Program 5.1

5.1 The extended ASCII character set includes those characters and symbols with ASCII codes between 80H and FFH. Modify Program 5.1 to print only these extended characters. Provide a complete program listing of your solution with comments.

5.2 Study the program listing shown in Figure 5–26. Behind each program line, add a meaningful comment. You may find it useful to enter the program into memory using DEBUG and run it. You should see the ASCII character set printed in bright green on a blue background (*Hint:* Refer to Appendix B.1 and B.3 for descriptions of the BIOS services and color codes used). Now answer the following questions about this program.

(a) What video mode does the program set?

(b) Which instruction defines the screen colors?

(c) What is the purpose of the instruction MOV CX,07D0? Try loading smaller numbers into CX and note the effect.

```
1446:0100 B400          MOV     AH,00
1446:0102 B002          MOV     AL,02
1446:0104 CD10          INT     10
1446:0106 B82009        MOV     AX,0920
1446:0109 B31A          MOV     BL,1A
1446:010B B9D007        MOV     CX,07D0
1446:010E CD10          INT     10
1446:0110 B402          MOV     AH,02
1446:0112 BA000A        MOV     DX,0A00
1446:0115 B700          MOV     BH,00
1446:0117 CD10          INT     10
1446:0119 B40E          MOV     AH,0E
1446:011B B000          MOV     AL,00
1446:011D CD10          INT     10
1446:011F FEC0          INC     AL
1446:0121 3C80          CMP     AL,80
1446:0123 75F8          JNZ     011D
1446:0125 CD20          INT     20
```

Figure 5–26.
Figure for Analysis and Design Question 5.2.

(d) What is the effect of changing the MOV AX,0920 instruction to MOV AX,0924?

(e) What change is required to display the ASCII characters in red on a white background?

Program 5.2

5.3 Determine the contents of registers AL, AH, BL, and BH after the following instructions have executed.

```
MOV     AX,EA7BH
MOV     BX,AX
AND     AL,3CH
OR      AH,0FH
XOR     BL,F0H
NOT     BH
```

Program 5.3

5.7 Refer to the listing for Program 5.3 in Figure 5–10. Assume the user enters the problem: *28+79=*. Determine the contents of the following just before the JNC instruction in 0139 executes:

(a) SI
(b) Byte Ptr [SI+1]
(c) Word Ptr [SI+2]
(d) Word Ptr [SI+5]
(e) AX
(f) BH
(g) CL
(h) DX
(i) CF

5.6 Write a program to display the two packed BCD numbers in register AX. Example: AX = 1234H, video display shows 1234. Provide a complete program listing of your solution with comments. (*Hint:* Call Program 5.2 as a subroutine twice; replace the INT 20H at the end of Program 5.2 with RET).

Table B–3.
Colors Available in 16-Color Text and Graphics Modes*

Intensity†	Red	Green	Blue	Hex	Description
0	0	0	0	00	Black
0	0	0	1	01	Blue
0	0	1	0	02	Green
0	0	1	1	03	Cyan
0	1	0	0	04	Red
0	1	0	1	05	Magenta
0	1	1	0	06	Brown
0	1	1	1	07	White (or gray)
1	0	0	0	08	Black (or dark gray)
1	0	0	1	09	Bright blue
1	0	1	0	0A	Bright green
1	0	1	1	0B	Bright cyan
1	1	0	0	0C	Bright red
1	1	0	1	0D	Bright magenta
1	1	1	0	0E	Yellow
1	1	1	1	0F	Bright white

*The high-order four bits of the attribute byte set the background color; the low-order four bits set the foreground color.
†When setting the background color, if the intensity bit is set, blinking is enabled.

5.1 Program 5.1: Displaying the ASCII Character Set

Introduction

The ASCII character set consists of all of the letters of the alphabet, the numbers 0–9, and various punctuation symbols and control codes. In all, 128 codes are defined (see Table 2.5). In Program 5.1, we create a COM program that displays a symbol for each of these codes on your PC's video monitor. This is accomplished by setting up a program loop that passes the ASCII codes one by one to the BIOS video service (interrupt 10H, service number 0EH). DEBUG is used to write and test the program, which will run on any 80x86 computer.

In this section we:

- Develop a machine language program using a top-down design approach.
- Set up a program loop using the compare and conditional jump instructions.
- Show how to document a machine language program.

New Instructions

MOV Destination, Source. The move instruction is used to copy data from one CPU register to another, from memory to a CPU register, or from a CPU register to memory. It cannot be used to move data from one memory location to another (move the data to a CPU register first, then to the new memory location).

When using this instruction, the size of the destination and source operands must match. The following are examples of valid and invalid move instructions:

```
MOV AL,BL    ;Valid-8-bit register to 8-bit register
MOV AL,BX    ;Invalid-Attempting to copy a 16-bit register
             into an 8-bit register
MOV BX,AL    ;Invalid-Although AL fits "inside" BX, the
             processor would not recognize which bits to
             replace
```

When accessing memory, the size of the register operand defines the size of the data moved:

```
MOV AL,[0800]     ;The byte in memory location 0800 is copied
                  into register AL
MOV EAX,[0800]    ;The four bytes (doubleword) beginning at
                  address 0800 are copied into register EAX
```

INC Destination. The increment instruction is used to add 1 to the contents of a CPU register or memory location. The flags are changed to reflect the result. For example, consider the following two instructions:

```
MOV AL,FF     ;AL = FF
INC AL        ;AL = FF + 1 = 00
```

The flags are affected as follows:

```
CF = X    ;Not affected
PF = 1    ;The number of 1s in the result is even
            (0 is considered even)
AF = 1    ;There is a carry from bit 3 to bit 4
ZF = 1    ;The result is 0
SF = 0    ;Bit 7 of the result is 0
OF = 0    ;There was no overflow
```

A similar instruction is DEC destination, which subtracts one from the destination operand and sets the flags accordingly.

CMP Operand1, Operand2. The compare instruction is used to compare a CPU register with another CPU register, memory location, or immediate data. This is done by subtracting operand2 from operand1 and setting the flags according to the result. Note that neither operand is affected, only the flags. Typically, a conditional jump instruction follows the compare.

JNZ Destination. This instruction transfers control to the instruction at the destination address if the zero flag is not set. When used with the compare instruction, it allows a program to determine if two operands are equal and branch accordingly.

```
CMP AL,BL    ;Compare AL with BL
JNZ 0121     ;If AL <> BL, transfer control to location 0121
INC CX       ;If AL = BL, the jump is not taken and control
              "falls into" this instruction
```

There are many different conditional jump instructions (see Table 4.2[f]) corresponding to the various tests possible. Some have different but equivalent mnemonics. For example, JE (jump if equal) is equivalent to JZ (jump if zero). If the target address is in another segment (a far jump), an unconditional jump must be used. For example, the following is invalid:

```
        JNZ New_Segment_Address
A1:     If zero, the program continues here
```

To accomplish this far jump, use the following:

```
        JZ A1
        JMP New_Segment_Address
A1:     If zero, the program continues here[1]
```

INT Type Number. This is the software interrupt instruction. It is used to call interrupt service routines (ISRs) without having to know the address of these routines. This is done by using the type number as an index into a table (the interrupt vector table, or IVT) that stores the addresses of the various ISRs.

