


Part 1: Answers to Even- Numbered Review Questions



CHAPTER 1

2. One-to-one.
4. Hardware device driver. Telecommunications program. Computer game. Embedded systems program, such as a controller for an automobile's fuel injection system.
6. There is a close interaction between assembly language programs and the operating system, and assembly programs frequently call operating system functions.
8. Yes, later processors always run programs written for earlier processors within the same family.
10. The assembler produces an object file (extension .obj), and the linker reads the object file and produces an executable file (extension .exe).
12. Assembly language is too detailed, and it does not have the high-level logic structures built into the language. Large applications would take too long to write.
14. A device driver is a specialized program that acts as a communication link between a hardware device and other programs. It is executed by the operating system.
16. The op code is 05.
18. (a) yes (b) no
20. (a) B5h = 181 (b) 17h = 23
22. (a) CF57 (b) EC0A
24. X = 58h (01011000) Y = 59h (01011001)
26. +16 is 00010000, so -16 is 11110000. (This question should have been: "What is the binary representation of -16?")
28. A byte is the smallest.
30. In decimal, 2^{16} is 65,536.

CHAPTER 2

2. Protected mode prevents a bug in one program from affecting other programs running at the same time. Almost any applications can benefit. For example, a word processing program that crashes does not prevent a graphics drawing program from continuing to run at the same time. Or, two programs monitoring output information from separate machines in a manufacturing system can operate independently of each other.
4. Expanded memory was designed to allow large data blocks such as spreadsheets and documents to be kept in memory. This data could be kept in memory above the 640K barrier imposed by DOS.
6. Conventional memory can only be accessed by the system bus, because it is external to the CPU. The system bus is much slower than the CPU, forcing the latter to wait for requested data.
8. BX is called a *base register* and BP is called a *base pointer*.
10. The CS register holds the base location of all executable instructions (code) in a program. The DS register is the default base location for variables. The CPU calculates their locations using the segment value in DS. The SS register contains the base location of the stack. The ES register is an additional base location for memory variables.
12. The CX register is used as a loop counter.
14. The CS (code segment) register.
16. The BP register.
18. Control flags: Interrupt flag, Trap flag, and Direction flag.
20. The Carry flag signals unsigned overflow.
22. The Parity flag. If there is an even number of bits, the Parity is even. If there is an odd number of bits, the Parity is odd.
24. The interrupt vector table.
26. No, because the two adapters have different base addresses.
28. The stack segment is the base location of the stack, pointed to by the SS register.
30. Absolute.
32. The CS and IP registers.
34. Right after reading the config.sys file during the DOS boot sequence.
36. AX = 1234h, BX = 15FAh

CHAPTER 3

2. Assembly language mnemonic example: MOV
4. Yes, following the instruction.

6. Microsoft CodeView and Borland Turbo Debugger.
8. Code example:

```
add cx,bx
```
10. Segment and the offset.
12. Integer constants: 101b, 36, 0AB5h, 4327q.
14. COUNT = 20
16. Examples of string constants:

```
"Sample string"  
'Another sample'  
"Sample with 'embedded' quotes"  
'Sample with "embedded" quotes' (alternate form)
```
18. The first line should end with a \ character:

```
extrn Writestring:proc, Clrscr:proc, \  
Readint:proc
```
20. A *directive* is a statement that affects either the program listing or the way machine code is generated. An instruction is only executed at runtime.
22. Identifier names may contain the following special characters: ?, _ (underscore), @, \$, . (period).
24. Examples of labels:

```
L1:  
Start:  
Begin_loop:  
??0001:
```
26. (*quote*) Segments are the building blocks of programs: The code segment is where program instructions are stored; the data segment contains all variables, and the stack segment contains the program's runtime stack. The *extra* segment can be used for any purpose, usually data.
28. Code example:

```
mov ax,@data  
mov ds,ax
```
30. immediate.
32. The ENDP directive.
34. a. register, immediate
b. register, direct
c. register, immediate
d. register, indirect
36. Code example:

```
filename db "myfile.dta"
```

38. Code example:

```
array dw 500 dup(1000h)
```

40. To clarify what the book said, IP can never be a destination operand. Segment registers can be destination operands, only if the source operand is a general-purpose register such as AX, BX, CX, DX, SI, BP, or DI.

42. Code example:

```
arrayptr dw intarray
```

44. AX = 0101h

CHAPTER 4

2. The assembler can create two files, proj1.lst and proj1.obj.
4. The R indicates that the operand is segment relocatable.
6. No, test.obj must be linked to produce the file test.exe.
8. Memory models: tiny, small, compact, medium, large, huge. (Optional: flat).
10. Each page of the listing file contains 55 vertical lines and 132 vertical columns.
12. A label outside the current module is *external*.
14. Referring to the example, if the code segment starts at 18400h, the data segment starts at 18430 and the stack starts at 18440.

16.

	Type	Length	Size
var1	4	5	20
var2	2	10	20
var3	1	1	1
msg	1	1	1

18. PTR is required because the notation [si] does not indicate the memory operand's size.
20. No, the program will halt before label L2 is reached.
22. Errors:

Line 5: Change .data to .code.

Line 6: Insert the following two lines:

```
mov ax,@data
mov ds,ax
```

Line 8: Remove the ", 1" from the INC instruction.

Line 9: Transpose lines 9 and 10.

Line 12: Change to .data.

Line 13: Insert DW after value1.


Line 14: Insert DW after value2.

24. a. AX = 003Ah

b. BX = 0120h

c. BX = 003Ah

d. AX = 011Eh

e. ptr2 = 0124h 

CHAPTER 5

2. A software interrupt is activated when a program executes the INT instruction.
4. The numbers in the interrupt vector table are 32-bit pointers to interrupt service routines that are either part of the operating system or have been installed by software applications.
6. Because each computer has slightly different hardware, with devices made by different manufacturers. An application program would be enormously complex if it had to take into account all the possible hardware configurations of the computers running the program.
8. DOS function calls are more hardware compatible than BIOS interrupts, and they are part of the memory-resident part of DOS.
10. Redirection provides flexibility to users that want programs to read input/output data from various sources.
12. NUL.
14. Code example:

```
.data
message db 'Hello, world!$'      (1)
.code
mov ax,@data                    (2)
mov ds,ax
mov ah,9 ; string output        (3)
mov dx,offset message           (4)
int 21h
```
16. Use INT 16h when reading extended keys from the keyboard.
18. The chapter suggests using INT 21h function 6. (INT 16h can also be used.)
20. The carriage return (0Dh) character.

22. Choose any four colors, such as blue on white (0F1h).
24. Function 0Ah: Write a character only (no attribute) at the current cursor position.
26. Code example:
- ```
mov ah,1
mov ch,3
mov cl,4
int 10h
```
28. Code example:
- ```
mov ah,6
mov al,0
mov ch,0
mov cl,0
mov dh,0
mov dl,0
mov bh,70h          ; reverse video
int 10h
```
30. prog1 > prn
32. The stack grows downward when a value is pushed.

CHAPTER 6

2. Unsigned comparisons use the Carry and Zero flags.
4. The JCXZ instruction.
6. The JB instruction jumps base on unsigned comparison. The JL instruction is based on signed comparisons.
8. The JB instruction will be executed if the Carry flag is set.
10. Both AL and BL equal 3Fh. (The usual prologue code of `mov ax,@data \ mov ds,ax` is assumed to have been executed.)
12. AL = 01 and BL = 0CAh.
14. CX = 0268h, DX = 01FEh, SI = (offset val2), val2 = 3FD7h.

CHAPTER 7

2. The C and C++ languages contain bitwise shift operators, << and >>.
4. The ROL instruction copies the highest bit into the Carry flag and the lowest bit position. RCL, which is similar, copies the Carry flag into the lowest bit position.
6. The SAR instruction shifts to the right and replicates the Sign bit.
8. Code example:

```

    shr  al,1
    jnc  next
    or   al,10000000b
next:

```

10. Code example:

```
shr ax,4 ; multiply by 16
```

12. Code example:

```

; Multiply AL by 12: (AL * 8) + (AL * 4)
mov dl,al ; save copy of AL
shl al,3 ; multiply by 8
mov cl,al ; save result
mov al,dl ; get new copy of AL
shl al,2 ; multiply by 4
add cl,al ; add to sum
mov al,cl ; AL = AL * 12

```

14. ROR DL,4 (or, ROL DL,4)

16. Code examples:

```

; The following instructions shift AX one bit to the right
; and copy the shifted bit into the highest bit
; position of BX:
shr ax,1
jnc next
or bx,8000h
next:

```

```

; It is not possible to do the same with only a single
; instruction on the 80486. The closest match is the following:
; The SHRD instruction shifts BX one bit to the right
; and fills its highest bit position with the least significant
; bit of AX:
shrd bx,ax,1

```

18. Table:

Field	Mask	Shift	Width
fld1	F000h	0Ch	4
fld2	0D00h	9	3
fld3	01C0h	6	3
fld4	003Fh	0	6

20. Code example: Subtracting val1 from val2 will yield a negative number, of course. One improvement made to the sample code was to use a single index register to refer to all three operands:

```

mov ax,@data ; init data segment
mov ds,ax
mov cx,8 ; loop counter: 8 bytes
mov si,0 ; set index to start
clc ; clear carry flag
top:

```

```

mov    al,vall[si]    ; AL = vall
sbb   al,val2[si]    ; subtract val2 from AL
mov    result[si],al ; store the result
inc    si
loop  top

```

22. DX = 0002h and AX = 2200h.

24. After the division DX = 0004 (the remainder), and AX = 0123h (the quotient).

26. Code example that multiplies -5 by 3:

```

mov    al,-5
mov    bl,3
imul  bl
mov    vall,ax

```

28. Code example that divides 20000000h by 10h. We avoid a divide overflow condition by performing 16-bit division on the upper word of the quotient first. The remainder from this operation is added to the lower word of the quotient, and 16-bit division is performed again:

```

dividend dd 20000000h
result_lo dw 0
result_hi dw 0
.code
main proc
    mov    ax,@data                ; init data segment
    mov    ds,ax
    mov    dx,0
    mov    ax,word ptr dividend+2  ; high word of dividend
    mov    bx,10h                  ; divisor
    div    bx                      ; ax = quotient, dx = remainder
    mov    result_hi,ax            ; quotient, high word
    mov    ax,word ptr dividend    ; low word of dividend
    div    bx                      ; ax = quotient, dx = remainder
    mov    result_lo,ax           ; quotient, low word

```

CHAPTER 8

2. Code example:

```

mPushData macro
    push  ax
    push  bx
    push  cx
    push  dx
endm

mPopData macro
    pop   dx
    pop   cx
    pop   bx
    pop   ax
endm

```

4. Code example:

```

mReadArray macro Array, numWords
    local L1
    mov  cx,numWords
    mov  si,offset Array

L1:
    call Readint      ; input integer into AX
    call Crlf
    mov  [si],ax
    add  si,2
    loop L1
endm

```

6. Code example:

```

mTestJump macro dest,source,result,label
    test dest,source
    j&result label
endm

```

8. Code example. Arguments passed to this macro must be constants:

```

row = 12h
col = 16h

                                mLocate -2,20
                                (no code generated)

                                mLocate 10,20
0005 BB 0000                    1    mov    bx,0
0008 B4 02                      1    mov    ah,2
000A B6 14                      1    mov    dh,20
000C B2 0A                      1    mov    dl,10
000E CD 10                      1    int    10h

                                mLocate row,col
0010 BB 0000                    1    mov    bx,0
0013 B4 02                      1    mov    ah,2
0015 B6 16                      1    mov    dh,col
0017 B2 12                      1    mov    dl,row
0019 CD 10                      1    int    10h

```

10. Code example:

```

db 0,0,0,100
db 0,0,0,20
db 0,0,0,30

```

CHAPTER 9

-
2. The hexadecimal contents of ASCII 4096 would be 31h,30h,39h, and 36h.
 4. No, XLAT can only work with tables of 8-bit values.
 6. XLAT does not affect the flags.

8. AL = 37h, the value at offset 6 in the table. (The offset of ptr1, by the way, is irrelevant.)
10. To convert 302h (770d) from binary to ASCII decimal, we repeatedly divide the number by 10 and take each remainder as one of the resulting ASCII digits:

770 / 10 = 77 remainder 0 (convert to "0")

77 / 10 = 7 remainder 7 (convert to "7")

7 / 10 = 0 remainder 7 (convert to "7")

12. Converting ASCII "3F62" to binary requires multiplying each digit by 16:

(Sum	*	16)	+	digit	=	Sum
(0*	16)	+	3	=	3	
(3*	16)	+	F	=	63	
(63	* 16)	+	6	=	1014	
(1014	* 16)	+	2	=	16226	

14. DS = 1234h and SI = ABCDh.

16. Code Example:

```
.data
inputlist db 5,26,45,96,88,128
COUNT = ($ - inputList)

validchars db 32 dup(0) ; invalid chars: 0-31
            db 96 dup(0FFh) ; valid chars: 32-127
            db 128 dup(0) ; invalid chars: 128-255

.code
include library.inc

main proc
    mov ax,@data ; init data segment
    mov ds,ax

    mov bx,offset validchars
    mov si,offset inputlist
    mov cx,COUNT

getchar:
    mov al,[si]
    mov dl,al ; save the character
    xlat validchars ; look up in table
    cmp al,0FFh ; valid character?
    jne next ; no: get next character
    mov ah,2 ; display char in DL
    int 21h
next:
    inc si
    loop getchar
```

18. If AX equals 600h (1536d) before the loop starts, the value in DX each time the loop repeats is the remainder after dividing by 10: { 6, 3, 5, 1 }

CHAPTER 10

2. The longer string is truncated to the available space in the shorter string's storage area.
4. REP should be used. It automatically decrements CX.
6. DI will equal 000Bh.
8. Code example:

```

mov    ax,@data           ; init data segment
mov    ds,ax
mov    es,ax

lookforit:
std                    ; direction = down
mov    al,'@'            ; AL = byte to be found
mov    di,offset bigstring ; get string offset
add    di,biglen
dec    di                ; point to last byte
mov    cx,biglen
repnz scasb             ; repeat while NZ
inc    di                ; adjust DI when found

```

CHAPTER 11

2. Explanations of error codes:

ErrorNum/Function	Explanation
0Fh -> 0Eh	Trying to set default drive to nonexistent drive.
15h -> 36h	Diskette was not in drive.
0Fh -> 47h	Trying to get current directory of nonexistent drive.
03h -> 3Bh	Trying to set current directory to nonexistent path.
05h -> 39h	Trying to create subdirectory using the same name as an existing directory or file.
10h -> 3Ah	Trying to remove the current subdirectory.

4. Sector 33 (see Table 4).
6. It will take up 1 cluster, or 512 bytes (see Table 3).
8. The file has been deleted.
10. Date stamp for July 3, 1983:


```

year    = 0000011    (1983)

```

```
month    = 0111      (7)
day      = 00011     (3)
```

12. A value of 2Eh indicates that the filename starts with a period. The parent directory, for example, is indicated by a filename of "..". In DOS you move to the parent directory by typing the following command:

```
CD ..
```

14. File size = 00000020h, starting cluster = 0004h.

16. You need to adjust the stack pointer after calling INT 25h, with the following instruction:

```
add sp,2
```

18. A value of 18h indicates a volume or directory name, which the program does not display.

20. Code example: AX contains the previous cluster number, and DX contains the offset into the FAT. Calculate the new cluster number and place it in AX. We assume that DX contains the offset of the *new* cluster number, which might have been calculated as follows:

```
mov  dx,ax          ; copy the number
shr  dx,1           ; divide by 2
add  dx,ax          ; new cluster offset
```

Now we can use the offset in DX to obtain the new cluster number:

```
mov  bx,dx          ; use a base register
mov  dx,fattable[bx] ; DX = new cluster value
shr  ax,1           ; old cluster even?
jc   E1             ; no: keep high 12 bits
and  dx,0FFFh      ; yes: keep low 12 bits
jmp  E2
E1:  shr  dx,4       ; shift 4 bits to the right
E2:  mov  ax,dx      ; return new cluster number
```

CHAPTER 12

2. Yes, function 3Ch creates the file for both input and output. The read-only attribute only applies after the file is closed.
4. Error explanations:
- Rename file, Error 3: The new filename path contains directories that do not exist.
 - Delete file, Error 5: Access denied, meaning that the file is currently in use or its read-only attribute bit is set.
 - Set date/time, Error 6: An invalid file handle indicates that the file has not been opened.

- Remove directory, Error 10h: You cannot remove the current directory.
 - Rename file, Error 11h: Not same device. A file cannot be rename across disk boundaries.
 - Find first matching file, Error 12h: No more file handles can be opened.
6. No, when you delete a file, all you need is its file handle.
 8. An invalid file handle was used when trying to read the file. The file handle must match that of a file that is currently open for input or input/output.
 10. The buffer will contain "1234567890". Note: 0Dh and 0Ah are only inserted in the buffer if the user types fewer than 10 characters before pressing the Enter key.
 12. Yes, the file pointer is updated automatically.
 14. Yes, the file pointer can be moved using Function 42h.
 16. The offset would be 50 x 19, which equals 950.
 18. 11000101: store = 6 department = 5
 00101001: store = 1 department = 9
 01010101: store = 2 department = 21

CHAPTER 13

2. An underscore character is prepended to all external names.
4. Near.
6. Yes, certain registers must be preserved. The requirements will vary among different compilers written for the same language.
8. The calling program adds a value to the stack pointer that restores it to its state before the subroutine arguments were pushed on the stack.
10. The C linker is case-sensitive, so any names exported by the assembly language module must be recognized by the linker.
12. Near calls are required by the tiny, small, and compact memory models.
14. The flat memory model.
16. Stack frame, 1 parameter passed by far reference, large model. Let us assume that the current value of SP is F002. The parameter segment and offset were pushed first, then the segment:offset return address, then BP:

SS:F00A	SS:F008	SS:F006	SS:F004	SS:F002
(paramSeg)	(paramOfs)	(returnSeg)	(returnOfs)	BP
18. The __fastcall calling method uses registers for all parameters, rather than pushing parameters on the stack.
20. Inline assembly code is assembly language source code inserted into the body of any C++ procedure. A C++ inline function, on the other hand, is an entire function

written in C++ whose machine code is inserted into a C++ program at the point at which procedure is called. The latter approach eliminates the overhead of calling and returning from the function.

22. Inline code is not portable to different computer systems, so its enclosing C++ program is also not portable.
24. Yes.
26. Yes.
28. No.
30. Yes.
32. Yes.
34. It returns a count of the number of elements in the array.
36. If the C++ compiler is allowed to mangle the names used in subroutine calls, the linker cannot match these names to the names of external subroutines.
38. int = 2, enum = 2, float = 4, double = 8.
40. Code example:

```
mov  eax, [bp+6]
```
42. No significant difference was found in the code generation.

CHAPTER 14

2. A pointer variable might contain the address of an array, and be used to iterate over the array members. A pointer might contain the address of a procedure, and a program could call the procedure via the pointer. Pointers are also used when passing arguments to procedures by reference.
4. Both instructions move the same value to SI. The first, using OFFSET, moves an immediate value that must be known at assembly time. The second instruction, using LEA, is able to determine the location of a variable at runtime, and is therefore more flexible.
6. The CLI instruction, which clears the Interrupt flag, disables hardware interrupts until the flag is later set. This is done by system procedures that handle requests from hardware devices. It is important that no other devices interrupt these procedures until they have finished. (This information was taken from Chapter 15, p. 545.)
8. HLT suspends a program until a hardware interrupt occurs. WAIT suspends the main CPU until a coprocessor signals that it has completed an operation.
10. Errors:
 - a. Immediate port address must be between 0 and 255.
 - b. Second operand must be AL or AX.

- c. First operand must be AL or AX. Second operand must be immediate value or DX.
- d. Operands are reversed, and the immediate operand must be between 0 and 255.
- 12. Variables should not be in the code segment because the CPU will assume their offsets are from the segment pointed to by the DS register.
- 14. The ESC instruction passes an instruction and/or operand to the floating-point coprocessor. It is automatically inserted by the assembler into a program's object code before each floating-point instruction.
- 16. The segment identified as type 'STACK' is used when initializing the SS register when the program is loaded into memory.
- 18. In a COM program, CS is set to the starting location of the program segment prefix (PSP). In an EXE program, CS contains the location of the next byte following the PSP.
- 20. The EXE header record specifies the minimum number of paragraphs required by the program. This is used when allocating memory.
- 22. The segment_B will begin at location 0F110h, the next even paragraph boundary.
- 24. A near call pushes the current location counter on the stack and loads the 16-bit offset of the called procedure into IP. A far call pushes the current code segment and location counter on the stack and loads CS and IP with the segment and offset of the called procedure. The far call takes longer because of the greater number of values that have to be pushed and loaded.
- 26. Code example with corrections. The assembler caught errors 1-3, and the linker caught error 4:

```

title Segment Example

cseg segment 'CODE'                ; error 1
assume ds:dseg, ss:sseg           ; error 2
main proc
    mov ax,dseg
    mov ds,ax
    mov bx,value1+2
    jmp L1
    push ax
    pop ax
L1: mov ax,4C00h
    int 21h
main endp
cseg ends

dseg segment
    value1 dw 1000h,2000h
dseg ends                               ; error 3

sseg segment STACK                 ; error 4
    db 100h dup('S')
sseg ends
end main

```

28. Transient programs are loaded into memory long enough to be executed, and then the memory they occupied is released when they finish.
30. A copy of the DOS command tail is stored at offset 80h in the program segment prefix area (at the beginning of the program).
32. Load module.

CHAPTER 15

2. The 8253 Programmable Interval Timer/Counter.
4. The control unit (also called the bus interface unit).
6. The arithmetic logic unit (ALU).
8. The control unit.
10. Code examples:

```
dd +1.5E+02          ; decimal short real
dd 3F800000r        ; encoded short real
dq +1.5E+10         ; decimal long real
dq 3F00000000000000r ; encoded long real
dt 123456789012345678 ; 10-byte real in BCD format
dq 123456789d       ; 8-byte decimal integer
```
12. Interrupt 9.
14. File I/O has a lower priority than keyboard I/O, so the key will be placed in the buffer first.
16. It is store at segment 0, offset 10h x 4, which is 0000:0040h.
18. Math coprocessor instructions cannot reference the main CPU registers.

Part 2: Solutions to Programming Exercises

CHAPTER 2

Chapt 2 Ex 1: Testing Registers

```
-a 100
1D99:0100 mov ax,1
1D99:0103 mov bx,2
1D99:0106 mov cx,3
1D99:0109 mov dx,4
1D99:010C mov si,5
1D99:010F mov di,6
1D99:0112 mov bp,7
1D99:0115 mov sp,8
```

The following registers do not accept immediate values:

```
1D99:0115 mov ds,8
                ^ Error
1D99:0115 mov es,8
                ^ Error
1D99:0115 mov cs,8
                ^ Error
1D99:0115 mov ss,8
                ^ Error
1D99:0115 mov ss,8
                ^ Error
1D99:0115 mov ip,8
                ^ Error
```

Immediate values cannot be assigned to segment registers or to IP.

Chapt 2 Ex 2: Display the ROM BIOS Date

```
-d FFFF:0005
FFFF:0000 30 31 2F-33 30 2F 39 36 00 FC 00      01/30/96...
```

Chapt 2 Ex 3: Carry Flag

```
-a 100
1D99:0100 mov al,ff
1D99:0102 add al,1
```

```

1D99:0104 mov bx,1
1D99:0107 sub bx,2
1D99:010A
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0100 NV UP EI PL NZ NA PO NC
1D99:0100 B0FF MOV AL,FF
-t

AX=00FF BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0102 NV UP EI PL NZ NA PO NC
1D99:0102 0401 ADD AL,01
-t

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0104 NV UP EI PL ZR AC PE CY
1D99:0104 BB0100 MOV BX,0001
-t

AX=0000 BX=0001 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0107 NV UP EI PL ZR AC PE CY
1D99:0107 83EB02 SUB BX,+02
-t

AX=0000 BX=FFFF CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=010A NV UP EI NG NZ AC PE CY

```

Chapt 2 Ex 4: ROM BIOS Area

```

-r ds
DS 1D99
:F000
-d 0 FF
F000:0000 41 4D 49 42 49 4F 53 28-43 29 41 4D 49 30 32 2F AMIBIOS(C)AMI02/
F000:0010 30 32 2F 31 39 39 35 20-44 61 74 65 3A 2D 30 32 02/1995 Date:-02
F000:0020 2F 30 32 2F 39 35 20 28-43 29 31 39 38 35 2D 31 /02/95 (C)1985-1
F000:0030 39 39 32 2C 41 4D 49 41-6D 65 72 69 63 61 6E 20 992,AMIAmerican
F000:0040 4D 65 67 61 74 72 65 6E-64 73 20 49 6E 63 2E 2C Megatrends Inc.,
F000:0050 28 43 29 31 39 39 33 2D-31 39 39 35 2C 20 49 6E (C)1993-1995, In
F000:0060 74 65 6C 20 43 6F 72 70-6F 72 61 74 69 6F 6E 43 tel CorporationC
F000:0070 6F 70 79 72 69 67 68 74-20 49 6E 74 65 6C 20 43 opyright Intel C
F000:0080 6F 72 70 6F 72 61 74 69-6F 6E 41 6C 6C 20 52 69 orporationAll Ri
F000:0090 67 68 74 73 20 52 65 73-65 72 76 65 64 2E 41 6C ghts Reserved.Al
F000:00A0 6C 20 50 72 6F 64 75 63-74 20 6E 61 6D 65 73 20 l Product names
F000:00B0 61 72 65 20 74 72 61 64-65 6D 61 72 6B 73 20 6F are trademarks o
F000:00C0 66 20 74 68 65 69 72 20-72 65 73 70 65 63 74 69 f their respecti
F000:00D0 76 65 20 43 6F 6D 70 61-6E 69 65 73 2E 00 00 00 ve Companies....
F000:00E0 66 60 2E A0 F3 04 E8 32-03 0F B6 C0 50 B4 00 B0 f`.....2....P...
F000:00F0 01 BB 0B 00 B9 00 00 66-33 FF 66 33 F6 E9 BC E9 .....f3.f3....

