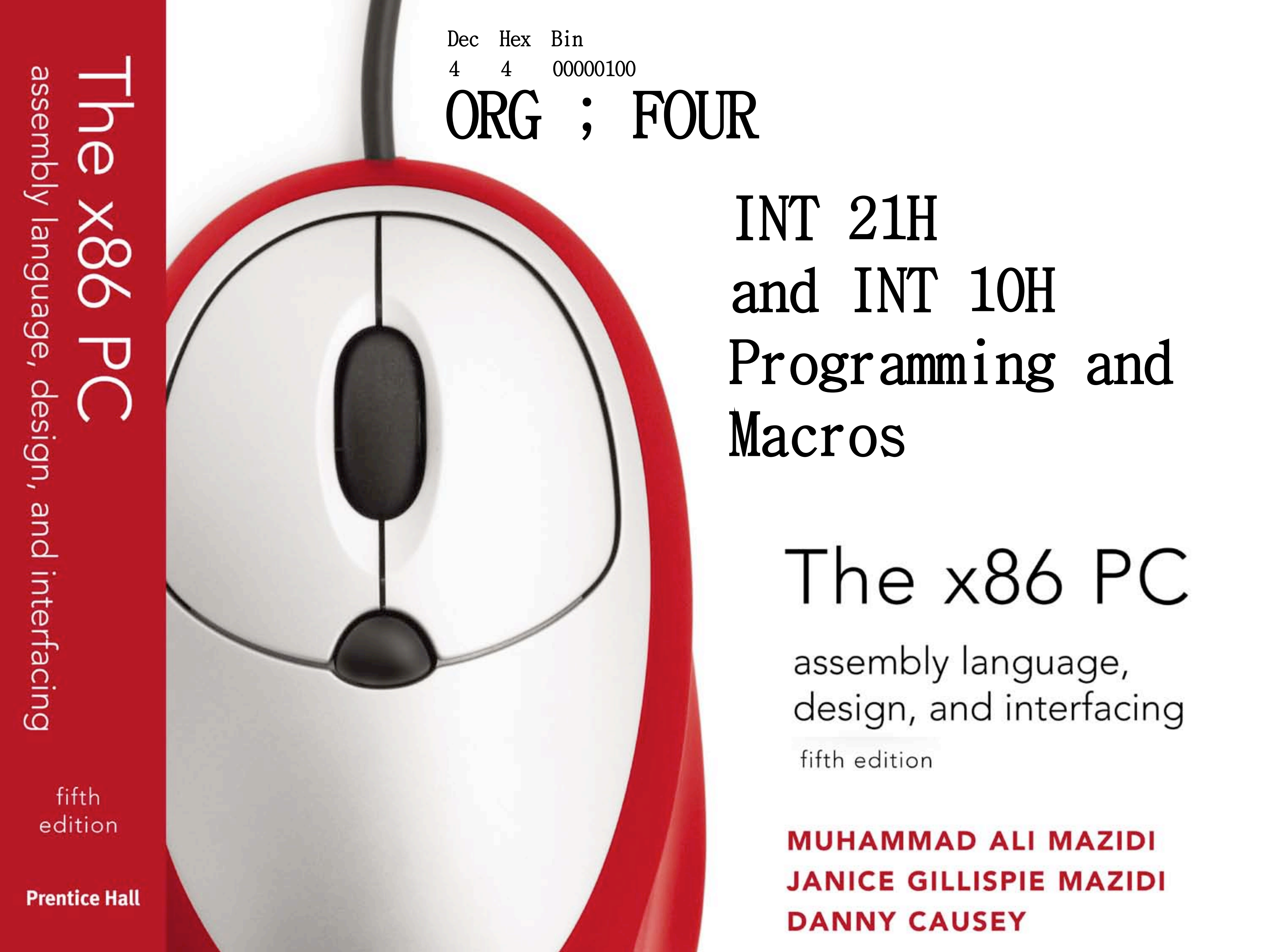
Dec	Hex	Bin
4	4	00000100

ORG ; FOUR

INT 21H and INT 10H Programming and Macros



OBJECTIVES

this chapter enables the student to:

- Use INT 10H function calls to:
  - Clear the screen.
  - Set the cursor position.
  - Write characters to the screen in text mode.
  - Draw lines on the screen in graphics mode.
  - Change the video mode.
- Use INT 21H function calls to:
  - Input characters from the keyboard.
  - Output characters to the screen.
  - Input or output strings.

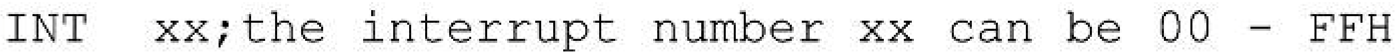
OBJECTIVES (*cont*)

this chapter enables the student to:

- Use the LABEL directive to set up structured data items.
- Code Assembly language instructions to define and invoke macros.
- Explain how macros are expanded by the assembler.
- Use the LOCAL directive to define local variables within macros.
- Use the INCLUDE directive to retrieve macros from other files.

4.0: INT 21H and 10H

- The INT instruction is somewhat like a FAR call.
  - Saves CS:IP and the flags on the stack and goes to the subroutine associated with that interrupt.



```
INT  xx;the interrupt number xx can be 00 - FFH
```

  - In x86 processors, 256 interrupts, numbered 00 to FF.
    - INT 10H and INT 21H are the most widely used with various functions selected by the value in the AH register.

4.1: BIOS INT 10H PROGRAMMING

- INT 10H subroutines are burned into the ROM BIOS.
  - Used to communicate with the computer's screen video.
    - Manipulation of screen text/graphics can be done via INT 10H.
- Among the functions associated with INT 10H are changing character or background color, clearing the screen, and changing the location of the cursor.
  - Chosen by putting a specific value in register AH.

4.1: BIOS INT 10H PROGRAMMING
monitor screen in text mode

- The monitor screen in the x86 PC is divided into 80 columns and 25 rows in normal text mode.
  - Columns are numbered from 0 to 79.
  - Rows are numbered 0 to 24.

The top left corner has been assigned 00,00, the top right 00,79.

Bottom left is 24,00, bottom right 24,79.

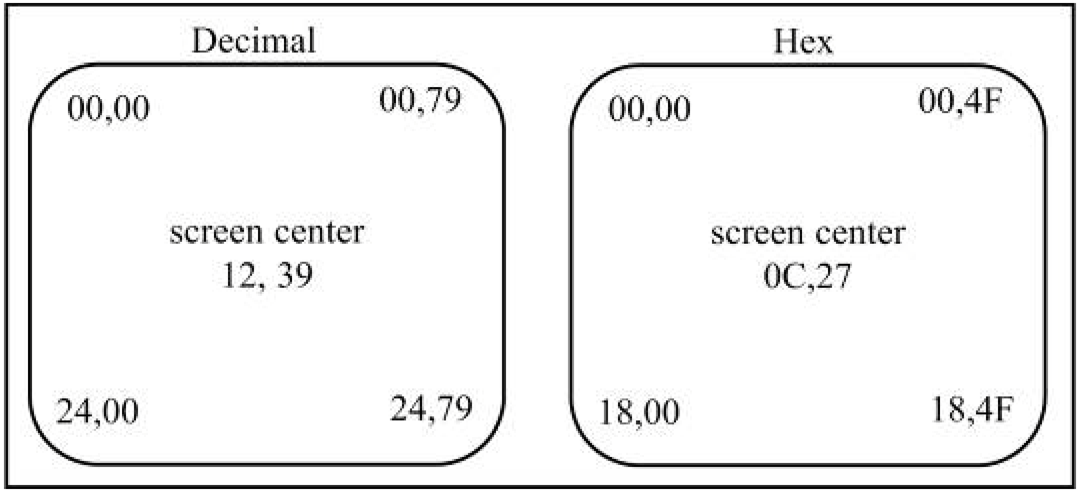
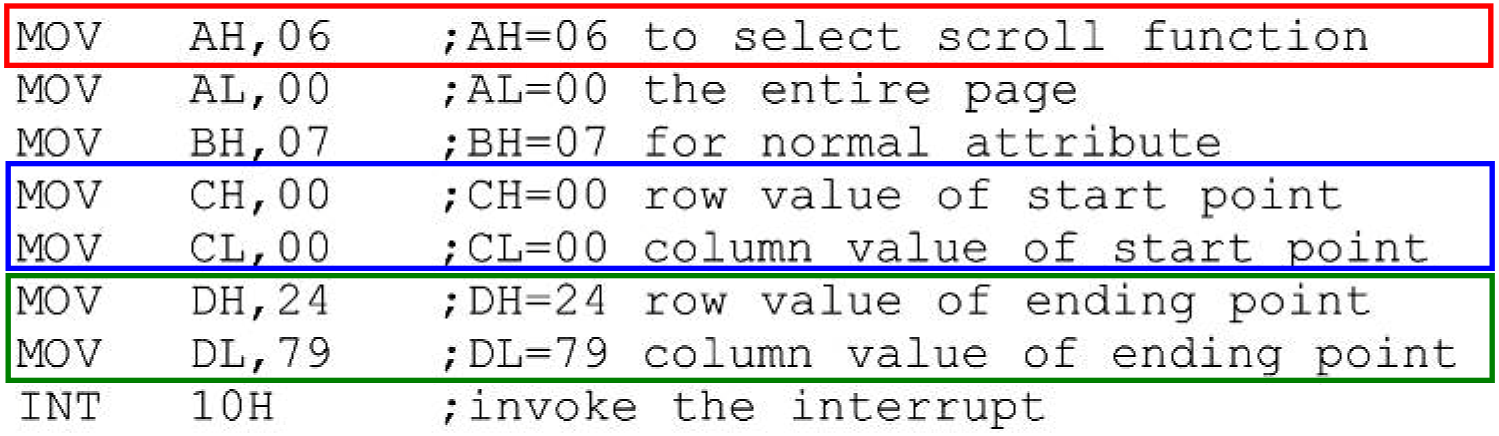