[1]When using the 8086 (and DEBUG), the target address for a conditional jump is limited to +127 or −128 bytes relative to the address of the first byte of the next instruction. The 386 and later processors are limited to +32,767 to −32,768 in a 64K segment or $+2^{31} - 1$ to -2^{31} in a 4 GB segment.

When executed, the INT instruction pushes the flags and current address onto the stack. This allows the IRET instruction (return from interrupt) to pop these values back off the stack, thereby returning control to the calling program.

Many of the interrupt type numbers are reserved for use by the processor (see Table 4–5). MS-DOS and PC-DOS make extensive use of software interrupts to provide services and enhanced functions to applications programs (see Appendix B for a description of these interrupts).

New DOS Features

BIOS INT 10H, Service 00 (Set Video Mode). Service 00 is used to configure the video interface card to a particular mode. Appendix B.1 lists the basic operating modes. Many video cards support additional modes via their onboard BIOS. Program 5.1 takes advantage of the fact that accessing this function clears the screen (even if the same video mode is specified).[2] Analysis and Design Question 5.2 explores using this service to set the video mode to 80 × 25 16-color text.

BIOS INT 10H, Service 02 (Set Cursor Position). Service 02 is used to move the cursor to a particular column (register DL) and row (register DH) on the video screen. The upper left corner of the screen has the coordinates 0,0. Depending on the video mode, several display pages may be supported. This allows the screen contents to be changed very quickly by selecting a new display page. Register BH holds the page number, which in Program 5.1 is set to 0.

BIOS INT 10H, Service 0EH (Write character in TTY mode) Service 0EH is used to write one character (passed in register AL) to the screen at the current cursor position in the current screen mode. In the color graphics modes, register BL holds the color attribute (see Appendix B.3).

DOS INT 20H (Program Terminate). INT 20H is used to end a program and return control to DOS. It should only be used with "simple" programs that do not open files or change the value of register CS.

Problem Statement

Write an 80x86 program that prints the ASCII character set codes 0–127 beginning on line 10, column 0 of the PC's video monitor.

Discussion

Program Outline. Figure 5–1 outlines the program solution. Note the top-down approach. We begin by identifying three tasks the program must accomplish:

1. Clear the screen and position the cursor on line 10, column 0.
2. Print the ASCII characters.
3. Return to DOS.

Next we work our way down, refining each task into ever-smaller pieces. We end up with a word description of all of the steps required. In the right-hand column, we convert these steps to machine code instructions.

[2]To override this, add 80H to the video mode code in register AL. In this way, the new video mode will be set, but the screen will *not* be cleared.

I. Clear Screen and Position Cursor on Line 10 Column 0		
A. (Re)Set Video Mode to 80 × 25		`mov ax, 0002`
1. BIOS INT 10H, service 00		`int 10h`
i. AH = 0 'Service 0		
ii. AL = 2 'Video mode 2		`mov ah, 2`
B. Move cursor to line 10 column 0		`mov dx, 0a00`
1. BIOS INT 10H, service 2		`mov bh, 0`
i. AH = 2		`int 10h`
ii. DX = 0A00 (row = 10, column = 0)		
2. BH = 0 (page 0)		
II. Print the ASCII Characters		`mov ax, 0e00`
A. Print ASCII code	`II.A`	`int 10h`
1. BIOS INT 10H, service 0EH		
i. AH = 0EH		
ii. AL = character code (first code is 0)		
B. Increment code and repeat II.A until all 128 codes have been printed		`inc al`
		`cmp al, 80h`
1. AL = AL + 1		`jnz II.A`
2. AL = 128?		
i. No: goto II.A		
ii. Yes: continue		
III. Return to DOS		`int 20h`
A. INT 20H		

Figure 5–1.
Outline for Program 5.1.

Program Listing. The next step is to type the program into DEBUG, beginning at address 0100H. This is shown in Figure 5–2. As DEBUG assembles the instructions, the address for the JNZ instruction can now be determined. The last step is to name the program (***n prog1.com***), load register CX with the program length (***rcx 1B***), and write the program to disk (**w**).

To run the program, type ***g*** (or exit to DOS and type ***prog1***). Figure 5–3 shows the result. The first 32 characters are control codes that display as "odd" symbols. However, beginning with 20H (space) and 21H (exclamation mark), each succeeding character appears as expected.

Documenting the Program. Assembly language programs can be very difficult to understand, especially if the author chooses not to annotate each program line. Figure 5–4 shows Program 5.1 without these comments. Although this is the same program, it would be very difficult to identify its function without a great deal of study.[3]

[3]You might argue that it is a waste of time typing comments into a DEBUG program, as they cannot be saved as part of the program file. And this is true. However, by creating a *text* version of the program (PROG1.TXT) and then "pasting" this file into DEBUG, the annotated version of the program can be saved. This method was described at the end of Section 4.2.

```
-a100
1415:0100 mov     ax,0002          ;BIOS service 0, video mode 2
1415:0103 int     10               ;Set video mode and clear screen
1415:0105 mov     ah,2             ;BIOS service 2
1415:0107 mov     dx,0a00          ;Row 10, column 0
1415:010A mov     bh,0             ;Page 0
1415:010C int     10               ;Position cursor
1415:010E mov     ax,0e00          ;BIOS service 0E, first character 0
1415:0111 int     10               ;Print character
1415:0113 inc     al               ;Next
1415:0115 cmp     al,80            ;Done?
1415:0117 jnz     0111             ;No: loop again
1415:0119 int     20               ;Yes: back to DOS
1415:011B
-
```

Figure 5–2.
DEBUG listing for Program 5.1.

```
☺☻♥♦♣
►◄↕‼¶§▬↨↑↓→←∟↔▲▼ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]
^_`abcdefghijklmnopqrstuvwxyz{|}~⌂
```

Figure 5–3.
Sample output for Program 5.1.

```
mov   ax, 0002
int   10
mov   ah, 2
mov   dx, 0a00
mov   bh, 0
int   10
mov   ax, 0e00
int   10
inc   al
cmp   al, 80
jnz   0111
int   20
```

Figure 5–4.
Instruction listing for Program 5.1 without comments.

5.2 Program 5.2: BCD to ASCII Conversion

Introduction

BCD numbers consist of the binary combinations 0000 through 1001. One byte can, therefore, store two BCD numbers. This is referred to as a *packed* BCD number. In ASCII, the numbers 0 through 9 have the hexadecimal codes 30H through 39H. Therefore, one byte stores one ASCII number. Program 5.2 converts the packed BCD number in register AL to two ASCII numbers, which are then passed to the BIOS video service and displayed on the system monitor. For example, if AL = 12H, the screen displays 12. This is accomplished by separating the two BCD numbers and adding 30H (*the ASCII bias*) to each digit. DEBUG is again used to write and test the program, which will run on any 80x86 DOS computer.