```

Chapt 2 Ex 5: Dumping Memory Variables

```
-E 200 36
```

```

-A 100
1D99:0100 MOV DL,[200]
1D99:0104 MOV [201],DL

-D 200,201                (before executing the program)
1D99:0200 36 FF

-T
-T
-D 200,201
1D99:0200 36 36          (after executing the program)

```

Chapt 2 Ex 6: Overflow Flag

```

-A 100
1D99:0100 MOV AL,7F      (+127)
1D99:0102 ADD AL,1
1D99:0104
-T

AX=007F BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0102 NV UP EI PL NZ NA PO NC
1D99:0102 0401          ADD     AL,01
-T

AX=0080 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0104 OV UP EI PL NZ AC PO NC
(The Overflow flag has been set, but not the Carry flag)

-A 100
1D99:0100 MOV AL,1
1D99:0102 SUB AL,2
1D99:0104
-T

AX=0001 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0102 NV UP EI NG NZ NA PO NC
1D99:0102 2C02          SUB     AL,02
-T

AX=00FF BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0104 NV UP EI NG NZ AC PE CY
(Carry flag set, but not the Overflow flag)

```

Chapt 2 Ex 7: Resetting the Instruction Pointer

```

-A 100
1D99:0100 mov ax,2000
1D99:0103 mov si,ax
1D99:0105 mov bx,si
1D99:0107 mov ds,bx
1D99:0109 int 20
1D99:010B

```

T
T
T
T
T

R IP
100

Chapt 2 Ex 8: Evaluating the Flags

```
-a 100
1D99:0100 mov al,FF
1D99:0102 inc al
1D99:0104 sub al,2
1D99:0106 mov dl,al
1D99:0108 add dx,2
1D99:010B int 20
1D99:010D
```

-t

```
AX=00FF BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0102 NV UP EI PL NZ NA PO NC
1D99:0102 FEC0          INC      AL
```

-t

```
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0104 NV UP EI PL ZR AC PE NC
```

>> Comments:

>> ZR = 1 because AL was rolled over to zero. CF = 0 because the INC
>> instruction does not set the Carry flag.

```
1D99:0104 2C02          SUB      AL,02
```

-t

```
AX=00FE BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0106 NV UP EI NG NZ AC PO CY
1D99:0106 88C2          MOV      DL,AL
```

>> Comments:

>> CF = 1 because subtracting 2 from 0 is an invalid unsigned operation.
>> OF = 0 because AL contains a valid signed result. SF = 1 because
>> the value in AL is negative.

-t

```
AX=00FE BX=0000 CX=0000 DX=00FE SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0108 NV UP EI NG NZ AC PO CY
1D99:0108 83C202        ADD      DX,+02
```

-t

Chapt 2 Ex 11: Stack Manipulation

```

-a 100
1D99:0100 mov ax,0102          (leading zeros are optional)
1D99:0103 mov bx,0304
1D99:0106 mov cx,0506
1D99:0109 mov dx,0708
1D99:010C push ax
1D99:010D push bx
1D99:010E push cx
1D99:010F push dx
1D99:0110 pop ax
1D99:0111 pop bx
1D99:0112 pop cx
1D99:0113 pop dx
1D99:0114

```

The stack, after tracing through Step B:

```

-d ss:FFE6
1D99:FFE0  08 07-06 05 04 03 02 01 00 00

```

The registers, after tracing through Step C:

```

AX=0708 BX=0506 CX=0304 DX=0102 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0114  NV UP EI PL NZ NA PO NC

```

The registers were popped off the stack in the same order as the PUSH instructions, resulting in their values being reversed.

Chapt 2 Ex 12: Add 8-bit Values

(Enter data values into memory)

```

-e 100 10,20,30

```

(Assemble the program)

```

-a 100
1D99:0100 mov dl,[100]
1D99:0104 add dl,[101]
1D99:0108 add dl,[102]
1D99:010C

```

(Trace the program)

```

-t

```

```

AX=0000 BX=0000 CX=0000 DX=008A SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0104  NV UP EI PL NZ NA PO NC
1D99:0104 02160101      ADD      DL,[0101]
DS:0101=16

```

```

-t

```

```

AX=0000 BX=0000 CX=0000 DX=00A0 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0108  NV UP EI NG NZ AC PE NC

```

```

1D99:0108 02160201      ADD     DL,[0102]
DS:0102=00
-t

AX=0000  BX=0000  CX=0000  DX=00A0  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1D99  ES=1D99  SS=1D99  CS=1D99  IP=010C   NV UP EI NG NZ NA PE NC

(DL = A0, the sum)

```

Chapt 2 Ex 13: Add 16-bit Values

(Enter data values into memory, using the DW directive.)

```

-a 200
1D99:0200 dw 0102,0304,0506,0

```

(Assemble the program.)

```

-a 100
1D99:0100 mov ax,[200]
1D99:0103 add ax,[202]
1D99:0107 add ax,[204]
1D99:010B mov [206],ax

```

(Trace the program.)

```

-t

AX=0102  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1D99  ES=1D99  SS=1D99  CS=1D99  IP=0103   NV UP EI PL NZ NA PO NC
1D99:0103 03060202      ADD     AX,[0202]
DS:0202=0304
-t

```

```

AX=0406  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1D99  ES=1D99  SS=1D99  CS=1D99  IP=0107   NV UP EI PL NZ NA PE NC
1D99:0107 03060402      ADD     AX,[0204]
DS:0204=0506
-t

```

```

AX=090C  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1D99  ES=1D99  SS=1D99  CS=1D99  IP=010B   NV UP EI PL NZ NA PE NC
1D99:010B A30602      MOV     [0206],AX
DS:0206=0000
-t

```

```

AX=090C  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1D99  ES=1D99  SS=1D99  CS=1D99  IP=010E   NV UP EI PL NZ NA PE NC

```

(Dump location 0206, containing 090C, the sum.)

```

-d 206,207
1D99:0200 0C 09

```

Chapt 2 Ex 14: Machine Bytes

```
-a 100
1D99:0100 mov ax,20
1D99:0103 mov bx,10
1D99:0106 add ax,bx
1D99:0108 int 20
1D99:010A
```

(Display the machine language bytes.)

```
-d 100,109
1D99:0100 B8 20 00 BB 10 00 01 D8-CD 20
```

(Create a data definition at offset 0200.)

```
-a 200
1D99:0200 db B8,20,00,BB,10,00,01,D8,CD,20
1D99:020A
-g = 200
```

Program terminated normally

Chapt 2 Ex 15: Signed Numbers

```
-a 100
1D99:0100 mov ax,7FFF
1D99:0103 inc ax
1D99:0104 mov bx,8000
1D99:0107 dec bx
1D99:0108 int 20
1D99:010A
-g 108
```

```
AX=8000 BX=7FFF CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D99 ES=1D99 SS=1D99 CS=1D99 IP=0108 OV UP EI PL NZ AC PE NC
```

Explanation:

After the instruction at 0103, CF = 0, ZF = 0, OF = 1 (signed overflow), because (+32767 + 1) produces a result that is out of range.

After the instruction at 0107, CF = 0, ZF = 0, OF = 1 (signed overflow), because (-32768 - 1) produces a result that is out of range.

CHAPTER 3**Chapt 3 Ex 1: Program Trace**

```
.model small
.stack 100h
.data
.code
main proc
```

```

    mov ax,@data          ; init data segment
    mov ds,ax

    mov ax,1234h
    mov bx,ax
    mov cx,ax
    add ch,al             ; CF = 0, SF = 0, ZF = 0, OF = 0
    add bl,ah             ; CF = 0, SF = 0, ZF = 0, OF = 0
    add ax,0FFFFh        ; CF = 1, SF = 0, ZF = 0, OF = 0
    dec bx                ; CF = 1, SF = 0, ZF = 0, OF = 0
    inc ax                ; CF = 1, SF = 0, ZF = 0, OF = 0

    mov ax,4c00h         ; end program
    int 21h
main endp
end main

```

Chapt 3 Ex 2: Define and Display 8-bit Numbers

; Note: Each number is the ASCII code of a digit,
; producing output of "1234".

```

.model small
.stack 100h

.data
array db 31h,32h,33h,34h

.code
main proc
    mov ax,@data          ; init data segment
    mov ds,ax

    mov ah,2
    mov dl,array
    int 21h

    mov ah,2
    mov dl,array+1
    int 21h

    mov ah,2
    mov dl,array+2
    int 21h

    mov ah,2
    mov dl,array+3
    int 21h

    mov ax,4c00h         ; end program
    int 21h
main endp
end main

```

Chapt 3 Ex 3: Arithmetic Sums

```

.model small
.stack 100h

.data
ThreeBytes db 10h,20h,30h
TheSum     db ?

.code
main proc
    mov ax,@data           ; init data segment
    mov ds,ax

    mov al,ThreeBytes
    add al,ThreeBytes+1
    add al,ThreeBytes+2
    mov TheSum,al

    mov ax,4c00h           ; end program
    int 21h
main endp
end main

```

Chapt 3 Ex 4: Uppercase Conversion

```

; Note: subtracting 32 will only convert a character to
; uppercase if you know that the starting character is
; lowercase. See the OR instruction in Chapter 6 for a
; better method.

.model small
.stack 100h
.data
aString db "this is a string containing lowercase letters"
strSize = ($ - aString)

.code
main proc
    mov ax,@data           ; init data segment
    mov ds,ax

    mov si,offset aString
    mov cx,strSize
L1: sub [si],32
    inc si
    Loop L1

    mov ax,4c00h           ; end program
    int 21h
main endp
end main

```

Chapt 3 Ex 5: Extended Registers (80386)

```

.model small
.stack 100h
.386
.data
longVals dd 12345h,87654h

.code
main proc
    mov ax,@data          ; init data segment
    mov ds,ax

    mov eax,12345678h
    mov ebx,eax
    mov ecx,longVals
    mov edx,longVals+4

    mov si,offset longVals
    mov eax,[si]

    mov ax,4c00h          ; end program
    int 21h
main endp
end main

```

Chapt 3 Ex 6: Simple Number Sequence

```

; Generate number sequence 1,2,4,8,...,1000

.model small
.stack 100h
.386

.data
array dw 1,11 dup(0)
.code
main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    mov di,offset array
    mov cx,12

L1:
    mov ax,[di]           ; get current value
    add ax,ax              ; double it

    inc di
    mov [di],ax           ; save it for the next loop iteration
    Loop L1

    mov ax,4c00h          ; end program
    int 21h
main endp

```

```
end main
```

Chapt 3 Ex 7: Fibonacci Numbers

```
; Generate number sequence 1,2,3,5,8,13,21,...46368

; To display a 16-bit integer, call the Writeint procedure
; (explained more fully in Chapter 4).

.model small
.stack 100h
.386

LOOP_COUNT = 22
.data
array dw 1,1,LOOP_COUNT dup(0)

.code
extrn Writeint:proc, Crlf:proc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    mov di,offset array
    mov bx,10

    mov ax,[di]          ; display the first two values
    call Writeint
    call Crlf
    add di,2
    mov ax,[di]
    call Writeint
    call Crlf
    add di,2

    mov cx,LOOP_COUNT

L1:
    mov ax,[di-4]        ; sum the previous two values
    add ax,[di-2]
    mov [di],ax          ; save current value
    call Writeint        ; display it
    call Crlf
    add di,2
    Loop L1

    mov ax,4c00h         ; end program
    int 21h
main endp
end main
```

CHAPTER 4

From Chapter 4 onward, the book's Link Library (irvine.lib) is used by our solution programs. The following include file provides the necessary external declarations for the programs in this chapter:

```
; LIBRARY.INC - include file for Irvine.lib

extrn Clrscr:proc, Crlf:proc, Delay_seconds:proc, Get_time:proc, \
    Gotoxy:proc, Readchar:proc, Readkey:proc, Scroll:proc, \
    Seconds_today:proc, Set_videoseg:proc

extrn Show_time:proc, Waitchar:proc, \
    Writestring_direct:proc, Writechar_direct:proc, \
    PackedToBin:proc, Readint:proc, Readlong:proc, Writebcd:proc

extrn Writeint:proc, Writeint_signed:proc, Writelong:proc, \
    Random_range:proc, Random32:proc, Randomize:proc

extrn Readstring:proc, Str_copy:proc, Str_length:proc, Str_ucase:proc, \
    Writestring:proc, Write_errorstr:proc
```

Chapt 4 Ex 1: Simple Calculation Problem

```
.model small
.stack 100h

.data
prompt1 db "Enter the first integer: ",0
prompt2 db "Enter the second integer: ",0
result db "The sum of the integers is: ",0
num1 dw ?

.code
include library.inc

main proc
    mov ax,@data           ; init data segment
    mov ds,ax

    call ClrScr
    mov dx,0500h
    call Gotoxy

    ; Ask for the first number.

    mov dx,offset prompt1
    call Writestring
    call Readint
    mov num1,ax
    call Crlf

    ; Ask for the second number.

    mov dx,offset prompt2
    call Writestring
```

```
    call Readint
    call Crlf

; Calculate and display the sum.

    add  ax,num1
    mov  dx,offset result
    call Writestring
    mov  bx,10
    call Writeint
    call Crlf

    mov  ax,4c00h           ; end program
    int  21h
main endp
end main
```

Chapt 4 Ex 2: Using the Offset Operator

```
.model small
.stack 100h

.data
byteArray db 10,20,30
wordArray dw 1000,2000,3000

.code
include library.inc

main proc
    mov  ax,@data           ; init data segment
    mov  ds,ax
    call ClrScr

    mov  ax,0
    mov  bx,offset byteArray
    mov  al,[bx]
    mov  bx,10
    call Writeint
    call Crlf

    mov  ax,0
    mov  bx,offset byteArray+1
    mov  al,[bx]
    mov  bx,10
    call Writeint
    call Crlf

    mov  ax,0
    mov  bx,offset byteArray+2
    mov  al,[bx]
    mov  bx,10
    call Writeint
    call Crlf
```

```

    mov  bx,offset wordArray
    mov  ax,[bx]
    mov  bx,10
    call Writeint
    call Crlf

    mov  bx,offset wordArray+2
    mov  ax,[bx]
    mov  bx,10
    call Writeint
    call Crlf

    mov  bx,offset wordArray+4
    mov  ax,[bx]
    mov  bx,10
    call Writeint
    call Crlf

    mov  ax,4c00h           ; end program
    int  21h
main endp
end main

```

Chapt 4 Ex 3: Add a List of 16-Bit Numbers

```

.model small
.stack 100h

.data
wordArray dw 1000,2000,3000
sum       dw ?

.code
include library.inc

main proc
    mov  ax,@data           ; init data segment
    mov  ds,ax
    call ClrScr

    mov  sum,0
    mov  cx,3               ; loop counter
    mov  bx,offset wordArray

L1:  mov  ax,[bx]           ; get an integer
     push bx
     mov  bx,10             ; display the integer
     call Writeint
     call Crlf
     pop  bx
     add  sum,ax            ; add it to the sum
     add  bx,type wordArray ; point to next value
     loop L1

```

```

    mov ax,sum
    mov bx,10          ; display the sum
    call Writeint
    call Crlf

    mov ax,4c00h      ; end program
    int 21h
main endp
end main

```

Chapt 4 Ex 4: Fixing an Overflow Problem

; The secret to getting this program to work is to use
; a 16-bit accumulator. The original program neglected
; to set AL to zero, so we remember to set AX to zero
; in the current program.

```

.model small
.stack 100h

.data
aList db 6Fh,0B4h,1Fh
sum    dw 0          ; changed to a 16-bit variable

.code
include library.inc

main proc
    mov ax,@data      ; init data segment
    mov ds,ax
    call ClrScr

    mov bx,offset aList
    mov si,2
    mov ax,0          ; clear the accumulator
    mov dh,0
    mov dl,[bx]
    add ax,dx         ; use 16-bit accumulator
    mov dl,[bx+1]
    add ax,dx
    mov dl,[bx+si]
    add ax,dx
    mov sum,ax

    mov bx,16        ; display the sum in hexadecimal (optional)
    call Writeint
    call Crlf

    mov ax,4c00h     ; end program
    int 21h
main endp
end main

```

Chapt 4 Ex 5: Sum of Values, Using Indexed Addressing

```

; The sum is A3FFh. Notice the calculation of LOOP_COUNT, which
; automatically adjusts if additional values are added to
; the array.

.model small
.stack 100h

.data
wlist dw 1000h,2000h,7000h,03FFh
LOOP_COUNT = ($ - wlist) / (type wlist)

.code
include library.inc

main proc
    mov ax,@data            ; init data segment
    mov ds,ax
    call ClrScr

    mov cx,LOOP_COUNT
    mov si,offset wlist
    mov ax,0

L1:
    add ax,[si]            ; watch the Carry flag for
    add si,type wlist      ; unsigned overflow!
    loop L1

    mov ax,4c00h           ; end program
    int 21h
main endp
end main

```

Chapt 4 Ex 6: Generate Random Numbers

```

; Requires at least an 80386 processor.

.model small
.stack 100h
.386

LOOP_COUNT = 20
.data
array dd LOOP_COUNT dup(0)

.code
include library.inc

main proc
    mov ax,@data            ; init data segment
    mov ds,ax
    call ClrScr

```

```

    mov cx,LOOP_COUNT
    mov si,offset array

L1:
    mov eax,101                ; specify range: 0 - 100
    call Random_range          ; generate random integer
    mov [si],eax               ; save the number
    mov bx,10                  ; decimal radix
    call Writelong              ; display it
    call Crlf
    add si,type array
    loop L1

    mov ax,4c00h                ; end program
    int 21h
main endp
end main

```

Chapt 4 Ex 7: Display Random String

```

; Fill and display a null-terminated string with
; random characters between A and Z.

.model small
.stack 100h
.386

STRING_SIZE = 50
.data
randString db STRING_SIZE dup (0),0

.code
include library.inc

main proc
    mov ax,@data                ; init data segment
    mov ds,ax
    call ClrScr
    mov cx,STRING_SIZE
    mov si,offset randString

L1:
    mov eax,26                  ; specify range: 0 - 25
    call Random_range           ; generate random integer
    add al,'A'                  ; offset to start of alphabet
    mov [si],al                 ; save in string
    inc si
    loop L1

; Display the string.

    mov dx,offset randString
    call Writestring
    call Crlf

```

```

        mov ax,4c00h           ; end program
        int 21h
main endp
end main

```

Chapt 4 Ex 8: Display at Random Locations

```

; Display a character 500 times at random screen locations.

.model small
.stack 100h
.386

COUNT = 500

.code
include library.inc

main proc
    mov ax,@data           ; init data segment
    mov ds,ax
    call ClrScr
    mov cx,COUNT

L1:
    mov eax,25             ; specify row number
    call Random_range      ; generate random integer
    mov dh,al
    mov eax,80             ; specify column number
    call Random_range      ; generate random integer
    mov dl,al
    call Gotoxy            ; locate cursor
    mov ah,2               ; display the character
    mov dl,'X'
    int 21h
    Loop L1

    mov ax,4c00h           ; end program
    int 21h
main endp
end main

```

Chapt 4 Ex 9: Using the = and EQU Directives

```

.model small
.stack 100h
.386
one = 1
two = 2
three = one * two + 10 / 4
four = 418 mod 6
five = four - one
string equ <"This, naturally, is a string",0>

```

```

.data
msg db string

.code
include library.inc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax

    mov al,one
    mov bl,two
    mov cl,three
    mov dl,four
    mov ax,one
    mov bx,two
    mov cx,three
    mov dx,four
    mov si,five

    mov dx,offset msg
    call Writestring
    call Crlf

    mov ax,4c00h          ; end program
    int 21h
main endp
end main

```

Chapt 4 Ex 10: Copy a String

```

; Copy a string using indirect addressing. Display the
; string after it is copied.

.model small
.stack 100h
.386

.data
string db "Source string",0    ; string to be copied
STRSIZE = ($ - string)

dest db 80 dup(0)             ; destination

.code
include library.inc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call Clrscr
    mov si,offset string
    mov di,offset dest
    mov cx,STRSIZE

```

```

L1:
    mov  al,[si]           ; get character from the source
    mov  [di],al          ; copy it to the destination
    inc  si               ; increment both pointers
    inc  di
    loop L1               ; repeat loop

    mov  dx,offset dest
    call Writestring
    call Crlf

    mov  ax,4c00h         ; end program
    int  21h
main endp
end main

```

Chapt 4 Ex 11: Copy a String Backwards

```

; Copy a string backwards using indirect addressing.
; Display the copied string.

.model small
.stack 100h
.386

.data
string db "Source string",0 ; string to be copied
STRSIZE = ($ - string) - 1

dest   db  80 dup(0)        ; destination

.code
include library.inc

main proc
    mov  ax,@data           ; init data segment
    mov  ds,ax
    call Clrscr
    mov  si,offset string
    add  si,(STRSIZE - 1)
    mov  di,offset dest
    mov  cx,STRSIZE

L1:
    mov  al,[si]           ; get character from the source
    mov  [di],al          ; copy it to the destination
    dec  si               ; move both pointers
    inc  di
    loop L1               ; repeat loop

    mov  dx,offset dest
    call Writestring
    call Crlf

```

```

        mov ax,4c00h           ; end program
        int 21h
main endp
end main

```

Chapt 4 Ex 12: Copy an Array

```

; Copy all numbers from array1 to array2. You can test it
; either by dumping memory after the loop, or by calling
; Writeint and displaying each integer on the screen.

```

```

.model small
.stack 100h
.386

COUNT = 5

.data
array1 dw 1000h,2000h,3000h,4000h,5000h
arraySize = ($ - array1) / (type array1)

array2 dw COUNT dup(0)      ; destination

.code
include library.inc

main proc
    mov ax,@data             ; init data segment
    mov ds,ax
    call Clrscr
    mov si,0
    mov cx,COUNT

    ; Notice that indexed addressing works well here, because
    ; you only need to use a single index register, SI.

L1:
    mov ax,array1[si]        ; get integer from array1
    mov array2[si],ax        ; copy it to array2
    add si,type array1       ; index to next position
    loop L1                  ; repeat loop

    mov ax,4c00h           ; end program
    int 21h
main endp
end main

```

Chapt 4 Ex 13: Array Insertion

```

; Using array1 from Exercise 12, insert 2500 in position 3.
; Note to instructors: This exercise is more challenging
; than any of the previous exercises in this chapter.