Figure 4-1 Cursor Locations (row, column)

4.1: BIOS INT 10H PROGRAMMING

screen clearing with INT 10H function 06H

- To clear the screen using INT 10H, these registers must contain certain values before INT 10H is called:
  - AH = 06, AL = 00, BH = 07, CX = 0000, DH = 24, DL = 79.



```
MOV    AH,06     ;AH=06 to select scroll function
MOV    AL,00     ;AL=00 the entire page
MOV    BH,07     ;BH=07 for normal attribute
MOV    CH,00     ;CH=00 row value of start point
MOV    CL,00     ;CL=00 column value of start point
MOV    DH,24     ;DH=24 row value of ending point
MOV    DL,79     ;DL=79 column value of ending point
INT    10H       ;invoke the interrupt
```

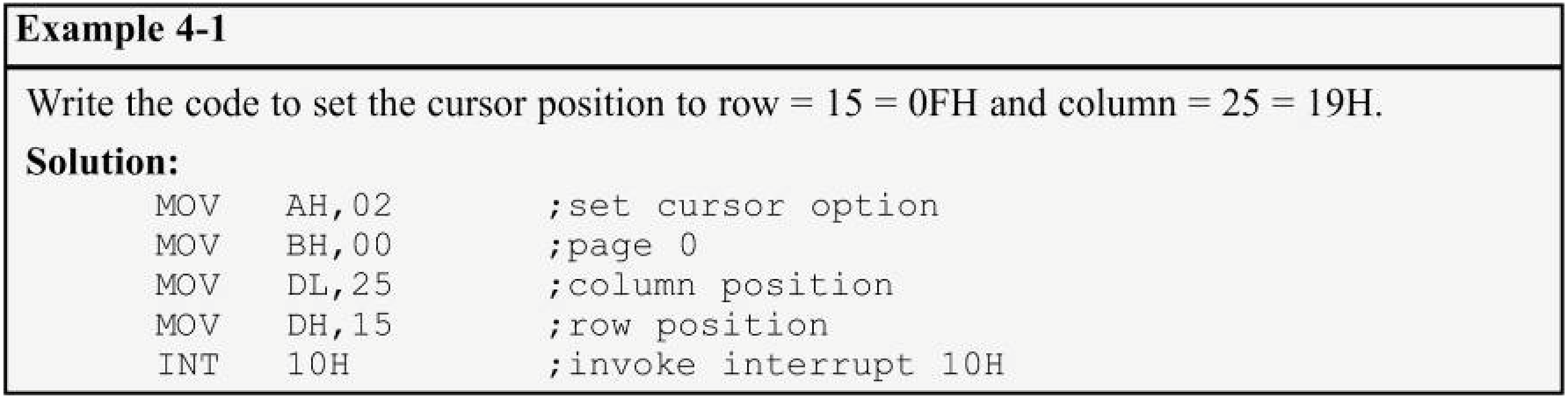
  - Option **AH = 06** calls the scroll function, to scroll upward.
  - **CH** & **CL** registers hold starting row & column.
  - **DH** & **DL** registers hold ending row & column.

4.1: BIOS INT 10H PROGRAMMING
setting the cursor to a specific location

- INT 10H function AH = 02 will change the position of the cursor to any location.
  - Desired position is identified by row/column values in DX.
    - Where DH = row and DL = column.
- Video RAM can have multiple pages of text.
  - When AH = 02, page zero is chosen by making BH = 00.
- After INT 10H (or INT 21H) has executed, registers not used by the interrupt remain unchanged.

4.1: BIOS INT 10H PROGRAMMING
setting the cursor to a specific location

- Example 4-1 demonstrates setting the cursor to a specific location.



Example 4-1

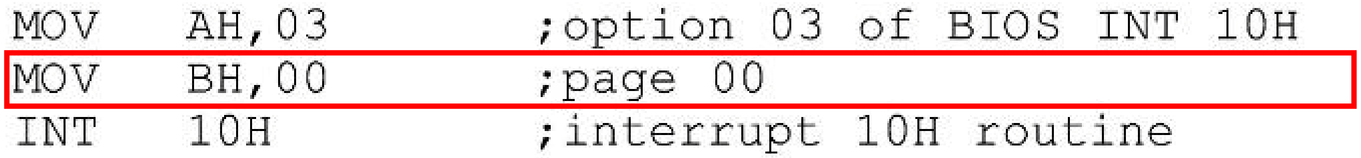
Write the code to set the cursor position to row = 15 = 0FH and column = 25 = 19H.

Solution:

```
        MOV     AH,02           ;set cursor option
        MOV     BH,00           ;page 0
        MOV     DL,25           ;column position
        MOV     DH,15           ;row position
        INT     10H             ;invoke interrupt 10H
```

4.1: BIOS INT 10H PROGRAMMING
get current cursor position

- In text mode, determine where the cursor is located at any time by executing the following:



```
MOV     AH,03          ;option 03 of BIOS INT 10H
MOV     BH,00          ;page 00
INT     10H            ;interrupt 10H routine
```

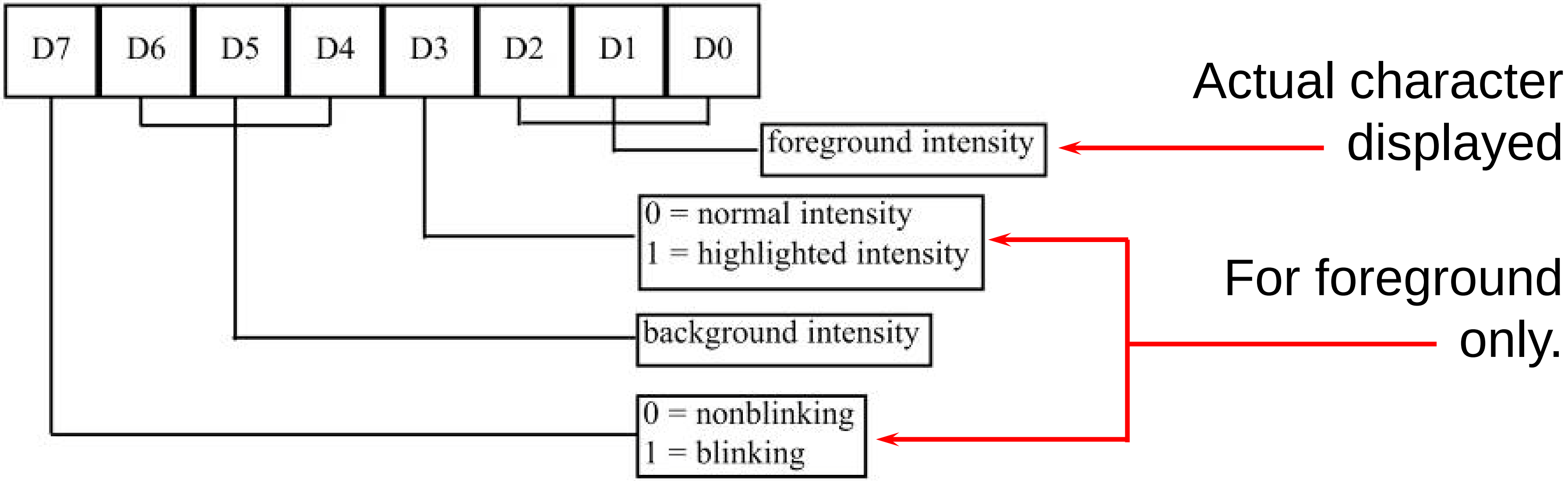
  - After execution of the program, registers DH & DL will have current row and column positions.
    - CX provides information about the shape of the cursor.
  - In text mode, **page 00** is chosen for the currently viewed page.

