Chapter 4
Part-3
Assembly Program Structure & I/O Instructions

PREPARED BY

AHMED AL MAROUF

LECTURER, DEPT. OF CSE

DAFFODIL INTERNATIONAL UNIVERSITY

Outline:

- Program Structure
  - Memory Models
  - Data Segment
  - Stack Segment
  - Code Segment
  - Putting All Together
- Input-Output Instructions
  - Interrupt instruction
  - INT 21h Operations

Program Structure

- Each machine language programs consist of code, data and stack. Each part occupies a memory segment.
- The same organization is reflected in an assembly language program.
- In assembly program, the code, data and stack are structures as program segments.
- Each segment is translated into a memory segment by the assembler.
- In 8086 microprocessor Bus Interface Unit (BIU) there are four different segments present:
    - Data Segment (DS)
    - Stack Segment (SS)
    - Code Segment (CS) and
    - Extra Segment (ES)

Memory Model (1/2)

- The size of the code and data a program can have is determined by specifying a ***memory model*** using **.MODEL** directive.
- Syntax: **.MODEL memory_model**
- The most frequently used memory models are **SMALL, MEDIUM, COMPACT** and **LARGE**.
- Unless there is a lot of code or data, the appropriate model is SMALL.
- The MODEL directive should come before any segment definition.

Memory Model (2/2)