```

```

; You may want to ask students to use the expression
; (TYPE array1) rather than the literal 2.

; Suggested enhancement: Ask the user for the value to be
; inserted and the insert position. Perform range checking
; to avoid accessing memory outside the array.

.model small
.stack 100h
.386

COUNT = 5
INSERT_POSITION = 2 ; first element is (0)
INSERT_VALUE = 2500h

; An extra position was added to the array to
; provide space for the elements that are moved.

.data
array1 dw 1000h,2000h,3000h,4000h,5000h,0

.code
include library.inc

main proc
    mov ax,@data ; init data segment
    mov ds,ax

    ; Make room for the new value by moving all subsequent
    ; elements toward the end of the array. We must start
    ; at the end of the array and work backward.

    mov si, COUNT * (type array1)
    mov cx, COUNT - INSERT_POSITION

L1:
    mov ax,array1[si-2] ; get integer from the array
    mov array1[si],ax ; move it back one position
    sub si,type array1 ; point to previous
    loop L1 ; repeat loop

    ; Insert the new value.

    mov array1+(INSERT_POSITION * 2), INSERT_VALUE

    ; Display the array to verify (optional).

    call Clrscr
    mov cx,COUNT+1
    mov si,0

L2:
    mov ax,array1[si] ; get integer from the array
    mov bx,16 ; display it in hexadecimal
    call Writeint
    call Crlf
    add si,type array1 ; point to next

```

```

        loop L2                ; repeat loop

        mov  ax,4c00h          ; end program
        int  21h
main endp
end main

```

Chapt 4 Ex 14: Copy an Array

```

; Copy and display an array of long integers.
; This is almost a duplicate of Exercise 10, which
; copies a string.

.model small
.stack 100h
.386

.data
array dd 1,2,3,4,5,6,7,8,9,10
COUNT = ($ - array) / (type array)

dest  dd COUNT dup(0)

.code
include library.inc

main proc
    mov  ax,@data            ; init data segment
    mov  ds,ax
    call Clrscr

    mov  cx,COUNT
    mov  si,0

L1:
    mov  eax,array[si]      ; get integer from the source
    mov  dest[si],eax       ; copy it to the destination
    mov  bx,10              ; display it in decimal
    call Writelong
    call Crlf
    add  si,type array      ; increment index
    loop L1                 ; repeat loop

    mov  ax,4c00h          ; end program
    int  21h
main endp
end main

```

Chapt 4 Ex 15: Shuffle an Array

```

; Create a sequentially numbered array of 50 integers,
; shuffle the array randomly, and display the array.

```

```

.model small
.stack 100h
.386
COUNT = 50

.data
array dw COUNT dup(0)
comma db ",",0

.code
include library.inc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax

    ; Create a sequentially numbered array.

    mov cx,COUNT
    mov si,0
    mov ax,1

L1: mov array[si], ax
    inc ax
    add si,type array
    Loop L1

    ; Shuffle the array randomly by choosing random indexes
    ; and exchanging the first element with the value in
    ; the randomly chosen position. Remember to multiply
    ; the random index in DI by 2 before using it as an offset.

    mov cx,COUNT          ; pass through the array once
    mov si,0

L2: mov eax,COUNT          ; rand value, 0..(COUNT - 1)
    call Random_range
    mov di,ax
    shl di,1              ; multiply by 2 for 16-bit values
    mov ax,array[si]      ; exchange [si] with [di]
    xchg ax,array[di]
    mov array[si],ax
    add si,type array      ; increment index
    loop L2                ; repeat loop

    ; Display the shuffled array.

    call Clrscr
    mov cx,COUNT
    mov si,0
    mov bx,10              ; decimal radix

L3: mov ax,array[si]
    call Writeint          ; display a number
    mov dx,offset comma    ; display ",",

```

```

        call Writestring
        add si,type array
        loop L3

        call Crlf
        mov ax,4c00h          ; end program
        int 21h
main endp
end main

```

Chapt 4 Ex 16: Reverse an Array

```

; Copy an array and reverse it at the same time.

.model small
.stack 100h
.386

.data
array dd 1,2,3,4,5,6,7,8,9,10
COUNT = ($ - array) / (type array)

dest dd COUNT dup(0)
dest_last = $ - (type array)

.code
include library.inc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call Clrscr

    mov cx,COUNT
    mov si,offset array
    mov di,dest_last

L1:
    mov eax,[si]          ; get integer from the source
    mov [di],eax          ; copy it to the destination
    add si,type array     ; increment index
    sub di,type array
    loop L1              ; repeat loop

; Display the copied array.

    mov cx,COUNT
    mov si,offset dest

L2:
    mov eax,[si]
    mov bx,10             ; display it in decimal
    call Writelong
    call Crlf

```

```

        add si,type dest      ; increment index
        loop L2

        mov ax,4c00h         ; end program
        int 21h
main endp
end main

```

Chapt 4 Ex 17: Delete an Element from an Array

```

; Write and test a procedure that deletes an element n
; from an array of 16-bit integers. Let n be an input
; parameter.

; Note: Unfortunately, this solution program requires
; the use of the SHL instruction and the JZ instruction.
; These will have to be explained by the instructor
; before students can complete the program. See the
; two lines marked NEEDS EXPLANATION.

.model small
.stack 100h
.386

.data
array dw 1000h,2000h,3000h,4000h,5000h
COUNT = ($ - array) / (type array)

prompt db "Enter element number to delete[1..5]: ",0

.code
include library.inc

main proc
    mov ax,@data      ; init data segment
    mov ds,ax

    mov dx,offset prompt
    call Writestring
    call Readint      ; get element num in AX
    dec ax            ; 0 is the first offset
    call DeleteMember

; Display the array to verify (optional).

    call Clrscr
    mov cx,COUNT-1
    mov si,0
L2:
    mov ax,array[si]  ; get integer from the array
    mov bx,16         ; display it in hexadecimal
    call Writeint
    call Crlf
    add si,type array ; point to next
    loop L2           ; repeat loop

```

```

        mov ax,4c00h          ; end program
        int 21h
main endp

; Delete a member at position n from the array. The
; value of n is passed in the AX register. Each subsequent
; member will be copied forward in the array to fill the
; gap created by the deleted member.

DeleteMember proc

        mov cx,COUNT-1      ; determine loop count
        sub cx,ax
        jz  DM2              ; NEEDS EXPLANATION

        mov si,ax           ; get value of n
        shl si,1            ; mult by 2 to get offset
                                ; NEEDS EXPLANATION
DM1:
        mov ax,array[si+2]  ; get integer from the array
        mov array[si],ax    ; move it back one position
        add si,type array   ; point to next
        loop DM1            ; repeat loop

DM2:
        ret
DeleteMember endp
end main

```

Chapt 4 Ex 18: Pointers

```

; Define an array of four words, define a pointer to
; the array, use the pointer to display the array.
; Easy exercise, comparable in difficulty to Exercises
; 1-5 in this chapter.

.model small
.stack 100h

.data
myArray dw 1000h,2000h,3000h,4000h,5000h,6000h
COUNT = ($ - myArray) / (type myArray)

aPointer dw myArray

.code
include library.inc

main proc
        mov ax,@data          ; init data segment
        mov ds,ax

        call ClrScr
        mov si,aPointer

```

```

        mov    cx,COUNT

L1:  mov    ax,[si]
      mov    bx,16
      call Writeint
      call Crlf
      add    si,type myArray
      loop  L1

      mov    ax,4c00h           ; end program
      int   21h
main  endp
end   main

```

CHAPTER 5

Chapt 5 Ex 1: Keyboard Scan Codes

```

; Display the keyboard scan code in hexadecimal for any
; key pressed by the user. Quit when Esc is pressed.
; NOTE: this program requires a single use of the JE
; instruction. See the line marked NEEDS EXPLANATION.

```

```

.model small
.stack 100h
.data

WAIT_FOR_KEY = 10h
ESC_KEY = 1Bh

.code
include library.inc

main proc
    mov  ax,@data           ; init data segment
    mov  ds,ax
    call ClrScr

L1:
    mov  ah,WAIT_FOR_KEY
    int  16h               ; get scan code in AH
    cmp  al,ESC_KEY       ; check for Esc key
    je   L2               ; NEEDS EXPLANATION

    mov  al,ah             ; move scan code to AL
    mov  ah,0
    mov  bx,16             ; hexadecimal radix
    call Writeint         ; display the scan code
    call Crlf
    loop L1

L2:
    mov  ax,4c00h         ; end program

```

```

    int 21h
main endp
end main

```

Chapt 5 Ex 2: Keyboard Status Indicator

```

; (Advanced assignment)
; Display the status of the Shift, CapsLock, and Alt keys
; in the lower right corner of the screen. Pass the color
; attribute as an argument.

; Note: This assignment is somewhat difficult because
; students will have to understand how to use the TEST and
; JZ instructions from Chapters 6 and 7.

; This program causes the keyboard status information to flicker
; because the program is continually redrawing the same
; information. As a challenge, suggest that students set a
; boolean flag for each status value that indicates its current
; state. Then they can check the flag and only draw on the
; screen when the flag has changed value. That should
; eliminate the flicker.

.model small
.stack 100h

SHIFTKEY_MASK = 11b
CAPSLOCK_MASK = 1000000b
ALTKEY_MASK   = 1000b

CHECK_KEYBOARD = 11h
GET_KYBD_FLAGS = 12h
STATUS_ROW    = 24
STATUS_COL    = 60
STATUS_LEN    = 20
ESC_KEY       = 1Bh

.data

; Define the status strings.
shiftOn db "SHIFT ",0
capsOn  db "CAPS  ",0
altOn   db "ALT   ",0
blanks  db STATUS_LEN dup(20h),0

.code
include library.inc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call ClrScr
    mov bl,0Fh           ; color attribute

A1: mov ah,CHECK_KEYBOARD ; wait for key

```

```

        int 16h
        cmp al,ESC_KEY           ; quit when Esc pressed
        je  Exit
        call ShowStatusKeys
        jmp Al

Exit:
        mov ax,4c00h             ; end program
        int 21h
main endp

; BL contains the attribute

ShowStatusKeys proc
        mov dh,STATUS_ROW
        mov dl,STATUS_COL
        mov ah,bl

        ; Clear the status line.
        mov si,offset blanks
        call Writestring_direct

        ; Get the keyboard flag byte.

        mov ah,GET_KYBD_FLAGS
        int 16h                 ; keyboard status in AL
        mov ah,bl               ; AH = video attribute

L1: test al,SHIFTKEY_MASK      ; check bits 0 and 1
        jz  L2                 ; skip if bits are clear
        mov si,offset shiftOn  ; bits set, so display
        call Writestring_direct ; string "Shift"

L2: test al,CAPSLOCK_MASK
        jz  L3
        mov si,offset capsOn
        call Writestring_direct

L3: test al,ALTKEY_MASK
        jz  L4
        mov dx,offset altOn
        call Writestring

L4:
        ret
ShowStatusKeys endp
end main

```

Chapt 5 Ex 3: Stack Operations

```

; Use the PUSH and POP instructions to display a list of 16-bit
; integers in reverse order on the console.

```

```

; Easy program. Uses indirect addressing with a loop.

```

```

.model small
.stack 100h

.data

aList dw 100h,200h,300h,400h,500h
COUNT = ($ - aList) / (type aList)

.code
include library.inc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call ClrScr
    mov cx,COUNT
    mov si,offset aList

    ; Push the numbers.

L1: mov ax,[si]
    push ax
    add si,type aList
    loop L1

    ; Pop and display the numbers.

    mov cx,COUNT

L2: pop ax                ; pop from stack
    mov bx,16             ; display in hexadecimal
    call Writeint
    call Crlf
    loop L2

    mov ax,4c00h         ; end program
    int 21h
main endp
end main

```

Chapt 5 Ex 4: Window Scroll Program

```

; Scroll a window and display a message in the window.
; (Do not use book library procedures.)

.model small
.stack 100h
.data
string db " This text is in the window.$"

.Code
main proc
    mov ax, @data      ; init data segment
    mov ds, ax

```

```

SetWindow:
    mov     ah, 06h        ; function 6, initialize a window
    mov     al, 0Fh        ; set lines to scroll at 15
    mov     bh, 70h        ; attribute: black on white
    mov     ch, 05h        ; start at row 5
    mov     cl, 0Ah        ; column 10
    mov     dh, 14h        ; end at row 20
    mov     dl, 46h        ; column 70
    int     10h

SetCursor:
    mov     ah, 02h        ; call function 2, set cursor position
    mov     bh, 00h        ; on video page 0 (current page)
    mov     dh, 0Ah        ; at row 10
    mov     dl, 14h        ; column 20
    int     10h

DisplayText:
    mov     ah, 09h        ; function 9, display string
    mov     dx, offset string ; offset of the string
    int     21h

Exit:
    mov     ax, 4C00h      ; exit program
    int     21h
main     endp
end main

```

Chapt 5 Ex 5: Enhanced Window Scroll Program

```

.model small
.stack 100h
.data
string db " This text is in the window.$"

.code
main proc
    mov     ax,@data      ; init data segment
    mov     ds,ax

SetWindow:
    mov     ah,6          ; scroll window up
    mov     al,0Fh        ; set lines to scroll at 15
    mov     bh,70h        ; inverse video attribute
    mov     ch,5          ; start at column 10
    mov     cl,0Ah        ; row 5
    mov     dh,14h        ; and end at column 70,
    mov     dl,46h        ; row 20
    int     10h

SetCursor:
    mov     ah,2          ; set cursor position
    mov     bh,0          ; on video page 0
    mov     dh,0Ah        ; at row 10
    mov     dl,14h        ; column 20

```

```
    int    10h

DisplayText:
    mov    ah,9           ; display a string
    mov    dx,offset string ; point to the string
    int    21h

WaitForKey:
    mov    ah,8           ; console input with
    int    21h           ; no echo

ScrollWindow:
    mov    ah,6           ; scroll window up
    mov    al,0           ; scroll all lines
    mov    bh,7           ; normal attribute
    mov    ch,7           ; start at row 7
    mov    cl,0Fh        ; column 15
    mov    dh,12h        ; and end at row 18
    mov    dl,44h        ; column 68
    int    10h

SetCursor2:
    mov    ah,2           ; set cursor position
    mov    bh,0           ; on video page 0
    mov    dh,0Ch        ; at row 12,
    mov    dl,26h        ; column 43
    int    10h

WriteCharacter:
    mov    ah,9           ; write character and
    mov    al,'A'         ; attribute at cursor position
    mov    bh,0           ; on video page 0
    mov    bl,87h        ; normal video, blinking
    mov    cx,1           ; write it one time
    int    10h

WaitForKey2:
    mov    ah,8           ; console input with
    int    21h           ; no echo.

ClearScreen:
    mov    ah,6           ; scroll a window
    mov    al,0           ; scroll all lines
    mov    bh,7           ; normal video attribute
    mov    ch,0           ; start at row 1
    mov    cl,0           ; column 1
    mov    dh,18h        ; and end at row 25
    mov    dl,50h        ; column 80
    int    10h

Exit:
    mov    ax,4C00h      ; end program
    int    21h
main endp
end main
```

Chapt 5 Ex 6: Keyboard Echo Program

```

; Input 10 characters from standard input and echo them
; to standard output. Use redirection operators when
; running the program from a command prompt.

.model small
.stack 100h

.code
main proc
    mov  ax,@data        ; init data segment
    mov  ds,ax
    mov  cx,10           ; loop counter

InputCharacter:
    mov  ah,1            ; character input with echo
    int  21h

OutputCharacter:
    mov  ah,2            ; character output
    mov  dl,al           ; character in dl
    int  21h
    loop InputCharacter  ; repeat 10 times

    mov  ax,4C00h       ; end program
    int  21h
main endp
end main

```

Chapt 5 Ex 7: Compressed Type Setup

```

; This program sets an Epson-compatible printer
; to compressed type.

.model small
.stack 100h
.code
main proc
    mov  ax,@data
    mov  ds,ax
    mov  ah,5            ; DOS: printer output
    mov  dl,15           ; code for Epson compressed type
    int  21h

    mov  ax,4C00h       ; return to DOS
    int  21h
main endp
end main

```

Chapt 5 Ex 8: Character String Printing

```

; This program writes a line of text to the printer

```

```

; and prints one of the words in compressed mode.
; This must be run on an Epson dot-matrix printer.

.model small
.stack 100h
.data
aString db "The word ",0Fh,"compressed",12h," is the only one "
        db "that is small.",0Dh,0Ah,"$"

.code
main proc
    mov     ax,@data
    mov     ds,ax

    mov     ah,9
    mov     dx,offset aString
    int     21h

    mov     ax,4C00h
    int     21h
main endp
end main

```

Chapt 5 Ex 9: String Input

```

; This program inputs a string from the keyboard and
; redisplayes it on the screen. Demonstrates DOS
; functions 0Ah and 9.

.model small
.stack 100h

.data
stringSize  db  80           ; size of input area
keysTyped   db  ?           ; number of chars input
inputString db  80 dup("$") ; input chars stored here
crlf        db  0Dh,0Ah,"$"

.code
main proc
    mov     ax,@data
    mov     ds,ax
    mov     ah,0Ah           ; DOS function: input string
    mov     dx,offset stringSize
    int     21h

    mov     ah,9             ; go to next screen line
    mov     dx,offset crlf
    int     21h

; The string contains a carriage return, but we
; need to insert an additional linefeed character so
; it can be displayed correctly.

    mov     ax,0

```

```

    mov    al,keysTyped
    mov    si,ax
    mov    inputString[si+1],0Ah ; linefeed character

; Echo the string to the console.

    mov    ah,9                ; output $-terminated string
    mov    dx,offset inputString
    int    21h

    mov    ax,4C00h            ; end program
    int    21h
main endp
end main

```

Chapt 5 Ex 10: Uppercase Conversion

```

; Create a loop that inputs lowercase letters. Convert
; each character to uppercase and display it. Halt
; when Ctrl-Break is pressed. (Hint: run this in the
; debugger.)

```

```

.model small
.stack 100h
.data
string db "Enter lowercase letters: $"

.code
main proc
    mov    ax,@data           ; init data segment
    mov    ds,ax

clear_window:
    mov    ah,7               ; scroll window down
    mov    al,0               ; clear entire window
    mov    cx,0               ; row 0, column 0
    mov    dx,184fh           ; to row 24, column 79
    mov    bh,70h             ; reverse video
    int    10h

set_cursor:
    mov    ah,2               ; set cursor position
    mov    dx,0913h           ; row 9, column 19
    mov    bh,0               ; video page 0
    int    10h

display_prompt:
    mov    ah,9               ; display a string
    mov    dx,offset string
    int    21h

input_character:
    mov    ah,8               ; input char, no echo
    int    21h               ; char is in AL

```

```

convert_character:
    sub     al,32d        ; convert AL to uppercase

display_character:
    mov     ah,2          ; display character
    mov     dl,al         ; character is in DL
    int     21h

wait_for_key:
    mov     ah,8          ; get character, no echo
    int     21h
    jmp     input_character ; get another character

Exit:
    mov     ax,4c00h      ; end program
    int     21h
main     endp
end main

```

Chapt 5 Ex 11: String with Attributes

```

; Use INT 10h to display the first 15 letters of the
; alphabet, Give each character a different color.
; Note: As a challenge problem, let students use
; a nested loop to show all possible backgrounds with
; all possible foregrounds.

```

```

.model small
.stack 100h
LOOP_COUNT = 15

.data
char     db 'A'
row      db 5
col      db 5
color    db 1

.code
main proc
    mov   ax,@data        ; init data segment
    mov   ds,ax
    mov   cx,LOOP_COUNT

L1:
    push  cx              ; save loop counter

    ; Set the cursor position.

    mov   ah,2            ; BIOS function
    mov   bh,0            ; video page 0
    mov   dh,row          ; set row and column
    mov   dl,col
    int   10h

    ; Display a character and its attribute (color).

```

```

    mov  ah,9           ; BIOS function
    mov  al,char       ; get the character
    mov  bh,0          ; video page 0
    mov  bl,color      ; video attribute
    mov  cx,1          ; character count for INT 10h
    int  10h

; Increment the color, character, and attribute.

    inc  col
    inc  char
    inc  color

    pop  cx            ; restore loop counter
    loop L1           ; repeat the loop

    mov  ax,4C00h      ; end program
    int  21h
main endp
end main

```

Chapt 5 Ex 12: Box Drawing Program

```

; Use INT 10h to draw a single-line box on the screen,
; with the upper left corner at row 5, column 10 and the
; lower right corner at row 20, column 70.

.model small
.stack 100h

ulrow = 5
ulcol = 10
lrrrow = 20
lrcol = 70
bwidth = (lrcol - ulcol) - 2
bheight =(lrrrow - ulrow) - 2

ulcorner = 0DAh
urcorner = 0BFh
llcorner = 0C0h
lrcorner = 0D9h
hbar      = 0C4h
vbar      = 0B3h
crlf      EQU <0Dh,0Ah>

.data
top  db ulcorner
    db bwidth dup (hbar)      ; horizontal line
    db urcorner,crlf,'$'

bottom db llcorner
    db bwidth dup (hbar)      ; horizontal line
    db lrcorner,crlf,'$'

```

```

side db vbar, bwidth dup(' '), vbar, crlf, '$'
row db ulrow
col db ulcol

.code
main proc
    mov ax,@data
    mov ds,ax

    call DrawBox

    mov ah,1          ; wait for keystroke
    int 21h
    mov ax,4C00h     ; end program
    int 21h
main endp

; Draw the complete box.

DrawBox proc
    ; Draw the top of the box.

    call locate      ; position the cursor
    mov ah,9         ; function: display string
    mov dx,offset top
    int 21h
    inc row

    ; Draw the sides of the box.

    mov cx,bheight   ; set loop count for box sides

L1: call locate
    mov ah,9         ; display string
    mov dx,offset side
    int 21h
    inc row
    loop L1

    ; Draw the bottom of the box.

    call locate
    mov ah,9         ; display string
    mov dx,offset bottom
    int 21h
    ret
DrawBox endp

; Locate the cursor at <row>, <col>.

locate proc
    push ax
    push bx
    push dx
    mov ah,2
    mov bh,0

```

```

        mov     dh,row
        mov     dl,col
        int     10h
        pop     dx
        pop     bx
        pop     ax
        ret
locate endp
end main

```

Chapt 5 Ex 13: Multiple Boxes

```

; Use INT 10h to draw several boxes on the screen of
; varying sizes and shapes.

; If INT 21h were used to display the boxes in this program,
; the solution program for Exercise 14 (Boxes with Colors)
; would be more difficult. For this reason, I used the
; Writechar_direct and Writestring_direct procedures
; from the link library to complete this program.
; Alternatively, you could use INT 10h function 9 to
; display each character.

; The first version of this program used the DH and DL
; registers to keep track of the current row and column.
; This method proved difficult to debug, however, so I
; switched to using the variables <row> and <column>. I
; found this easier to implement, and somewhat more flexible.

; Note: This program uses the JE instruction, which
; is not covered until Chapter 6.

.model small
.stack 100h
.286

; Define constants for the box drawing characters
ulcorner = 0DAh
urcorner = 0BFh
llcorner = 0C0h
lrcorner = 0D9h
hbar     = 0C4h
vbar     = 0B3h

TABLE_ENTRY_SIZE = 4

.data
; Format for table entry: top row, left column,
; bottom row, right column.

boxes    label byte
        db 5,1,20,10      ; row,col, row,col
        db 12,20,18,60
        db 1,5,3,10
        db 7,0,24,79

```

```

        db  9,25,18,75
        db  0FFh           ; sentinel value

boxWidth  dw  ?
boxHeight dw  ?
attribute db  7           ; white on black
row       db  ?
col       db  ?

.code
include library.inc

main proc
    mov  ax,@data
    mov  ds,ax
    mov  si,offset boxes   ; point to box table
    call Clrscr

L1:
    cmp  byte ptr [si],0FFh ; end of table?
    je   quit              ; yes: quit
    call draw_box          ; SI points to box information
    add  si,TABLE_ENTRY_SIZE ; get next entry
    jmp  L1

quit:
    mov  ax,4C00h          ; return to DOS
    int  21h
main endp

; Draw a single box, with parameters pointed to by SI

draw_box proc
    pusha
    mov  ch,0              ; calculate boxHeight
    mov  cl,[si+2]
    sub  cl,[si]
    dec  cl
    mov  boxHeight,cx

    mov  ch,0              ; calculate boxWidth
    mov  cl,[si+3]
    sub  cl,[si+1]
    dec  cl
    mov  boxWidth,cx

    mov  al,[si]           ; upper-left corner
    mov  row,al
    mov  al,[si+1]
    mov  col,al

    call draw_top          ; draw top of box
    inc  row               ; second row of box, left side

    mov  cx,boxHeight
    call draw_side         ; draw left side of box

```

```

    mov     al,[si+3]           ; right column
    mov     col,al
    mov     cx,boxHeight
    call    draw_side          ; draw right side of box

    mov     cx,boxWidth
    mov     al,[si+2]         ; lower-left row
    mov     row,al
    mov     al,[si+1]         ; lower-left column
    mov     col,al
    call    draw_bottom        ; draw bottom of box

    popa
    ret
draw_box endp

; Draw the side of a box, starting at position
; row,col, using CX as a counter.

draw_side proc
    mov     al,row           ; save the row
    push   ax

DS1:
    mov     al,vbar
    call    Outchar
    inc     row
    loop   DS1

    pop     ax               ; restore the row
    mov     row,al
    ret
draw_side endp

; Draw the top of the box by displaying the upper-left
; corner character, a straight line, and the upper-right
; corner character.

draw_top proc
    mov     al,col           ; save the column
    push   ax

    mov     al,ulcorner      ; lower-left corner char
    call    Outchar          ; display the character
    inc     col

    ; Draw a horizontal line.

    mov     cx,boxWidth

L2:
    mov     al,hbar           ; horizontal bar char
    call    Outchar
    inc     col
    loop   L2

```

```

    mov    al,urcorner    ; upper-right corner char
    call  Outchar
    inc    col

    pop    ax            ; restore the column
    mov    col,al
    ret
draw_top endp

; Draw the bottom of the box, starting at
; row,col, with a width specified in CX.

draw_bottom proc
    mov    al,col        ; save the column
    push  ax

    mov    al,llcorner   ; lower-left corner char
    call  Outchar
    inc    col

    ; Draw a horizontal line.

    mov    cx,boxWidth

DB1:
    mov    al,hbar
    call  Outchar
    inc    col
    loop  DB1

    mov    al,lrcorner
    call  Outchar

    pop    ax            ; restore the column
    mov    col,al
    ret
draw_bottom endp

; Output character in AL at current row,col position

Outchar proc
    push  dx
    mov    ah,attribute
    mov    dh,row
    mov    dl,col
    call  Writechar_direct
    pop    dx
    ret
Outchar endp
end main

```

Chapt 5 Ex 14: Boxes with Colors

```

; Draw several boxes on the screen of varying sizes and shapes.
; Add a color attribute byte to the box definitions. This is
; a good bonus assignment to follow Exercise 13.