4.1: BIOS INT 10H PROGRAMMING
changing the video mode

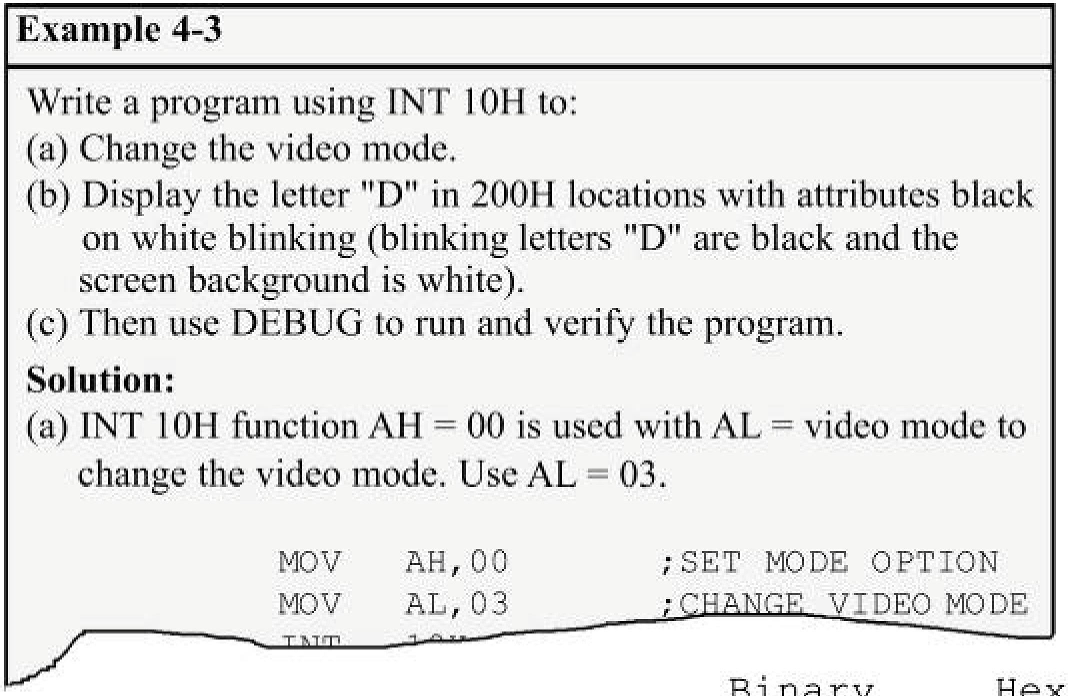
- To change the video mode, use INT 10H with AH = 00 and AL = video mode.

4.1: BIOS INT 10H PROGRAMMING
attribute byte in monochrome monitors

- An attribute associated with each character on the screen provides information to the video circuitry.
  - Character (foreground) & background color/intensity.
- The attribute byte for each character on the monochrome monitor is limited.



4.1: BIOS INT 10H PROGRAMMING
attribute byte in monochrome monitors



Example 4-3

Write a program using INT 10H to:
(a) Change the video mode.
(b) Display the letter "D" in 200H locations with attributes black on white blinking (blinking letters "D" are black and the screen background is white).
(c) Then use DEBUG to run and verify the program.

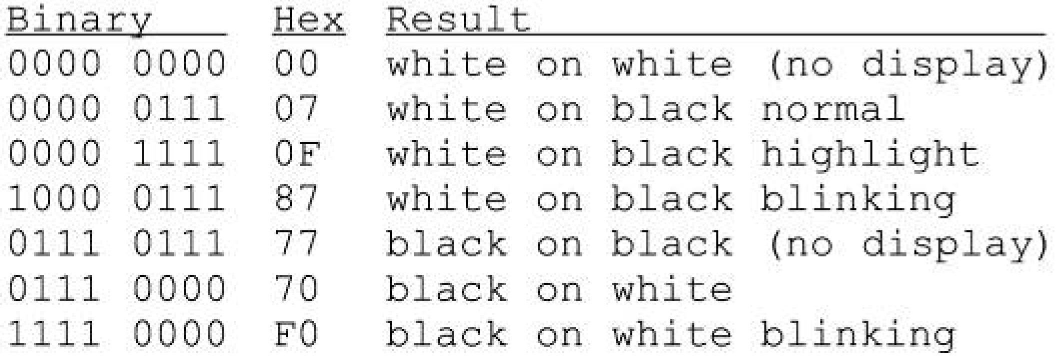
Solution:

(a) INT 10H function AH = 00 is used with AL = video mode to change the video mode. Use AL = 03.

```
        MOV    AH,00          ;SET MODE OPTION
        MOV    AL,03          ;CHANGE VIDEO MODE
        INT    10
```

See the entire example on page 134 of your textbook.

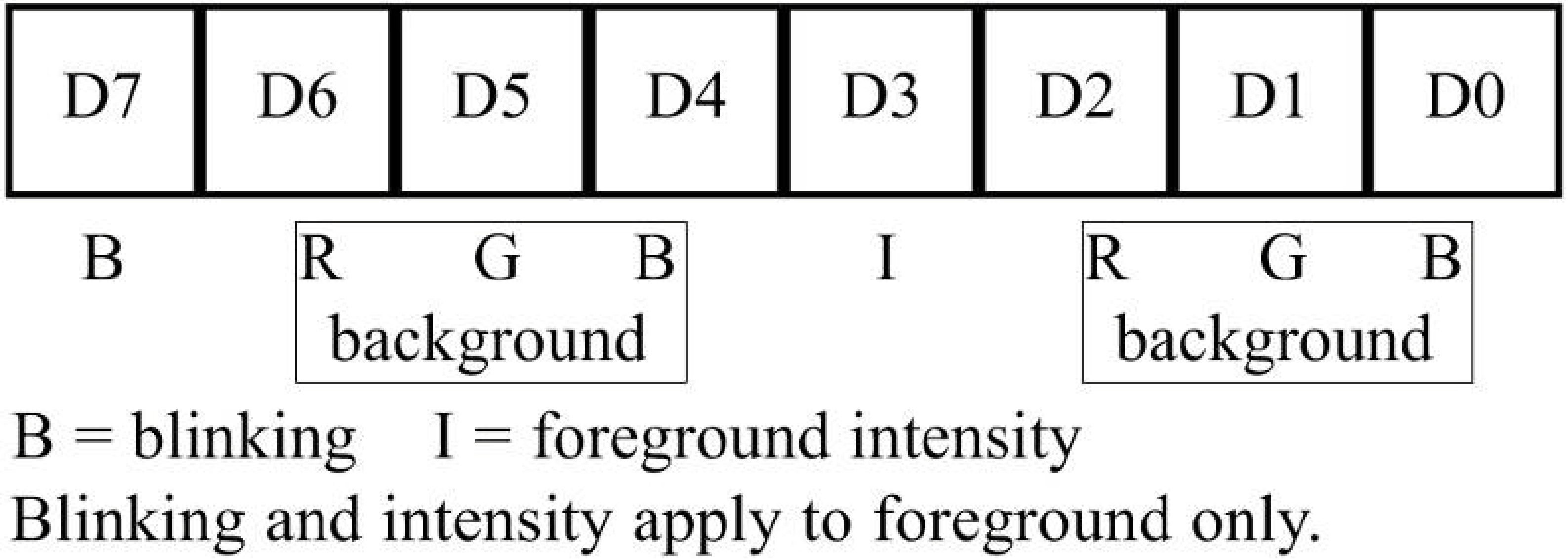
Possible variations of attributes in Fig. 4-2.



Binary	Hex	Result
0000 0000	00	white on white (no display)
0000 0111	07	white on black normal
0000 1111	0F	white on black highlight
1000 0111	87	white on black blinking
0111 0111	77	black on black (no display)
0111 0000	70	black on white
1111 0000	F0	black on white blinking

4.1: BIOS INT 10H PROGRAMMING
attribute byte in CGA text mode

- CGA mode is the common denominator for all color monitors, as S all color monitors & video circuitry are upwardly compatible,
  - – CGA attribute byte bit definition is as shown:



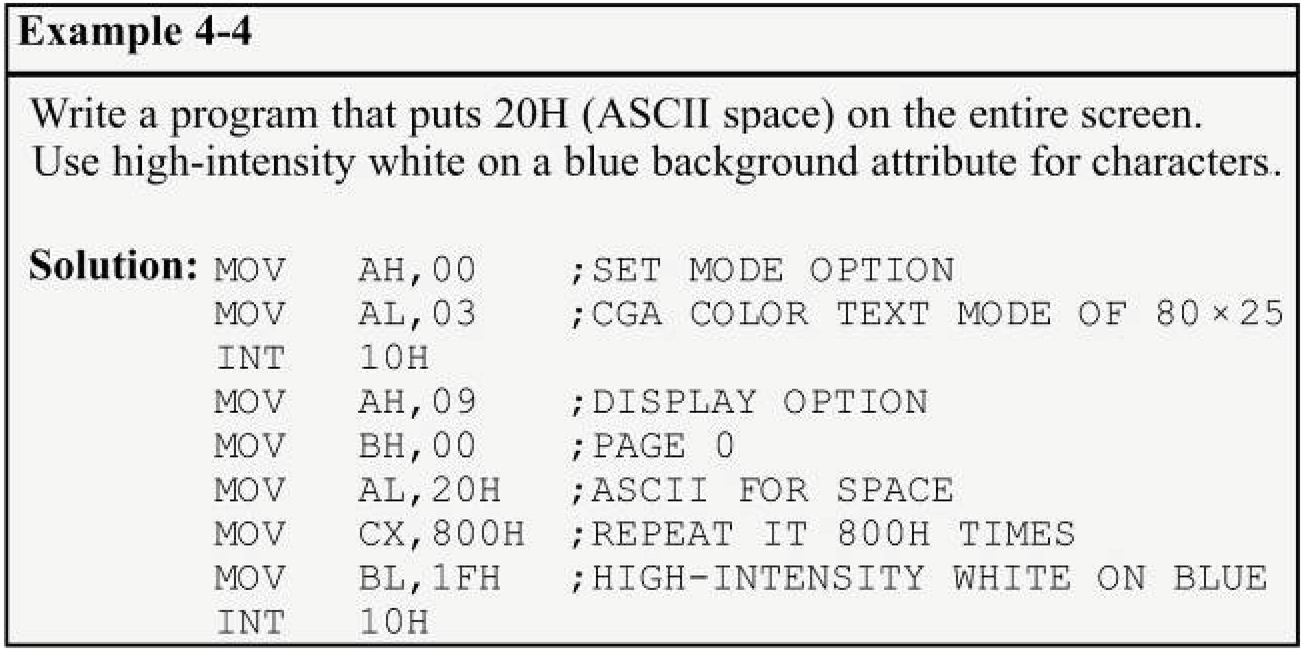
B = blinking I = foreground intensity

Blinking and intensity apply to foreground only.