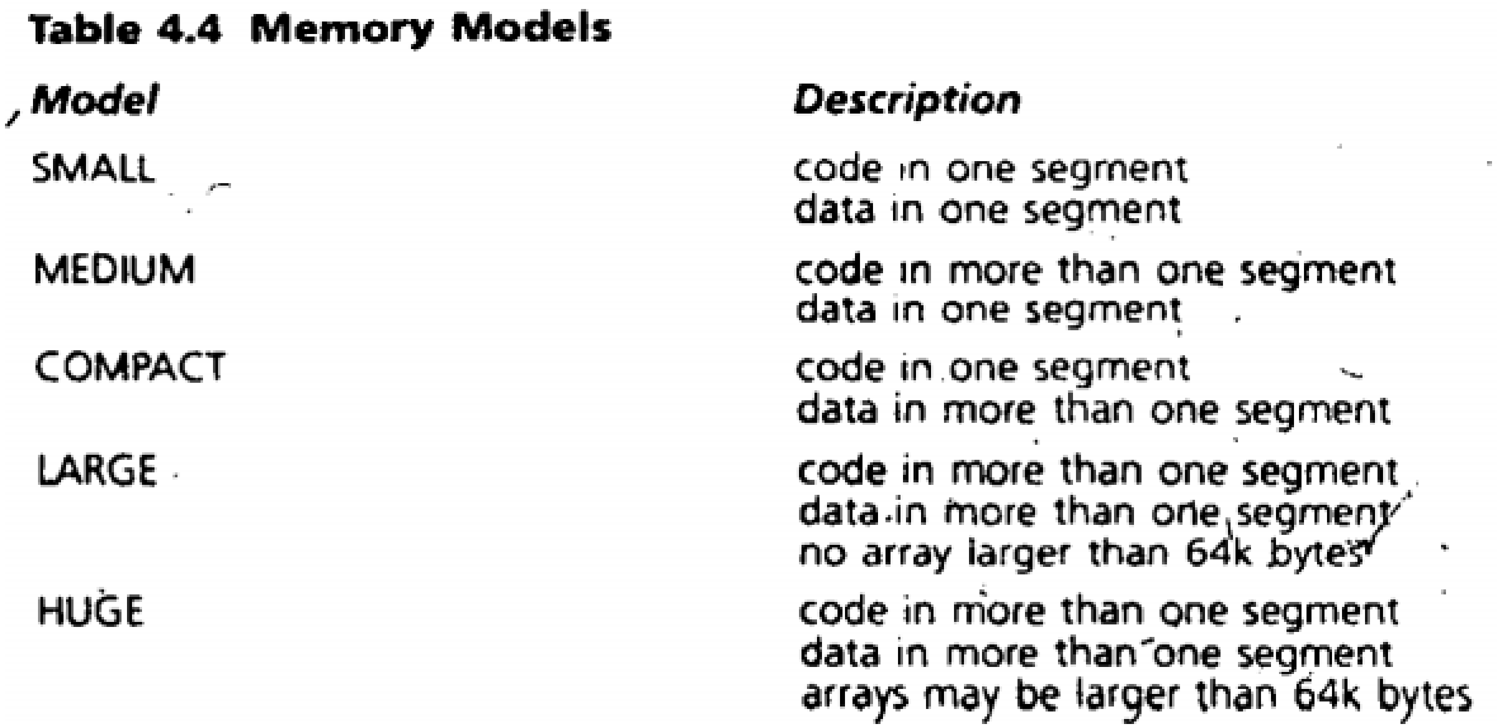


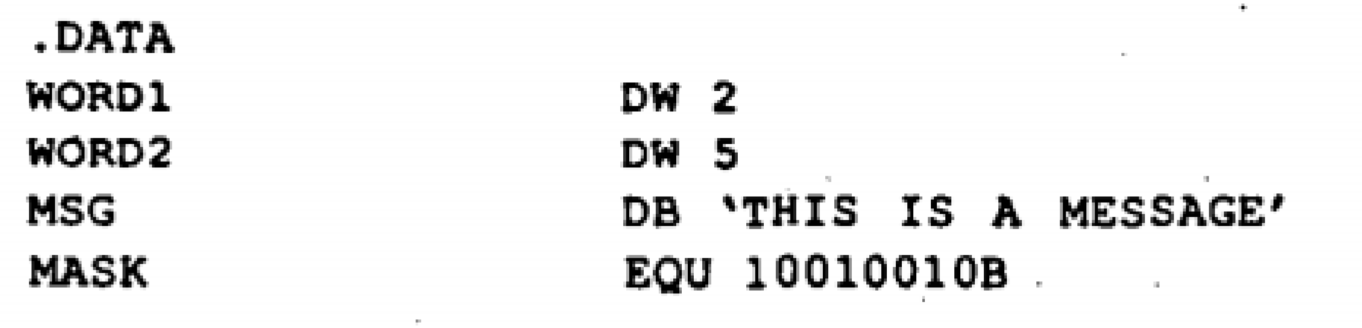
Table 4.4 Memory Models

Model	*Description*
SMALL	code in one segment data in one segment
MEDIUM	code in more than one segment data in one segment
COMPACT	code in one segment data in more than one segment
LARGE	code in more than one segment data in more than one segment no array larger than 64k bytes
HUGE	code in more than one segment data in more than one segment arrays may be larger than 64k bytes

- Unless there is a lot of code or data, the appropriate model is SMALL.
- The MODEL directive should come before any segment definition.

Data Segment

- A program’s ***data segment*** contains all the variable definitions.
- Constant definitions are often made here as well.
- However, they may be placed elsewhere in the program since no memory allocation is involved.
- To declare a data segment, we use the directive **.DATA**, followed by variables and constant declarations.
- Example:



```
.DATA
WORD1                        DW 2
WORD2                        DW 5
MSG                          DB ‘THIS IS A MESSAGE’
MASK                         EQU 10010010B
```

Stack Segment

- The purpose of the ***stack segment*** declaration is to set aside a block of memory to store the stack.
- The stack area should be big enough to contain the stack at its maximum size.
- Syntax: **.STACK size**

where size is an optional number that specifies the stack area size in bytes.