In this section we:

- Construct a logical mask to force selected bits of an operand low.
- Use the rotate and logical AND instructions to unpack a packed BCD number.
- Convert a BCD number to its ASCII equivalent.

New Instructions

AND Operand1, Operand2. The AND instruction functions like multiple AND gates, performing a bit-by-bit AND of the two operands, which may be a CPU register or memory location and another CPU register or immediate value. Operand1 stores the result. For example, if AL = 6DH and BL = 27H, the instruction AND AL, BL will produce the following result:

```
  AL = 6DH = 0110 1101
• BL = 27H = 0010 0111
-------------------------
  AL = 25H = 0010 0101
```

The AND instruction is often used as a "mask" or filter. In this way, only selected bits are allowed to pass through the filter. For example, suppose we want to examine only

bits 4–7 of register AL. We accomplish this by ANDing AL with 1111 0000 (F0H). Now if AL = 6DH:

```
  AL   = 6DH = 0110 1101
• mask = F0H = 1111 0000
-------------------------
  AL   = 60H = 0110 0000
```

Notice how the mask allows bits 4–7 of register AL to pass through the filter unchanged. Bits 0–3, however, are all forced to be zero.

One drawback to the AND instruction is that the data being tested is *changed* by the AND operation. For example, if we want to input data from port 27H and test if bit 7 is high we might use the sequence:

```
IN    AL,27H    ;Get input data
AND   AL,80H    ;Mask all bits except bit 7
JNZ   A1        ;Goto A1 if bit 7 is high, else continue
```

Now, however, if processing requires that we also test for bit 0 of the input data to be low, there is a problem. The AND instruction has "destroyed" the original data. The solution is to use the TEST instruction. TEST works exactly like the AND instruction but only sets the flags, the contents of the tested register are unchanged. In this example the program becomes:

```
IN      AL,27H    ;Get input data
TEST    AL,80H    ;Test bit 7
JNZ     A1        ;Go to A1 if bit 7 is high, else continue
TEST    AL,01H    ;Test bit 0
JZ      A2        ;Go to A2 if bit 0 is low, else continue
```

There are three other logical instructions: OR, exclusive-OR, and NOT. These are also performed bit by bit between the two operands. The OR function can be used to set selected bits, the exclusive-OR to invert selected bits, and the NOT to invert all bits (see Analysis and Design Question 5.3).

ROR Operand,CL. The ROR instruction rotates all of the bits of the operand, which may be a CPU register or memory location, right CL times. That is, if CL = 4, four rotate rights are performed. (Figure 4.5 diagrammed this instruction.) Beginning with the 386 processor, an immediate value may be specified. For example, if AL = 64H and the instruction ROR AL,2 is given, the result is:

```
0110 0100 -> first rotate: 0011 0010
0011 0010 -> second rotate: 0001 1001
```

Notice that rotating right is the same as dividing the operand by 2. Thus, 64H (100) when rotated right twice becomes 19H (25).

Other similar instructions are ROL, which rotates the operand left, and RCL and RCR, which rotate the operand through the carry flag (see Figure 4–5). The shift instructions are also similar, but the end bits are allowed to "drop off" and do not recirculate (again, see Figure 4–5).

ADD Operand1, Operand2. The add instruction is used to add two operands, which may be a CPU register or memory location and another CPU register or immediate value.

Operand1 stores the result. By selecting the appropriate operand size, 8-, 16-, and 32-bit additions can be performed. For example, if register AX = F3D9H and the instruction ADD AX,16A2H is given, the result is:

```
  AX = F3D9H
+      16A2H
-------------
       0A7BH and CF = 1
```

Because the addition of the last two bits generated a carry (1 + F = 10), the carry flag is set. The ADC instruction is similar but includes the carry flag as part of the addition. For example, consider the 64-bit addition problem EDX:ECX = EDX:ECX + EBX:EAX.

```
ADD ECX,EAX    ;Add the low-order registers
ADC EDX,EBX    ;Add the high-order registers plus carry from
                the low-order addition
```

Problem Statement

Write an 80x86 program that displays the packed BCD number in register AL on the system video monitor.

Discussion

Program Outline. Figure 5–5 outlines the program solution. The packed BCD number is assumed to be in register AL. The first number to be displayed should be the most significant digit (MSD). It is found by masking the least significant digit (LSD) and then rotating the (MSD) into the LSD position. The result is converted to ASCII by adding 30H. The BIOS video service is then called to display the result.

Extracting and converting the LSD is simpler, as no rotate operation is required. Note the importance of saving the original number: The MSD conversion process "wipes out" the AL register.

Program Listing. Figure 5–6 shows the annotated machine language program. It follows the outline exactly.

Testing the Program. To test the program, load register AL with a packed BCD number. For example, to load AL with 12H (and AH with 00), give the DEBUG command ***rax*** and then enter ***0012.*** Now type ***g*** for GO. The number 12 should appear followed by the usual *Program terminated normally* statement.

In the next section, we will convert Program 5.2 into a subroutine that can be called by any program requiring a packed BCD to ASCII conversion.

Figure 5–5.
Outline for Program 5.2.

I. Get Most Significant Digit (MSD), Convert to ASCII, and Display	
A. Save original number	`mov bl,al`
B. Force bits 0-3 low	`and al,f0h`
1. AND AL with F0H	
C. Rotate MSD into LSD position	`mov cl,4`
1. Four rotate rights	`ror al,cl`
D. Convert to ASCII	`add al,30h`
1. Add 30H to result	
E. Display result	`mov ah,0eh`
1. BIOS INT 10, video service 0EH	`int 10h`
i. AH = 0EH	
ii. AL = character to display	
II. Get Least Significant Digit, Convert to ASCII and Display	
A. Recover original number	`mov al,bl`
B. Force bits 4–7 low	`and al,0fh`
1. AND AL with 0FH	
C. Convert to ASCII	`add al,30h`
1. Add 30H to result	
D. Display result	`int 10h`
1. BIOS INT 10, video service 0EH	
i. AH = 0EH	
ii. AL = character to display	
III. Return to DOS	
A. INT 20	`int 20h`

Figure 5–6.
Listing for Program 5.2.

```
10B9:0100 mov     bl,al          ;Save original number
10B9:0102 and     al,f0          ;Force bits 0-3 low
10B9:0104 mov     cl,4           ;Four rotates
10B9:0106 ror     al,cl          ;Rotate MSD into LSD position
10B9:0108 add     al,30          ;Convert to ASCII
10B9:010A mov     ah,0e          ;BIOS video service 0E
10B9:010C int     10             ;Display character
10B9:010E mov     al,bl          ;Recover original number
10B9:0110 and     al,0f          ;Force bits 4-7 low
10B9:0112 add     al,30          ;Convert to ASCII
10B9:0114 int     10             ;Display character
10B9:0116 int     20             ;Return to DOS
```

5.3 Program 5.3: Two-Digit BCD Adder

Introduction

When adding BCD numbers, care must be taken to avoid results that produce invalid BCD numbers. For example, 1001 + 0001 yields 1010—the correct binary result, but an invalid BCD number. In Program 5.3 we develop an 80x86 program that inputs two two-digit BCD numbers from the system keyboard, adds the numbers, and displays the results on the system video display. For example, typing ***64*** + ***89*** = will produce the result 153 on the next line. This is accomplished by inputting the ASCII numbers to be added to a memory buffer, converting the numbers to packed BCD, performing the addition, adjusting the result for decimal, and finally sending the sum to Program 5.2, which displays the result.