```

```

; If the solution to Exercise 13 was designed correctly,
; the current program can be completed in about 10 minutes.
; Simply add an attribute byte to each table entry, change
; TABLE_ENTRY_SIZE, and initialize the attribute variable
; in the draw_box procedure.

```

```

; Note: This program uses the JE instruction, which
; is not covered until Chapter 6.

```

```

.model small
.stack 100h
.286

```

```

; Define constants for the box drawing characters
ulcorner = 0DAh
urcorner = 0BFh
llcorner = 0C0h
lrcorner = 0D9h
hbar     = 0C4h
vbar     = 0B3h

```

```

whiteOnBlack = 0Fh
blueOnWhite  = 71h
yellowOnBlue = 1Eh
magentaOnBlack = 0Dh
yellowOnBrown = 6Eh

```

```

TABLE_ENTRY_SIZE = 5

```

```

.data
; Format for table entry: top row, left column,
; bottom row, right column, attribute.

```

```

boxes    label byte
         db  5,1,20,10,whiteOnBlack
         db 12,20,18,60,blueOnWhite
         db  1,5,3,10,yellowOnBlue
         db  7,0,24,79,magentaOnBlack
         db  9,25,18,75,yellowOnBrown
         db 0FFh                ; sentinel value

```

```

boxWidth  dw  ?
boxHeight dw  ?
attribute  db  ?                ; color of box frame
row       db  ?
col       db  ?

```

```

.code
include library.inc

```

```

main proc
    mov  ax,@data
    mov  ds,ax
    mov  si,offset boxes    ; point to box table
    call Clrscr

```

```
L1:
    cmp    byte ptr [si],0FFh ; end of table?
    je     quit                ; yes: quit
    call   draw_box           ; SI points to box information
    add    si,TABLE_ENTRY_SIZE
    jmp    L1
```

```
quit:
    mov    ax,4C00h           ; return to DOS
    int    21h
main endp
```

; Draw a single box, with parameters pointed to by SI

```
draw_box proc
    pusha
    mov    ch,0                ; calculate boxHeight
    mov    cl,[si+2]
    sub    cl,[si]
    dec    cl
    mov    boxHeight,cx

    mov    ch,0                ; calculate boxWidth
    mov    cl,[si+3]
    sub    cl,[si+1]
    dec    cl
    mov    boxWidth,cx

    mov    al,[si+4]           ; initialize the attribute
    mov    attribute,al

    mov    al,[si]             ; set starting row,col values
    mov    row,al
    mov    al,[si+1]
    mov    col,al

    call   draw_top            ; draw top of box
    inc    row                 ; second row of box, left side

    mov    cx,boxHeight
    call   draw_side           ; draw left side of box

    mov    al,[si+3]           ; right column
    mov    col,al
    mov    cx,boxHeight
    call   draw_side           ; draw right side of box

    mov    cx,boxWidth
    mov    al,[si+2]           ; lower-left row
    mov    row,al
    mov    al,[si+1]           ; lower-left column
    mov    col,al
    call   draw_bottom        ; draw bottom of box

    popa
```

```
        ret
draw_box endp

; Draw the side of a box, starting at position
; row,col, using CX as a counter.

draw_side proc
    mov  al,row        ; save the row
    push ax

DS1:
    mov  al,vbar
    call Outchar
    inc  row
    loop DS1

    pop  ax            ; restore the row
    mov  row,al
    ret
draw_side endp

; Draw the top of the box by displaying the upper-left
; corner character, a straight line, and the upper-right
; corner character.

draw_top proc
    mov  al,col        ; save the column
    push ax

    mov  al,ulcorner  ; lower-left corner char
    call Outchar      ; display the character
    inc  col

    ; Draw a horizontal line.

    mov  cx,boxWidth

L2:
    mov  al,hbar      ; horizontal bar char
    call Outchar
    inc  col
    loop L2

    mov  al,urcorner  ; upper-right corner char
    call Outchar
    inc  col

    pop  ax            ; restore the column
    mov  col,al
    ret
draw_top endp

; Draw the bottom of the box, starting at
; row,col, with a width specified in CX.

draw_bottom proc
    mov  al,col        ; save the column
```

```

    push  ax

    mov  al,llcorner      ; lower-left corner char
    call Outchar
    inc  col

; Draw a horizontal line.

    mov  cx,boxWidth
DB1:
    mov  al,hbar
    call Outchar
    inc  col
    loop DB1

    mov  al,lrcorner
    call Outchar

    pop  ax              ; restore the column
    mov  col,al
    ret
draw_bottom endp

; Output character in AL at current row,col position

Outchar proc
    push dx
    mov  ah,attribute
    mov  dh,row
    mov  dl,col
    call Writechar_direct
    pop  dx
    ret
Outchar endp
end main

```

Chapt 5 Ex 15: Setting the Cursor Size

```

; Write three short procedures that set the cursor size to
; (1) a solid block, (2) top line, and (3) normal size.
;
; Note: On a VGA display, the cursor can either be on top, on
; bottom, or a solid block. Therefore, the mid-height cursor
; mentioned in the Exercise 14 is not possible. This error
; was corrected in the second printing.

.model small
.stack 100h
.data
solidMsg  db "Solid cursor: ",0
topMsg    db "Top-line cursor: ",0
normalMsg db "Normal cursor: ",0

.code
include library.inc

```

```
main proc
    mov     ax,@data
    mov     ds,ax

    call    solid_cursor
    call    top_cursor
    call    default_cursor

    mov     ax,4C00h
    int     21h
main endp

solid_cursor proc
    mov     dx,offset solidMsg
    call    Writestring
    mov     ah,1
    mov     ch,0
    mov     cl,7
    int     10h
    call    getch
    call    Crlf
    ret
solid_cursor endp

top_cursor proc
    mov     dx,offset topMsg
    call    Writestring
    mov     ah,1
    mov     ch,0
    mov     cl,1
    int     10h
    call    getch
    call    Crlf
    ret
top_cursor endp

default_cursor proc
    mov     dx,offset normalMsg
    call    Writestring
    mov     ah,1
    mov     ch,6
    mov     cl,7
    int     10h
    call    getch
    call    Crlf
    ret
default_cursor endp

getch proc
    mov     ah,1             ; wait for keystroke
    int     21h
    ret
getch endp
end main
```

Chapt 5 Ex 16: Blinking Message

```

; Display a blinking message in the lower right corner
; of the screen, wait for a keystroke, and erase the
; message.

; Note: you must switch to video mode 7 to enable the
; blink bit in the video attribute byte. Also, the program
; must be run in full-screen mode, not in a graphical window.

; This implementation uses INT 10h, function 9 to display
; a string with a chosen attribute. As a simple variation,
; students can call the Writestring_direct procedure
; from the book's link library.

.model small
.286
.stack 100h

blinking = 087h      ; blinking video attribute
normal   = 7         ; normal video attribute

.data
msg      db '<<Press any key to continue...>>'
MSG_SIZE = ($ - msg)
blank    db MSG_SIZE dup(' ')

row db 24
col db 79 - MSG_SIZE

.code
main proc
    mov ax,@data      ; init data segment
    mov ds,ax
    mov ah,0          ; set video mode to mode
    mov al,7          ; 7, to enable blinking
    int 10h

    mov bl,blinking   ; attribute
    mov dh,row
    mov dl,col
    mov si,offset msg
    mov cx,MSG_SIZE
    call show_text

    mov ah,1          ; wait for keystroke
    int 21h

    mov bl,normal     ; attribute
    mov si,offset blank
    mov cx,MSG_SIZE
    call show_text

    mov ax,4C00h      ; end program
    int 21h
main endp

```

```

; Write a string with a given attribute. Input:
; DH, DL = row, column position, SI = offset of message,
; CX = message length, BL = attribute.

show_text proc
    pusha

L1:
    push cx          ; save loop counter
    mov ah,2        ; set cursor position
    mov bh,0        ; video page 0
    int 10h         ; DX = row,col

    mov ah,9        ; write character & attribute
    mov al,[si]     ; character
    mov bh,0        ; video page 0
    mov cx,1        ; repetition count
    int 10h

    inc dl          ; increment column
    inc si          ; point to next character
    pop cx          ; restore loop counter
    loop L1

    popa
    ret
show_text endp
end main

```

Chapt 5 Ex 17: Null Attributes

```

; Write blanks with null attributes to line 10 on
; the screen. Then, write a string to the same
; screen location, using INT 21h.

; The text never appears, of course, because INT 21h
; does not modify existing screen attributes when it
; writes to the console. The null attributes make the
; text invisible.

.model small
.stack 100h
.data
message db "Can you see this string? $"

.code
main proc
    mov ax,@data    ; init data segment
    mov ds,ax

    ; Set the cursor position.

    mov ah,2        ; BIOS function
    mov bh,0        ; video page 0

```

```

    mov    dh,10          ; set row and column
    mov    dl,0
    int    10h

; Display a line of spaces with null attributes.

    mov    ah,9          ; display character/attribute
    mov    al,20h        ; blank character
    mov    bh,0          ; video page 0
    mov    bl,0          ; null video attribute
    mov    cx,50         ; character count
    int    10h

; Display a line of text, using INT 21h.

    mov    ah,9
    mov    dx,offset message
    int    21h

    mov    ax,4C00h      ; end program
    int    21h
main endp
end main

```

Chapt 5 Ex 18: Day Number of any Date

```

; Prompt the user for a date. Determine the day
; number and display the day of the week as a string.

; I recommend letting students use the link library
; procedures when inputting the date from the user.

; It is interesting to note whether students remember
; to save and restore the current system date.

; This implementation requires the use of CMP and JE
; instructions, which are covered in Chapter 6.

.model small
.stack 100h

.data
month db ?
day   db ?
year  dw ?

saveMonth db ?
saveDay   db ?
saveYear  dw ?

askMonth db "Month: ",0
askDay   db "Day: ",0
askYear  db "Year: ",0

; Define a table of day names, making sure that

```

```

; each entry is the same length by padding with
; spaces.

dayNames label byte
db "Sunday ",0
DAYSIZE = ($ - dayNames)
db "Monday ",0
db "Tuesday ",0
db "Wednesday",0
db "Thursday ",0
db "Friday ",0
db "Saturday ",0

.code
include library.inc

main proc
    mov ax,@data            ; init data segment
    mov ds,ax
    call ClrScr
    call SaveCurrentDate
    call InputDate         ; get date from user

    ; Set the current date to the user's values.

    mov ah,2Bh
    mov cx,year
    mov dh,month
    mov dl,day
    int 21h
    cmp al,0
    jne restoreDate       ; quit if not successful

    ; Get the date, including the dayOfWeek.

    mov ah,2Ah
    int 21h                ; AL = day of week
    mov dx,offset dayNames ; point to table
    mov ah,0
    cmp ax,0               ; Sunday?
    je  printName         ; yes: print now

    mov cx,ax              ; no: find table entry
L1: add dx,DAYSIZE        ; calculate position
    loop L1

printName:
    call Writestring
    call Crlf

restoreDate:
    call RestoreCurrentDate
    mov ax,4c00h          ; end program
    int 21h
main endp

```

```

; Restore the original system date.

RestoreCurrentDate proc
    mov  ah,2Bh
    mov  cx,saveYear
    mov  dh,saveMonth
    mov  dl,saveDay
    int  21h
    ret
RestoreCurrentDate endp

; Get the current date and save it.

SaveCurrentDate proc
    mov  ah,2Ah
    int  21h                ; AL = day of week
    mov  saveMonth,dh
    mov  saveDay,dl
    mov  saveYear,cx
    ret
SaveCurrentDate endp

; Ask the user for a new month, day, and year.

InputDate proc
    mov  dx,offset askMonth
    call Writestring
    call Readint
    call Crlf
    mov  month,al
    mov  dx,offset askDay
    call Writestring
    call Readint
    call Crlf
    mov  day,al
    mov  dx,offset askYear
    call Writestring
    call Readint
    call Crlf
    mov  year,ax
    ret
InputDate endp
end main

```

Chapt 5 Bonus Exercise: Line-Drawing Editor

```

; This program interactively draws a line on the screen using
; character graphics in text mode. When running this program,
; type Q to quit. Use the arrow keys to change direction
; and guide the path of the line.

; Challenging program.
; This is the solutin to an exercise from Chapter 5 in the First
; Edition. Solution by Bob Galivan.

```

```

.Model small
.Stack 100h

.Code
Main      Proc
    mov    ax, @data          ;initialize the data segment so the program
    mov    ds, ax            ;can address the variables

ClearScreen:
    mov    ah, 07h           ; calling BIOS function 7, scroll window down
    mov    al, 00h           ; blanking the entire screen
    mov    bh, 07h           ; in normal video
    mov    cx, 0000h         ; row 1, col 1 start
    mov    dx, 184Fh         ; row 25, col 80 finish
    int    10h

PlaceCursor:
    mov    ah, 02h           ; calling BIOS function 2, set cursor position
    mov    bh, 00h           ; on video page 0
    mov    dx, 0000h         ; at row 1, col 1
    int    10h

ReadCharacter:
    mov    ah, 07h           ; calling DOS function 8, character input
    int    21h               ; without echo

EvaluateCharacter:
    cmp    al, 'q'           ; time to quit program
    jne    CheckExtendChar   ; if user enters q
    jmp    Exit

CheckExtendChar:
    cmp    al, 00h           ; if al = 0, an extended key was pressed
    je     GetExtendChar     ; If not, it was an invalid key
    jmp    return            ; and we cycle around again

GetExtendChar:
    int    21h               ; do a second read to get the character

CheckLeftArrow:
    cmp    al, LArrow        ; was the key the left arrow key
    jne    CheckRightArrow   ; if not, check the next key
    mov    cl, direction      ; put the direction we are coming from into cl
    mov    direction, 'l'    ; and set the direction to left
    cmp    cl, 'l'           ; If we are coming from the left
    je     DrawLeftLine      ; then we continue to the left
    cmp    cl, 'u'           ; If we were going up,
    je     LtDrawTopRight    ; then we need a top right corner
    cmp    cl, 'd'           ; if we were going down,
    je     LtDrawBottomRight ; then we need a bottom right corner
    jmp    DrawLeftLine      ; If we get here, we were coming from the right

LtDrawBottomRight:
    mov    ah, 09h           ; Calling BIOS function 9, write character
    mov    al, BottomRight   ; and attribute. We write the bottom right
    mov    bh, 00h           ; character on vidoe page 0, with a normal
    mov    bl, 07h           ; attribute,
    
```

```

    mov     cx, 01h           ; one time
    int     10h
    jmp     SetLeftCursor    ; and then position the left cursor

LtDrawTopRight:
    mov     ah, 09h           ; calling BIOS function 9, write character
    mov     al, TopRight      ; and attribute. We write a top right character
    mov     bh, 00h           ; on video page 0
    mov     bl, 07h           ; with a normal attribute,
    mov     cx, 01h           ; one character
    int     10h
    jmp     SetLeftCursor    ; and then position the left cursor

DrawLeftLine:
    mov     ah, 09h           ; calling BIOS function 9, write character and
    mov     al, HorizLine     ; attribute. We are writing a horizontal line
    mov     bh, 00h           ; on video page 0
    mov     bl, 07h           ; with a normal attribute,
    mov     cx, 01h           ; one character
    int     10h

SetLeftCursor:
    mov     cl, MinCol        ; Put the minimum column into cl
    cmp     CurrentCol, cl    ; If we are below the minimum column,
    ja     PositionLeftCursor ; we cannot go further left. Otherwise
    jmp     PositionCursor    ; we move one column to the left

PositionLeftCursor:
    dec     CurrentCol        ; by decrementing the current column
    jmp     PositionCursor    ; and positioning the cursor

CheckRightArrow:
    cmp     al, RArrow        ; Was the key a right arrow ?
    jne     CheckUpArrow     ; if not, check the next key

CheckRightDirection:
    mov     cl, Direction     ; put the direction we are coming from
    mov     direction, 'r'    ; into cl, and set the new direction to right
    cmp     cl, 'r'           ; are we travelling right ?
    je     DrawRight         ; If so, draw the line to the right
    cmp     cl, 'u'           ; were we going up ?
    je     RtDrawTopLeft     ; then we draw the top left corner
    cmp     cl, 'd'           ; if we were going down,
    je     RtDrawBottomLeft  ; then we need the bottom left corner
    cmp     cl, ' '           ; A space means we have just started
    jmp     RtDrawTopLeft    ; so we draw a top left corner

RtDrawBottomLeft:
    mov     ah, 09h           ; calling BIOS function 9, write character
    mov     al, BottomLeft    ; and attribute. We write a bottom left character
    mov     bh, 00h           ; on video page 0
    mov     bl, 07h           ; with a normal attribute,
    mov     cx, 01h           ; one character
    int     10h
    jmp     SetRightCursor   ; and reset the cursor

RtDrawTopLeft:

```

```

    mov  ah, 09h           ; calling BIOS function 9, write character and
    mov  al, TopLeft      ; attribute. Writing a Top left character
    mov  bh, 00h          ; on video page 0
    mov  bl, 07h          ; with a normal attribute,
    mov  cx, 01h          ; one character
    int  10h
    jmp  SetRightCursor   ; and reset the cursor

DrawRight:
    mov  ah, 09h           ; If we got here, it is ok to write
    mov  al, HorizLine    ; the horizontal line character
    mov  bh, 00h          ; on video page 0
    mov  bl, 07h          ; with a normal attribute,
    mov  cx, 01h          ; write one character
    int  10h

SetRightCursor:
    mov  cl, MaxCol       ; put the maximum columns in cl to compare
    cmp  CurrentCol, cl   ; If we are below the maximum column,
    jb   PositionRightCursor ; we can reset the cursor
    jmp  PositionCursor   ; otherwise, we can go no further right

PositionRightCursor:
    inc  CurrentCol       ; increment the current column
    jmp  PositionCursor   ; and move the cursor one column to the right

CheckUpArrow:
    cmp  al, UpArrow      ; was the key an Up arrow ?
    jne  CheckDnArrow     ; If not, then we check the next key
    mov  cl, direction    ; save the direction we came from
    mov  direction, 'u'   ; and set it to where we are going
    cmp  cl, 'u'          ; were we going up ?
    je   DrawUpLine       ; then continue drawing
    cmp  cl, 'r'          ; if we were coming right, then
    je   UpDrawBottomRight ; then we need a bottom right corner
    cmp  cl, 'l'          ; if we were going left,
    je   UpDrawBottomLeft  ; then we need the bottom left corner

UpDrawBottomLeft:
    mov  ah, 09h           ; calling BIOS function 9, write character
    mov  al, BottomLeft   ; and attribute. Writing the bottom left corner
    mov  bh, 00h          ; on video page 0
    mov  bl, 07h          ; with a normal attribute
    mov  cx, 01h          ; write one character
    int  10h
    jmp  SetUpCursor      ; and resetting the cursor

UpDrawBottomRight:
    mov  ah, 09h           ; calling BIOS function 9, write character and
    mov  al, BottomRight  ; attribute. Writing the bottom right corner
    mov  bh, 00h          ; on video page 0
    mov  bl, 07h          ; with a normal attribute
    mov  cx, 01h          ; write one character
    int  10h
    jmp  SetUpCursor

DrawUpLine:

```

```

    mov  ah, 09h           ; It is OK to write the vertical
    mov  al, VertLine     ; line character
    mov  bh, 00h         ; on video page 0
    mov  bl, 07h         ; with a normal attribute
    mov  cx, 01h         ; write one character
    int  10h

SetUpCursor:
    mov  cl, MinRow      ; put the minimum rows in cl
    cmp  CurrentRow, cl  ; have we reached the minimum ?
    jbe  return          ; if so, there is nothing else to do
    dec  CurrentRow      ; otherwise, we decrement our row
    jmp  PositionCursor  ; and reposition the cursor

CheckDnArrow:
    cmp  al, DnArrow     ; was the key the down arrow ?
    jne  return          ; If not, then we return to the top of the loop
    mov  cl, direction   ; Else, we put the direction we are coming from
    mov  direction, 'd'  ; into cl, and set the new direction.
    cmp  cl, 'd'         ; are we still going down ?
    je   DrawDownLine   ; if so, continue
    cmp  cl, 'r'         ; If we were coming from the right
    je   DnDrawTopRight ; then we need a top right corner
    cmp  cl, 'l'         ; If we were coming from the left
    je   DnDrawTopLeft  ; then we need a top left corner

DnDrawTopLeft:
    mov  ah, 09h         ; calling BIOS function 9, write character and
    mov  al, TopLeft     ; attribute. Writing a top left corner
    mov  bh, 00h         ; on video page 0
    mov  bl, 07h         ; with a normal attribute
    mov  cx, 01h         ; write one character
    int  10h
    jmp  SetDownCursor   ; and resetting the cursor

DnDrawTopRight:
    mov  ah, 09h         ; calling BIOS function 9, write character and
    mov  al, TopRight    ; attribute. Writing a top right corner
    mov  bh, 00h         ; on video page 0
    mov  bl, 07h         ; with a normal attribute
    mov  cx, 01h         ; write one character
    int  10h
    jmp  SetDownCursor   ; and resetting the cursor

DrawDownLine:
    mov  ah, 09h         ; It is OK to write the vertical
    mov  al, VertLine     ; line character
    mov  bh, 00h         ; on video page 0
    mov  bl, 07h         ; with a normal attribute
    mov  cx, 01h         ; write one character
    int  10h

SetDownCursor:
    mov  cl, MaxRow      ; put the maximum rows in cl
    cmp  CurrentRow, cl  ; if we have exceeded them, we
    jae  return          ; return to the top of the loop
    inc  CurrentRow      ; otherwise, we increment the current row

```

```

        jmp     PositionCursor      ; and position the cursor

PositionCursor:
    mov     ah, 02h                ; Calling BIOS function 2, set cursor position
    mov     bh, 0                  ; on video page 0
    mov     dh, CurrentRow         ; at the current row
    mov     dl, currentCol         ; and the current Column
    int     10h

Return:
    jmp     ReadCharacter          ; return to the top of the loop for another
                                   ; round

Exit:
    mov     ax, 4C00h              ; load the terminate function
    int     21h                   ; and end the program
Main      Endp

.data
    currentRow    db  00h          ; What is our current row
    CurrentCol    db  00h          ; What is our current column
    MinRow        db  00h          ; what is our minimum row
    MinCol        db  00h          ; and column position
    MaxRow        db  18h          ; and our maximum row
    MaxCol        db  4Fh          ; and column position
    Direction     db  ' '         ; What direction are we going in

;Character definitions
    UpArrow       db  48h
    DnArrow       db  50h
    LArrow        db  4Bh
    RArrow        db  4Dh
    TopLeft       db  0DAh
    TopRight      db  0BFh
    BottomLeft    db  0C0h
    BottomRight   db  0D9h
    HorizLine     db  0C4h
    VertLine      db  0B3h

end Main

```

CHAPTER 6

Chapt 6 Ex 1: Reading Keyboard Arrow Keys

```

; Write a procedure called GetArrow that inputs a keyboard scan code
; with INT 16h, checks to see if the key is a left arrow, right
; arrow, etc., and returns an integer in AX identifying the arrow.

;-----
; Suggested improvement (after reading chapter 7): Create a table of
; strings representing the key names, and index into the table to
; get the appropriate string. This requires the MUL operator:

```

```

;keyNames \
; db "None      ",0
; db "Up arrow  ",0
; db "Right arrow",0
; db "Down arrow ",0
; db "Left arrow ",0
;-----

.model small
.stack 100h

LEFT = 4Bh
RIGHT = 4Dh
UP = 48h
DOWN = 50h
NONE = 0

.data
rightMsg db "Right arrow",0
leftMsg  db "Left arrow",0
upMsg    db "Up arrow",0
downMsg  db "Down arrow",0

.code
include library.inc

main proc
    mov ax,@data      ; init data segment
    mov ds,ax
    call ClrScr

L1:  call GetArrow
    cmp ax,0          ; not an arrow key?
    je  Quit          ; quit
    jmp L1

Quit:
    mov ax,4c00h      ; end program
    int 21h
main endp

; Input a keystroke, return one of the following values in AX,
; depending on which key was pressed: 1=up, 2=right, 3=down,
; 4=left, 0=none.

GetArrow proc
    push dx
    mov ah,10h        ; input keystroke (AH = scan code)
    int 16h
    mov dl,ah         ; make copy of scan code

    cmp dl,UP
    jne GA1
    mov ax,1
    mov dx,offset upMsg

```

```

        call Writestring
        jmp GA5

GA1:
    cmp dl,RIGHT
    jne GA2
    mov ax,2
    mov dx,offset rightMsg
    call Writestring
    jmp GA5

GA2:
    cmp dl,DOWN
    jne GA3
    mov ax,3
    mov dx,offset downMsg
    call Writestring
    jmp GA5

GA3:
    cmp dl,LEFT
    jne GA4
    mov ax,4
    mov dx,offset leftMsg
    call Writestring
    jmp GA5

GA4:
    mov ax,0                ; unknown key

GA5:
    call Crlf
    pop dx
    ret
GetArrow endp
end main

```

Chapt 6 Ex 2: Vertical Bar Menu

```

; Display a vertical menu and let the user move a highlighted
; bar up and down across menu choices by pressing keyboard
; arrow keys.