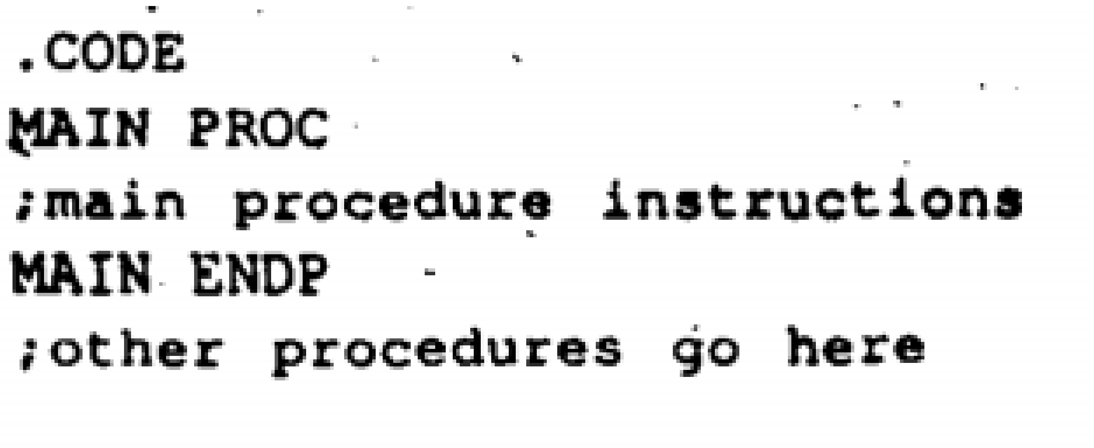
- Example: **.STACK 100h**
- This sets aside 100h bytes for the stack area.
- If size is omitted, 1 KB is set aside for the stack area.

Code Segment

- The ***code segment*** contains a program’s instructions.
- Syntax: **.CODE name**

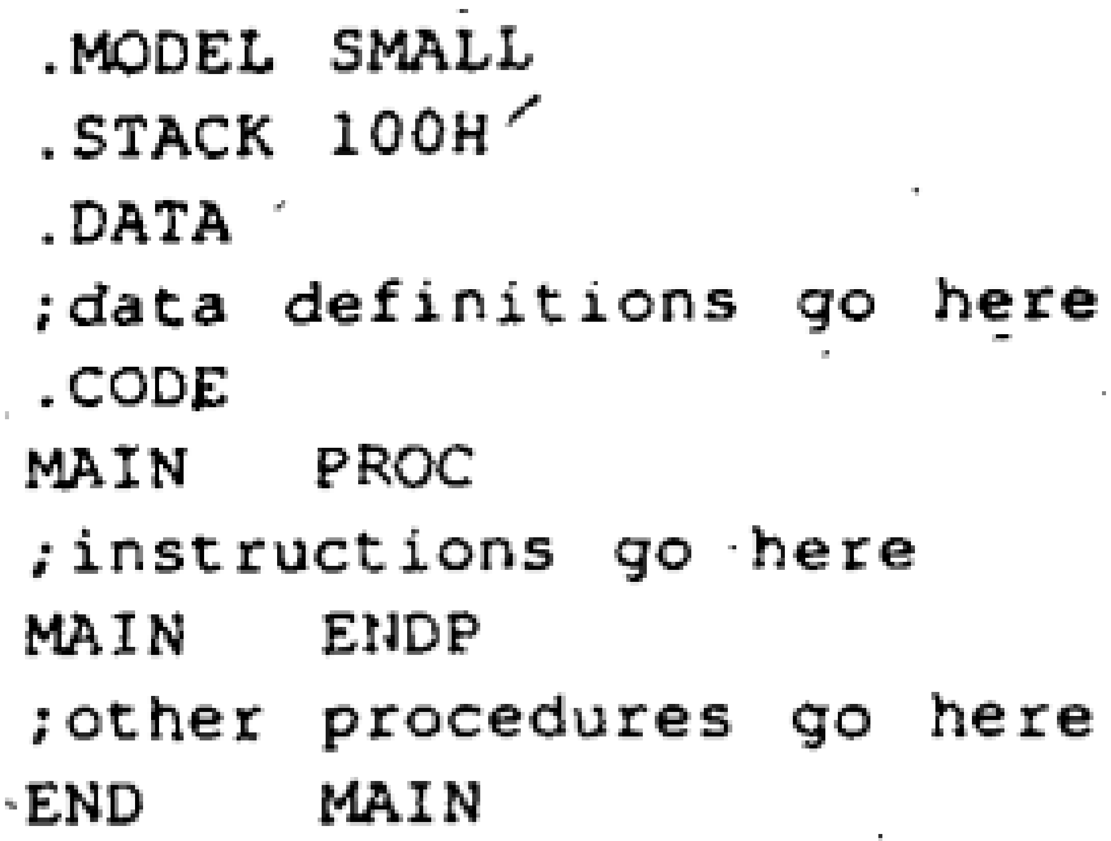
where name is the optional name of the segment.