In this section we:

- Input and store in a memory buffer a string of characters from the system keyboard.
- Convert an ASCII number to BCD
- Add two packed BCD numbers, ensuring that the sum remains a valid BCD number.
- Link two programs by calling the second as a subroutine.

New Instructions

SUB Operand1, Operand2. The subtract instruction is used to subtract two operands, which may be a CPU register or memory location and another CPU register or immediate value. Operand1 stores the result. By selecting the appropriate operand size, 8-, 16, and 32-bit subtractions can be performed. For example, if register EAX = 12345678H and the instruction SUB EAX,246E7A10H is given, the result is:

```
  EAX = 1234 5678H
-       246E 7A10H
        EDC5 DC68H and CF = 1
```

Because the subtraction of the last two bits required a borrow, the carry flag is set. Like the ADC instruction discussed in Section 5.2, SBB is used with multiple subtractions. If the carry flag is set, this instruction subtracts 1 from the result (in effect accounting for the borrow in the previous subtraction). For example, assume the 32-bit subtraction DX:CX − BX:AX is to be performed with DX:CX = 0002:9000 and BX:AX = 0001:A000:

```
SUB    CX,AX    ;First subtract AX from CX and set flags.
                ;9000 - A000 = F000 and CF = 1 (a borrow).
SBB    DX,BX    ;Now subtract BX from DX with borrow.
                ;0002 - 0001 - 1 = 0000
```

The result, stored in DX:CX, is 0000:F000.

DAA. This instruction performs a decimal adjust for addition on the quantity in register AL. As such, it is used to ensure that the addition of two BCD numbers contains a valid BCD result. It works by noting if the result of an addition has resulted in a carry out of bit 7 (CF = 1) or bit 3 (AF = 1), or if the high- or low-order nibbles of the result are greater than 9. If either condition is met, the DAA instruction automatically adds 6 to the corresponding nibble. Consider the following example:

```
  47H
+ 24H
  6BH    ;AF and CF = 0, but low-order nibble >9
+ 06H    ;So add 6
  71H    ;Corrected result (47 + 24 = 71)
```

Sometimes both digits will need to be corrected:

```
  47H
+ 59H
  A0H    ;AF = 1 and high-order nibble >9
+ 66H    ;So add 66H
1 06H    ;Corrected result (47 + 59 = 106)
```

Note that in this last result CF = 1, representing the carry from the addition of A + 6. When subtracting BCD numbers, the DAS instruction can be used with similar effect. The following two points should be noted when using the DAA and DAS instructions:

1. Both instructions are restricted to adjusting only the number in register AL.
2. Neither instruction will convert a binary number to BCD: They should only be used following an addition (DAA) or subtraction (DAS) operation.

CALL Destination. This instruction transfers control to another program or procedure at the destination address. In Real Mode, the address may be within the current code segment (a *near* call) or to a procedure in a different code segment (a *far* call). In Protected Mode, the destination may specify a call gate to allow transfers between different privilege levels.

Procedures (or subroutines) simplify the programming task because they allow sections of code to be used over and over within other programs. Indeed, the concept of modular programming is one in which a program is broken into small modules, or procedures, which are then called by the main program. This allows several programmers to work on the same project, each with responsibility for one module only. Another advantage is that libraries of procedures can be maintained. Programmers can "pick and choose" from these libraries and avoid reinventing the wheel with each new program they write.

RET. When a CALL instruction is executed, the processor first pushes the address of the instruction immediately following the CALL (the return address) onto the stack (see box on page 174 for a review of stack operation). This ranges from two bytes for a near call (only the 16-bit instruction pointer need be saved) to four bytes for a Real Mode far call (2 bytes each for IP and CS) to six bytes for a Protected Mode far call (4 bytes for IP and 2 for CS). With the return address safely stored on the stack, control then trans-

fers to the called procedure. To return control to the calling program, an RET (return from subroutine) instruction is given. This pops the top of the stack back into IP (and CS, for a far call). It is important, therefore, that each CALL instruction be balanced with a corresponding RET instruction. In addition, if multiple CALLs are performed (nested subroutines), they must be returned in the reverse order in which they were called.

New DOS Features

DOS INT 21H, Function 0AH (Buffered Keyboard Input). Function 0AH is used to input characters from the keyboard to a memory buffer whose address is specified in DS:DX. The first byte of the buffer should hold the total buffer size. The second byte is updated by DOS and represents the actual number of bytes input to the buffer. The keyboard characters themselves are stored beginning with the third byte of the buffer. DOS closes the buffer when the return character (0DH) is input.

Function 0AH is useful when complete strings of keyboard input are required rather than the single bytes input by BIOS INT 16H service 00. In addition, input characters are automatically echoed to the screen, and the backspace key can be used to edit input before pressing return.

Problem Statement

Write an 80x86 program that adds two packed BCD numbers input from the system keyboard and computes and displays the result on the system video monitor. Data should be input in the form: ***64+89=***. The answer, *153,* should appear on the following line.

Discussion

Program Outline. Figure 5–7 is an outline for Program 5.3. Four major tasks are identified:

1. Get two two-digit ASCII numbers to be added and convert to BCD.
2. Add the numbers.
3. Display decimal result in ASCII.
4. Return to DOS.

The user is expected to enter the two two-digit numbers in the form:

64+89=

The result will appear on the line below. Figure 5–8 shows how the input buffer would look for the above numbers. The first part of the program sets up an eight-byte input buffer and then calls function 0AH. Because this buffer will be placed at the end of the program, and this address is not yet known, we use the name *buffer_address* to temporarily mark this location. The cursor is then positioned on the next line by outputting a line feed character via BIOS service 0E.

Figure 5–9 shows how the first two ASCII numbers are converted to packed BCD. In the first step, 30H is subtracted from each number. Next, the LSD is rotated left four bits. Finally, the 16-bit MSD/LSD is rotated four bits right. This packs the two BCD numbers into a single register.