Figure 4-3 CGA Attribute Byte

4.1: BIOS INT 10H PROGRAMMING
attribute byte in CGA text mode

- The background can take eight different colors by combining the prime colors **red**, **blue**, and **green**.
- The foreground can be any of 16 different colors by combining **red**, **blue**, **green**, and **intensity**



Example 4-4

Write a program that puts 20H (ASCII space) on the entire screen. Use high-intensity white on a blue background attribute for characters.

Solution:

```
MOV    AH,00    ;SET MODE OPTION
MOV    AL,03    ;CGA COLOR TEXT MODE OF 80×25
INT    10H
MOV    AH,09    ;DISPLAY OPTION
MOV    BH,00    ;PAGE 0
MOV    AL,20H   ;ASCII FOR SPACE
MOV    CX,800H  ;REPEAT IT 800H TIMES
MOV    BL,1FH   ;HIGH-INTENSITY WHITE ON BLUE
INT    10H
```

Example 4-4 shows the use of the attribute byte in CGA mode.

4.1: BIOS INT 10H PROGRAMMING
attribute byte in CGA text mode

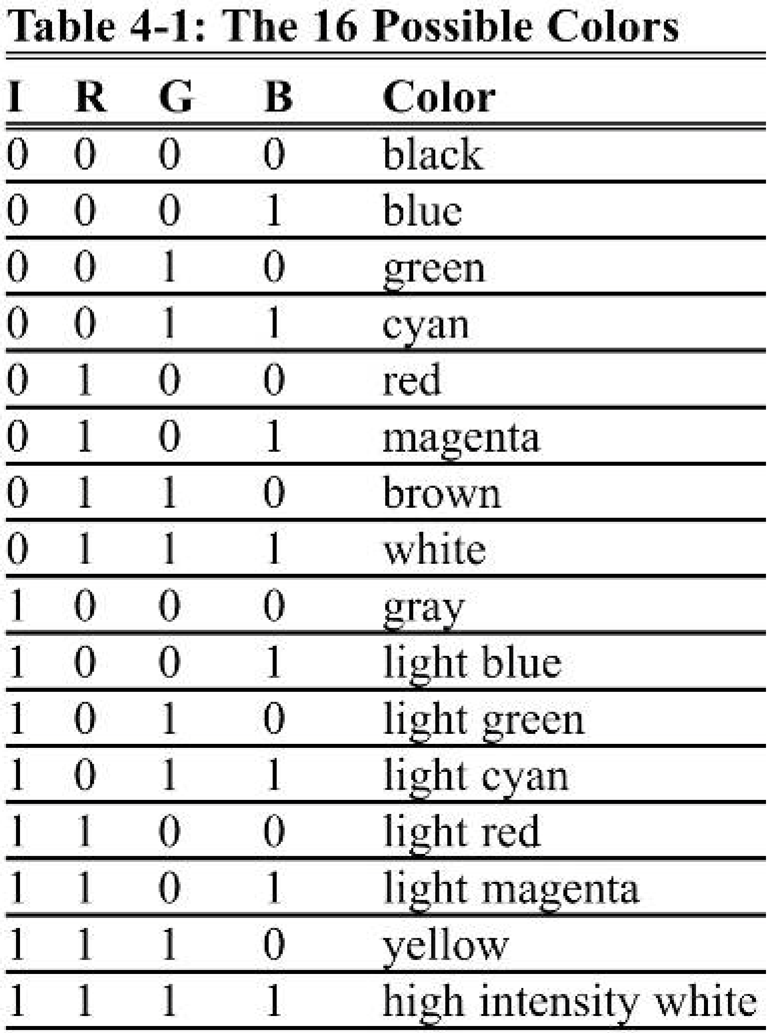
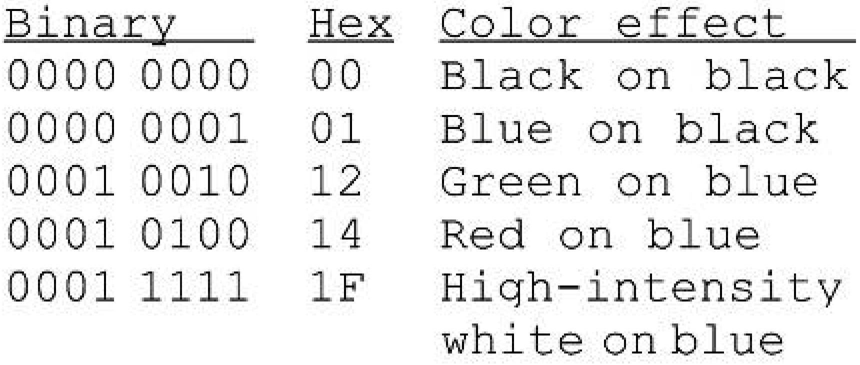


Table 4-1: The 16 Possible Colors

I	R	G	B	Color
0	0	0	0	black
0	0	0	1	blue
0	0	1	0	green
0	0	1	1	cyan
0	1	0	0	red
0	1	0	1	magenta
0	1	1	0	brown
0	1	1	1	white
1	0	0	0	gray
1	0	0	1	light blue
1	0	1	0	light green
1	0	1	1	light cyan
1	1	0	0	light red
1	1	0	1	light magenta
1	1	1	0	yellow
1	1	1	1	high intensity white

Some possible CGA colors and variations.



Binary	Hex	Color effect
0000 0000	00	Black on black
0000 0001	01	Blue on black
0001 0010	12	Green on blue
0001 0100	14	Red on blue
0001 1111	1F	High-intensity white on blue

4.1: BIOS INT 10H PROGRAMMING
graphics: pixel resolution & color

- In text mode, the screen is viewed as a matrix of rows and columns of characters.
  - In graphics mode, a matrix of horizontal & vertical pixels.
    - Number of pixels depends on monitor resolution & video board.
- Two facts associated with every pixel on the screen must be stored in the video RAM:
  - Location of the pixel, and Attributes. (color and intensity)
    - The higher the number of pixels and colors, the larger the amount of memory that is needed to store them
    - Memory requirements go up with resolution & number of colors.
  - CGA mode can have a maximum of 16K bytes of video memory due to its inherent design structure.

4.1: BIOS INT 10H PROGRAMMING
graphics: modes

- Text mode of 80 × 25 characters.
  - A total of 2K (80 × 25 = 2000) for characters, plus 2K for attributes, as each character has one attribute byte.
    - Each screen (frame) takes 4K, which results in CGA supporting a total of four pages of data, where each page represents one full screen.
- In this mode, 16 colors are supported.
  - To select this mode, use AL = 03 for mode selection in INT 10H option AH = 00.

4.1: BIOS INT 10H PROGRAMMING
graphics: modes

- Graphics mode of 320 × 200. (medium resolution)
  - 64,000 pixels. (320 columns × 200 rows = 64,000)
    - Dividing total video RAM of 128K bits (16K × 8 bits = 128K) by 64,000 pixels gives 2 bits for the color of each pixel.
- 2 bits give four possibilities, thus 320 × 200 resolution CGA can support no more than 4 colors.
  - To select this mode, use AL = 04.

4.1: BIOS INT 10H PROGRAMMING graphics: modes

- Graphics resolution of 640 × 200. (high resolution)
  - 128,000 pixels. (200 × 640 = 128,000)
    - Dividing gives 1 bit (128,000/128,000 = 1) for color, which can can be on (white) or off (black).
- 640 × 200 high-resolution CGA can support only black and white.
  - To select this mode, use AL = 06.
- With a fixed amount of video RAM, the number of supported colors decreases as resolution increases.
  - To create more colors in video boards there must be memory available to store the extra colors.