```

```

; The challenge in writing this program is to create
; procedures that have clearly defined tasks. In this way,
; we can reduce the amount of redundant code.

```

```

.model small
.286
.stack 100h

```

```

ESC_KEY = 1Bh

UP_ARROW = 48h
DOWN_ARROW = 50h

.data
screenPosition  db  5,10      ; upper-left row, column
numberOfEntries dw  3         ; number of entries
unselectedColor db  70h      ; color of unselected entries
selectedColor   db  20h      ; color of selected entries

menuText  \
  db  "Choice One   ",0
ENTRY_SIZE = ($ - menuText)
  db  "Choice Two   ",0
  db  "Choice Three ",0

currentRow dw  0
prompt db "Press the Esc key to cancel the menu.",0

.code
include library.inc

main proc
  mov  ax,@data          ; init data segment
  mov  ds,ax
  call ClrScr
  mov  dx,offset prompt
  call Writestring

  call DrawVerticalMenu
  mov  currentRow,0

L1:
  call ProcessVerticalKey
  cmp  al,ESC_KEY
  jne  L1

  mov  ax,4c00h          ; end program
  int  21h
main endp

; Read the keyboard and move the menu bar according to the
; vertical arrow key that was pressed. Exit if the user presses
; the Esc key

ProcessVerticalKey proc
  mov  ah,10h           ; read keyboard
  int  16h
  cmp  al,ESC_KEY      ; exit if Esc pressed
  je   PK4
  cmp  ah,UP_ARROW
  jne  PK2
  cmp  currentRow,0    ; if currentRow > 0 then
  jbe  PK2
  mov  ah,unselectedColor ; unselect the current entry

```

```

    call DrawEntry
    dec  currentRow          ; decrement the row number
    mov  ah,selectedColor   ; select the new entry
    call DrawEntry

PK2:
    cmp  ah,DOWN_ARROW
    jne  PK4
    mov  dx,numberOfEntries ; get number of entries
    dec  dx                 ; adjust to zero-based index
    cmp  dx,currentRow     ; any rows below currentRow?
    jbe  PK4               ; if so,

    mov  ah,unselectedColor ; unselect the current entry
    call DrawEntry
    inc  currentRow        ; increment the row number
    mov  ah,selectedColor  ; select the new entry
    call DrawEntry

PK4:
    ret
ProcessVerticalKey endp

```

; Draw the menu. The first item is automatically selected.

```

DrawVerticalMenu proc
    pusha
    mov  di,offset selectedColor
    mov  ah,[di]           ; selected attribute
    mov  currentRow,0
    call DrawEntry        ; draw the first entry

    mov  di,offset numberOfEntries
    mov  cx,[di]          ; number of entries
    dec  cx               ; (remaining number)
    mov  di,offset unselectedColor
    mov  ah,[di]         ; unselected attribute

```

```

DM1:
    inc  currentRow
    call DrawEntry
    loop DM1

```

```

    popa
    ret
DrawVerticalMenu endp

```

; Draw the menu entry identified by currentRow with
; the attribute in AH.

```

DrawEntry proc
    pusha
    mov  cx,currentRow
    mov  di,offset screenPosition
    mov  dh,[di]          ; starting screen row
    mov  dl,[di+1]       ; starting screen column

```

```

    mov  si,offset menuText

; Use a loop to advance the menuText table
; pointer to the right entry and to increment
; the screen row position.

    cmp  cx,0
    je   DE2
DE1:
    inc  dh
    add  si,ENTRY_SIZE      ; next text entry
    loop DE1

DE2:
    call Writestring_direct ; display current entry
    popa
    ret
DrawEntry endp
end main

```

Chapt 6 Ex 3: Horizontal Bar Menu

```

; Display a horizontal menu and let the user move a highlighted
; bar left and right across menu choices by pressing keyboard
; arrow keys.

; The program solution to Exercise 2 was used as a starting
; pointer for this program.

.model small
.286
.stack 100h

ESC_KEY = 1Bh

LEFT_ARROW = 4Bh
RIGHT_ARROW = 4Dh

.data
screenPosition  db  5,0      ; upper-left row, column
numberOfEntries dw  3        ; number of entries
unselectedColor db  70h     ; color of unselected entries
selectedColor   db  20h     ; color of selected entries

menuText  \
db  "ChoiceOne  ",0
ENTRY_SIZE = ($ - menuText)
db  "ChoiceTwo  ",0
db  "ChoiceThree",0

currentCol dw  0
prompt db  "Press the Esc key to cancel the menu.",0

.code
include library.inc

```

```

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call ClrScr
    mov dx,offset prompt
    call Writestring

    call DrawHorizontalMenu
    mov currentCol,0

L1:
    call ProcessHorizontalKey
    cmp al,ESC_KEY
    jne L1

    mov ax,4c00h          ; end program
    int 21h
main endp

; Read the keyboard and move the menu bar according to the
; horizontal arrow key that was pressed. Exit if the user presses
; the Esc key

ProcessHorizontalKey proc
    mov ah,10h            ; read keyboard
    int 16h
    cmp al,ESC_KEY        ; exit if Esc pressed
    je PK4
    cmp ah,LEFT_ARROW
    jne PK2
    cmp currentCol,0      ; if currentCol > 0 then
    jbe PK2
    mov ah,unselectedColor ; unselect the current entry
    call DrawEntry
    dec currentCol        ; decrement the column number
    mov ah,selectedColor  ; select the new entry
    call DrawEntry

PK2:
    cmp ah,RIGHT_ARROW
    jne PK4

    mov dx,numberOfEntries ; get number of entries
    dec dx                  ; adjust to zero-based index
    cmp dx,currentCol      ; any columns after currentCol?
    jbe PK4                ; no: skip
    mov ah,unselectedColor ; unselect the current entry
    call DrawEntry
    inc currentCol         ; increment the column number
    mov ah,selectedColor  ; select the new entry
    call DrawEntry

PK4:
    ret
ProcessHorizontalKey endp

```

; Draw the menu. The first item is automatically selected.

```

DrawHorizontalMenu proc
    pusha
    mov di,offset selectedColor
    mov ah,[di]          ; selected attribute
    mov currentCol,0
    call DrawEntry      ; draw the first entry

    mov di,offset numberOfEntries
    mov cx,[di]         ; number of entries
    dec cx              ; (remaining number)
    mov di,offset unselectedColor
    mov ah,[di]        ; unselected attribute

DM1:
    inc currentCol
    call DrawEntry
    loop DM1

    popa
    ret
DrawHorizontalMenu endp

; Draw the menu entry identified by currentCol with
; the attribute in AH.

DrawEntry proc
    pusha
    mov cx,currentCol
    mov di,offset screenPosition
    mov dh,[di]         ; starting screen row
    mov dl,[di+1]       ; starting screen column
    mov si,offset menuText

    ; Use a loop to advance the menuText table
    ; pointer to the right entry and to increment
    ; the screen column position.

    cmp cx,0
    je DE2
DE1:
    add dl,ENTRY_SIZE   ; screen column
    add si,ENTRY_SIZE   ; next text entry
    loop DE1

DE2:
    call Writestring_direct ; display current entry
    popa
    ret
DrawEntry endp
end main

```

Chapt 6 Ex 4: Alphabetic Input

```

; Input and display only letters A-Z from the keyboard
; until the Enter key is pressed.

; In this implementation we accept only capital letters.

.model small
.stack 100h

.code
include library.inc

main proc
    mov ax,@data           ; init data segment
    mov ds,ax
    call ClrScr

L1: mov ah,10h             ; wait for a keystroke
    int 16h
    cmp al,0Dh            ; Enter key pressed?
    je Quit
    cmp al,'A'
    jb L2
    cmp al,'Z'
    ja L2
    mov ah,2              ; Display the character
    mov dl,al
    int 21h

L2: jmp L1

Quit:
    mov ax,4c00h          ; end program
    int 21h
main endp
end main

```

Chapt 6 Ex 5: Signed Integer Input

```

; Let the user input a signed decimal integer
; from the console. Use a finite-state diagram, and
; Display an error message if invalid input is entered.

; Note: This solution program is identical to the one
; in Example 12, Chapter 6.

.model small
.stack 100h

DOS_CHAR_INPUT = 1
ENTER_KEY = 0Dh

.data
InvalidInputMessage db "Invalid input",0dh,0ah,0

.code

```

```

extrn Writestring:proc, Crlf:proc, Clrscr:proc

main proc
    mov ax,@data
    mov ds,ax
    call Clrscr

StateA:
    call Getnext          ; read next char into AL
    cmp al,'+'           ; leading + sign?
    je StateB            ; go to State B
    cmp al,'-'           ; leading - sign?
    je StateB            ; go to State C
    call Isdigit         ; ZF = 1 if AL contains a digit
    jz StateC
    call DisplayError    ; invalid input found
    jmp Exit

StateB:
    call Getnext          ; read next char into AL
    call Isdigit         ; ZF = 1 if AL contains a digit
    jz StateC
    call DisplayError    ; invalid input found
    jmp Exit

StateC:
    call Getnext          ; read next char into AL
    jz Exit              ; quit if Enter pressed
    call Isdigit         ; ZF = 1 if AL contains a digit
    jz StateC
    call DisplayError    ; invalid input found
    jmp Exit

Exit:
    call Crlf
    mov ax,4c00h
    int 21h
main endp

; Read char from standard input into AL,
; set ZF = 1 if Enter key was read.

Getnext proc
    mov ah,DOS_CHAR_INPUT ; read standard input
    int 21h                ; AL = character
    cmp al,ENTER_KEY
    ret
Getnext endp

; Set ZF = 1 if the character in AL is a digit.

Isdigit proc
    cmp al,'0'
    jb Al
    cmp al,'9'
    ja Al

```

```

        test ax,0      ; set ZF = 1
Al: ret
Isdigit endp

; Display error message.

DisplayError proc
    push dx
    mov dx,offset InvalidInputMessage
    call WriteString
    call Crlf
    pop dx
    ret
DisplayError endp
end main

```

Chapt 6 Ex 6: Real Number Input (Unsigned)

```

; Input an unsigned real number from the keyboard.
; Reject any characters except digits
; and a single decimal point. Quit when the Enter
; key is pressed.

.model small
.stack 100h

ENTER_KEY = 0Dh

.data
prompt db "Input a real number: ",0

.code
include library.inc

main proc
    mov ax,@data
    mov ds,ax
    call Clrscr
    mov dx,offset prompt ; display a prompt
    call Writestring

StateA:
    call Readkey          ; read character into AL
    cmp al,ENTER_KEY
    je quit
    call IsDigit
    jz StateB

StateB:
    call Readkey          ; read next char into AL
    cmp al,ENTER_KEY
    je quit              ; quit if Enter pressed
    call IsDigit         ; ZF = 1 if AL contains a digit
    jz StateB

```

```

        call IsPoint
        je   StateC

StateC:
    call Readkey      ; read next char into AL
    cmp  al,ENTER_KEY
    je   quit        ; quit if Enter pressed
    call Isdigit     ; ZF = 1 if AL contains a digit
    jz   StateC      ; repeat for more digits

quit:
    mov  ax,4C00h    ; end program
    int  21h
main  endp

; Return ZF = 1 if the character in AL is a digit.

IsDigit proc
    cmp  al,'0'      ; is the ASCII code < '0'?
    jb  ID1
    cmp  al,'9'      ; is the ASCII code > '9'?
    ja  ID1
    call Echo1
    test ax,0        ; set ZF = 1
ID1:  ret
IsDigit endp

; Return ZF = 1 if the character in AL is a decimal point.

IsPoint proc
    cmp  al,'.'
    jne IP1
    call Echo1
    test ax,0
IP1:  ret
IsPoint endp

Echo1 proc
    mov  dl,al       ; echo the character in AL
    mov  ah,2
    int  21h
    ret
Echo1 endp
end  main

```

Chapt 6 Ex 7: Real Number Input with Leading Sign

```

; Input a signed real number from the keyboard.
; Reject any characters except a leading sign, digits,
; and a single decimal point. Quit when the Enter
; key is pressed.

```

```

; Note: The approach called for by this exercise and the
; previous one is a little different from the one in the
; chapter (Example 12). In the current program, we are

```

```
; told to ignore invalid keystrokes. This is done by
; placing a JMP instruction at the end of each state that
; returns to the same state label.

.model small
.stack 100h

ENTER_KEY = 0Dh

.data
prompt db "Input a real number: ",0

.code
include library.inc

main proc
    mov ax,@data
    mov ds,ax
    call Clrscr
    mov dx,offset prompt ; display a prompt
    call Writestring

StateA:
    call Readkey
    cmp al,ENTER_KEY
    je quit
    call IsSign
    jz StateB
    call IsDigit
    jz StateC
    jmp StateA ; ignore other characters

StateB:
    call Readkey ; read character into AL
    cmp al,ENTER_KEY
    je quit
    call IsDigit
    jz StateC
    jmp StateB ; ignore other characters

StateC:
    call Readkey ; read next char into AL
    cmp al,ENTER_KEY
    je quit ; quit if Enter pressed
    call IsDigit ; ZF = 1 if AL contains a digit
    jz StateC
    call IsPoint
    je StateD
    jmp StateC ; ignore other characters

StateD:
    call Readkey ; read next char into AL
    cmp al,ENTER_KEY
    je quit ; quit if Enter pressed
    call Isdigit ; ZF = 1 if AL contains a digit
    jz StateD ; repeat for more digits
```

```

        jmp Stated          ; ignore other characters

quit:
    mov ax,4C00h          ; end program
    int 21h
main endp

; Return ZF = 1 if the character in AL is a digit.

IsDigit proc
    cmp al,'0'           ; is the ASCII code < '0'?
    jb ID1
    cmp al,'9'           ; is the ASCII code > '9'?
    ja ID1
    call Echo1
    test ax,0             ; set ZF = 1
ID1: ret
IsDigit endp

; Return ZF = 1 if the character in AL is a decimal point.

IsPoint proc
    cmp al','.'
    jne IP1
    call Echo1
    test ax,0
IP1: ret
IsPoint endp

IsSign proc
    cmp al,'+'
    je IS1
    cmp al,'-'
    je IS1
    jmp IS2

IS1: call Echo1
    test ax,0

IS2: ret
IsSign endp

Echo1 proc
    mov dl,al            ; echo the character in AL
    mov ah,2
    int 21h
    ret
Echo1 endp
end main

```

Chapt 6 Ex 8: Reverse an Array

```

; Reverse an array of 16-bit integers, in place. Display
; the array before and after the reversal.
; Note: some of the code from the Chapter 4 Exercise 16

```

```

; solution program was transplanted and adapted for this
; program.

.model small
.stack 100h
.data
array dw 1,2,3,4,5,6,7,8,9,10
COUNT = ($ - array) / (type array)
array_last = $ - (type array)

.code
include library.inc

main proc
    mov ax,@data           ; init data segment
    mov ds,ax
    call ClrScr
    call ShowArray

    ; Reverse the array.

    mov cx,(COUNT / 2)
    mov si,offset array
    mov di,array_last

L1:
    mov ax,[si]           ; get integer from the source
    xchg [di],ax          ; exchange with destination
    mov [si],ax           ; save back in source
    add si,type array     ; increment index
    sub di,type array
    loop L1               ; repeat loop

    call ShowArray

    mov ax,4c00h          ; end program
    int 21h
main endp

; Display the array.

ShowArray proc
    mov si,offset array
    mov cx,COUNT

SA1:
    mov ax,[si]
    mov bx,10             ; display it in decimal
    call Writeint
    call Crlf
    add si,type array     ; increment index
    loop SA1
    ret
ShowArray endp
end main

```

Chapt 6 Ex 9: Deviation of Array Values

```

; Calculate and display the amount of variance between
; each element of a 16-bit integer array and the array's
; arithmetic mean.

; Note: This solution program requires an understanding
; of the 32-bit DIV instruction. See the line in the
; program listing labeled NEEDS EXPLANATION.

.model small
.stack 100h
.386
.data
array dw 1000,1100,2000,2200,3000,3300,4000,4400
      dw 5000,5500,6000,6600,7000,7700,8000,8800
COUNT = ($ - array) / (type array)

mean dd ?
showMeanMsg db "Arithmetic mean (integer): ",0
heading db 0dh,0ah
         db "Value - Variance",0dh,0ah
         db "-----",0dh,0ah,0

.code
include library.inc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call ClrScr
    call CalculateMean

; Display the mean.

    mov dx,offset showMeanMsg
    call Writestring
    mov ebx,10            ; display EAX in decimal
    mov eax,mean
    call Writelong
    call Crlf

; Calculate and display each of the array values
; and their variances.

    mov dx,offset heading
    call Writestring

    mov cx,COUNT
    mov eax,0            ; use 32-bit accumulator
    mov si,offset array
L1:
    mov ax,[si]          ; get array value
    call Writeint        ; display it
    call Writespaces
    sub ax,word ptr mean ; subtract the mean

```

```

        jns     L2                ; result negative?
        neg     ax                ; convert to positive

L2:
        call   Writeint          ; display the difference
        call   Crlf
        add    si,type array     ; point to next array value
        loop   L1

        mov    ax,4c00h          ; end program
        int   21h
main endp

; Calculate and store the arithmetic mean.

CalculateMean proc
    pusha
    mov     cx,COUNT
    mov     eax,0                ; use 32-bit accumulator
    mov     si,offset array
CM1:
    movzx   edx,word ptr[si] ; move and zero-extend
    add     eax,edx             ; add to accumulator
    add     si,type array
    loop   CM1

    ; Divide EDX:EAX by EBX, giving EAX.

    mov     edx,0
    mov     ebx,COUNT
    div     ebx                 ; NEEDS EXPLANATION
    mov     mean,eax
    popa
    ret
CalculateMean endp

Writespaces proc
.data
tspaces db " - ",0
.code
    push   dx
    mov    dx,offset tspaces
    call  Writestring
    pop    dx
    ret
Writespaces endp
end main

```

Chapt 6 Ex 10: Display a String in Reverse

```

; Input a string and redisplay it with all of its
; characters reversed.

```

```

.model small
.stack 100h

```

```

.data
string db 128 dup(0)           ; the input string

.code
include library.inc

main proc
    mov     ax,@data
    mov     ds,ax
    call    Clrscr

    mov     dx,offset string
    call    Readstring         ; output: AX = length
    call    ReverseString
    call    Crlf

    mov     ax,4c00h
    int     21h
main endp

; Display a string in reverse. DX points to the
; string and AX contains its length.

ReverseString proc
    mov     cx,ax              ; loop counter
    mov     si,dx              ; point to string
    add     si,ax              ; point to last char
    dec     si

RS1:
    mov     dl,[si]           ; get character
    cmp     dl,0
    je     RS2
    mov     ah,2              ; display it
    int     21h
    dec     si
    loop   RS1

RS2:
    ret
ReverseString endp
end main

```

Chapt 6 Ex 11: Reverse the Words in a String

```

; Let the user input a string. Redisplay the
; string with the words in reverse order.

.model small
.stack 100h

.data
buffer db 81                 ; maximum length as specified in
book

```

```

        db      ?                ; length of data keyed
string  db      81 dup (?)      ; the input string
crlf   db      0Dh,0Ah,'$'

.code
main    proc
        mov     ax,@data        ; establish segment register
contents
        mov     ds,ax          ; so that data may be accessed
        call    ReverseString   ; Call main routine
        mov     ax,4c00h
        int     21h            ; Exit the program
main    endp

ReverseString proc
        call    get_string      ; get a string, set si to the end
        or     si,si           ; if ZF, then no data was entered
        jz     zero_length     ; can't reverse a zero length string
next_word:
        mov     cx,0           ; set length of word to zero
again:
        cmp     string-1[si], 20h ; have we reached a space ?
        je     a_word          ; word ends in one or more spaces
        inc     cx              ; add 1 to length of the word
        dec     si              ; move toward the front of the
string
        jz     done            ; until we reach the very beginning
        jmp    again           ; check the next character
a_word:
        call    show_word       ; display a word
        mov     cx,1           ; we know we have a space
sep_len:
        dec     si              ; move towards beginning of string
        jz     done            ; until we reach the very beginning
        cmp     string-1[si], 20h ; is the seperator over yet?
        jne    sep_done        ; if not, print it
        inc     cx              ; and increase the length counter
        jmp    sep_len         ; keep looking for a space
sep_done:
        call    show_word       ; don't care if word or separator
        jmp    next_word       ; next word
done:    call    show_word       ; display final word
zero_length:
        ret
ReverseString endp

get_string proc                    ; get string, set SI to end
        mov     ah,0ah          ; buffered character input (a
string)
        mov     dx,offset buffer ; point dx to data storage area
        int     21h            ; request service (get a string)
        mov     al,buffer+1
        cbw
        mov     si,ax           ; the length of the string
        mov     dx, offset crlf ; send a carrige return linefeed
        mov     ah, 09h

```

```

        int     21h
        ret                                ; return to calling routine
get_string endp

show_word proc                                ; display a word. CX = length
    mov     ah,40h                          ; write characters specified in CX
    mov     bx,1                            ; to the standard output
    lea     dx,string[si]                   ; starting address of data
    int     21h
    ret
show_word endp
end main

```

Chapt 6 Ex 12: Count the Words in a File

```

; Read a text file from standard input and count the
; number of words and characters. Use INT 21h
; Function 6.

; This is a fairly easy program to write, compared to later
; exercises in this chapter. This solution program assumes
; that words are separated by a single space. If the file
; contains consecutive spaces, each space would be counted
; as a word. As a bonus exercise, ask students to
; include logic that skips consecutive spaces.

; If students use INT 21h function 6, the program is
; almost impossible to debug with an interactive debugger.
; Students might want to temporarily use Function 1 and
; designate a special key such as Ctrl-Z to signal end
; of input.

.model small
.stack 100h
.data
charCount dw 0                ; character count
wordCount dw 0                ; word count

charMsg db "Number of characters: ",0
wordMsg db "Number of words:   ",0

.code
include library.inc

main proc
    mov     ax,@data           ; init data segment
    mov     ds,ax
    call   ClrScr

L1: mov     ah,6                ; DOS function 6
    mov     dl,0FFh           ; input a character
    int     21h                ; AL = character
    jz     L8                  ; ZF = 1 if no more data
    inc     charCount          ; increment character count
    cmp     al,' '             ; space (end of word) ?