Returning to the outline, note that register SI is pointed at the base of the input buffer (*buffer_address*). The *indexed-plus-displacement* addressing mode is then used to

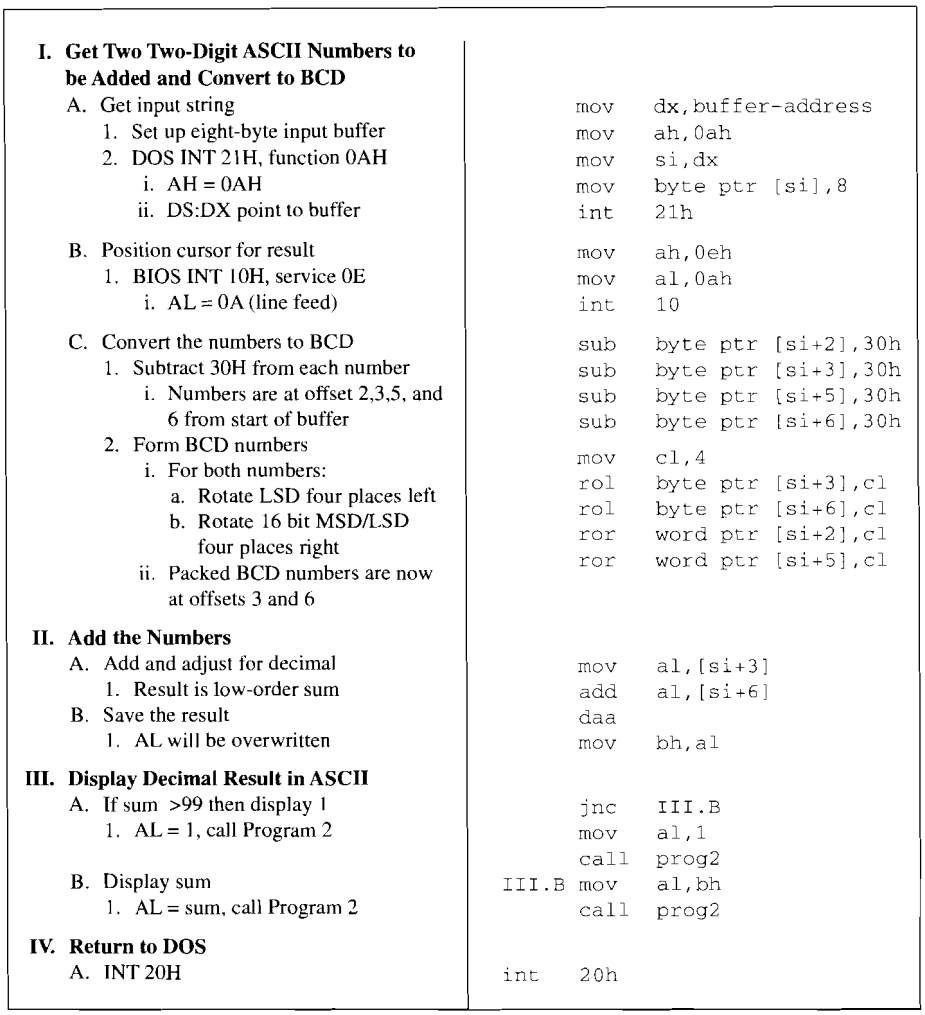


Figure 5–7.
Outline for Program 5.3.

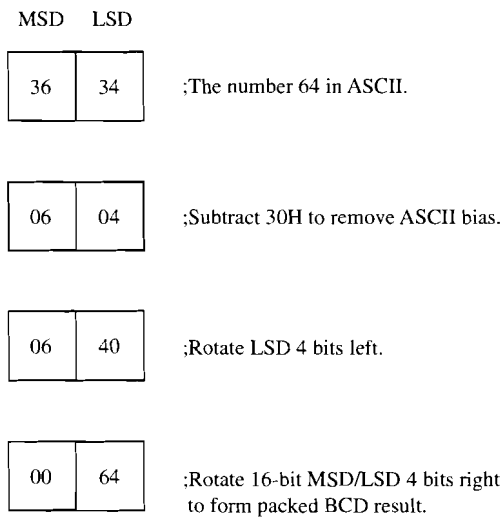
select each digit one by one, subtract the ASCII bias (30H), and perform the rotates. In this way, the absolute address of each byte in the buffer need not be given.

Once the numbers have been packed, they can be added and adjusted for decimal. Remember, however, that this must take place in register AL. Thus, the first number (at SI + 3) is brought into register AL and then added to the second (at SI + 6). If a carry is generated, the result must be greater than 99 and the program first outputs a 1, and then the sum. This test occurs with the JNC instruction. The numbers to be displayed are then loaded into AL and Program 5.2 is called as a subroutine. This program converts the packed BCD number in AL to ASCII and displays it.

Offset:	0	1	2	3	4	5	6	7
	Working size of buffer in bytes	Number of bytes in the buffer	Digit 1 Number 1 6	Digit 2 Number 1 4	Operator +	Digit 1 Number 2 8	Digit 2 Number 2 9	Operator =
ASCII Value			36H	34H	2BH	38H	39H	3DH

Figure 5–8.
Eight-byte input buffer storing the problem "64 + 89 =."

Figure 5–9.
Three steps are required to convert a two-digit ASCII number to packed BCD.



Program Listing. Figure 5–10 provides the complete listing for Program 5.3 with the buffer, conditional jump, and call addresses resolved.[4] The input buffer is placed after the program at address 0160 (the program ends in 015D).[5] Program 5.2 has been added to the bottom of the program, beginning at address 0147. The two call instructions then reference this address. Also note that the last instruction in the Program 5.2 procedure has been changed to an RET so that control returns to Program 5.3 after each call.

Program Trace. When testing a program such as this, it is a good idea to single-step the program to avoid a program crash (a loss of program control requiring the computer to be rebooted). Figure 5–11 illustrates the process using DEBUG's trace (*t*) command. Comments have been inserted in bold print to make the trace easier to follow. The display command (**d**) is used to view the input buffer at appropriate times.

[4]As mentioned in Chapter 4, this is not as simple as it looks. "Dummy" addresses must be inserted when the program is first entered. The DEBUG listing is then studied to determine the actual addresses required, and the program edited accordingly.

[5]Recall that a COM program requires that all of the segment registers have the same value. Therefore, address 0160 in the data segment is the same as address 0160 in the code segment.

```
10B9:0100 mov     dx,0160                       ;Point DX at input buffer
10B9:0103 mov     ah,0a                         ;DOS function 0AH
10B9:0105 mov     si,dx                         ;Point Si at input buffer
10B9:0107 mov     byte ptr [si],8               ;8 byte buffer
10B9:010A int     21                            ;Get the two numbers
10B9:010C mov     ah,0e                         ;BIOS video service
10B9:010E mov     al,0a                         ;ASCII line feed
10B9:0110 int     10                            ;
10B9:0112 sub     byte ptr [si+2],30            ;Convert each digit to BCD
10B9:0116 sub     byte ptr [si+3],30
10B9:011A sub     byte ptr [si+5],30
10B9:011E sub     byte ptr [si+6],30
10B9:0122 mov     cl,4                          ;Four rotates
10B9:0124 rol     byte ptr [si+3],cl            ;Form LSD
10B9:0127 rol     byte ptr [si+6],cl
10B9:012A ror     word ptr [si+2],cl            ;Add to MSD
10B9:012D ror     word ptr [si+5],cl
10B9:0130 mov     al,[si+3]                     ;Fetch first BCD number
10B9:0133 add     al,[si+6]                     ;Add to second
10B9:0136 daa                                   ;Keep results decimal
10B9:0137 mov     bh,al                         ;Save results
10B9:0139 jnc     0142                          ;Check for hundredths digit
10B9:013B mov     al,1                          ;Set hundredths digit
10B9:013D call    0147                          ;Display it
10B9:0140 mov     al,bh                         ;Recover low order result
10B9:0142 call    0147                          ;Display low order result
10B9:0145 int     20                            ;Return to DOS


;*********************************************************************
;  This is Program 5.2 modified to end with an RET instruction
;*********************************************************************
10B9:0147 mov     bl,al                         ;Save original number
10B9:0149 and     al,f0                         ;Force bits 0-3 low
10B9:014B mov     cl,4                          ;Four rotates
10B9:014D ror     al,cl                         ;Rotate MSD into LSD
10B9:014F add     al,30                         ;Convert to ASCII
10b9:0151 mov     ah,0e                         ;BIOS video service 0E
10B9:0153 int     10                            ;Display character
10B9:0155 mov     al,bl                         ;Recover original number
10B9:0157 and     al,0f                         ;Force bits 4-7 low
10B9:0159 add     al,30                         ;Convert to ASCII
10B9:015B int     10                            ;Display character
10B9:015D ret                                   ;Return to calling program

10B9:0160                                       ;Input buffer begins here
```

