


PROGRAM	COMMENTS
MOV SI,1200H	Initialize memory location for array size
MOV CL,[SI]	Number of comparisons in CL
L4 : MOV SI,1200H	Initialize memory location for array size
MOV DL,[SI]	Get the count in DL
INC SI	Go to next memory location
MOV AL,[SI]	Get the first data in AL
L3 : INC SI	Go to next memory location
MOV BL,[SI]	Get the second data in BL
CMP AL,BL	Compare two data's
JNB L1 	If $AL < BL$ go to L1
DEC SI	Else, Decrement the memory location
MOV [SI],AL	Store the smallest data
MOV AL,BL	Get the next data AL
JMP L2	Jump to L2
L1 : DEC SI	Decrement the memory location
MOV [SI],BL	Store the greatest data in memory location
L2 : INC SI	Go to next memory location
DEC DL	Decrement the count
JNZ L3	Jump to L3, if the count is not reached zero
MOV [SI],AL	Store data in memory location
DEC CL	Decrement the count
JNZ L4	Jump to L4, if the count is not reached zero
HLT	Stop

DESCENDING ORDER

SORTING IN DESCENDING ORDER:

- Load the array count in two registers C_1 and C_2 .
- Get the first two numbers.
- Compare the numbers and exchange if necessary so that the two numbers are in descending order.
- Decrement C_2 .
- Get the third number from the array and repeat the process until C_2 is 0.
- Decrement C_1 and repeat the process until C_1 is 0.

Note: change the coding **JNB L1** into **JB L1** in the **LINE 10**

LARGEST, smallest NUMBER IN AN ARRAY

FINDING LARGEST NUMBER:

- Load the array count in a register C_1 .
- Get the first two numbers.
- Compare the numbers and exchange if the number is small.
- Get the third number from the array and repeat the process until C_1 is 0.

FINDING SMALLEST NUMBER:

- Load the array count in a register C_1 .
- Get the first two numbers.
- Compare the numbers and exchange if the number is large.
- Get the third number from the array and repeat the process until C_1 is 0.

LARGEST NUMBER

PROGRAM	COMMENTS
MOV SI,1200H	Initialize array size
MOV CL,[SI]	Initialize the count
INC SI	Go to next memory location
MOV AL,[SI]	Move the first data in AL
DEC CL	Reduce the count
L2 : INC SI	Move the SI pointer to next data
CMP AL,[SI]	Compare two data's
JNB L1	If $AL > [SI]$ then go to L1 (no swap)
MOV AL,[SI]	Else move the large number to AL
L1 : DEC CL	Decrement the count
JNZ L2	If count is not zero go to L2
MOV DI,1300H	Initialize DI with 1300H
MOV [DI],AL	Else store the biggest number in 1300 location
HLT	Stop

SMALLEST NUMBER

PROGRAM	COMMENTS
MOV SI,1200H	Initialize array size
MOV CL,[SI]	Initialize the count
INC SI	Go to next memory location
MOV AL,[SI]	Move the first data in AL
DEC CL	Reduce the count
L2 : INC SI	Move the SI pointer to next data
CMP AL,[SI]	Compare two data's
JB L1	If $AL < [SI]$ then go to L1 (no swap)
MOV AL,[SI]	Else move the large number to AL
L1 : DEC CL	Decrement the count

Modular Programming

- Generally , industry-programming projects consist of thousands of lines of instructions or operation code.
- The size of the modules are reduced to a humanly comprehensible and manageable level.
- Program is composed from several **smaller modules**. Modules could be developed by separate teams concurrently. OBJ modules (Object modules).
- The .OBJ modules so produced are combined using a LINK program.
- Modular programming techniques **simplify the software development process**

CHARACTERISTICS of module:

1. Each module is independent of other modules.
2. Each module has one input and one output.
3. A module is small in size.
4. Programming a single function per module is a goal

Advantages of Modular Programming:

- It is easy to write, test and debug a module.
- Code can be reused.
- The programmer can divide tasks.
- Re-usable Modules can be re-used within a program

DRAWBACKS:

Modular programming requires extra time and memory

MODULAR PROGRAMMING:

1.LINKING & RELOCATION

2.STACKS

3.Procedures

4.Interrupts & Interrupt Routines

5.Macros

LINKING & RELOCATION

LINKER

- A **linker** is a program used to join together several object files into one large object file.
- The linker produces a link file which contains the binary codes for all the combined modules.