```

```

        jne L2                ; no
        inc wordCount        ; yes - increment count
L2:
        jmp L1

L8:
        inc wordCount        ; count the last word

; Display the results.

        mov bx,10            ; decimal radix
        mov dx,offset charMsg ; "Number of characters: "
        call Writestring
        mov ax,charCount
        call Writeint
        call Crlf

        mov dx,offset wordMsg ; "Number of words: "
        call Writestring
        mov ax,wordCount
        call Writeint
        call Crlf

        mov ax,4c00h        ; end program
        int 21h
main endp
end main

```

Chapt 6 Ex 13: Console Input Field

```

; Write a procedure that displays an input field on
; the screen, waits for user input, and returns
; the characters typed. Terminate when either the
; Enter key or Tab key is pressed.

; This program is particularly challenging because of the
; many possible variations in user input. For example, the
; user might press function keys. Or, the user might press
; the Bksp key before entering any other characters. This
; program definitely requires the use of an interactive
; debugger.

; This program would be easier to write if the STRUC directive
; were available, but STRUC is not introduced until Chapter 8.
; You may want to let students read ahead to that topic before doing
; this project.

; An important strategy in this program is to make the code as
; transportable as possible to related exercises in this chapter
; (14,15,16,17).

.model small
.286
.stack 100h
.data

```

```

; The following data structure is required by the
; ShowInputField procedure.

field1Length = 10

field1 \
    dw field1Length
    db 5,12                ; row, column position
    db 70h                 ; attribute
    db ?                   ; count actual chars typed
    db field1Length dup(0),0

.code
include library.inc

main proc
    mov ax,@data           ; init data segment
    mov ds,ax
    call ClrScr

    mov si,offset field1
    call ShowInputField

    mov ax,4c00h           ; end program
    int 21h
main endp

; SI points to a structure containing the following input
; field data: field length (word), screen row (byte), screen
; column (byte), color attribute (byte), input buffer (array
; of byte).

ENTER_KEY = 0Dh
TAB_KEY = 9
BKSP_KEY = 8
FILL_CHAR = '.'

ShowInputField proc
    pusha

    mov cx,[si]            ; field length
    mov dh,[si+2]         ; screen row
    mov dl,[si+3]         ; screen column
    mov ah,[si+4]         ; color attribute

    mov di,si              ; don't touch SI!
    add di,6               ; point to buffer
    mov al,FILL_CHAR

    push dx                ; save row,column

SIF1:                      ; fill with dots
    call Writechar_direct
    inc dl
    loop SIF1

```

```

        pop dx                ; restore row,column

; Begin entering characters into buffer.

        call Gotoxy          ; position cursor
        mov cx,0             ; character count

SIF2:
        mov ah,8             ; get keystroke, no echo
        int 21h              ; AL = ASCII code
        cmp al,0             ; extended key?
        jne SIF2A            ; no
        int 21h              ; yes - read extra kbyd byte
        jmp SIF2             ; look for another key

SIF2A:
        call CheckCharacter  ; echo normal characters
        cmp al,ENTER_KEY    ; look for exit keys
        je SIF9
        cmp al,TAB_KEY
        je SIF9
        cmp al,BKSP_KEY     ; look for backspace
        je SIF3
        mov [di],al         ; save the key
        inc di
        inc cx               ; add to character count
        cmp cx,[si]         ; max characters reached?
        jb SIF2
        jmp SIF9            ; exit if not

SIF3:
        cmp cx,0             ; first keystroke?
        je SIF2             ; yes: get another keystroke
        mov dx,offset EraseString ; erase the previous char
        call Writestring
        dec di               ; back up in the buffer
        dec cx               ; decrement character count
        jmp SIF2            ; get another keystroke

SIF9:
        mov [si+6],cx        ; save buffer count
        popa
        ret

.data
EraseString db BKSP_KEY, FILL_CHAR, BKSP_KEY,0
.code
ShowInputField endp

; Check the character in AL, and be careful not to
; echo certain characters.

CheckCharacter proc
        push di
        push dx

```

```

; The following special keys are not to be echoed.

CF1:
    cmp    al,ENTER_KEY      ; don't echo the Enter key
    je     CF2
    cmp    al,TAB_KEY        ; don't echo the Tab key
    je     CF2
    cmp    al,BKSP_KEY       ; don't echo the Bksp key
    je     CF2
    call   EchoKey

CF2:
    pop    dx
    pop    di
    ret
CheckCharacter endp

; Echo the character in AL, preserving all registers.

EchoKey proc
    push  ax
    push  dx
    mov   ah,2
    mov   dl,al
    int  21h
    pop   dx
    pop   ax
    ret
EchoKey endp
end main

```

Chapt 6 Ex 14: Console Input Field with Keyboard Filter

```

; Write a procedure that displays an input field on
; the screen, waits for user input, and returns
; the characters typed. Terminate when either the
; Enter key or Tab key is pressed. Pass a pointer to a null-
; terminated string containing the set of characters that the
; user will be permitted to type.

; Note: This is an extension of Exercise 13, of course, so we
; will use its solution program as a starting point.

; The FindInString procedure developed for this exercise is
; general enough to be useful in other programs.

.model small
.286
.stack 100h

ENTER_KEY = 0Dh
TAB_KEY = 9
BKSP_KEY = 8

.data

```

```

; The following data structure is required by the
; ShowInputField procedure.

field1Length = 10

field1 \
    dw field1Length
    db 5,12                ; row, column position
    db 70h                ; attribute
    db ?                  ; count actual chars typed
    dw field1Filter       ; pointer to filter string
    db field1Length dup(0),0

field1Filter db "0123456789",TAB_KEY,BKSP_KEY,ENTER_KEY,0

.code
include library.inc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call ClrScr

    mov si,offset field1
    call ShowInputField

    mov ax,4c00h          ; end program
    int 21h
main endp

; SI points to a structure containing the following input
; field data: field length (word), screen row (byte), screen
; column (byte), color attribute (byte), input buffer (array
; of byte).

FILL_CHAR = '.'

ShowInputField proc
    pusha

    mov cx,[si]           ; field length
    mov dh,[si+2]         ; screen row
    mov dl,[si+3]         ; screen column
    mov ah,[si+4]         ; color attribute

    mov di,si             ; don't touch SI!
    add di,8              ; point to buffer
    mov al,FILL_CHAR

    push dx               ; save row,column

SIF1:                    ; fill with dots
    call Writechar_direct
    inc dl
    loop SIF1

```

```

    pop dx                ; restore row,column

; Begin entering characters into buffer.

    call Gotoxy          ; position cursor
    mov cx,0            ; character count

SIF2:
    mov ah,8            ; get keystroke, no echo
    int 21h             ; AL = ASCII code
    cmp al,0            ; extended key?
    jne SIF2A           ; no
    int 21h             ; yes - read extra kbyd byte
    jmp SIF2            ; look for another key

SIF2A:
    call CheckFilter    ; check for valid characters
    cmp al,0            ; if AL = 0, then
    je SIF2             ; skip this character
    cmp al,ENTER_KEY    ; look for exit keys
    je SIF9
    cmp al,TAB_KEY
    je SIF9
    cmp al,BKSP_KEY     ; look for backspace
    je SIF3
    mov [di],al         ; save the key
    inc di
    inc cx               ; add to character count
    cmp cx,[si]         ; max characters reached?
    jb SIF2
    jmp SIF9            ; exit if not

SIF3:
    cmp cx,0            ; first keystroke?
    je SIF2             ; yes: get another keystroke
    mov dx,offset EraseString ; erase the previous char
    call Writestring
    dec di              ; back up in the buffer
    dec cx              ; decrement character count
    jmp SIF2            ; get another keystroke

SIF9:
    mov [si+6],cx       ; save buffer count
    popa
    ret

.data
EraseString db BKSP_KEY, FILL_CHAR, BKSP_KEY,0
.code
ShowInputField endp

; Check the character in AL against the current field's
; filter string. If it matches, echo the character. If it
; does not match, set AL to 0 so it will not be processed
; any further.

```

```

CheckFilter proc
    push di
    push dx

    mov di,[si+6]      ; get pointer to filter string
    call FindInString ; find AL in [di] string.
    jz  CF1           ; char found if ZF = 1
    mov al,0         ; not found, set AL = 0
    jmp CF2          ; and exit immediately

    ; The following special keys are not to be echoed.

CF1:
    cmp al,ENTER_KEY ; don't echo the Enter key
    je  CF2
    cmp al,TAB_KEY   ; don't echo the Tab key
    je  CF2
    cmp al,BKSP_KEY  ; don't echo the Bksp key
    je  CF2
    call EchoKey
    test al,0        ; set ZF = 1

CF2:
    pop dx
    pop di
    ret
CheckFilter endp

; Given a null-terminated string pointed to by DI and a
; character in AL, set ZF = 1 if the character exists in
; the string. Otherwise, clear ZF to 0.

FindInString proc
    pusha

FIS1:
    cmp [di],0      ; end of string?
    je  FIS2        ; if so, quit with ZF = 0
    cmp [di],al     ; check current character
    je  FIS3        ; if match found, exit with ZF = 1
    inc di          ; move to next character
    jmp FIS1

FIS2:
    or  al,1        ; clear ZF = 0

FIS3:
    popa
    ret
FindInString endp

; Echo the character in AL, preserving all registers.

EchoKey proc
    push ax

```

```

    push dx
    mov  ah,2
    mov  dl,al
    int  21h
    pop  dx
    pop  ax
    ret
EchoKey endp
end main

```

Chapt 6 Ex 15: Customer Account Program

```

; Display an account screen with input fields. As the
; user presses the Tab key, move to each subsequent field.
; End the input screen when the Enter key is pressed.

.model small
.286
.stack 100h

.data
    ScreenTitle    db  'ACCOUNT INPUT SCREEN',0
    AcctTitle      db  '    ACCT NUM:',0
    NameTitle      db  '    LAST NAME:',0
    PrevBalTitle   db  'PREV BALANCE:',0
    PymtsTitle     db  '    PAYMENTS:',0
    CrTitle        db  '    CREDITS:',0

StartOfPrompts   \
    dw  0419h          ; this section contains both
    dw  offset ScreenTitle ; screen positions for prompts
    dw  060Fh          ; and the addresses of the prompts
    dw  offset NameTitle  ; The first entry is the row and
    dw  080Fh          ; column position for the first
    dw  offset AcctTitle  ; prompt. The second entry is the
    dw  0A0Fh          ; address of the first prompt.
    dw  offset PrevBalTitle ; the display routine goes through
    dw  0C0Fh          ; this portion of the data, using
    dw  offset PymtsTitle ; these values to write to the
    dw  0E0Fh          ; screen
    dw  offset CrTitle

LastNameLen = 30
AcctNumLen = 6
PrevBalLen = 11
PaymentsLen = 11
CreditsLen = 11

LastName \
    dw  LastNameLen
    db  6,29          ; row, column position
    db  70h          ; attribute
    db  ?            ; count actual chars typed
    db  LastNameLen dup(0),0

```

```

AcctNum \
    dw AcctNumLen
    db 8,29                ; row, column position
    db 70h                ; attribute
    db ?                  ; count actual chars typed
    db AcctNumLen dup(0),0

PrevBal \
    dw PrevBalLen
    db 10,29              ; row, column position
    db 70h                ; attribute
    db ?                  ; count actual chars typed
    db PrevBalLen dup(0),0

Payments \
    dw PaymentsLen
    db 12,29              ; row, column position
    db 70h                ; attribute
    db ?                  ; count actual chars typed
    db PaymentsLen dup(0),0

Credits \
    dw CreditsLen
    db 14,29              ; row, column position
    db 70h                ; attribute
    db ?                  ; count actual chars typed
    db CreditsLen dup(0),0

    LineLength    dw 50
    HorizLine     db 196
    VertLine      db 179
    TopLeft       db 218
    Botleft       db 192
    TopRight      db 191
    BotRight      db 217

.Code
include library.inc

main proc
    mov     ax, @data        ; init data segment
    mov     ds, ax
    call   SetupScreen      ; Draw the input form
    call   WritePrompts    ; write prompts to the screen
    call   GetFormData      ; Get the user input

Exit:
    mov     ax, 4C00h        ; end program
    int     21h
main endp

; Get each of the input fields. If the Carry flag is set by
; ShowInputField, the user has pressed a key that ends
; all input for the form.

GetFormData proc

```

```

    mov si,offset LastName
    call ShowInputField
    jc GF9
    mov si,offset AcctNum
    call ShowInputField
    jc GF9
    mov si,offset PrevBal
    call ShowInputField
    jc GF9
    mov si,offset Payments
    call ShowInputField
    jc GF9
    mov si,offset Credits
    call ShowInputField
    jc GF9

GF9:
    ret
GetFormData endp

; SI points to a structure containing the following input
; field data: field length (word), screen row (byte), screen
; column (byte), color attribute (byte), input buffer (array
; of byte).

ENTER_KEY = 0Dh
TAB_KEY = 9
BKSP_KEY = 8
FILL_CHAR = '.'

ShowInputField proc
    pusha

    mov cx,[si]                ; field length
    mov dh,[si+2]              ; screen row
    mov dl,[si+3]              ; screen column
    mov ah,[si+4]              ; color attribute

    mov di,si                  ; don't touch SI!
    add di,6                   ; point to buffer
    mov al,FILL_CHAR

    push dx                    ; save row,column

SIF1:                          ; fill with dots
    call Writechar_direct
    inc dl
    loop SIF1
    pop dx                      ; restore row,column

; Begin entering characters into buffer.

    call Gotoxy                ; position cursor
    mov cx,0                   ; character count

SIF2:

```

```

        mov  ah,8                ; get keystroke, no echo
        int  21h                ; AL = ASCII code
        cmp  al,0               ; extended key?
        jne  SIF2A              ; no
        int  21h                ; yes - read extra kbyd byte
        jmp  SIF2               ; look for another key

SIF2A:
        call CheckCharacter     ; echo normal characters
        cmp  al,ENTER_KEY      ; look for exit keys
        je   SIF8              ; end of Form
        cmp  al,TAB_KEY        ;
        je   SIF9              ; end of Field
        cmp  al,BKSP_KEY       ; look for backspace
        je   SIF3              ;
        mov  [di],al           ; save the key
        inc  di
        inc  cx                 ; add to character count
        cmp  cx,[si]           ; max characters reached?
        jb  SIF2               ;
        jmp  SIF9              ; end of Field

SIF3:
        cmp  cx,0               ; first keystroke?
        je   SIF2              ; yes: get another keystroke
        mov  dx,offset EraseString ; erase the previous char
        call Writestring
        dec  di                 ; back up in the buffer
        dec  cx                 ; decrement character count
        jmp  SIF2              ; get another keystroke

SIF8:
        stc                    ; signal end of form
        jmp  SIF10

SIF9:
        cld                    ; signal end of field

SIF10:
        mov  [si+6],cx         ; save buffer count
        popa
        ret

.data
EraseString db BKSP_KEY, FILL_CHAR, BKSP_KEY,0
.code
ShowInputField endp

; Check the character in AL, and be careful not to
; echo certain characters.

CheckCharacter proc
        push di
        push dx

; The following special keys are not to be echoed.

```

```

CF1:
    cmp  al,ENTER_KEY      ; don't echo the Enter key
    je   CF2
    cmp  al,TAB_KEY       ; don't echo the Tab key
    je   CF2
    cmp  al,BKSP_KEY      ; don't echo the Bksp key
    je   CF2
    call EchoKey

CF2:
    pop  dx
    pop  di
    ret
CheckCharacter endp

; Echo the character in AL, preserving all registers.

EchoKey proc
    push ax
    push dx
    mov  ah,2
    mov  dl,al
    int  21h
    pop  dx
    pop  ax
    ret
EchoKey endp

;-----

SetupScreen    proc
    push  dx
    call  Clrscr
    mov   dx, 050Ah          ; cursor will be at row 5, col 10
    call  PlaceCursor
    call  DrawHorizontal     ; draw the top line of the box
    call  DrawVertical       ; and the left side
    mov   dx, 0F0Ah         ; move cursor to row 15, col 10
    call  PlaceCursor
    call  DrawHorizontal     ; Draw the bottom line
    mov   dx, 053Ch         ; Place the cursor at row 5, col 60
    call  DrawVertical       ; and draw the right side
    call  DrawCorners
    pop   dx
    ret
SetupScreen    endp

PlaceCursor    proc
    mov   ah, 02h           ; BIOS function set cursor position
    mov   bh, 00h          ; on video page 0
    int  10h               ; coordinates passed in dx
    ret
PlaceCursor    endp

DrawHorizontal proc

```

```

        mov     ah, 09h           ; BIOS function write character
        mov     al, horizLine     ; the horizontal line
        mov     bh, 00h           ; on video page 0
        mov     bl, 07h           ; with a normal attribute
        mov     cx, LineLength    ; and size of the line
        int     10h
        ret
DrawHorizontal endp

DrawVertical    proc
        mov     cx, 0Ah           ; The box is 15 high
        mov     al, VertLine
        mov     bx, 0007         ; page 0 with a normal attribute

DrawIt:
        push    ax               ; this routine draws a vertical line
        push    bx               ; by incrementing the dh register,
        push    cx               ; which contains the row at which the
        mov     cx, 1            ; character is to be written
        int     10h             ; the column in dl remains constant
        inc     dh               ; The registers must be saved since
        call    PlaceCursor     ; the same ones are used for drawing
        pop     cx               ; and positioning the cursor
        pop     bx
        pop     ax
        loop   DrawIt
        ret
DrawVertical    endp

DrawOneChar    proc
        mov     ah, 09h           ; BIOS function write character
        mov     bx, 0007h        ; video page 0 with normal attribute
        mov     cx, 1            ; only one character to write
        int     10h
        ret
DrawOneChar    endp

DrawCorners    proc
        mov     al, BotRight     ; load the corner character
        call    DrawOneChar      ; and draw it
        mov     dx, 050Ah        ; move cursor to 5,10
        call    PlaceCursor
        mov     al, TopLeft      ; Load the top left char
        call    DrawOneChar      ; and draw it
        mov     dx, 053Ch        ; move to top right corner
        call    PlaceCursor
        mov     al, TopRight     ; and draw that character
        call    DrawOneChar
        mov     dx, 0F0Ah        ; and move to the bottom left
        call    PlaceCursor
        mov     al, BotLeft      ; and draw the corner
        call    DrawOneChar
        ret
DrawCorners    endp

WritePrompts   proc

```

```

        mov     cx, 6                ; number of prompts to write
        mov     bx, offset StartOfPrompts ; and the beginning of the data
WriteLoop:
        mov     dx, [bx]            ; loads the cursor position into dx
        call    PlaceCursor        ; and places it
        add     bx, 2                ; move bx to the next table entry
        mov     dx, [bx]            ; which is the address of the prompt
        call    Writestring        ; load it into dx, and write it out
        add     bx, 2                ; move bx to the next entry, which is
        loop    WriteLoop          ; a cursor position, and repeat.
        ret
WritePrompts endp
end main

```

Chapt 6 Ex 16: Customer Account Program, Version 2.

```

; Display an account screen with input fields. As the
; user presses the Tab key, move to each subsequent field.
; End the input screen when the Enter key is pressed.
; For the Previous Balance, Payments, and Credits fields,
; allow only signed numbers to be entered.

; Notes:
; This Exercise is easily the most difficult one to
; complete up to this point in the chapter. It requires
; careful debugging and should be a lot of fun for students
; who like challenges.

; This solution program builds on the solution
; program for Exercise 15, and adapts the the Console
; Input Field with keyboard filter program developed for
; Exercise 14. Due to the specialized nature of checking
; a signed number, I decided to include a pointer in each
; field's record structure that contains the address
; of an appropriate filter procedure. (In Exercise 14, we
; simply stored a pointer to a filter string.) Students have
; not yet tried using indirect procedure calls, so they will
; have to read page 503 to understand the following line
; of code:

; call near ptr [si+FILTER_OFFSET] ; call filter procedure

; If no filter procedure is specified, the special constant
; NO_FILTER is inserted in the procedure pointer.

; Although it was not required, this solution program
; allows the user to backspace over characters and
; fields are displayed in reverse video. These features
; were already implemented in Exercise 14. In other
; words, this solution program is identical to the
; one written for Exercise 17.

; -----
; Another way to implement this solution program is to adapt
; the Finite State Machine program from Exercise 5 in this

```

```

; chapter to the current program. The primary difference is
; that Exercise 5 does not require the programmer to store
; the input characters in a buffer.

.model small
.286
.stack 100h

.data
    ScreenTitle    db 'ACCOUNT INPUT SCREEN',0
    AcctTitle      db '  ACCT NUM:',0
    NameTitle      db '  LAST NAME:',0
    PrevBalTitle   db 'PREV BALANCE:',0
    PymtsTitle     db '  PAYMENTS:',0
    CrTitle        db '  CREDITS:',0

StartOfPrompts   \
    dw 0419h                ; this section contains both
    dw offset ScreenTitle  ; screen positions for prompts
    dw 060Fh                ; and the addresses of the prompts
    dw offset NameTitle    ; The first entry is the row and
    dw 080Fh                ; column position for the first
    dw offset AcctTitle    ; prompt. The second entry is the
    dw 0A0Fh                ; address of the first prompt.
    dw offset PrevBalTitle ; the display routine goes through
    dw 0C0Fh                ; this portion of the data, using
    dw offset PymtsTitle   ; these values to write to the
    dw 0E0Fh                ; screen
    dw offset CrTitle

LastNameLen = 30
AcctNumLen = 6
PrevBalLen = 11
PaymentsLen = 11
CreditsLen = 11

NO_FILTER = -1

LastName \
    dw LastNameLen
    db 6,29                ; row, column position
    db 70h                 ; attribute
    db ?                   ; count actual chars typed
FILTER_OFFSET = ($ - LastName)
    dw NO_FILTER
BUFFER_OFFSET = ($ - LastName) ; used by ShowInputField
    db LastNameLen dup(0),0

AcctNum \
    dw AcctNumLen
    db 8,29                ; row, column position
    db 70h                 ; attribute
    db ?                   ; count actual chars typed
    dw NO_FILTER
    db AcctNumLen dup(0),0

```

```

PrevBal \
  dw PrevBalLen
  db 10,29                ; row, column position
  db 70h                  ; attribute
  db ?                    ; count actual chars typed
  dw SignedNumberFilter
  db PrevBalLen dup(0),0

Payments \
  dw PaymentsLen
  db 12,29                ; row, column position
  db 70h                  ; attribute
  db ?                    ; count actual chars typed
  dw SignedNumberFilter
  db PaymentsLen dup(0),0

Credits \
  dw CreditsLen
  db 14,29                ; row, column position
  db 70h                  ; attribute
  db ?                    ; count actual chars typed
  dw SignedNumberFilter
  db CreditsLen dup(0),0

  LineLength      dw 50
  HorizLine       db 196
  VertLine        db 179
  TopLeft         db 218
  Botleft         db 192
  TopRight        db 191
  BotRight        db 217

.Code
include library.inc

main proc
  mov  ax, @data          ; init data segment
  mov  ds, ax
  call SetupScreen       ; Draw the input form
  call WritePrompts     ; write prompts to the screen
  call GetFormData       ; Get the user input

Exit:
  mov  ax, 4C00h         ; end program
  int  21h
main endp

; Get each of the input fields. If the Carry flag is set by
; ShowInputField, the user has pressed a key that ends
; all input for the form.

GetFormData proc
  mov  si,offset LastName
  call ShowInputField
  jc   GF9
  mov  si,offset AcctNum

```

```

    call ShowInputField
    jc  GF9
    mov  si,offset PrevBal
    call ShowInputField
    jc  GF9
    mov  si,offset Payments
    call ShowInputField
    jc  GF9
    mov  si,offset Credits
    call ShowInputField
    jc  GF9

GF9:
    ret
GetFormData endp

; SI points to a structure containing the following input
; field data: field length (word), screen row (byte), screen
; column (byte), color attribute (byte), input buffer (array
; of byte).

ENTER_KEY = 0Dh
TAB_KEY   = 9
BKSP_KEY  = 8
FILL_CHAR = '.'

ShowInputField proc
    pusha
    mov  decimalCount,0

    mov  cx,[si]           ; field length
    mov  dh,[si+2]        ; screen row
    mov  dl,[si+3]        ; screen column
    mov  ah,[si+4]        ; color attribute

    mov  di,si            ; don't touch SI!
    add  di,BUFFER_OFFSET ; point to buffer
    mov  al,FILL_CHAR

    push dx               ; save row,column

SIF1:
    call Writechar_direct ; fill with dots
    inc  dl
    loop SIF1
    pop  dx               ; restore row,column

; Begin entering characters into buffer.

    call Gotoxy           ; position cursor
    mov  cx,0            ; character count

SIF2:
    mov  ah,8             ; get keystroke, no echo
    int  21h             ; AL = ASCII code
    cmp  al,0            ; extended key?