Figure 5–10.
Listing for Program 5.3. Note that Program 2 is used as a subroutine. The input buffer begins at address 0160.

```
C:\PROGRAMS>debug prog3.com
-u100      ;Check the first few lines to verify that the program has loaded.
10DC:0100 BA6001        MOV     DX,0160
10DC:0103 B40A          MOV     AH,0A
10DC:0105 89D6          MOV     SI,DX
10DC:0107 C60408        MOV     BYTE PTR [SI],08
10DC:010A CD21          INT     21
10DC:010C B40E          MOV     AH,0E
10DC:010E B00A          MOV     AL,0A
10DC:0110 CD10          INT     10
10DC:0112 806C0230      SUB     BYTE PTR [SI+02],30
10DC:0116 806C0330      SUB     BYTE PTR [SI+03],30
10DC:011A 806C0530      SUB     BYTE PTR [SI+05],30
10DC:011E 806C0630      SUB     BYTE PTR [SI+06],30
-g 112     ;Run at full speed to address 0112 (breakpoint).
64+89=     ;Enter two test numbers in the expected format.

AX=0E0A  BX=0000  CX=005E  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=0112   NV UP EI PL NZ NA PO NC
10DC:0112 806C0230      SUB     BYTE PTR [SI+02],30                DS:0162=36
-
-d160 110  ;Display the input buffer and note that the numbers are as expected.
10DC:0160  08 06 36 34 2B 38 39 3D-0D D9 21 D8 43 D9 21 D8   ..64+89=..!.C.!.
-
-t         ;Begin tracing instructions one at a time.

AX=0E0A  BX=0000  CX=005E  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=0116   NV UP EI PL NZ NA PE NC
10DC:0116 806C0330      SUB     BYTE PTR [SI+03],30                DS:0163=34
-t

AX=0E0A  BX=0000  CX=005E  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=011A   NV UP EI PL NZ NA PO NC
10DC:011A 806C0530      SUB     BYTE PTR [SI+05],30                DS:0165=38
-t

AX=0E0A  BX=0000  CX=005E  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=011E   NV UP EI PL NZ NA PO NC
10DC:011E 806C0630      SUB     BYTE PTR [SI+06],30                DS:0166=39
-t

AX=0E0A  BX=0000  CX=005E  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=0122   NV UP EI PL NZ NA PE NC
10DC:0122 B104          MOV     CL,04
-
-d160 110   ;Display the input buffer again and note the ASCII bias has been removed.
10DC:0160  08 06 06 04 2B 08 09 3D-0D D9 21 D8 43 D9 21 D8   ....+..=..!.C.!.
-
-t          ;Next we trace the two rotate left instructions.
```

(continued on next page)

Figure 5–11.
Using DEBUG's trace command to single-step Program 5.3. Comments are shown in boldface print.

```
AX=0E0A  BX=0000  CX=0004  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=0124   NV UP EI PL NZ NA PE NC
10DC:0124 D24403        ROL     BYTE PTR [SI+03],CL               DS:0163=04
-t

AX=0E0A  BX=0000  CX=0004  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=0127   NV UP EI PL NZ NA PE NC
10DC:0127 D24406        ROL     BYTE PTR [SI+06],CL               DS:0166=09
-t


AX=0E0A  BX=0000  CX=0004  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=012A   OV UP EI PL NZ NA PE NC
10DC:012A D34C02        ROR     WORD PTR [SI+02],CL               DS:0162=4006
-
-d160 110   ;Check the input buffer again.  Note the LSDs have been rotated left.
10DC:0160  08 06 06 40 2B 08 90 3D-0D D9 21 D8 43 D9 21 D8   ...@+..=..!.C.!.
-
-t        ;Now trace the two rotate rights and move the first number to AL.

AX=0E0A  BX=0000  CX=0004  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=012D   OV UP EI PL NZ NA PE NC
10DC:012D D34C05        ROR     WORD PTR [SI+05],CL               DS:0165=9008
-t        ;

AX=0E0A  BX=0000  CX=0004  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=0130   OV UP EI PL NZ NA PE CY
10DC:0130 8A4403        MOV     AL,[SI+03]                        DS:0163=64
-t

AX=0E64  BX=0000  CX=0004  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=0133   OV UP EI PL NZ NA PE CY
10DC:0133 024406        ADD     AL,[SI+06]                        DS:0166=89
-
-d160 110  ;The two BCD numbers have been formed.  AL holds the first number (64).
10DC:0160  08 06 00 64 2B 00 89 3D-0D D9 21 D8 43 D9 21 D8   ...d+..=..!.C.!.
-
-t

AX=0EED  BX=0000  CX=0004  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=0136   NV UP EI NG NZ NA PE NC
10DC:0136 27            DAA
-t       ;Add the two numbers.  Note result is not decimal.

AX=0E53  BX=0000  CX=0004  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=0137   NV UP EI PL NZ AC PE CY
10DC:0137 88C7          MOV     BH,AL
-t      ;But the DAA fixes that problem.  Note carry flag is set.
```

Figure 5–11.
(*continued*)

```
AX=0E53  BX=5300  CX=0004  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=0139   NV UP EI PL NZ AC PE CY
10DC:0139 7307          JNB     0142
-t      ;The result is saved in BH.

AX=0E53  BX=5300  CX=0004  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=013B   NV UP EI PL NZ AC PE CY
10DC:013B B001          MOV     AL,01
-t      ;The jump is not taken and AL is therefore set to 1.

AX=0E01  BX=5300  CX=0004  DX=0160  SP=FFFE  BP=0000  SI=0160  DI=0000
DS=10DC  ES=10DC  SS=10DC  CS=10DC  IP=013D   NV UP EI PL NZ AC PE CY
10DC:013D E80700        CALL    0147


-g      ;Run the rest of the program at full speed and the correct result appears.
0153
Program terminated normally
-
```

Figure 5–11.
(continued)

Self-Review 5.3 (Answers on page 268)

5.3.1 For the instruction sequence below, determine the contents of register AL (a) after the add instruction and (b) after the DAA instruction.

```
MOV    AL,28H
ADD    AL,35H
DAA
```

5.3.2 When using DOS INT 21 function 0AH (buffered keyboard input), if DX = 0200H, the user keyboard characters are stored beginning at offset address _____.