- Inside a code segment, instructions are organized as procedures.
- Example:



```
.CODE
MAIN PROC
;main procedure instructions
MAIN ENDP
;other procedures go here
```

Putting it Together



```
.MODEL  SMALL
.STACK  100H
.DATA
;data  definitions  go  here
.CODE
MAIN    PROC
;instructions  go  here
MAIN    ENDP
;other  procedures  go  here
END     MAIN
```

Input-Output Instructions using Interrupt

- **INT Instruction:**
- To invoke a DOS or BIOS routine, the **INT** (interrupt) instruction is used.
- Syntax: **INT interrupt_number**

where interrupt_number is a number that specifies a routine.

- For example, INT 16h invokes a BIOS routine that performs keyboard input

INT 21h Operations

- INT 21h is used to invoke a large number of DOS functions such as input and output.

Function Number	Task	Functionality
1	Single-key input	Input: AH = 1 Output: AL = ASCII code if character key is pressed AL = 0, if non-character key is pressed
2	Single-character output	Input: AH = 2 DL = ASCII code of the display character or control character Output: AL = ASCII code of the display character or control character
9	Character string output	Input: DX = offset address of string The string must end with a "$" character

INT 21h functions

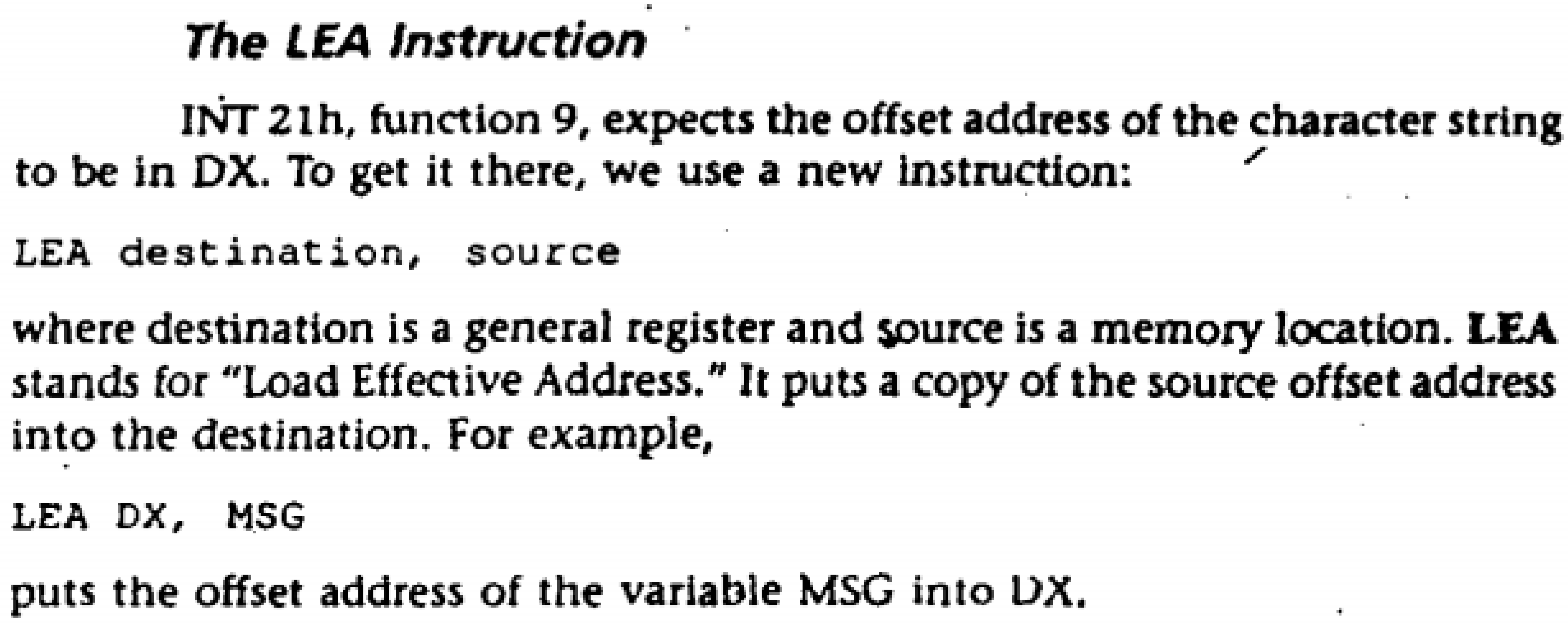
- ***Single-key input:***

```
MOV AH, 1        ;input key function
INT 21h          ;ASCII code in AL
```

