it store the offset of the label RESULT into DI register.
- **MOV [DI], AX** ➔ This instruction stores the result available in AX into the address pointed to by DI. I.e, address of the RESULT.
- **MOV AH, 4CH** ➔ 4CH is a function call for return back to DOS prompt, after executing the program.
- **INT 21H** ➔ DOS function calls available under INT 21H instruction.
- In DOS , the hardware like memory, keyboard , CRT display , hard disk can be handled with the help of the instruction INT 21H

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

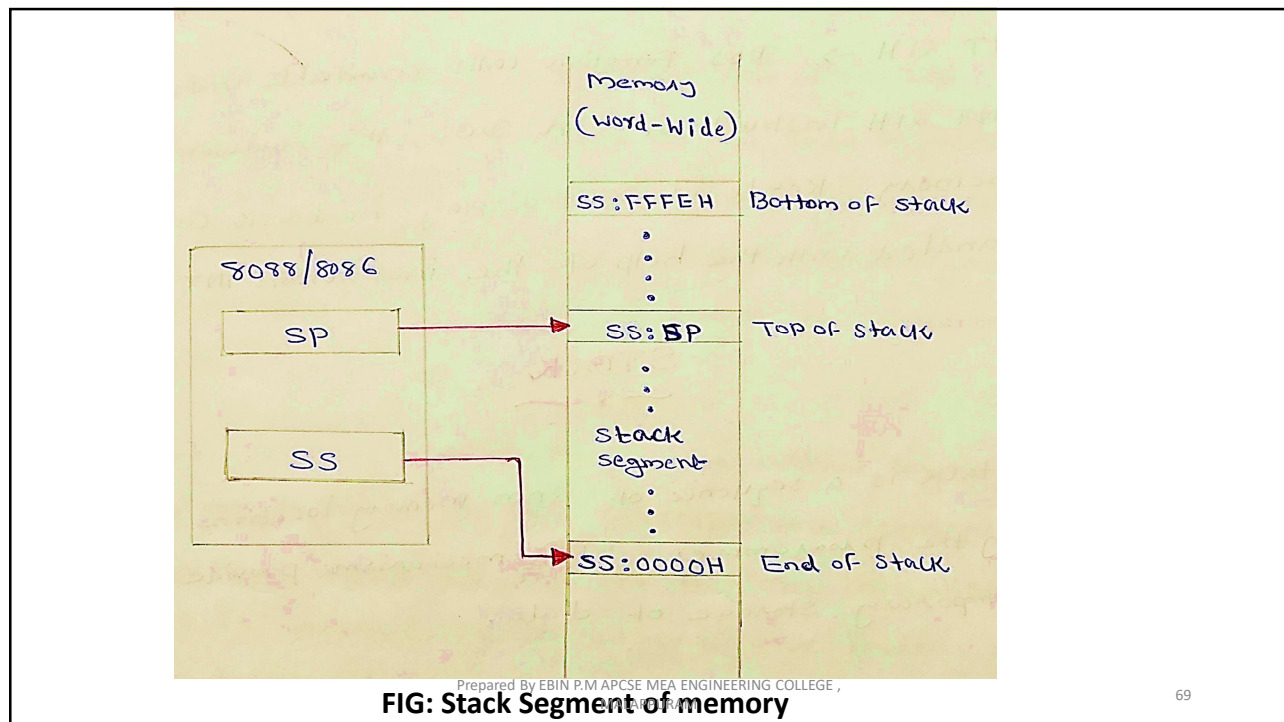
67

STACK

- Stack is a sequence of **RAM memory locations** defined by the programmer. Stack mechanism provide a **temporary storage of data**.
- Stack is a block of memory locations which is **accessed using the SP and SS** registers.
- Stack is a Last In First Out (LIFO) data segment.
- In the case of subroutine and interrupt, the address of re-entry in to the main program (**return address**) may be stored in the stack.
- The process of **storing the data** in the stack is called **Pushing into** the stack.
- The process of transferring the data back from the stack to the CPU register is called **popping off** the stack.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