The linker program is invoked using the following options.

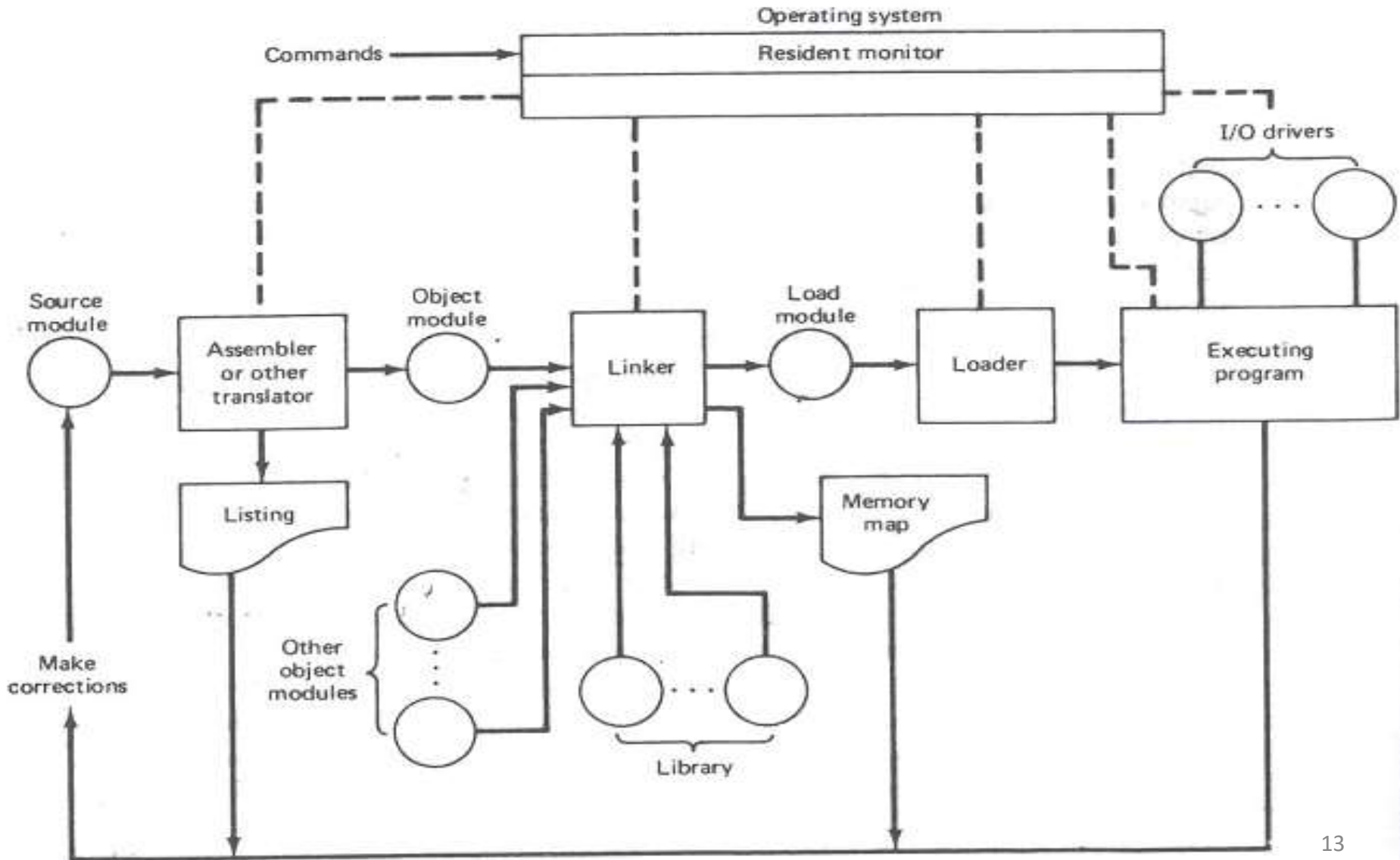
C> LINK

or

C>LINK MS.OBJ

- The **loader** is a part of the operating system and places codes into the memory after reading the '.exe' file
- A program called **locator** *reallocates* the linked file and creates a file for permanent location of codes in a standard format.