- ***Single Character Output:***

```
MOV AH, 2        ;display character function
MOV DL, ‘A’      ;character is ‘A’
INT 21h          ;display character
```

LEA (Load Effective Address)



The LEA Instruction

INT 21h, function 9, expects the offset address of the character string to be in DX. To get it there, we use a new instruction:

```
LEA destination, source
```

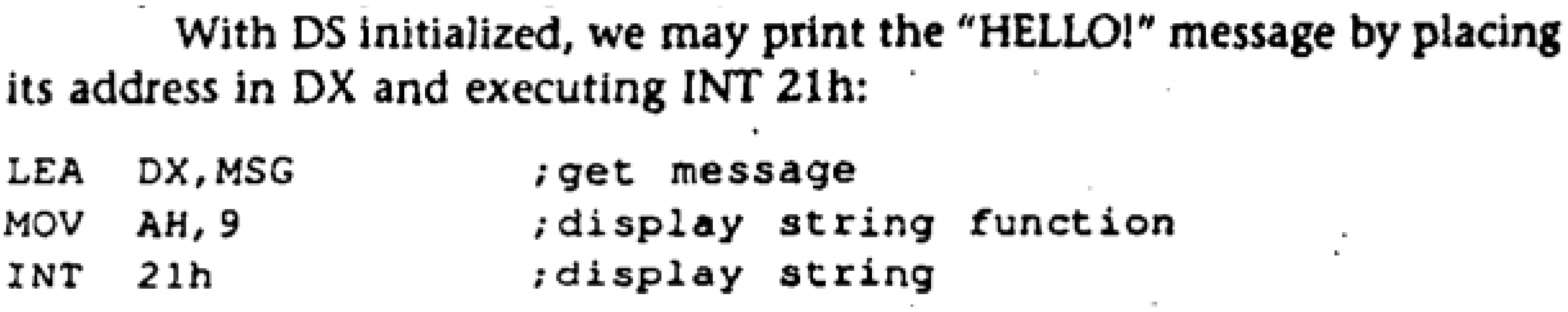
where destination is a general register and source is a memory location. **LEA** stands for "Load Effective Address." It puts a copy of the source offset address into the destination. For example,

```
LEA DX, MSG
```

puts the offset address of the variable MSG into DX.