5.3.3 One byte can store _____ ASCII character(s) or _____ BCD digit(s).

5.4 Program 5.4: 80x86 Music Machine

Introduction

Today's PC is a multimedia computer capable of displaying full-motion video with stereo sound. This is accomplished via the addition of CD-ROM drives and sound cards with on-board digital signal processors. Program 5.4 shows how to create computer music through a much less sophisticated method: the PC's integrated 8253/54 timer and onboard

CHAPTER 5

80x86 Programming Techniques

About this Chapter

Chapter 5 is a key chapter as it is here that your students will learn how to select 80x86 instruction mnemonics to form useful programs. Seven different programming examples are presented, with the new instructions and programming techniques highlighted. If additional practice is required, the Analysis and Design Questions section of the chapter presents an additional 19 new or modified versions of these programs.

My experience has always been that students do not like to plan out their machine language programs. Like BASIC programmers, they use a method of "trial and error." Unfortunately, in a programming environment like Debug, this can be very frustrating. I have found the outline method of program development presented in this chapter to work well. Students prefer developing their programs on a computer to drawing flowcharting symbols, and the ease with which ideas can be inserted, deleted, promoted, and demoted allows them to focus on the program's logic. I hope you will try this method.

Lab Projects

1. The Analysis and Design Questions section of the chapter poses several new or modified programming challenges that can be used as lab projects. The table below provides a brief description each project.

Problem No.	New	Modified
5.1		Program 5.1 modified to print the extended ASCII character set.
5.2	Similar to Program 5.1 but displays green text on a blue background. The program is given but its operation must be analyzed.	
5.5	Simple three line program to produce a BEEP (BEEP.COM)	
5.6		Calls Program 5.2 twice to display the packed BCD number in register AX.
5.7	Good project that explores the DAA instruction to build a program that adds two four-digit BCD numbers.	
5.9		Similar to Program 5.4 but calculates the frequency of each new note to be played instead of using a lookup table.
5.10	DOS batch file that uses the if errorlevel command.	
5.11	Simple program that waits for a keystroke and then returns control to DOS (YES.COM).	
5.12	DOS batch file using YES.COM to confirm a user's choice to format a disk (see also Problem 5.19).	
5.14	Program that uses the LOOP and LODSW instructions to search a block of memory for a particular 16-bit word. (See also Problem 5.26)	

5.15		Changes Program 5.6 so that only the drive letters A or B are accepted.
5.16	Good interfacing project requiring a hardware and software solution. The XLAT instruction is used to look up the seven-segment code of the number input via a DIP switch.	
5.17		Simple change to Program 5.7 to allow 0 to 9.9s time delays.
5.18		More complex change to Program 5.7 to allow time delays up to 999s.
5.19		Change to Problem 5.12 to allow a 5s caution message to appear.
5.21		Modification to Program 5.4 so that each note includes a 16-bit time delay value. A complete "song" is then stored in memory and played.
5.24		Program to fill a block of memory using the STOSD instruction.
5.25		Program that locates the first occurance of a particular letter in a string using the SCASB instruction.
5.26		Program to search a block of memory for a particular 16-bit word using the SCASW instruction.

2. Program 5.7 requires access to the 32-bit registers of the 80x86 processors. Analysis and Design question 5.20 explores the operation of this program using Debug32's procedure trace command.
3. Analysis and Design question 5.23 explores the BIOSDATE.COM program by loading it with DEBUG and tracing program execution. Because two diffewrent data segments are used this can be instructive.

Self-Test Answers

1. AH
2. Sequence 2
3. 128
4. 50H
5. 17H
6. 79 - 82 = F7 (CF=1) -13 = E4 -1 = E3
7. (a) CBH, (b) 31H
8. (b)
9. LODSB
10. (a) port address > 255, (b) destination must be accumulator, (d) immediate data not allowed
11. i15
12.
```
in   al,57     ;Read data
or   al,c0     ;Set bits 6 and 7
out  57,al     ;Output data
```
13. Insert
14. 21H, AL
15. (b)

16. PUSH BX
17. 4096, DS:0300H, 200H
18. LOOPE
19. 0100H
21. ROR EAX,10H
22. 00B7 1B00
23. SCAS, REP
24. MOV AX,1000H
    MOV DS,AX
25. 1F4 (500), 9000:0300, 8400:7000

Analysis and Design Questions

5.1

```
mov     ax,0002         ;BIOS service 0
mov     al,2            ;Video mode 2
int     10
mov     ah,2            ;BIOS service 2
mov     dx,0a00         ;Row 10, column 0
mov     bh,0            ;Page 0
int     10              ;Home cursor
mov     ax,0e80         ;BIOS service 0E
mov     al,80           ;First character is 80H
int     10              ;Print character
inc     al              ;Next
jnz     0113            ;More characters? Loop again
int     20              ;When AL=0 program is done
```

5.2

```
mov     ah,0            ;BIOS service 0
mov     al,2            ;Video mode 2
int     10              ;Set video mode and clear screen
mov     ax,0920         ;Service 9, ASCII space
mov     bl,1a           ;Bright green on blue
mov     cx,7d0          ;Fill screen (80 x 25)
int     10              ;Set screen color
mov     ah,2            ;BIOS service 2
mov     dx,0a00         ;Row 10, column 0
mov     bh,0            ;Page 0
int     10              ;Set cursor
mov     ah,0e           ;BIOS servce 0E
mov     al,0            ;First character is 0
int     10              ;Print character
```

```
inc     al              ;Next
cmp     al,80           ;Done?
jnz     011d            ;No: loop again
int     20              ;Yes: back to DOS
```

(a) Video mode 2: 80 x 25 16-color text.

(b) mov bl,1a (1 = blue background, A = bright green text).

(c) The screen has 80 x 25 = 2000 characters = 07D0H. This instruction causes all of these locations to be written with the character in AL (space) in the specified color.

(d) The '$' symbol is written to the entire screen (instead of a space).

(e) Change the mov bl,1a to mov bl,74.