4.1: BIOS INT 10H PROGRAMMING
INT 10H and pixel programming

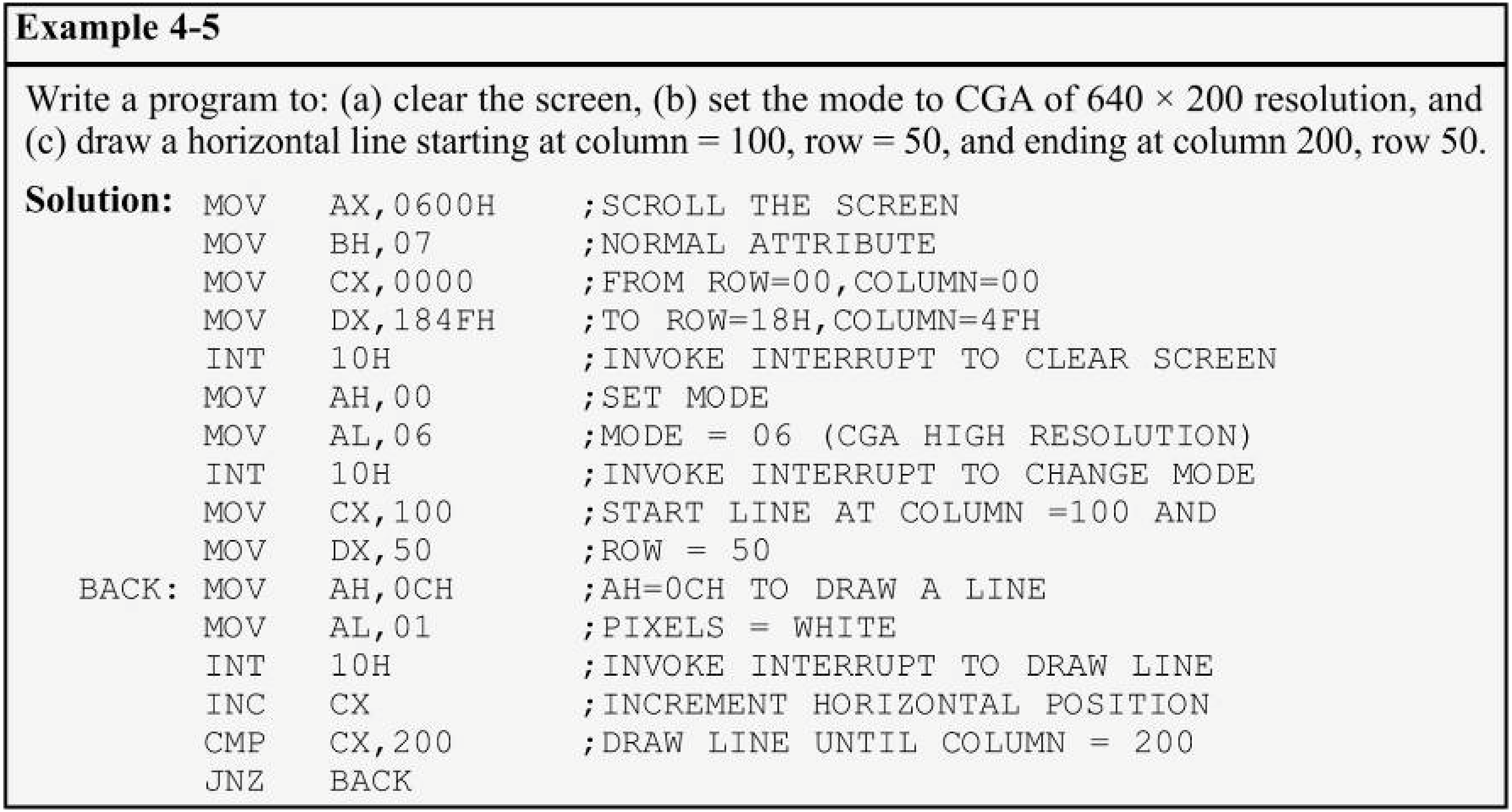
- To address a single pixel on the screen, use INT 10H with AH = 0CH.
  - The X (column) and Y (row) coordinates of the pixel must be known, and vary, depending on monitor resolution.
    - Registers are CX = the column point (the X coordinate) and DX = the row point. (Y coordinate)
  - To turn the pixel on/off, AL=1 or AL=0 for black and white.
    - The value of AL can be modified for various colors.
- If the display mode supports more than one page, BH = page number.

4.1: BIOS INT 10H PROGRAMMING
drawing lines in graphics mode

- To draw a horizontal line, choose row/column values to point to the beginning of the line and increment the column until it reaches the end of the line.
  - To draw a vertical line, increment the vertical value held by the DX register, and keep CX constant.
  - Linear equation $y = mx + b$ can be used for any line.

4.1: BIOS INT 10H PROGRAMMING
drawing lines in graphics mode

Drawing a horizontal line



Example 4-5

Write a program to: (a) clear the screen, (b) set the mode to CGA of 640 × 200 resolution, and (c) draw a horizontal line starting at column = 100, row = 50, and ending at column 200, row 50.

Solution:

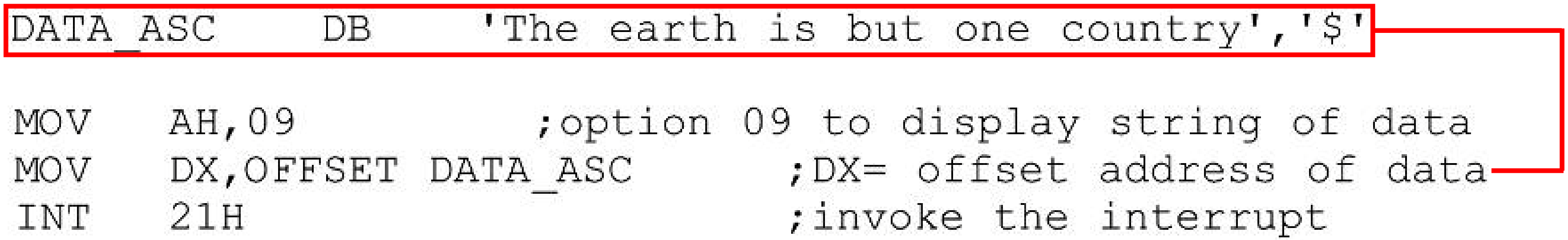
```
        MOV   AX,0600H    ;SCROLL THE SCREEN
        MOV   BH,07       ;NORMAL ATTRIBUTE
        MOV   CX,0000     ;FROM ROW=00,COLUMN=00
        MOV   DX,184FH    ;TO ROW=18H,COLUMN=4FH
        INT   10H         ;INVOKE INTERRUPT TO CLEAR SCREEN
        MOV   AH,00       ;SET MODE
        MOV   AL,06       ;MODE = 06 (CGA HIGH RESOLUTION)
        INT   10H         ;INVOKE INTERRUPT TO CHANGE MODE
        MOV   CX,100      ;START LINE AT COLUMN =100 AND
        MOV   DX,50       ;ROW = 50
  BACK: MOV   AH,0CH      ;AH=0CH TO DRAW A LINE
        MOV   AL,01       ;PIXELS = WHITE
        INT   10H         ;INVOKE INTERRUPT TO DRAW LINE
        INC   CX          ;INCREMENT HORIZONTAL POSITION
        CMP   CX,200      ;DRAW LINE UNTIL COLUMN = 200
        JNZ   BACK
```

4.2: DOS INTERRUPT 21H

- In previous chapters, a fixed set of data was defined in the data segment & results viewed in a memory dump.
  - This section uses information inputted from the keyboard, and displayed on the screen.
    - A much more dynamic way of processing information.
- When the OS is loaded, INT 21H can be invoked to perform some extremely useful functions.
  - Commonly referred to as DOS INT 21H function calls.
    - In contrast to BIOS-ROM based INT 10H.

4.2: DOS INTERRUPT 21H Option 09 outputting a data string the monitor

- INT 21H can send a set of ASCII data to the monitor.
  - Set AH = 09 and DX = offset address of the ASCII data.
    - Displays ASCII data string pointed at by DX until it encounters the dollar sign "$".
- The data segment and code segment, to display the message "*The earth is but one country*" :

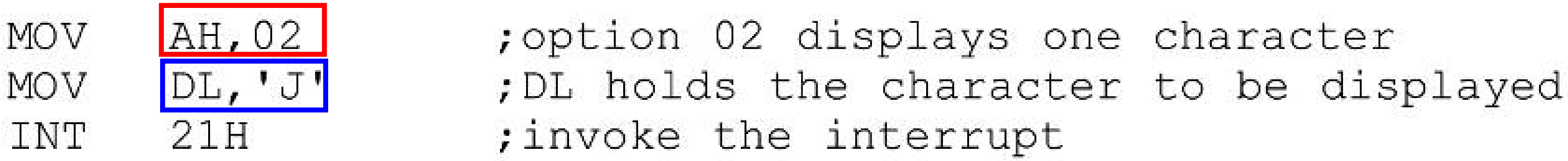


```
DATA_ASC     DB     'The earth is but one country','$'

MOV    AH,09             ;option 09 to display string of data
MOV    DX,OFFSET DATA_ASC      ;DX= offset address of data
INT    21H                     ;invoke the interrupt
```

4.2: DOS INTERRUPT 21H Option 02 outputting a single character

- To output only a single character, **02** is put in **AH**, and **DL** is loaded with the character to be displayed.
- The following displays the letter "*J*" :

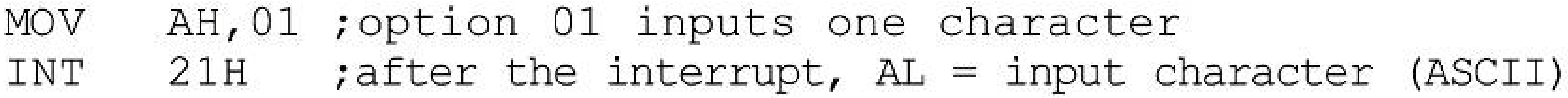


```
MOV     AH,02          ;option 02 displays one character
MOV     DL,'J'         ;DL holds the character to be displayed
INT     21H            ;invoke the interrupt
```

  – This option can also be used to display '$' on the monitor as the string display option (option 09) will not display '$‘.