INT 21h functions

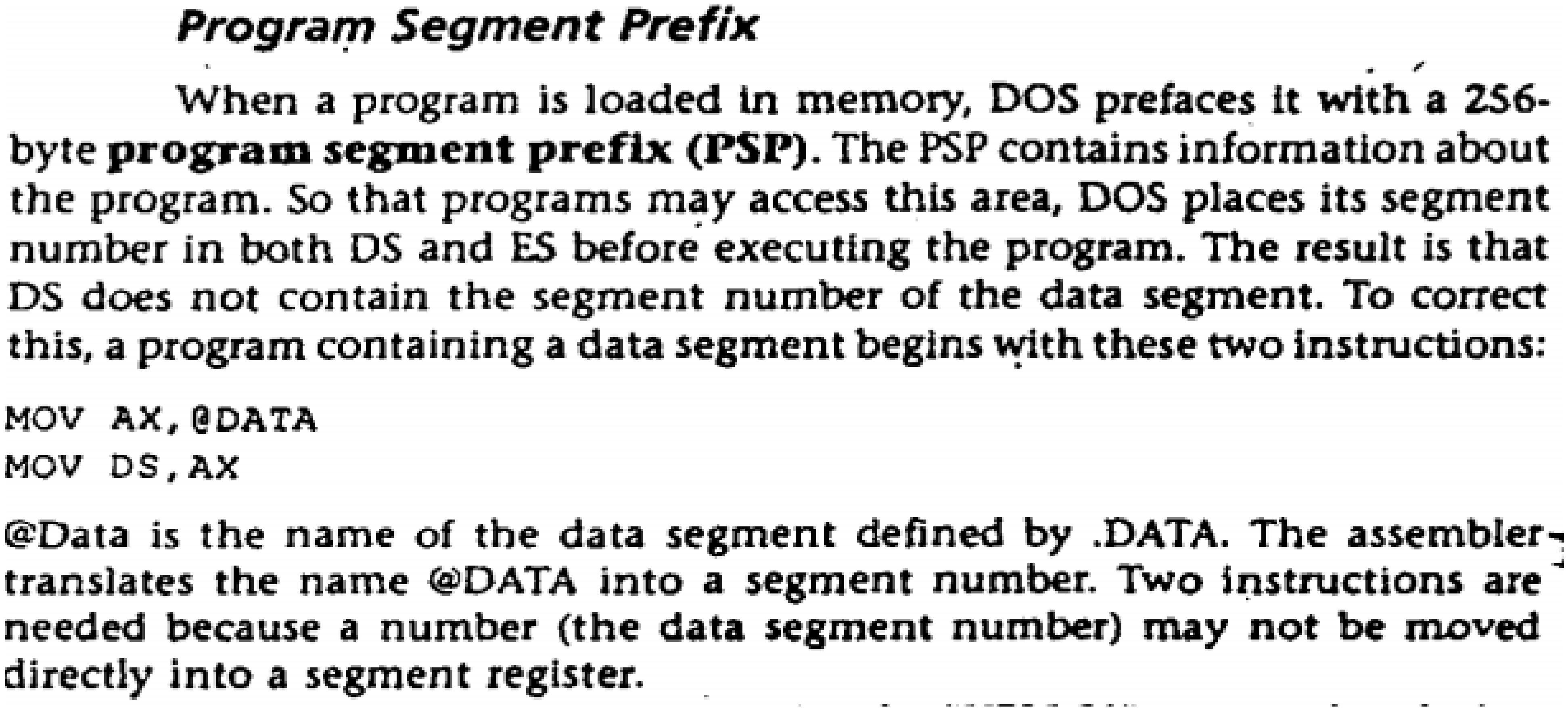
- Character String Output:



With DS initialized, we may print the "HELLO!" message by placing its address in DX and executing INT 21h:

```
LEA   DX,MSG           ;get message
MOV   AH,9             ;display string function
INT   21h              ;display string
```