68

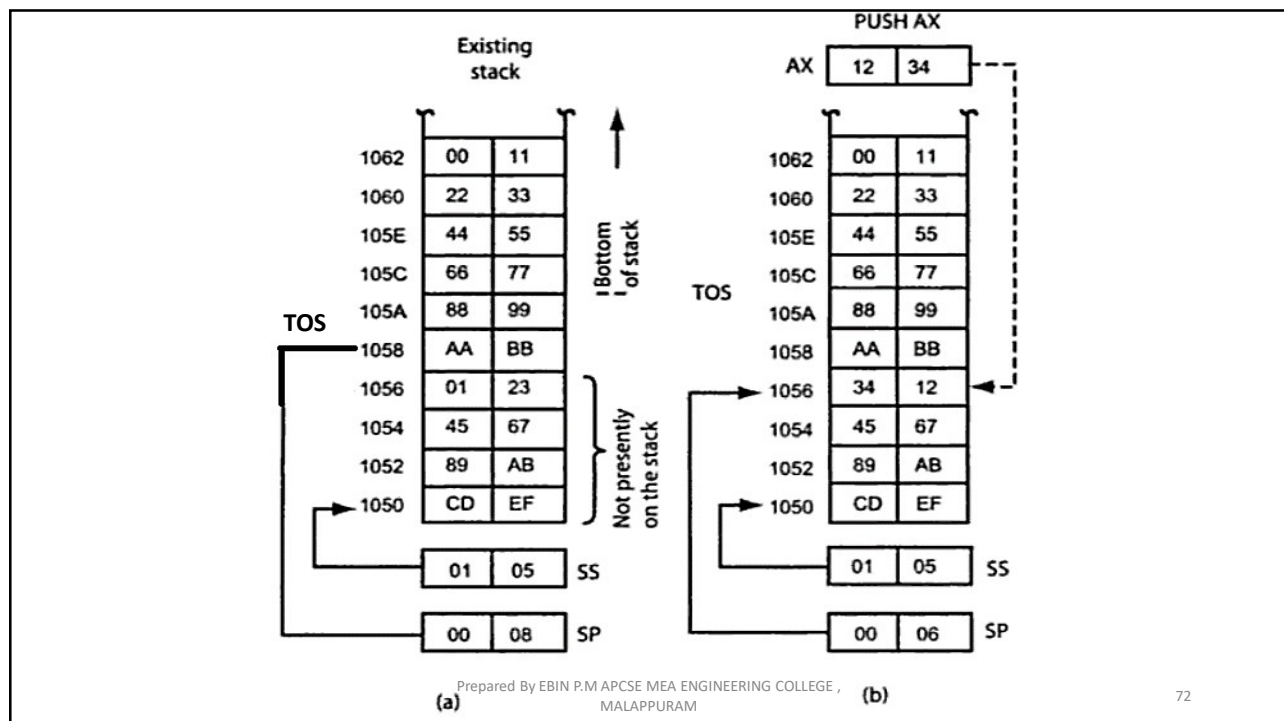


- **Top of stack** ➔ Physical address of the last storage location in the stack to which data were pushed. This address is obtained from the content of SS and SP (SS: SP)
- **Bottom of stack** ➔ At the microcomputer's startup, the value in SP is initialized to FFEH. Combining this value with the current value in SS gives the highest address word location in the stack (SS: FFEH), called bottom of the stack.
- 8086 can push **data** and **address** information on to the stack from its internal register or a storage location in memory.
- Data transferred to and from the stack are word-wide, not byte-wide

- Each time a word is to be **pushed** on to the stack, the **value in SP** is automatically **decremented by two**
- Therefore, **stack grows down** in memory **from the bottom of the stack (SS: FFFEH) towards the end of the stack (SS: 0000H)**
- Each time a word is **popped** from the top of stack, the **value in SP** is automatically **incremented by two**

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

71



Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

72

➤ figure (a), is the state of stack prior to execution of the PUSH AX instruction.

➤ The stack segment register contains 105H. Bottom of stack address is derived from SS and offset FFFE_H

$$\begin{aligned}\text{Bottom of stack address} &= 1050\text{H} + \text{FFFEH} \\ &= \text{1104EH}\end{aligned}$$

➤ SP contains 0008H. Top of stack can be derived from SS and SP

$$\begin{aligned}\text{Top of stack} &= 1050\text{H} + 0008\text{H} \\ &= \text{1058H}\end{aligned}$$

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

73

PUSH OPERATION

➤ AX initially contains the number 1234H.

➤ Execution of push instruction causes the stack pointer to be decremented by two.

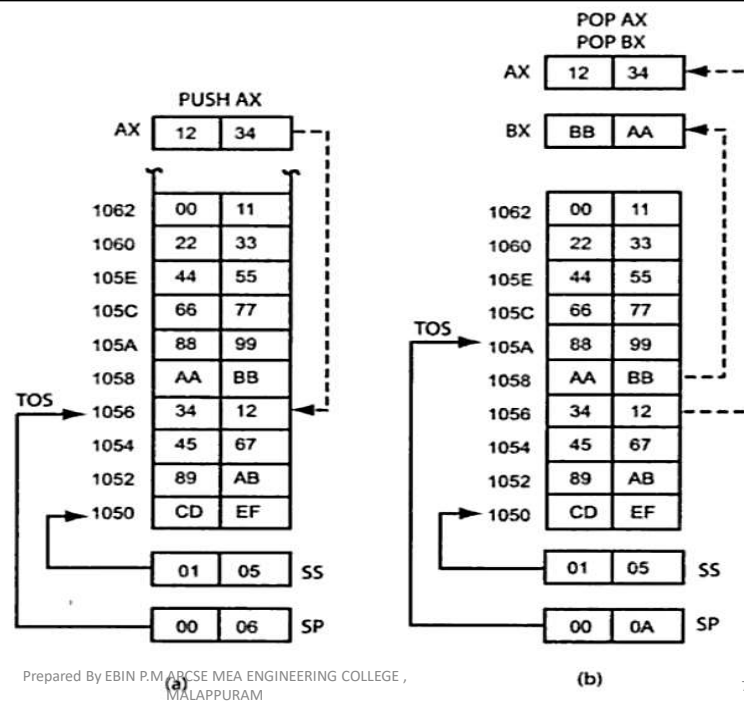
➤ Therefore the next stack access is to the location corresponding to address 1056H. This location is where the value in AX is pushed.