5.3 AL=38H; AH=EFH; BL=8BH; BH=15H

5.4

```
-rax
AX 0000
:26
```

Load register AL with 26 - the number to be converted.

```
-r
AX=0026  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=0100   NV UP EI PL NZ NA PO NC
0E6E:0100 88C3          MOV     BL,AL
-t
```

The number in AL is copied to BL.

```
AX=0026  BX=0026  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=0102   NV UP EI PL NZ NA PO NC
0E6E:0102 24F0          AND     AL,F0
-t
```

The low four bits of AL are forced to be 0.

```
AX=0020  BX=0026  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=0104   NV UP EI PL NZ NA PO NC
0E6E:0104 B104          MOV     CL,04
-t
```

CL is loaded with four to prepare for the four rotates.

```
AX=0020  BX=0026  CX=0004  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=0106   NV UP EI PL NZ NA PO NC
0E6E:0106 D2C8          ROR     AL,CL
```

The high four bits of AL are rotated into the low four bits.

```
AX=0002  BX=0026  CX=0004  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=0108   NV UP EI PL NZ NA PO NC
0E6E:0108 0430          ADD     AL,30
-t
```

The ASCII offset of 30 is added to AL forming the ASCII character.

```
AX=0032  BX=0026  CX=0004  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
```

```
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=010A   NV UP EI PL NZ NA PO NC
0E6E:010A B40E          MOV     AH,0E
-t
AH is loaded with the video service code 0E.
AX=0E32  BX=0026  CX=0004  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=010C   NV UP EI PL NZ NA PO NC
0E6E:010C CD10          INT     10
-p
2 ← The number 2 appears on the screen.
AX=0E32  BX=0026  CX=0004  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=010E   NV UP EI PL NZ NA PO NC
0E6E:010E 88D8          MOV     AL,BL
-t
The original number is revcovered from BL.
AX=0E26  BX=0026  CX=0004  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=0110   NV UP EI PL NZ NA PO NC
0E6E:0110 240F          AND     AL,0F
The high four bits are forced to 0.
AX=0E06  BX=0026  CX=0004  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=0112   NV UP EI PL NZ NA PE NC
0E6E:0112 0430          ADD     AL,30
-t
The ASCII offset is added.
AX=0E36  BX=0026  CX=0004  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=0114   NV UP EI PL NZ NA PE NC
0E6E:0114 CD10          INT     10
-p
6 ← The number 6 appears on the screen.
AX=0E36  BX=0026  CX=0004  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0E6E  ES=0E6E  SS=0E6E  CS=0E6E  IP=0116   NV UP EI PL NZ NA PE NC
0E6E:0116 CD20          INT     20
-p

Program terminated normally ← The program ends.
```

5.5

I. Print Control G	
A. BIOS INT 10H, service 0EH	
1. AH = 0EH	mov ah,oe
2. AL = 07 (control-G)	mov al,07
3. INT 10	int 10
II. Return to DOS	
A. DOS INT 20	int 20

```
mov  ah,oe     ;Video BIOS service to AH
mov  al,07     ;ASCII code for "Bell"
int  10        ;"Display the character" (ring the bell)
int  20        ;Return to DOS
```

5.7(a) SI = 0160 (b) Byte Ptr [SI+1] = 6 (c) Word Ptr [SI + 2] = 2800
(d) Word Ptr [SI + 5] = 7900 (e) AX = 0E07 (f) BH = 07 (g) CL = 04
(h) DX = 0160 (i) CF = 1

5.6

```
mov     dx,ax           ;Save original number
mov     al,dh           ;Get most significant BCD numbers
call    010e            ;Convert and display
mov     al,dl           ;Get least significant BCD numbers
call    010e            ;Convert and display
int     20              ;Return to DOS

;Program 2 Begins Here (address 010e)

mov     bl,al           ;Save original number
and     al,f0           ;Force bits 0-3 low
mov     cl,4            ;Four rotates
ror     al,cl           ;Rotate MSD into LSD position
add     al,30           ;Convert to ASCII
mov     ah,0e           ;BIOS service 0E
int     10              ;Display character
mov     al,bl           ;Recover original number
and     al,0f           ;Force bits 4-7 low
add     al,30           ;Convert to ASCII
int     10              ;Display character
ret                     ;Return to calling program
```

5.7

4-Digit BCD Adder	
I. Add AX and BX	
A. Add the LSDs without carry	
1. Add AL and BL and adjust the result for decimal	add al,bl daa
2. Save result	mov cl,al
B. Add the MSDs plus carry	
1. Move BH to BL and AH to AL	mov bl,bh mov al,ah
2. Add with carry AL and BL and adjust for decimal	adc al,bl daa
C. Assemble the result in AX	
1. Move MSD result to AH	mov ah,al
2. Move LSD result to AL	mov al,cl
II. Return to DOS	
A. INT 20H	int 20

5.8

Tone	Octave 1	Octave 2	Octave 3	Octave 4	Octave 5	Octave 6	Octave 7	Octave 8
C	11C33	8E89	4744	23A2	11D1	8E9	474	23A
C#	10D6A	86B5	4342	21A1	10D0	868	434	21A
D	FD4F	7F00	3F80	1FC0	FDF	7F0	3F8	1FC
D#	F040	77D1	3BE9	1DF4	EFB	77E	3BF	1DF
E	E242	7121	3890	1C48	E24	712	389	1C4
F	D5CD	6AA8	3564	1AB2	D59	6AC	356	1AB
F#	C9C5	64AB	3263	1932	C99	64C	326	193
G	BE3D	5F1F	2F8F	17C8	BE4	5F2	2F9	17C
G#	B344	59CE	2CE7	1671	B39	59D	2CE	167
A	A97D	54BE	2A5F	1530	A98	54C	2A6	153
A#	A02B	4FF2	2802	13FF	9FF	500	280	140
B	96D6	4B8B	25BD	12E1	970	4B8	25C	12E

Note: This table is based on the exact values shown in Table 5.1.

5.9

I. Initialize	
A. Set up note counter	
1. CL=9 ;8 notes + 1	mov cl,9
B. Get starting note	
1. Tone A, octave 1 = A97D	
2. Save in BX	mov bx,a97d
II. Each Time a Key is Pressed, Play a New Note	
A. Wait for a key to be pressed	
1. BIOS INT 16H, service 00	
i. AH=0	mov ah,0
ii. Keyboard character returned in AL (value = "don't care")	int 16
B. Play the note	
1. Program timer	
i. Port 43 = B6H	mov al,b6

ii. Port 42 = divisor low, then high	out 43,al
	mov al,bl
	out 42,al
	mov al,bh
	out 42,al

2. Enable speaker	
i. Set bits 0-1 of Port 61	in al,61 or al,3 out 61,al
C. Check for last note	
1. CL = CL - 1 ;Decrement note counter	dec cl
2. CL = 0?	jz III
i. Yes: Goto III	
ii. No: Prepare next note	
a. Shift right one bit ;Divide divisor by 2	shr bx,1
b. Goto IIA	jmp IIA
III. Return to DOS when all Eight Notes have been Played	
A. Turn off speaker	
1. Reset bits 0-1 of port 61	in al,61 and al,fc out 61,al
B. DOS INT 20H	int 20

5.10

```
@echo off

if errorlevel 0 if not errorlevel 1 goto a1
if errorlevel 1 if not errorlevel 2 goto a2
if errorlevel 2 if not errorlevel 3 goto a3
if errorlevel 3 if not errorlevel 4 goto a4
goto end

:a1
ver
```