Creation and execution of a program



Loader

- >Loader is a utility program which takes object code as input prepares it for execution and loads the executable code into the memory .
- >Loader is actually responsible for initializing the process of execution.

Functions of loaders:

- 1.It allocates the space for program in the memory(**Allocation**)
- 2.It resolves the code between the object modules(**Linking**)
3. some address dependent locations in the program, address constants must be adjusted according to allocated space(**Relocation**)
4. It also places all the machine instructions and data of corresponding programs and subroutines into the memory .(**Loading**)

Relocating loader (BSS Loader)

- When a single subroutine is changed then all the subroutine needs to be reassembled.
- The binary symbolic subroutine (BSS) loader used in IBM 7094 machine is relocating loader.
- In BSS loader there are many procedure segments
- The assembler reads one sourced program and assembles each procedure segment independently

- The output of the relocating loader is the object program
- The assembler takes the source program as input; this source program may call some external routines.

SEGMENT COMBINATION:

ASM-86 assembler regulating the way segments with the same name are concatenated & sometimes they are overlaid.

Form of segment directive:

Segment name **SEGEMENT** **Combine-type**

Possible combine-type are:

- PUBLIC
- COMMON
- STACK
- AT
- MEMORY

Procedures

CALL & RET instruction

- **Procedure** is a part of code that can be called from your program in order to make some **specific task**. Procedures make program more **structural and easier to understand**.

- syntax for procedure declaration:

name PROC

..... ; here goes the code

..... ; of the procedure ...

RET

name ENDP

here **PROC** is the procedure name.(used in top & bottom)

RET - used to return from OS. **CALL**-call a procedure

PROC & ENDP – compiler directives

CALL & RET - instructions

EXAMPLE 1 (call a procedure)

ORG 100h

CALL m1

MOV AX, 2

RET ; return to operating system.

m1 PROC

MOV BX, 5

RET ; return to caller.

m1 ENDP

END

- The above example calls procedure m1, does **MOV BX, 5** & returns to the next instruction after **CALL: MOV AX, 2**.

Example 2 : several ways to pass parameters to procedure

ORG 100h

MOV AL, 1

MOV BL, 2

CALL m2

CALL m2

CALL m2

CALL m2

RET

; return to operating system.

m2 PROC

MUL BL

; AX = AL * BL.

RET

; return to caller.

m2 ENDP

END

value of AL register is update every time the procedure is called.

final result in AX register is 16 (or 10h)

STACK

PUSH & POP instruction

- Stack is an area of memory for keeping **temporary data**.
- STACK is used by **CALL & RET** instructions.
 - PUSH -stores 16 bit value in the stack.**
 - POP -gets 16 bit value from the stack.**
- PUSH and POP instruction are especially useful because we don't have too much registers to operate
 1. Store original value of the register in stack (using PUSH).
 2. Use the register for any purpose.
 3. Restore the original value of the register from stack (using POP).

Example-1 (store value in STACK using PUSH & POP)

ORG 100h

MOV AX, 1234h

PUSH AX ; store value of AX in stack.

MOV AX, 5678h ; modify the AX value.

POP AX ; restore the original value of AX.

RET

END

Example 2: use of the stack is for exchanging the values

```
ORG 100h
MOV AX, 1212h      ; store 1212h in AX.
MOV BX, 3434h      ; store 3434h in BX
PUSH AX            ; store value of AX in stack.
PUSH BX            ; store value of BX in stack.
POP AX             ; set AX to original value of BX.
POP BX             ; set BX to original value of AX.
RET
END
```

push 1212h and then 3434h, on pop we will first get 3434h and only after it 1212h

MACROS

How to pass parameters using macros-6/8 Mark

- Macros are just like procedures, but not really.
- Macros exist **only** until your code is **compiled**
- **After compilation** all macros are **replaced** with **real instructions**
- several macros to make coding easier(**Reduce large & complex programs**)

Example (Macro definition)

```
name MACRO [parameters,...]  
<instructions>  
ENDM
```