4.2: DOS INTERRUPT 21H Option 01 inputting a single character, with echo

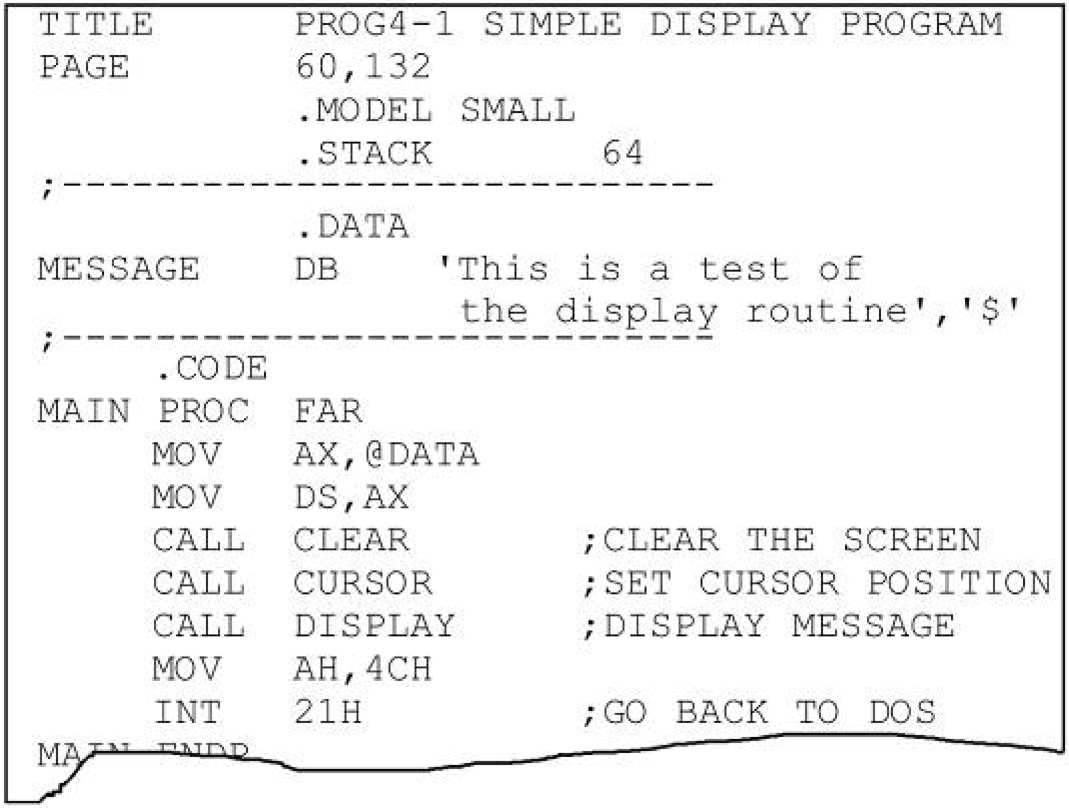
- This functions waits until a character is input from the keyboard, then echoes it to the monitor.
  - After the interrupt, the input character will be in AL.



```
MOV    AH,01 ;option 01 inputs one character
INT    21H   ;after the interrupt, AL = input character (ASCII)
```

4.2: DOS INTERRUPT 21H Option 01 inputting a single character, with echo

- Program 4-1 combines INT 10H and INT 21H.



```
TITLE       PROG4-1  SIMPLE  DISPLAY  PROGRAM
PAGE        60,132
            .MODEL  SMALL
            .STACK        64
;-----------------------------
            .DATA
MESSAGE     DB      'This  is  a  test  of
                     the  display  routine','$'
;-----------------------------
      .CODE
MAIN  PROC    FAR
      MOV     AX,@DATA
      MOV     DS,AX
      CALL    CLEAR          ;CLEAR  THE  SCREEN
      CALL    CURSOR         ;SET  CURSOR  POSITION
      CALL    DISPLAY        ;DISPLAY  MESSAGE
      MOV     AH,4CH
      INT     21H            ;GO  BACK  TO  DOS
MAIN  ENDP
```

The program does the following:

(1) Clears the screen.

(2) Sets the cursor to the center of the screen.

(3) Displays the message *"This is a test of the display routine".*

See the entire program listing on page 139 of your textbook.

4.2: DOS INTERRUPT 21H Option 0AH inputting a data string from the keyboard

- A means by which one can get keyboard data from & store it in a predefined data segment memory area.
  - Register AH = 0AH.
  - DX = offset address at which the string of data is stored.
    - Commonly referred to as a buffer area.
- DOS requires a buffer area be defined in the data segment.
  - The first byte specifies the size of the buffer.
  - The number of characters from the keyboard is in the second byte.
  - Keyed-in data placed in the buffer starts at the third byte.

4.2: DOS INTERRUPT 21H Option 0AH inputting a data string from the keyboard

- This program accepts up to six characters from the keyboard, including the return (carriage return) key.
  - Six buffer locations were reserved, and filled with FFH.

```
ORG    0010H
DATA1  DB     6,?,6 DUP (FF);0010H=06, 0012H to 0017H = FF

       MOV    AH,0AH              ;string input option of INT 21H
       MOV    DX,OFFSET DATA1     ;load offset address of buffer
       INT    21H                 ;invoke interrupt 21H
```

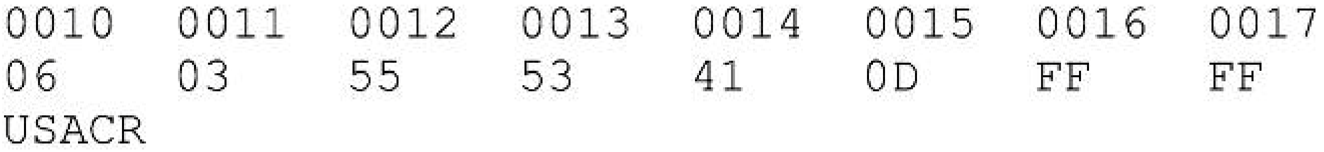
  - Memory contents of offset 0010H:

```
0010  0011  0012  0013  0014  0015  0016  0017
06    00    FF    FF    FF    FF    FF    FF
```

  - The PC won’t exit INT 21H until it encounters a RETURN.

4.2: DOS INTERRUPT 21H Option 0AH
inputting a data string from the keyboard

- Assuming the data entered through the keyboard was "USA" <RETURN>, the contents of memory locations starting at offset 0010H would look like:

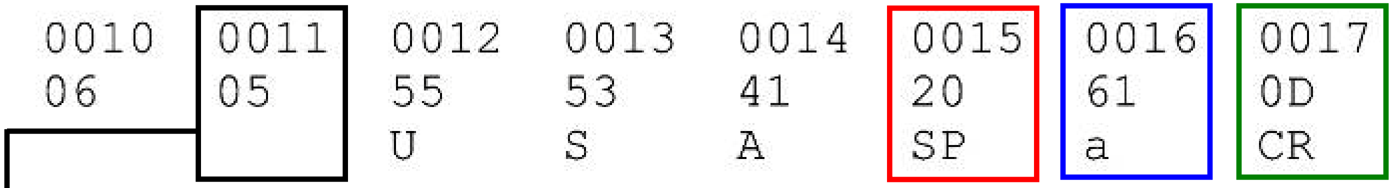


```
0010   0011   0012   0013   0014   0015   0016   0017
06     03     55     53     41     0D     FF     FF
USACR
```

  - 0010H = 06 DOS requires the size of the buffer here.
  - 0011H = 03 The keyboard was activated three times (excluding the RETURN key) to key in letters **U**, **S**, and **A**.
  - 0012H = 55H ASCII hex value for letter **U**.
  - 0013H = 53H ASCII hex value for letter **S**.
  - 0014H = 41H ASCII hex value for letter **A**.
  - 0015H = 0DH ASCII hex value for **CR**. (carriage return)

4.2: DOS INTERRUPT 21H
inputting more than buffer size

- Entering more than six characters (five + the CR = 6) will cause the computer to sound the speaker.
  - The contents of the buffer will look like this:

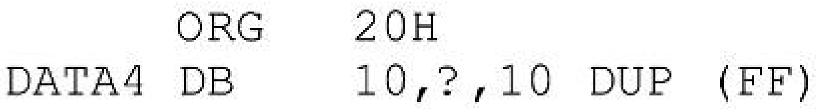


0010	0011	0012	0013	0014	0015	0016	0017
06	05	55	53	41	20	61	0D
		U	S	A	SP	a	CR

  - Location **0015** has **ASCII 20H** for <SPACE>
  - Location **0016** has **ASCII 61H** for "a“.
  - Location **0017** has **0D** for <RETURN> key.
  - The actual length is **05** at memory offset **0011H**.