Program Segment Prefix



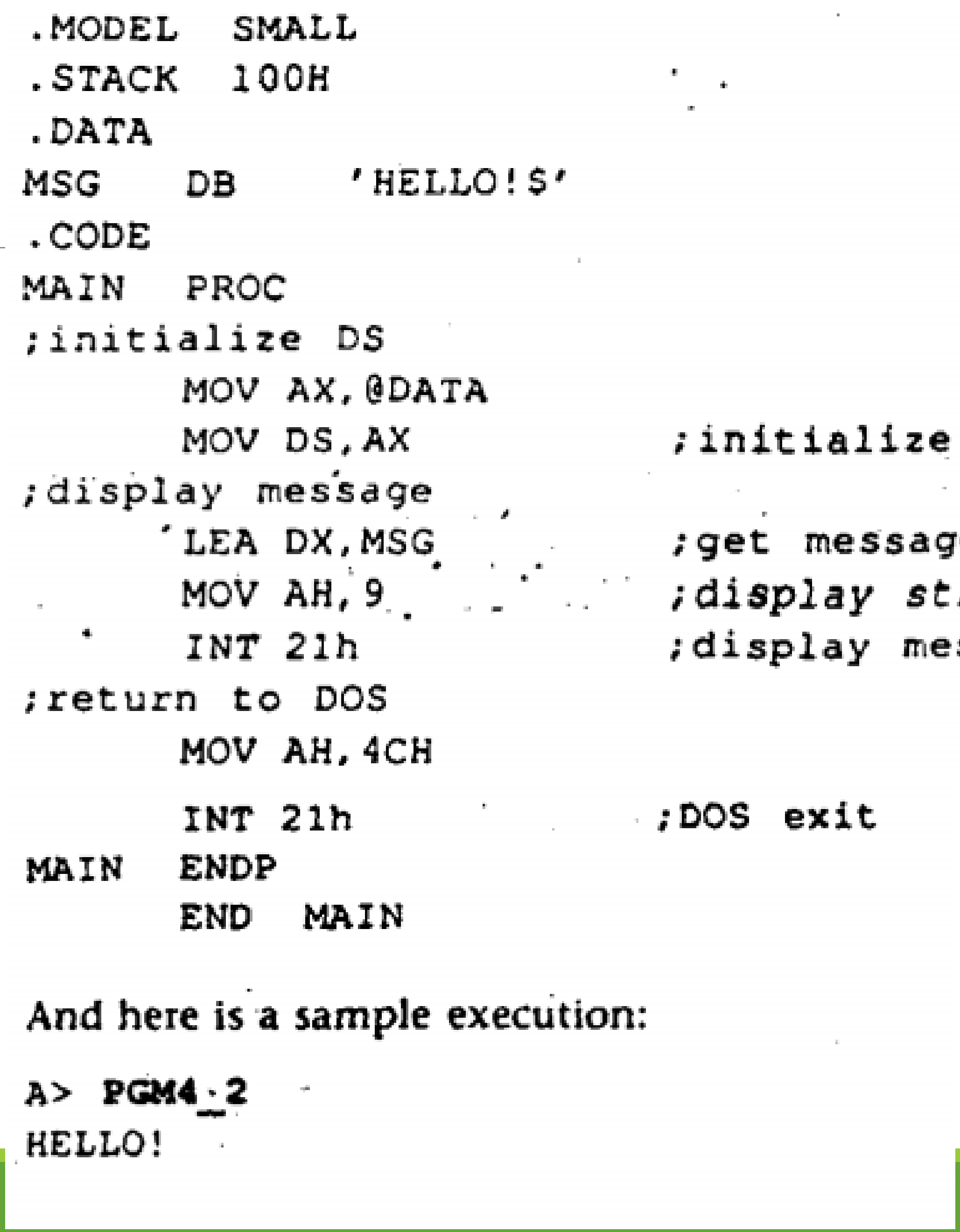
Program Segment Prefix

When a program is loaded in memory, DOS prefaces it with a 256-byte **program segment prefix (PSP)**. The PSP contains information about the program. So that programs may access this area, DOS places its segment number in both DS and ES before executing the program. The result is that DS does not contain the segment number of the data segment. To correct this, a program containing a data segment begins with these two instructions:

```
MOV  AX,@DATA
MOV  DS,AX
```

@Data is the name of the data segment defined by .DATA. The assembler translates the name @DATA into a segment number. Two instructions are needed because a number (the data segment number) may not be moved directly into a segment register.

Example



```
.MODEL  SMALL
.STACK  100H
.DATA
MSG    DB     'HELLO!$'
.CODE
MAIN   PROC
;initialize DS
       MOV AX,@DATA
       MOV DS,AX                ;initialize DS
;display message
       LEA DX,MSG               ;get message
       MOV AH,9                 ;display string function
       INT 21h                  ;display message
;return to DOS
       MOV AH,4CH
       INT 21h                  ;DOS exit
MAIN   ENDP
       END  MAIN
```

And here is a sample execution:

```
A> PGM4_2
HELLO!
```

Thank You