Example1 : Macro Definitions

SAVE

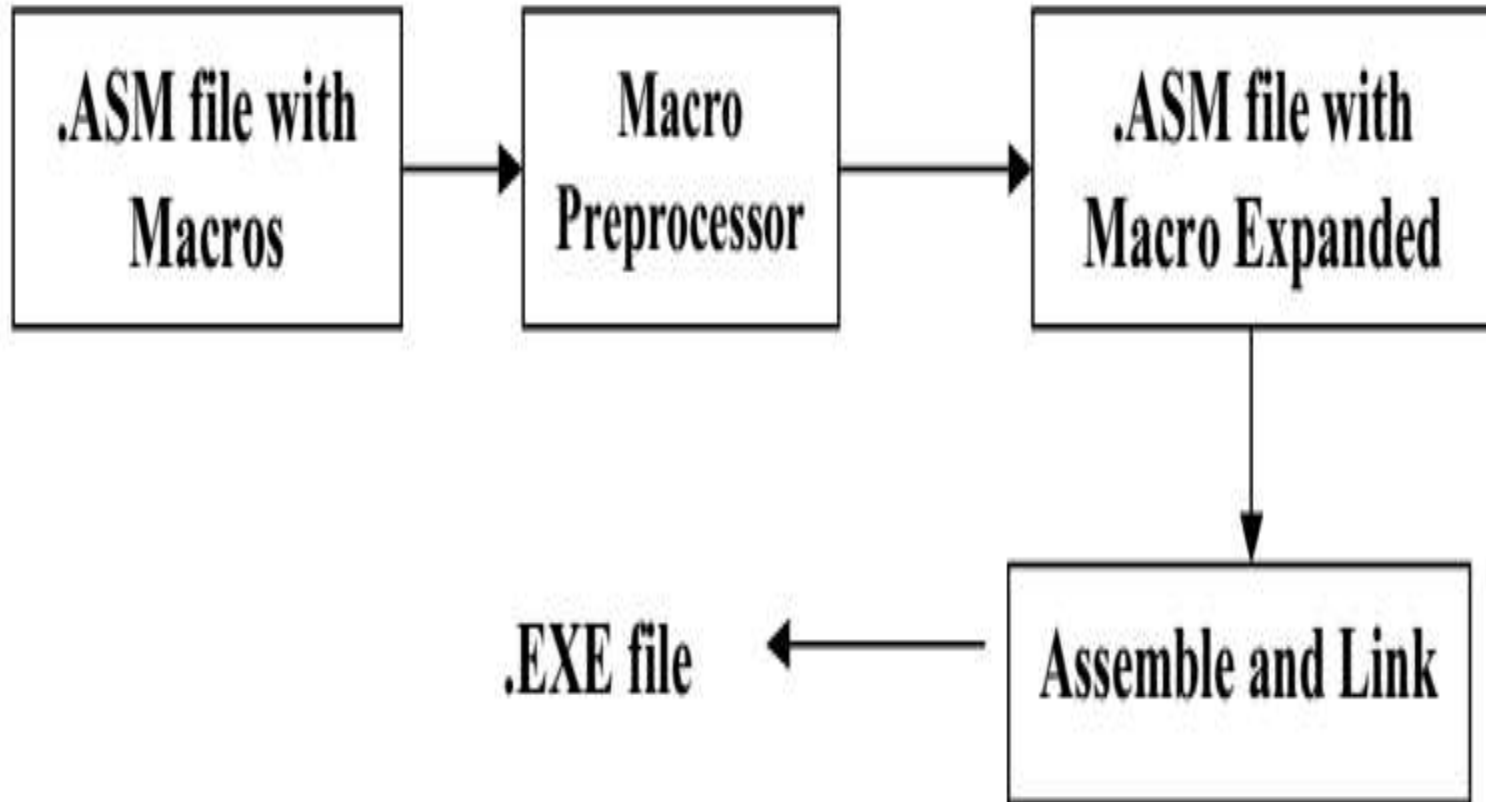
```
MACRO  
PUSH AX  
PUSH BX  
PUSH CX  
ENDM
```

definition of MACRO name **SAVE**

RETREIVE

```
MACRO  
POP CX  
POP BX  
POP AX  
ENDM
```

Another definition of MACRO name **RETREIVE**



MACROS with Parameters

Example:

```
COPY MACRO x, y ; macro named COPY with  
2 parameters{x, y}
```

```
PUSH AX
```

```
MOV AX, x
```

```
MOV y, AX
```

```
POP AX
```

```
ENDM
```