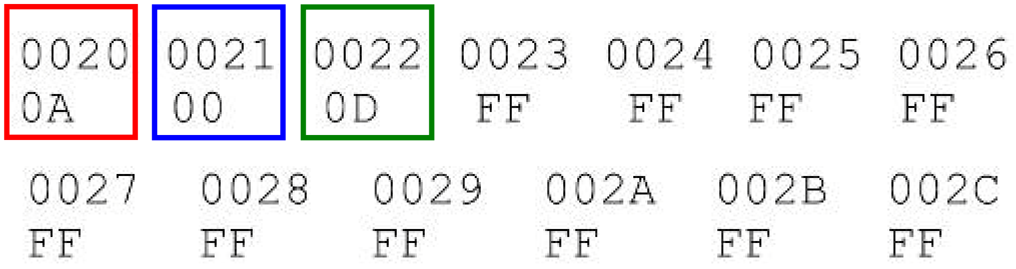
4.2: DOS INTERRUPT 21H
inputting more than buffer size

- If only the CR key is activated & no other character:



```
       ORG    20H
DATA4  DB     10,?,10 DUP (FF)
```

  - **0AH** is placed in memory 0020H.
  - **0021H** is for the count.
  - 0022H IS the first location to have data that was entered.



```
0020 0021 0022 0023 0024 0025 0026
0A   00   0D   FF   FF   FF   FF

0027 0028 0029 002A 002B 002C
FF   FF   FF   FF   FF   FF
```

CR is *not* included in the count.

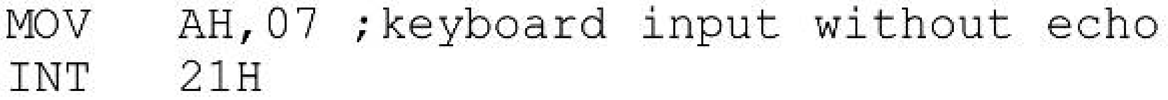
  - If only the <RETURN> key is activated, **0022H** has **0DH**, the hex code for CR.
    - The actual number of characters entered is **0** at location **0021**.

4.2: DOS INTERRUPT 21H
use of carriage return and line feed

- In Program 4-2, the EQU statement is used to equate CR (carriage return) with its ASCII value of 0DH, and LF (line feed) with its ASCII value of 0AH.
  - See pages 141 & 142
- Program 4-3 prompts the user to type in a name with a maximum of eight letters.
  - The program gets the length and prints it to the screen.
  - See page 143.
- Program 4-4 demonstrates many functions described in this chapter.
  - See pages 144 & 145.

4.2: DOS INTERRUPT 21H Option 07 keyboard input without echo

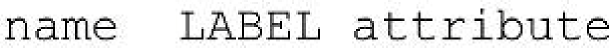
- Option 07 requires the user to enter a single character, which is not displayed (or echoed) on the screen.
  - The PC waits until a single character is entered and provides the character in AL.



```
MOV    AH,07 ;keyboard input without echo
INT    21H
```

4.2: DOS INTERRUPT 21H
using LABEL to define a string buffer

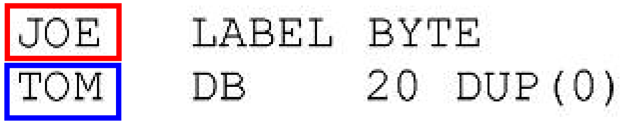
- The LABEL directive can be used in the data segment to assign multiple names to data.



```
name   LABEL attribute
```

  - Used to assign the same offset address to two names.
  - The attribute can be:
    - BYTE; WORD; DWORD; FWORD; QWORD; TBYTE.

- In the following:

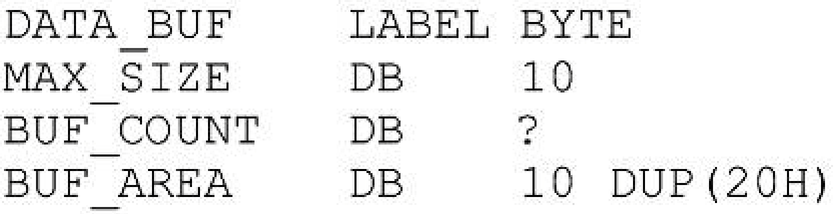


```
JOE    LABEL  BYTE
TOM    DB     20  DUP(0)
```

  - The offset address assigned to ***JOE*** is the same offset address for ***TOM*** since the LABEL directive does not occupy any memory space.

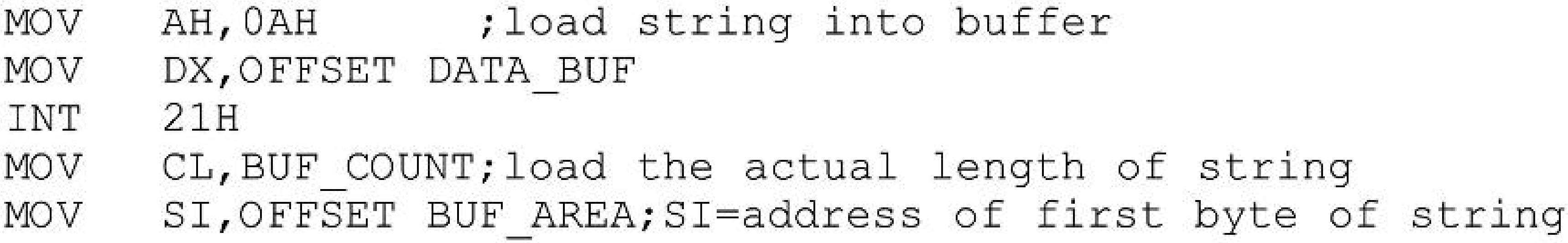
4.2: DOS INTERRUPT 21H
using LABEL to define a string buffer

- Use this directive to define a buffer area for the string keyboard input:



```
DATA_BUF    LABEL BYTE
MAX_SIZE    DB    10
BUF_COUNT   DB    ?
BUF_AREA    DB    10  DUP(20H)
```

- In the code segment the data can be accessed by name as follows:



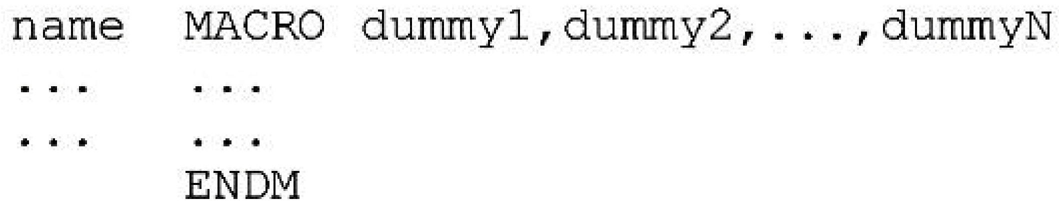
```
MOV    AH,0AH        ;load string into buffer
MOV    DX,OFFSET DATA_BUF
INT    21H
MOV    CL,BUF_COUNT;load the actual length of string
MOV    SI,OFFSET BUF_AREA;SI=address of first byte of string
```

4.3: WHAT IS A MACRO, AND HOW IS IT USED?

- There are applications in Assembly language programming where a group of instructions performs a task used repeatedly.
  - It does not make sense to rewrite them every time.
- Macros allow the programmer to write the task once only & invoke it whenever, wherever it is needed.
  - Reduces time to write code & reduce possibility of errors.

4.3: WHAT IS A MACRO, AND HOW IS IT USED?
MACRO definition

- Every macro definition must have three parts:



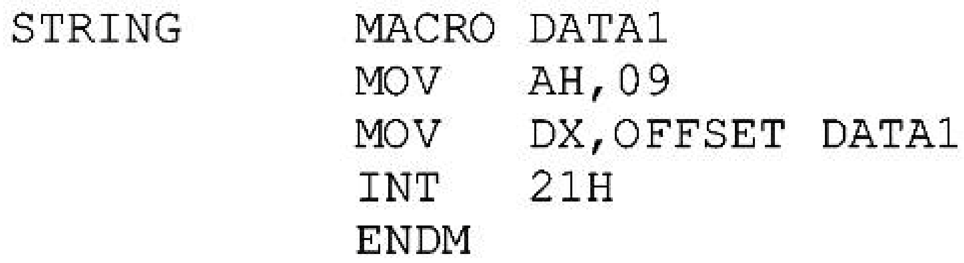
```
name   MACRO dummy1,dummy2,...,dummyN
...    ...
...    ...
       ENDM
```

  - The MACRO directive indicates the beginning of the macro definition, ENDM directive signals the end.
    - In between is the *body* of the macro.
  - The name must be unique and must follow Assembly language naming conventions.
    - Dummies are names, parameters, or registers that are mentioned in the body of the macro.
    - The macro can be invoked (or called) by its name, and appropriate values substituted for dummy parameters.