➤ The MSB of AX (ie, 12H) resides in memory address 1057H and LSB of AX (ie, 34H) is held in memory address 1056H

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

74

POP OPERATION



75

- The execution of the first instruction POP AX, causes the 8086 to read the value from the Top of the stack and put it in to AX register as 1234H
- SP is incremented to give 0008H and the other read operation POP BX, causes the value BBAAH to be loaded into the BX register.
- SP is incremented once more and now equals 000AH. Therefore , the new top of stack is at address 105AH

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

76

MACROS

- Suppose a number of instructions are appearing again and again in the main program, the listings become lengthy.
- So a macro definition (i.e. a **Label**) is assigned with the repeatedly appearing string of instructions.
- The process of assigning a label or macro-name to the string is called defining a macro
- The macro name or macro definition is then used through out the main program to refer to that string of instructions.
- A macro with in a macro is called **nested macro**

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

77

- When a group of instructions are to be used several times to perform the same task in a program and they are too small to be written as a procedure , then they can be defined as a macro.
- Macro is a small group of instructions, and is called by its name.
- The process of replacing the macro with the instructions it represent is called **expanding the macro**.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

78

- Macro can be defined anywhere in a program using the directives **MACRO** and **ENDM**

➤ Eg:

```

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

```

- ❖ SEG is used to substitute the segment address of the label MSG

- ❖ If AH contains 09H ,then CRT display is to be used for displaying a message.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

79

- DISPLAY is the macro name , which should be used in the actual program
- The above macro DISPLAY displays the message MSG on the CRT screen
- A macro may also be used to represent statements and directives. The following example shows a macro containing statements.

```

STRING MACRO
    MSG1 DB 0AH, 0DH, " Programs", 0AH, "$"
    MSG2 DB 0AH, 0DH, "Retry, Abort, Fail ", 0AH, "$"
ENDM

```

- Calling a macro means , inserting the statements and instructions represented by the macro directly at the place of the macro name in the program.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

80

PROCEDURE

- When a group of instructions are to be used several times to perform a same task in the program, then we can write them as a separate subprogram called procedure or subroutine.
- Procedure can be call in a program using **CALL** instruction
- Procedures are written and assembled as separate program modules and stored in memory.
- When a procedure is called in the main program , the program control is transferred to procedure and after executing the procedure the program control is transferred back to main program.
- **RET** is used to return the control to main program

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

81

- If the processor encounters a CALL instruction , then it save the content of **PC (Program Counter)** in stack and load the subroutine address in PC.
- The content of the PC that is saved in the stack is the address of the instruction next to CALL in the main program
- When the subroutine address is loaded in PC , the processor start executing the subroutine. The last instruction in the subroutine will be RET instruction.
- When RET is executed, the processor move the top of the stack memory to program counter (PC). Now the execution control is returned to main program.

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

82

Advantages :

1. **Modular programming** → The various task in the program can be developed as separate modules and called in the program.
2. Reduction in the amount of work and program development time
3. Reduces memory requirement for program storage

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

83

PROCEDURE	MACRO
Accessed by CALL and RET mechanism	Accessed with name given to macro when defined
Machine codes for instructions are stored in memory once	Machine codes are generated for instructions in the macro each time it is called
Parameters are passed in registers, memory locations or stack	Parameters are passed as part of statement which calls macro
Executes slower than macro	Executes faster than procedure
Used for more instruction operation	Used for less instruction operation

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

84

PARAMETER PASSING TECHNIQUES

1. Passing parameters through registers

- Here the data is placed in registers by the main program, and these registers are used by procedure.

Solve $S = (A+B) - (C+D)$

```
MOV AX, [1100]
MOV BX, [1102]
MOV CX, [1104]
MOV DX, [1106]
CALL SL-EQ
MOV [1200], AX
HLT
```

SL-EQ

```
ADD AX, BX
ADD CX, DX
SUB AX, CX
RET
```

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

85

2. Passing parameters through memory

```
MOV DI, 2500
MOV SI, 2000
MOV CX, 0AH
```

```
LI: CALL SQ
    MOV [DI], AL
    INC DI
    LOOP LI
    HLT
```

```
SQ
    MOV AL, [SI]
    MOV BL, AL
    MUL BL
    INC SI
    RET
```

Prepared By EBIN P.M APCSE MEA ENGINEERING COLLEGE ,
MALAPPURAM

86