```

```

    jne  SIF2A                ; no
    int  21h                 ; yes - read extra kbyd byte
    jmp  SIF2                ; look for another key

SIF2A:
    cmp  [si+FILTER_OFFSET],NO_FILTER ; filter procedure specified?
    jne  SIF2B                ; yes: skip next two lines
    call EchoKey              ; no: echo the character
    jmp  SIF3                ; and continue

SIF2B:
    call near ptr [si+FILTER_OFFSET] ; call filter procedure
    jnz  SIF2                ; invalid character if ZF = 0

SIF3:
    cmp  al,ENTER_KEY        ; look for exit keys
    je   SIF8                ; end of Form
    cmp  al,TAB_KEY          ; look for tab key
    je   SIF9                ; end of Field
    cmp  al,BKSP_KEY        ; look for backspace
    je   SIF5                ; backspace key
    mov  [di],al             ; save the key
    inc  di
    inc  cx                  ; add to character count
    cmp  cx,[si]             ; max characters reached?
    jb  SIF2                ; below max characters
    jmp  SIF9                ; end of Field

SIF5:
    cmp  cx,0                ; first keystroke?
    je   SIF2                ; yes: get another keystroke
    mov  dx,offset EraseString ; erase the previous char
    call Writestring
    dec  di                  ; back up in the buffer
    dec  cx                  ; decrement character count
    jmp  SIF2                ; get another keystroke

SIF8:
    stc                      ; signal end of form
    jmp  SIF10

SIF9:
    cld                      ; signal end of field

SIF10:
    mov  [si+6],cx           ; save buffer count
    popa
    ret

.data
EraseString db BKSP_KEY, FILL_CHAR, BKSP_KEY,0
.code
ShowInputField endp

; Custom procedure for processing a signed number. Input
; parameters: AL = character, CX = current character count.

```

```
; Output: ZF = 1 if the character is valid, or ZF = 0 if
; the character is invalid.
```

```
SignedNumberFilter proc
    pusha
    cmp  cx,0                ; first digit?
    jne  SNF8                ; no
    mov  decimalCount,0     ; yes: init decimal count

SNFA:
    mov  di,offset ValidChars ; get filter string
    call FindInString       ; is AL in the string?
    jnz  SNF8                ; no, exit with beep

    cmp  al,'+'              ; numeric sign?
    je   SNF1
    cmp  al,'-'
    je   SNF1
    cmp  al,'.'              ; decimal point?
    jne  SNF3                ; no: continue
    cmp  decimalCount,0     ; already used before?
    ja   SNF8                ; yes: exit with beep
    inc  decimalCount       ; no: increment count
    jmp  SNF3                ; continue

SNF1:    ; numeric sign found
    cmp  cx,0                ; is it the first character?
    jne  SNF8                ; no: exit with beep
    jmp  SNF3                ; yes: accept it

    ; The following special keys are not to be echoed.

SNF3:
    cmp  al,ENTER_KEY       ; don't echo the Enter key
    je   SNF9
    cmp  al,TAB_KEY         ; don't echo the Tab key
    je   SNF9
    cmp  al,BKSP_KEY        ; don't echo the Bksp key
    je   SNF9
    call EchoKey
    test al,0                ; set ZF = 1: Filter passed
    jmp  SNF9

SNF8:    ; invalid character: sound a beep
    mov  ah,2
    mov  dl,7
    int  21h

SNF9:
    popa
    ret

.data
    ValidChars db ".+-0123456789",TAB_KEY,BKSP_KEY,ENTER_KEY,0
    decimalCount db ?
.code
```

```
SignedNumberFilter endp
```

```
; Given a null-terminated string pointed to by DI and a
; character in AL, set ZF = 1 if the character exists in
; the string. Otherwise, clear ZF to 0.
```

```
FindInString proc
    pusha
```

```
FIS1:
    cmp [di],0      ; end of string?
    je  FIS2        ; if so, quit with ZF = 0
    cmp [di],al     ; check current character
    je  FIS3        ; if match found, exit with ZF = 1
    inc di          ; move to next character
    jmp FIS1
```

```
FIS2:
    or  al,1        ; clear ZF = 0
```

```
FIS3:
    popa
    ret
```

```
FindInString endp
```

```
; Echo the character in AL, preserving all registers.
```

```
EchoKey proc
    push ax
    push dx
    mov  ah,2
    mov  dl,al
    int  21h
    pop  dx
    pop  ax
    ret
```

```
EchoKey endp
```

```
;-----
```

```
SetupScreen    proc
    push  dx
    call  Clrscr
    mov   dx, 050Ah          ; cursor will be at row 5, col 10
    call  PlaceCursor
    call  DrawHorizontal     ; draw the top line of the box
    call  DrawVertical       ; and the left side
    mov   dx, 0F0Ah          ; move cursor to row 15, col 10
    call  PlaceCursor
    call  DrawHorizontal     ; Draw the bottom line
    mov   dx, 053Ch          ; Place the cursor at row 5, col 60
    call  DrawVertical       ; and draw the right side
    call  DrawCorners
    pop   dx
    ret
SetupScreen    endp
```

```

PlaceCursor    proc
    mov     ah, 02h           ; BIOS function set cursor position
    mov     bh, 00h         ; on video page 0
    int     10h             ; coordinates passed in dx
    ret
PlaceCursor    endp

DrawHorizontal proc
    mov     ah, 09h         ; BIOS function write character
    mov     al, horizLine   ; the horizontal line
    mov     bh, 00h         ; on video page 0
    mov     bl, 07h         ; with a normal attribute
    mov     cx, LineLength  ; and size of the line
    int     10h
    ret
DrawHorizontal endp

DrawVertical   proc
    mov     cx, 0Ah         ; The box is 15 high
    mov     al, VertLine    ;
    mov     bx, 0007        ; page 0 with a normal attribute

DrawIt:
    push    ax              ; this routine draws a vertical line
    push    bx              ; by incrementing the dh register,
    push    cx              ; which contains the row at which the
    mov     cx, 1           ; character is to be written
    int     10h            ; the column in dl remains constant
    inc     dh              ; The registers must be saved since
    call   PlaceCursor     ; the same ones are used for drawing
    pop     cx              ; and positioning the cursor
    pop     bx
    pop     ax
    loop   DrawIt
    ret
DrawVertical   endp

DrawOneChar    proc
    mov     ah, 09h         ; BIOS function write character
    mov     bx, 0007h       ; video page 0 with normal attribute
    mov     cx, 1           ; only one character to write
    int     10h
    ret
DrawOneChar    endp

DrawCorners    proc
    mov     al, BotRight    ; load the corner character
    call   DrawOneChar      ; and draw it
    mov     dx, 050Ah       ; move cursor to 5,10
    call   PlaceCursor
    mov     al, TopLeft     ; Load the top left char
    call   DrawOneChar      ; and draw it
    mov     dx, 053Ch       ; move to top right corner
    call   PlaceCursor
    mov     al, TopRight    ; and draw that character

```

```

        call DrawOneChar
        mov dx, 0F0Ah ; and move to the bottom left
        call PlaceCursor
        mov al, BotLeft ; and draw the corner
        call DrawOneChar
        ret
DrawCorners endp

WritePrompts proc
    mov cx, 6 ; number of prompts to write
    mov bx, offset StartOfPrompts ; and the beginning of the data
WriteLoop:
    mov dx, [bx] ; loads the cursor position into dx
    call PlaceCursor ; and places it
    add bx, 2 ; move bx to the next table entry
    mov dx, [bx] ; which is the address of the prompt
    call Writestring ; load it into dx, and write it out
    add bx, 2 ; move bx to the next entry, which is
    loop WriteLoop ; a cursor position, and repeat.
    ret
WritePrompts endp
end main

```

Chapt 6 Ex 17: Customer Account Program, Version 3.

```

; Display an account screen with input fields. As the
; user presses the Tab key, move to each subsequent field.
; End the input screen when the Enter key is pressed.
; For the Previous Balance, Payments, and Credits fields,
; allow only signed numbers to be entered. Allow the user to
; use the Backspace key to correct entries, and show each
; field in reverse video.

```

```

; Notes:
; This Exercise is easily the most difficult one to
; complete up to this point in the chapter. It requires
; careful debugging and should be a lot of fun for students
; who like challenges.

```

```

; This solution program builds on the solution
; program for Exercise 15, and adapts the the Console
; Input Field with keyboard filter program developed for
; Exercise 14. Due to the specialized nature of checking
; a signed number, I decided to include a pointer in each
; field's record structure that contains the address
; of an appropriate filter procedure. (In Exercise 14, we
; simply stored a pointer to a filter string.) Students have
; not yet tried using indirect procedure calls, so they will
; have to read page 503 to understand the following line
; of code:

```

```

; call near ptr [si+FILTER_OFFSET] ; call filter procedure

```

```

; If no filter procedure is specified, the special constant
; NO_FILTER is inserted in the procedure pointer.

```

```

; -----
; Another way to implement this solution program is to adapt
; the Finite State Machine program from Exercise 5 in this
; chapter to the current program. The primary difference is
; that Exercise 5 does not require the programmer to store
; the input characters in a buffer.

.model small
.286
.stack 100h

.data
    ScreenTitle    db 'ACCOUNT INPUT SCREEN',0
    AcctTitle      db '  ACCT NUM:',0
    NameTitle      db '  LAST NAME:',0
    PrevBalTitle   db 'PREV BALANCE:',0
    PymtsTitle     db '  PAYMENTS:',0
    CrTitle        db '  CREDITS:',0

StartOfPrompts    \
    dw 0419h          ; this section contains both
    dw offset ScreenTitle ; screen positions for prompts
    dw 060Fh          ; and the addresses of the prompts
    dw offset NameTitle  ; The first entry is the row and
    dw 080Fh          ; column position for the first
    dw offset AcctTitle  ; prompt. The second entry is the
    dw 0A0Fh          ; address of the first prompt.
    dw offset PrevBalTitle ; the display routine goes through
    dw 0C0Fh          ; this portion of the data, using
    dw offset PymtsTitle ; these values to write to the
    dw 0E0Fh          ; screen
    dw offset CrTitle

LastNameLen = 30
AcctNumLen = 6
PrevBalLen = 11
PaymentsLen = 11
CreditsLen = 11

NO_FILTER = -1

LastName \
    dw LastNameLen
    db 6,29          ; row, column position
    db 70h           ; attribute
    db ?            ; count actual chars typed
FILTER_OFFSET = ($ - LastName)
    dw NO_FILTER
BUFFER_OFFSET = ($ - LastName) ; used by ShowInputField
    db LastNameLen dup(0),0

AcctNum \
    dw AcctNumLen
    db 8,29          ; row, column position
    db 70h           ; attribute

```

```

    db ?                ; count actual chars typed
    dw NO_FILTER
    db AcctNumLen dup(0),0

PrevBal \
    dw PrevBalLen
    db 10,29            ; row, column position
    db 70h              ; attribute
    db ?                ; count actual chars typed
    dw SignedNumberFilter
    db PrevBalLen dup(0),0

Payments \
    dw PaymentsLen
    db 12,29            ; row, column position
    db 70h              ; attribute
    db ?                ; count actual chars typed
    dw SignedNumberFilter
    db PaymentsLen dup(0),0

Credits \
    dw CreditsLen
    db 14,29            ; row, column position
    db 70h              ; attribute
    db ?                ; count actual chars typed
    dw SignedNumberFilter
    db CreditsLen dup(0),0

    LineLength    dw 50
    HorizLine     db 196
    VertLine      db 179
    TopLeft       db 218
    Botleft       db 192
    TopRight      db 191
    BotRight      db 217

.Code
include library.inc

main proc
    mov     ax, @data        ; init data segment
    mov     ds, ax
    call    SetupScreen     ; Draw the input form
    call    WritePrompts    ; write prompts to the screen
    call    GetFormData     ; Get the user input

Exit:
    mov     ax, 4C00h       ; end program
    int     21h
main endp

; Get each of the input fields. If the Carry flag is set by
; ShowInputField, the user has pressed a key that ends
; all input for the form.

GetFormData proc

```

```

    mov si,offset LastName
    call ShowInputField
    jc GF9
    mov si,offset AcctNum
    call ShowInputField
    jc GF9
    mov si,offset PrevBal
    call ShowInputField
    jc GF9
    mov si,offset Payments
    call ShowInputField
    jc GF9
    mov si,offset Credits
    call ShowInputField
    jc GF9

GF9:
    ret
GetFormData endp

; SI points to a structure containing the following input
; field data: field length (word), screen row (byte), screen
; column (byte), color attribute (byte), input buffer (array
; of byte).

ENTER_KEY = 0Dh
TAB_KEY = 9
BKSP_KEY = 8
FILL_CHAR = '.'

ShowInputField proc
    pusha
    mov decimalCount,0

    mov cx,[si]                ; field length
    mov dh,[si+2]              ; screen row
    mov dl,[si+3]              ; screen column
    mov ah,[si+4]              ; color attribute

    mov di,si                  ; don't touch SI!
    add di,BUFFER_OFFSET      ; point to buffer
    mov al,FILL_CHAR

    push dx                    ; save row,column

SIF1:                          ; fill with dots
    call Writechar_direct
    inc dl
    loop SIF1
    pop dx                      ; restore row,column

; Begin entering characters into buffer.

    call Gotoxy                ; position cursor
    mov cx,0                   ; character count

```

```

SIF2:
    mov  ah,8                ; get keystroke, no echo
    int  21h                ; AL = ASCII code
    cmp  al,0               ; extended key?
    jne  SIF2A              ; no
    int  21h                ; yes - read extra kbyd byte
    jmp  SIF2               ; look for another key

SIF2A:
    cmp  [si+FILTER_OFFSET],NO_FILTER ; filter procedure specified?
    jne  SIF2B              ; yes: skip next two lines
    call EchoKey            ; no: echo the character
    jmp  SIF3               ; and continue

SIF2B:
    call near ptr [si+FILTER_OFFSET] ; call filter procedure
    jnz  SIF2               ; invalid character if ZF = 0

SIF3:
    cmp  al,ENTER_KEY       ; look for exit keys
    je   SIF8               ; end of Form
    cmp  al,TAB_KEY         ; look for tab
    je   SIF9               ; end of Field
    cmp  al,BKSP_KEY        ; look for backspace
    je   SIF5               ; backspace
    mov  [di],al            ; save the key
    inc  di
    inc  cx                 ; add to character count
    cmp  cx,[si]            ; max characters reached?
    jb  SIF2                ; below max
    jmp  SIF9               ; end of Field

SIF5:
    cmp  cx,0               ; first keystroke?
    je   SIF2               ; yes: get another keystroke
    mov  dx,offset EraseString ; erase the previous char
    call Writestring
    dec  di                 ; back up in the buffer
    dec  cx                 ; decrement character count
    jmp  SIF2               ; get another keystroke

SIF8:
    stc                     ; signal end of form
    jmp  SIF10

SIF9:
    clc                     ; signal end of field

SIF10:
    mov  [si+6],cx          ; save buffer count
    popa
    ret

.data
EraseString db BKSP_KEY, FILL_CHAR, BKSP_KEY,0
.code

```

```

ShowInputField endp

; Custom procedure for processing a signed number. Input
; parameters: AL = character, CX = current character count.
; Output: ZF = 1 if the character is valid, or ZF = 0 if
; the character is invalid.

SignedNumberFilter proc
    pusha
    cmp cx,0                ; first digit?
    jne SNF8                ; no
    mov decimalCount,0     ; yes: init decimal count

SNFA:
    mov di,offset ValidChars ; get filter string
    call FindInString       ; is AL in the string?
    jnz SNF8                ; no, exit with beep

    cmp al,'+'              ; numeric sign?
    je SNF1
    cmp al,'-'
    je SNF1
    cmp al,'.'              ; decimal point?
    jne SNF3                ; no: continue
    cmp decimalCount,0     ; already used before?
    ja SNF8                 ; yes: exit with beep
    inc decimalCount       ; no: increment count
    jmp SNF3                ; continue

SNF1:    ; numeric sign found
    cmp cx,0                ; is it the first character?
    jne SNF8                ; no: exit with beep
    jmp SNF3                ; yes: accept it

; The following special keys are not to be echoed.

SNF3:
    cmp al,ENTER_KEY       ; don't echo the Enter key
    je SNF9
    cmp al,TAB_KEY         ; don't echo the Tab key
    je SNF9
    cmp al,BKSP_KEY        ; don't echo the Bksp key
    je SNF9
    call EchoKey
    test al,0               ; set ZF = 1: Filter passed
    jmp SNF9

SNF8:    ; invalid character: sound a beep
    mov ah,2
    mov dl,7
    int 21h

SNF9:
    popa
    ret

```

```

.data
    ValidChars db ".+-0123456789",TAB_KEY,BKSP_KEY,ENTER_KEY,0
    decimalCount db ?
.code
SignedNumberFilter endp

; Given a null-terminated string pointed to by DI and a
; character in AL, set ZF = 1 if the character exists in
; the string. Otherwise, clear ZF to 0.

FindInString proc
    pusha

FIS1:
    cmp [di],0      ; end of string?
    je FIS2        ; if so, quit with ZF = 0
    cmp [di],al    ; check current character
    je FIS3        ; if match found, exit with ZF = 1
    inc di         ; move to next character
    jmp FIS1

FIS2:
    or al,1        ; clear ZF = 0

FIS3:
    popa
    ret
FindInString endp

; Echo the character in AL, preserving all registers.

EchoKey proc
    push ax
    push dx
    mov ah,2
    mov dl,al
    int 21h
    pop dx
    pop ax
    ret
EchoKey endp

;-----

SetupScreen proc
    push dx
    call Clrscr
    mov dx, 050Ah      ; cursor will be at row 5, col 10
    call PlaceCursor
    call DrawHorizontal ; draw the top line of the box
    call DrawVertical   ; and the left side
    mov dx, 0F0Ah      ; move cursor to row 15, col 10
    call PlaceCursor
    call DrawHorizontal ; Draw the bottom line
    mov dx, 053Ch      ; Place the cursor at row 5, col 60
    call DrawVertical   ; and draw the right side

```

```

        call    DrawCorners
        pop     dx
        ret
SetupScreen    endp

PlaceCursor    proc
    mov     ah, 02h           ; BIOS function set cursor position
    mov     bh, 00h         ; on video page 0
    int     10h             ; coordinates passed in dx
    ret
PlaceCursor    endp

DrawHorizontal proc
    mov     ah, 09h         ; BIOS function write character
    mov     al, horizLine   ; the horizontal line
    mov     bh, 00h         ; on video page 0
    mov     bl, 07h         ; with a normal attribute
    mov     cx, LineLength  ; and size of the line
    int     10h
    ret
DrawHorizontal endp

DrawVertical    proc
    mov     cx, 0Ah         ; The box is 15 high
    mov     al, VertLine    ;
    mov     bx, 0007        ; page 0 with a normal attribute

DrawIt:
    push    ax              ; this routine draws a vertical line
    push    bx              ; by incrementing the dh register,
    push    cx              ; which contains the row at which the
    mov     cx, 1           ; character is to be written
    int     10h            ; the column in dl remains constant
    inc     dh              ; The registers must be saved since
    call    PlaceCursor     ; the same ones are used for drawing
    pop     cx              ; and positioning the cursor
    pop     bx
    pop     ax
    loop   DrawIt
    ret
DrawVertical    endp

DrawOneChar    proc
    mov     ah, 09h         ; BIOS function write character
    mov     bx, 0007h       ; video page 0 with normal attribute
    mov     cx, 1           ; only one character to write
    int     10h
    ret
DrawOneChar    endp

DrawCorners    proc
    mov     al, BotRight    ; load the corner character
    call    DrawOneChar     ; and draw it
    mov     dx, 050Ah       ; move cursor to 5,10
    call    PlaceCursor     ;
    mov     al, TopLeft     ; Load the top left char

```

```

        call    DrawOneChar          ; and draw it
        mov     dx, 053Ch           ; move to top right corner
        call    PlaceCursor
        mov     al, TopRight        ; and draw that character
        call    DrawOneChar
        mov     dx, 0F0Ah           ; and move to the bottom left
        call    PlaceCursor
        mov     al, BotLeft         ; and draw the corner
        call    DrawOneChar
        ret
DrawCorners      endp

WritePrompts    proc
        mov     cx, 6               ; number of prompts to write
        mov     bx, offset StartOfPrompts ; and the beginning of the data
WriteLoop:
        mov     dx, [bx]           ; loads the cursor position into dx
        call    PlaceCursor        ; and places it
        add     bx, 2               ; move bx to the next table entry
        mov     dx, [bx]           ; which is the address of the prompt
        call    Writestring         ; load it into dx, and write it out
        add     bx, 2               ; move bx to the next entry, which is
        loop   WriteLoop           ; a cursor position, and repeat.
        ret
WritePrompts    endp
end main

```

Chapt 6 Ex 18: Sorting an Array

```

; Display, sort, and redisplay an array of signed integers.
; The original solution program was contributed by Kenneth C.
; Stahl (1990). The current version is by Kip Irvine (1998).

```

```

.model small
.286
.stack 100
cr = 0dh ;Carriage return
lf = 0ah ;Line feed

.data
beforeMsg db "Array before sorting",cr,lf,0
afterMsg  db cr,lf,"Array after Sorting",cr,lf,0

integer_array \
    dw 16,178,41,-36,982,154,-3,76,453,-21,18,90,413,-233,142
    dw 41,564,4,84,68,15,964,-38,815,150,901,713,852,384,520
    dw 213,912,145,-630,412,241,970,125,302,745,145,-135,102,278
    dw 145,640,-161,152,127,890
ARRAY_COUNT = ($ - integer_array) / (type integer_array)

.code
include library.inc

main    proc
        mov     ax,@data          ; Set up

```

```

        mov     ds,ax                ; data segment

        mov     dx,offset beforeMsg
        call    Writestring
        call    print_array          ; Print the array

        mov     dx,offset integer_array
        mov     cx,ARRAY_COUNT
        call    bubble_sort          ; Sort the array

        mov     dx,offset afterMsg
        call    Writestring
        call    print_array          ; Print the array again

        mov     ax,4C00h             ; Function:  exit w/o error
        int     21h                 ; Dos services
main     endp

; Sort an array of integers. Input parameters: DX points to the
; beginning of the array and CX contains the array size.

bubble_sort proc
        pusha
        dec cx                      ; decrement count by 1

Sort1:
        push cx                    ; Save the outer loop count on the stack.
        mov  si,dx                 ; get pointer to first element
        mov  di,dx
        add  di,type integer_array ; point to second element

Sort2:
        mov  ax,[si]              ; Get an array item
        cmp  [di],ax              ; Compare the items
        jge Sort3                 ; if [si] <= [di] then no swap needed
        xchg ax,[di]              ; Swap the two values
        mov  [si],ax

Sort3:
        add  si,type integer_array
        add  di,type integer_array
        loop Sort2                ; inner loop

        pop  cx                    ; Retrieve the outer count
        loop Sort1                ; outer loop

        popa
        ret
bubble_sort endp

; Display the array with four numbers per line.

print_array proc
        pusha
        mov  si,offset integer_array
        mov  cx,ARRAY_COUNT

```

```

        mov  dx,0                ; column counter

PA1:
        mov  ax,[si]             ; get a value
        call Writeint_signed     ; display it
        push dx
        mov  dx,offset tabStr
        call Writestring
        pop  dx
        inc  dx                  ; column counter
        cmp  dx,4                ; last column reached?
        jb   PA2
        call Crlf                ; yes: write EOLN
        mov  dx,0

PA2:
        add  si,2
        loop PA1

        call Crlf
        popa
        ret

.data
tabStr db 9,0
.code
print_array endp
end main

```

Chapt 6 Ex 19: Binary Search

```

; Sort an array of signed integers. Search the array for
; a single integer, using the Binary Search method.

.model small
.286
.stack 100h

.data
array \
dw 16,178,41,-36,982,154,-3,76,453,-21,18,90,413,-233,142
dw 41,564,4,84,68,15,964,-38,815,150,901,713,852,384,520
dw 213,912,145,-630,412,241,970,125,302,745,145,-135,102,278
dw 145,640,-161,152,127,890
ARRAY_COUNT = ($ - array) / (type array)

askForValue      db "Enter value to find: ",0
foundPositionMsg db "The value was found in position ",0
notFoundMsg      db "The value was not found.",0

.code
include library.inc

main proc
    mov  ax,@data                ; init data segment
    mov  ds,ax
    call ClrScr