4.3: *WHAT IS A MACRO, AND HOW IS IT USED?*

MACRO definition

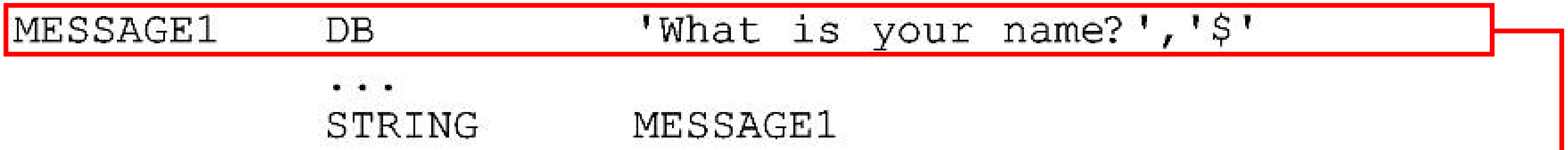
- A macro for displaying a string of data using the widely used function 09 of INT 21H:



```
STRING      MACRO DATA1
            MOV   AH,09
            MOV   DX,OFFSET DATA1
            INT   21H
            ENDM
```

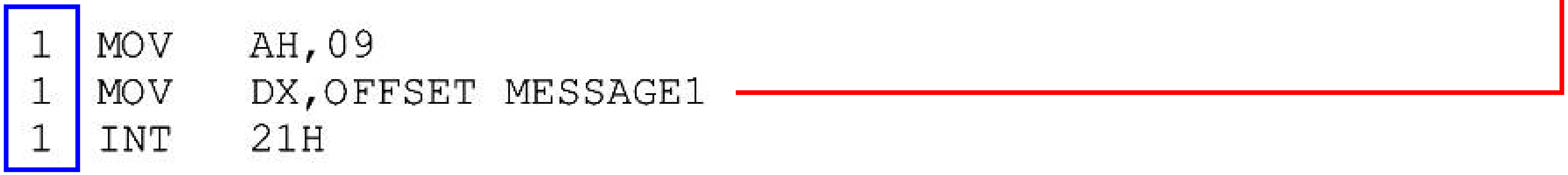
4.3: WHAT IS A MACRO, AND HOW IS IT USED?
MACRO definition

- In the following code segment, the macro can be invoked by its name with the user's actual data:
  - Instruction "STRING MESSAGE1" invokes the macro.



```
MESSAGE1      DB              'What is your name?','$'
              ...
              STRING          MESSAGE1
```

– The assembler expands the macro by providing the following code in the .LST file:



```
1  MOV    AH,09
1  MOV    DX,OFFSET MESSAGE1
1  INT    21H
```

– The (**1**) indicates that the code is from the macro.

  - Earlier versions of MASM, used a plus sign (+).

4.3: *WHAT IS A MACRO, AND HOW IS IT USED?*

comments in a macro

- Two types of comments in the macro:
  - Listable; Nonlistable.
- Comments preceded by one semicolon (;) will show up in the ".lst" file when the program is assembled.
  - Those preceded by a double semicolon (;;) will not.

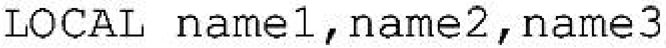
4.3: *WHAT IS A MACRO, AND HOW IS IT USED?*

comments in a macro

- Three directives designed to make programs that use macros more readable, affecting the ".lst" file, with no effect on the ".obj" or ".exe" files:
  - **.LALL (List ALL)** will list all instructions/comments preceded by a single semicolon in the ".lst" file.
  - **.SALL (Suppress ALL)** suppresses the listing of the macro body and the comments.
    - Used to make the list file shorter and easier to read
      - Will not eliminate any opcode from the object file.
  - **.XALL (eXecutable ALL)** is used to list only the part of the macro that generates opcodes.
    - The default listing directive.

4.3: WHAT IS A MACRO, AND HOW IS IT USED?
LOCAL directive and its use in macros

- If a macro is expanded more than once, and there is a label in the label field of the body of the macro, these labels must be declared as LOCAL.
  - Otherwise, an assembler error would be generated when the same label was encountered in two or more places.
- Rules which must be observed in the macro body:
  - 1. All labels in the label field must be declared LOCAL.
  - 2. LOCAL directive must be right after the MACRO directive.
  - 3. The LOCAL directive can be used to declare all names and labels at once.

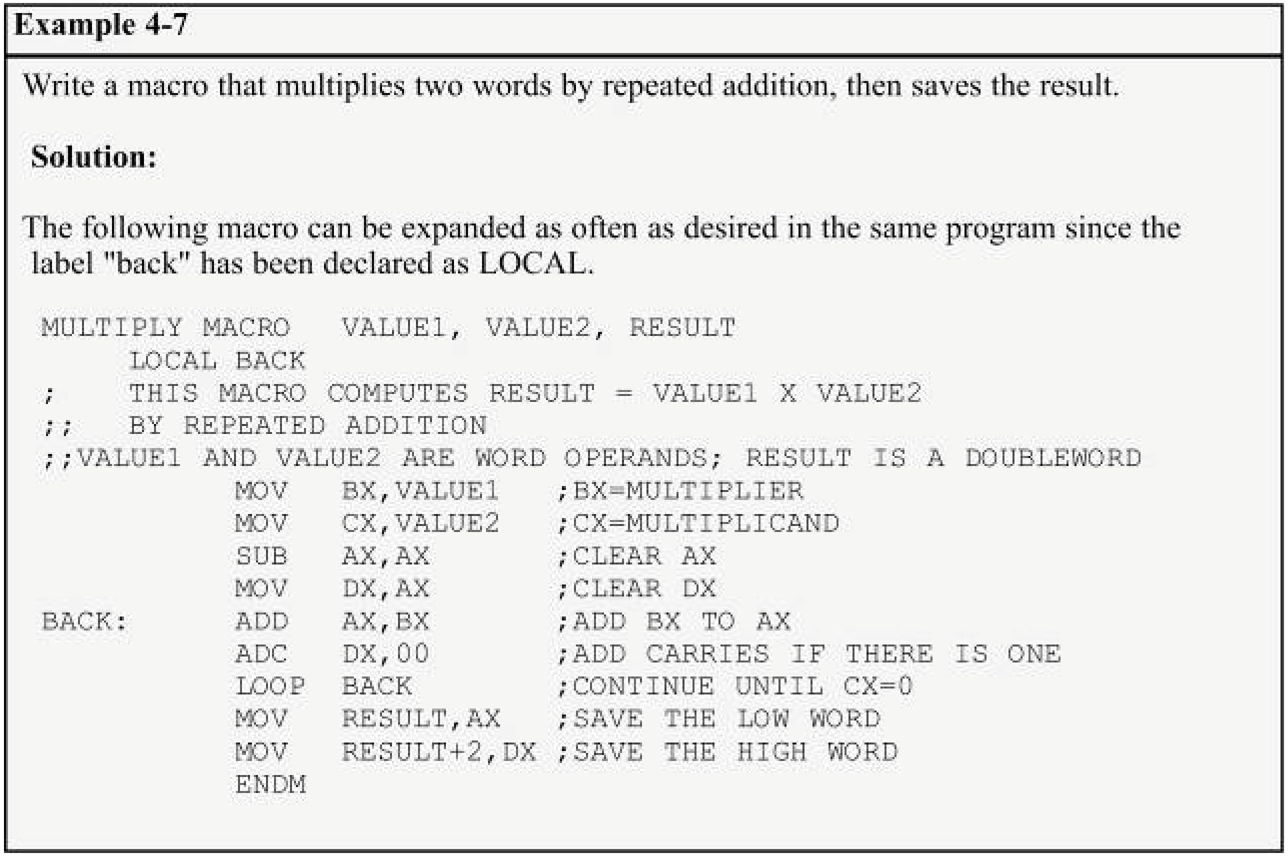


```
LOCAL name1,name2,name3
```

4.3: *WHAT IS A MACRO, AND HOW IS IT USED?*

LOCAL directive and its use in macros

- In example 4-7, the "BACK" label is defined as LOCAL right after the MACRO directive.



Example 4-7

Write a macro that multiplies two words by repeated addition, then saves the result.

Solution:

The following macro can be expanded as often as desired in the same program since the label "back" has been declared as LOCAL.

```
MULTIPLY MACRO   VALUE1, VALUE2, RESULT
        LOCAL BACK
;       THIS MACRO COMPUTES RESULT = VALUE1 X VALUE2
;;      BY REPEATED ADDITION
;;VALUE1 AND VALUE2 ARE WORD OPERANDS; RESULT IS A DOUBLEWORD
                 MOV    BX,VALUE1    ;BX=MULTIPLIER
                 MOV    CX,VALUE2    ;CX=MULTIPLICAND
                 SUB    AX,AX        ;CLEAR AX
                 MOV    DX,AX        ;CLEAR DX
BACK:            ADD    AX,BX        ;ADD BX TO AX
                 ADC    DX,00        ;ADD CARRIES IF THERE IS ONE
                 LOOP   BACK         ;CONTINUE UNTIL CX=0
                 MOV    RESULT,AX    ;SAVE THE LOW WORD
                 MOV    RESULT+2,DX  ;SAVE THE HIGH WORD
                 ENDM
```

4.3: *WHAT IS A MACRO, AND HOW IS IT USED?* INCLUDE directive

- The INCLUDE directive allows a programmer to write macros, save them in a file, and later bring them into any file.
  - Used to bring this file into any ".asm" file, to allow the program can call any of the macros as needed.
    - See Program 4 -7 on pages 155-157.
- In the list file of Program 4-7, the letter "C" in front of the lines indicates that they are copied from another file and included in the present file.

Dec	Hex	Bin
4	4	00000100

ENDS ; FOUR