```

```

    mov     dx,offset array
    mov     cx,ARRAY_COUNT

; Sort and display the array.

    call    BubbleSort           ; sort the array
    call    PrintArray          ; display the array

; Ask the user for a value to find in the array. Search
; for the value, and display its position if found.

    mov     dx,offset askForValue
    call    Writestring
    call    Readint             ; AX = value to find
    call    Crlf
    call    BinarySearch
    cmp     ax,-1               ; not found? (-1)
    je     L2                   ; yes: skip next lines
    mov     dx,offset foundPositionMsg
    call    Writestring
    mov     bx,10               ; decimal radix
    call    Writeint            ; show position where found
    jmp     L3

L2:
    mov     dx,offset notFoundMsg ; "Not found"
    call    Writestring

L3:
    call    Crlf
    mov     ax,4c00h           ; end program
    int     21h
main endp

; Sort an array of 16-bit signed integers. Input parameters:
; DX points to the beginning of the array and CX contains
; the array size.

BubbleSort proc
    pusha
    dec cx                    ; decrement count by 1

Sort1:
    push  cx                  ; Save the outer loop count on the stack:
    mov   si,dx               ; get pointer to first element
    mov   di,dx
    add   di,2                ; point to second element

Sort2:
    mov   ax,[si]             ; Get an array item
    cmp   [di],ax             ; Compare the items
    jge   Sort3               ; if [si] <= [di] then no swap needed
    xchg  ax,[di]             ; Swap the two values
    mov   [si],ax

Sort3:

```

```
    add    si,2                ; point to next element
    add    di,2
    loop  Sort2                ; inner loop

    pop    cx                  ; Retrieve the outer count
    loop  Sort1                ; outer loop

    popa
    ret
BubbleSort endp
```

```
; Search the array for the value in AX. The array length
; is in CX. If a match is found, return
; its array position in AX. If no match is found, set AX to -1.
```

```
BinarySearch proc
    push bx
    push si

    mov    bx,ax              ; BX = search value
    mov    first,0
    mov    last,cx
    dec    last
```

```
BS1: ; (top of loop)
```

```
; while first <= last:
    mov    ax,first
    cmp    ax,last
    ja     BS5                ; exit loop
```

```
; mid = (last + first) / 2;
```

```
    mov    ax,last
    add    ax,first
    shr    ax,1
    mov    mid,ax
```

```
; if (values[mid] < searchval )
;   first = mid + 1;
```

```
    mov    si,mid
    shl    si,1
    cmp    array[si],bx
    jge    BS2
```

```
    mov    ax,mid
    inc    ax
    mov    first,ax
    jmp    BS4
```

```
; else if( values[mid] > searchVal )
;   last = mid - 1;
```

```
BS2:
    mov    si,mid
```

```

        shl  si,1
        cmp  array[si],bx
        jle  BS3
        mov  ax,mid
        dec  ax
        mov  last,ax
        jmp  BS4

; else return mid;

BS3:
        mov  ax,mid ; value found: return its position
        jmp  BS9

; endif

BS4:
        jmp  BS1 ; continue the loop

BS5: ; (search failed)

        mov  ax,-1
        jmp  BS9

BS9:
        pop  si
        pop  bx
        ret

.data
first dw ?
last  dw ?
mid   dw ?
.code
BinarySearch endp

; Display the array with four numbers per line.

PrintArray proc
        pusha
        mov  si,offset array
        mov  cx,ARRAY_COUNT
        mov  dx,0 ; column counter

PA1:
        mov  ax,[si] ; get a value
        call Writeint_signed ; display it
        push dx
        mov  dx,offset tabStr
        call Writestring
        pop  dx
        inc  dx ; column counter
        cmp  dx,4 ; last column reached?
        jb  PA2
        call Crlf ; yes: write EOLN
        mov  dx,0

```

```

PA2:
    add    si,2
    loop  PA1

    call  Crlf
    popa
    ret

.data
tabStr db 9,0
.code
PrintArray endp
end main

```

Chapt 6 Ex 20: File Encryption Program

```

; Encrypt a file using a 128-byte encryption key. XOR each
; plaintext character with its matching byte in the key.

; Notes:
; The primary difficulty in getting this program to work lies
; in the filtering of control characters by DOS. The encryption
; operation frequently creates output files containing 1Ah (EOF)
; characters, for example. If you try to read such a file with
; INT 21h functions 6,7,or 8, they stop at the 1Ah character.

; The solution used here is to read the file with INT 21h
; function 3Fh (Read from File or Device), which is explained
; on page 153 in Chapter 5. Writing to the file is easiest
; with INT 21h function 40h (Write to File or Device), which is
; explained on page 452 (Chapter 12). Therefore, you can ask
; students to read just that one page.

.model small
.286
.stack 100h
.data
key db "alkwiourwsja.zx,vmc098230.jfsd0s8sfnm,nn,n34,5n3,$"
    db "8 83,fpd;n8mnbv'a90xbib;gmgfi lf,duutb8f9fkemsmdm#"
    db "8sm,dsmuwmnwpocnwy^%8@#$$%^&x"

KEY_SIZE = ($ - key)
BUF_SIZE = KEY_SIZE

bytesRead dw 0
buffer db BUF_SIZE dup(0)
handle dw 0

.code
include library.inc

main proc
    mov  ax,@data           ; init data segment
    mov  ds,ax
    call ClrScr

```

```
        call Encrypt

        mov ax,4c00h          ; end program
        int 21h
main endp

Encrypt proc
; Read from the file into the input buffer and
; encrypt all characters in the buffer with their
; matching values in the encryption key.

E1:
    call ReadBuf
    mov si,0
    mov cx,bytesRead
    cmp cx,0
    je E9

E2:
        ; Do the encryption
    mov al,key[si]
    xor buffer[si],al
    inc si
    loop E2

; Write encrypted buffer to standard output.

    mov ah,40h
    mov bx,1
    mov cx,bytesRead
    mov dx,offset buffer
    int 21h

    cmp bytesRead,BUF_SIZE
    je E1

E9:
    ret
Encrypt endp

; Read from standard input into the buffer.

ReadBuf proc
    pusha

    mov ah,3Fh          ; read a block from input
    mov bx,handle      ; file/device handle
    mov cx,BUF_SIZE
    mov dx,offset buffer
    int 21h
    mov bytesRead,ax

    popa
    ret
ReadBuf endp
end main
```

CHAPTER 7

7.8.1 Bit Manipulation

Ex 1: Backwards Binary Display

```

Title  Chapt 7 (7.8.1) Ex 1: Backwards Binary Display
; Display the binary bits in a 8-bit variable backwards.

7Title  Chapt 7 (7.8.1) Ex 1: Backwards Binary Display
; Display the binary bits in a 8-bit variable backwards.

.model small
.stack 100h
.data
sample db 84h

.code
include library.inc

main proc
    mov  ax,@data           ; init data segment
    mov  ds,ax
    call ClrScr

    mov  al,sample
    mov  cx,8               ; loop counter

L1:  mov  dl,'0'
     shr  al,1              ; bit shifted into Carry flag?
     jnc  L2                ; no: continue
     mov  dl,'1'            ; yes: display a '1'

L2:  push ax
     mov  ah,2              ; display 1 or 0
     int  21h
     pop  ax
     loop L1

     call Crlf
     mov  ax,4c00h          ; end program
     int  21h
main endp

end main
.model small
.stack 100h
.data
sample db 84h

.code

```

```

include library.inc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call ClrScr

    mov al,sample
    mov cx,8             ; loop counter

L1: mov dl,'0'
    shr al,1             ; bit shifted into Carry flag?
    jnc L2               ; no: continue
    mov dl,'1'           ; yes: display a '1'

L2: push ax
    mov ah,2             ; display 1 or 0
    int 21h
    pop ax
    loop L1

    call Crlf
    mov ax,4c00h         ; end program
    int 21h
main endp
end main

```

Ex 2: Packed Decimal Conversion

Title Chapt 7 (7.8.1), Ex 2: Packed Decimal Conversion

; Convert a 10-byte packed decimal number to individual
; ASCII digits and display the number.

```

.model small
.286
.stack 100h
.data
packedval dt 00273645193846571425

.code
include library.inc
main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call ClrScr

    mov si,offset packedval
    add si,9
    mov cx,10

L1: mov dl,byte ptr [si]
    shr dl,4
    or dl,30h
    mov ah,2              ; display high 4 bits

```

```

    int 21h
    mov dl,byte ptr [si]
    and dl,0Fh           ; display low 4 bits
    or  dl,30h
    int 21h
    dec si
    loop L1

    mov ax,4c00h         ; end program
    int 21h
main endp
end main

```

Ex 3: Shifting Multiple Bytes

Title Chapt 7 (7.8.1) Ex 3: Shifting Multiple Bytes

```

; Use SHRD to shift three consecutive operands to the right.
; Word operands are used because SHRD only operates on words
; or doublewords.

```

```

.model small
.stack 100h
.386

.data
word1 dw 0123h           ; after: 03h
word2 dw 4567h           ; after: B4h
word3 dw 89ABh           ; after: 6Fh

.code

main proc
    mov ax,@data         ; init data segment
    mov ds,ax

    mov ax,word1
    mov bx,word2
    mov cx,word3

    shrd cx,bx,4
    shrd bx,ax,4
    shr  ax,4

    mov word1,ax
    mov word2,bx
    mov word3,cx

    mov ax,4c00h         ; end program
    int 21h
main endp
end main

```

Ex 4: Room Schedule

```

Title  Chapt 7 (7.8.1) Ex 4: Room Schedule

; Using a room schedule record, query a bit-mapped field
; called roomstatus. Extract the individual subfields
; from roomstatus.

; ----- Field Format-----
;   Bit
; Position          Usage
; 0-1              Type of room (0,1,2,3)
; 2-7              Number of seats (0-63)
; 8-12             Department ID (0-31)
; 13               Overhead Projector (0,1)
; 14               Blackboard (0,1)
; 15               P.A. System (0,1)

.Model small
.Stack 100h
.data
    RoomStatus      dw    0110011101011101b
    RoomType        db    ?
    NumSeats        db    ?
    DeptID          db    ?
    Projector       db    ?
    BlackBoard     db    ?
    PASystem       db    ?

    typeCheck      db    03h          ; to AND the room status
    SeatsOffset    db    02h          ; Seats begins 2 bits from beginning
    SeatCheck      dw    3Fh          ; To AND the number of seats
    DeptOffset     db    06h          ; Department ID begins 8 bits from
beginning
    DeptCheck      dw    07h          ; to AND the Department ID
    ProjectorOffs  db    05h          ; 13 bits to the projector position
    ProjectorCheck dw    01h          ; Value to test projector
    BlackBOffset  db    01h
    BlackboardCk   dw    01h

.Code
Main      Proc
    mov     ax, @data          ; initialize the data segment so the
program
    mov     ds, ax            ; can find the variables
    mov     ax, RoomStatus    ; Load the Room status info into ax

ExtractType:
    push   ax                 ; Save the ax register for next step
    and    al, TypeCheck      ; Extract the room type
    mov    Roomtype, al       ; and save the result in the vari-
able
    pop    ax                 ; restore ax to original value

ExtractSeats:

```

```

    mov     cl, SeatsOffset           ; cx contains number of bits from
bit 0
    shr     ax, cl                   ; move the seats information into al
    push    ax                       ; save ax for the next step
    and     ax, SeatCheck            ; AND al with 00111111b to clear
high bits
    mov     NumSeats, al             ; place the value into NumSeats
    pop     ax                       ; restore ax to it's original value

ExtractDept:
    mov     cl, DeptOffset           ; cx contains the number of bits
from 0 the department begins
    shr     ax, cl                   ; move the department ID into al
    push    ax                       ; save ax for the next step
    and     ax, DeptCheck            ; AND al with 00000111b to extract
Dept ID
    mov     DeptID, al              ; store the value of Dept ID in the
variable
    pop     ax

ExtractProjector:
    mov     cl, ProjectorOffs        ; bits from start of string to
Projector position
    shr     ax, cl                   ; move projector into position
    push    ax                       ; save ax for the next step
    and     ax, ProjectorCheck       ; extract the projector bit
    mov     Projector, al            ; and save the value
    pop     ax                       ; restore ax

ExtractBlackBoard:
    shr     ax, 1                    ; move the blackboard value into
position
    push    ax                       ; save ax for the next step
    and     ax, Blackboardck         ; Extract the blackboard value
    mov     Blackboard, al           ; and save in the variable
    pop     ax

ExtractPA:
    shr     ax, 1                    ; move the PA bit into position
    mov     PASystem, al             ; and store it in the variable

Exit:
    mov     ax, 4C00h                ;load the terminate function
    Int     21h                      ;and end the program
Main     Endp
End Main

```

7.8.2 Bit-Mapped Sets

Ex 1: Display the Set Members

```

Title  Chapt 7 (7.8.2), Ex 1: Display the Set members

; Write a procedure called Display that displays a list

```

```
; of members of a bit-mapped set.

; This exercise is the second in a group of seven in Section
; 7.8.2 that deal with bit-mapped sets. In the solutions
; given here, the set is limited to 32 members, allowing
; the set to be contained in a single 32-bit integer. Each
; procedure name in this series is prefixed with "Set_".

.model small
.386
.stack 100h

.data

theSet dd 8000101Bh    ; set data, for testing

SET_SIZE = (type theSet) * 8    ; 32 bits

.code
include library.inc

main proc
    mov ax,@data        ; init data segment
    mov ds,ax
    call ClrScr

    mov eax,theSet
    call Set_Display

    mov ax,4c00h        ; end program
    int 21h
main endp

; Display the Set members, located in EAX. Start with
; bit 0, which is member 0.

Set_Display proc
    pusha
    mov cx,0            ; member counter

SD1:
    shr eax,1
    jnc SD2

    push ax
    mov ax,cx
    mov bx,10
    call WriteInt
    mov dx,offset isaMember
    call Writestring
    pop ax

SD2:
    inc cx
    cmp cx,SET_SIZE
    jb SD1
```

```

        popa
        ret

.data
isaMember db " is a member",0dh,0ah,0
.code
Set_Display endp
end main

```

Ex 2: Evaluate a Subset

Title Chapt 7 (7.8.2), Ex 2: Evaluate a Subset

```

; Write a procedure called SubSet that returns a value
; of 1 if the set in EBX is a subset of the set in EAX.
; Otherwise, return 0.

; This exercise is the second in a group of seven in Section
; 7.8.2 that deal with bit-mapped sets. In the solutions
; given here, the set is limited to 32 members, allowing
; the set to be contained in a single 32-bit integer. Each
; procedure name in this series is prefixed with "Set_".

.model small
.386
.stack 100h

.data
set_A dd 8000101Bh      ; set data, for testing
set_B dd 80101018h      ; sample subset (try different values)

subsetFound db "EBX is a subset of EAX.",0dh,0ah,0
noSubset     db "EBX is not a subset of EAX.",0dh,0ah,0

.code
include library.inc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call ClrScr

    mov eax,set_A         ; check for subset
    mov ebx,set_B
    call Set_SubSet

    cmp ax,1              ; display result if found
    jne M2
    mov dx,offset subsetFound
    call Writestring
    jmp M3

M2:
    mov dx,offset noSubset

```

```

        call Writestring

M3:
    mov ax,4c00h        ; end program
    int 21h
main endp

; Return 1 in AX if the set in EBX is a subset of the
; set in EAX. Otherwise, return 0.

Set_SubSet proc
    and  eax,ebx        ; clear all nonessential bits
    cmp  eax,ebx        ; compare the remaining ones
    je   IM1           ; same? then we have a subset
                        ; otherwise,
    mov  ax,0          ; indicate no subset found
    jmp  IM2           ; and exit

IM1: mov  ax,1        ; indicate subset found

IM2: ret
Set_SubSet endp
end main

```

Ex 3: Check for Set Membership

Title Chapt 7 (7.8.2), Ex 3: Check for Set membership

```

; Write a procedure called IsMember that returns a value
; of 1 if the member passed in BL is a member of the set
; identified by EAX. Otherwise, return 0.

; This exercise is the third in a group of seven in Section
; 7.8.2 that deal with bit-mapped sets. In the solutions
; given here, the set is limited to 32 members, allowing
; the set to be contained in a single 32-bit integer. Each
; procedure name in this series is prefixed with "Set_".

.model small
.386
.stack 100h

.data
theSet dd 8000101Bh    ; set data, for testing
membertoFind dw ?

prompt      db "Enter member number to find (0-31): ",0
memberIsThere db "This IS a member of the set.",0dh,0ah,0

SET_SIZE = (type theSet) * 8    ; 32 bits

.code
include library.inc

```

```

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call ClrScr

    mov dx,offset prompt  ; ask user for member to find
    call Writestring
    call Readint
    mov memberToFind,ax
    call Crlf

    mov eax,theSet       ; search for the member
    mov bx,memberToFind
    call Set_IsMember

    cmp ax,1             ; display result if found
    jne M2
    mov dx,offset memberIsThere
    call Writestring

M2:
    mov ax,4c00h         ; end program
    int 21h
main endp

; Return 1 in AX if the member passed in BL is a member
; of the set in EAX. Otherwise, return 0.

Set_IsMember proc
    push ecx
    push edx

    mov edx,1           ; bit mask for searching
    mov cl,bl           ; get member number
    shl edx,cl          ; shift mask into position
    test eax,edx        ; member found?
    jnz IM1             ; yes
    mov ax,0            ; no: set return value
    jmp IM2             ; and exit

IM1: mov ax,1          ; indicate member found

IM2: pop edx
     pop ecx
     ret
Set_IsMember endp
end main

```

Ex 4: Count the Set Members

Title Chapt 7 (7.8.2), Ex 4: Count the Set Members

```

; Write a procedure called Count that returns the
; number of members in the set identified by EAX.
; This exercise is the fourth in a group of seven in Section

```

; 7.8.2 that deal with bit-mapped sets. In the solutions
 ; given here, the set is limited to 32 members, allowing
 ; the set to be contained in a single 32-bit integer. Each
 ; procedure name in this series is prefixed with "Set_".

```
.model small
.386
.stack 100h
.data
theSet dd 8000101Bh    ; set data, for testing
SET_SIZE = (type theSet) * 8    ; 32 bits

.code
include library.inc
main proc
    mov ax,@data        ; init data segment
    mov ds,ax
    call ClrScr

    mov eax,theSet      ; count members in set
    call Set_Count
    mov ax,cx           ; CX = return value

    mov bx,10          ; display in decimal
    call Writeint
    call Crlf

    mov ax,4c00h       ; end program
    int 21h
main endp

; Count the number of members in the set contained in
; EAX.The count is returned in CX.

Set_Count proc
    push eax
    push dx

    mov cx,SET_SIZE
    mov dx,0           ; member counter

SC1: shr  eax,1        ; member in lowest bit?
    jnc  SC2          ; if not, skip
    inc  dx           ; yes: increment count
SC2: loop SC1

    mov  cx,dx        ; return count in CX
    pop  dx
    pop  eax
    ret
Set_Count endp

end main
```

Ex 5: Set Union

Title Chapt 7 (7.8.2), Ex 5: Set Union

```
; Write a procedure called Union that creates a new set
; from the combined members of two existing sets.
; Very easy exercise! Only three lines of code in
; the Set_Union procedure.

; This exercise is the fifth in a group of seven in Section
; 7.8.2 that deal with bit-mapped sets. In the solutions
; given here, the set is limited to 32 members, allowing
; the set to be contained in a single 32-bit integer. Each
; procedure name in this series is prefixed with "Set_".
```

```
.model small
.386
.stack 100h

.data
set_A dd ?
SET_SIZE = (type set_A) * 8      ; 32 bits

set_B dd 80101010h
set_C dd 20004013h

.code
include library.inc

main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call ClrScr

    ; Display the two input sets.

    mov eax,set_B
    call Set_Display
    call Crlf

    mov eax,set_C
    call Set_Display
    call Crlf

    ; Create the union of sets B and C.

    mov ebx,set_B
    mov ecx,set_C
    call Set_Union        ; union is in EAX
    mov set_A,eax        ; save it

    ; Display the resulting set.

    mov eax,set_A
    call Set_Display

    mov ax,4c00h          ; end program
    int 21h
```

```

main endp

; Form the union of the sets passed in EBX and ECX.
; Return the union set in EAX.

Set_Union proc
    mov  eax,ebx
    or   eax,ecx
    ret
Set_Union endp

; Display the Set members, located in EAX. Start with
; bit 0, which is member 0. (This procedure was borrowed from
; the solution program for Exercise 1 in Section 7.8.2.)

Set_Display proc
    pusha
    mov  cx,0           ; member counter

SD1:
    shr  eax,1
    jnc  SD2

    push ax
    mov  ax,cx
    mov  bx,10
    call WriteInt
    mov  dx,offset isaMember
    call Writestring
    pop  ax

SD2:
    inc  cx
    cmp  cx,SET_SIZE
    jb   SD1

    popa
    ret

.data
isaMember db " is a member",0dh,0ah,0
.code
Set_Display endp

end main

```

Ex 6: Set Difference

Title Chapt 7 (7.8.2), Ex 6: Set Difference

```

; Write a procedure called Difference that returns
; a set that represents the difference between two
; sets.

```

```
; This exercise is the sixth in a group of seven in Section
; 7.8.2 that deal with bit-mapped sets. In the solutions
; given here, the set is limited to 32 members, allowing
; the set to be contained in a single 32-bit integer. Each
; procedure name in this series is prefixed with "Set_".
```

```
.model small
.386
.stack 100h
```

```
.data
set_A dd ?
SET_SIZE = (type set_A) * 8      ; 32 bits
```

```
set_B dd 80101010h
set_C dd 80001010h
```

```
label1 db "Set A contains:",0dh,0ah,0
label2 db "Minus the members of set B:",0dh,0ah,0
label3 db "Results in the following set:",0dh,0ah,0
```

```
.code
include library.inc
```

```
main proc
    mov ax,@data          ; init data segment
    mov ds,ax
    call ClrScr
```

```
; Display the two input sets.
```

```
    mov dx,offset label1
    call Writestring
    mov eax,set_B
    call Set_Display
    call Crlf
```

```
    mov dx,offset label2
    call Writestring
    mov eax,set_C
    call Set_Display
    call Crlf
```

```
; Find the difference between sets B and C.
```

```
    mov ebx,set_B
    mov ecx,set_C
    call Set_Difference    ; result is in EAX
    mov set_A,eax        ; save it
```

```
; Display the resulting set.
```

```
    mov dx,offset label3
    call Writestring
    mov eax,set_A
    call Set_Display
```

```

        mov ax,4c00h          ; end program
        int 21h
main endp

; Find the difference between the sets in EBX and ECX.
; (Members that are in EBX but not in ECX.)
; Return the difference set in EAX.

Set_Difference proc
    push ecx
    mov  eax,ebx             ; get the first set
    not  ecx                 ; reverse bits in second set
    and  eax,ecx             ; remove common bits
    pop  ecx                 ; and you have the difference
    ret
Set_Difference endp

; Display the Set members, located in EAX. Start with
; bit 0, which is member 0. (This procedure was adapted from
; the solution program for Exercise 1 in Section 7.8.2.)

Set_Display proc
    pusha
    mov  cx,0                ; member counter

SD1:
    shr  eax,1
    jnc  SD2

    push ax
    mov  ax,cx
    mov  bx,10
    call WriteInt
    call Crlf
    pop  ax

SD2:
    inc  cx
    cmp  cx,SET_SIZE
    jb  SD1

    popa
    ret
Set_Display endp
end main

```

Ex 7: Fill Randomly

Title Chapt 7 (7.8.2), Ex 7: Fill Randomly

; Create a procedure called FillRandom that fills a
; set with a pseudorandom sequence of bits.

; This exercise is the seventh in a group of seven in Section

```

; 7.8.2 that deal with bit-mapped sets. In the solutions
; given here, the set is limited to 32 members, allowing
; the set to be contained in a single 32-bit integer. Each
; procedure name in this series is prefixed with "Set_".

.model small
.386
.stack 100h
SET_SIZE = 32
.code
include library.inc

main proc
    mov ax,@data           ; init data segment
    mov ds,ax
    call ClrScr

    call Set_FillRandom    ; EAX contains the set
    call Set_Display

    mov ax,4c00h          ; end program
    int 21h
main endp

; Fill a set (in EAX) with a pseudorandom sequence of bits.

Set_FillRandom proc
    push ecx
    push edx
    mov cx,SET_SIZE

FR1:
    mov eax,2             ; generate pseudorandom integer
    call Random_range     ; between 0 and 1
    or  edx,eax           ; insert the random bit in EDX
    shl  edx,1            ; shift left, prepare for next bit
    loop FR1              ; repeat for all bits

    mov  eax,edx          ; return the set in EAX
    pop  edx
    pop  ecx
    ret
Set_FillRandom endp

; Display the Set members, located in EAX. Start with
; bit 0, which is member 0. (This procedure was adapted from
; the solution program for Exercise 1 in Section 7.8.2.)

Set_Display proc
    pusha
    mov cx,0              ; member counter

SD1:
    shr  eax,1
    jnc  SD2

```

```

        push ax
        mov  ax,cx
        mov  bx,10
        call WriteInt
        call Crlf
        pop  ax

SD2:
        inc  cx
        cmp  cx,SET_SIZE
        jb  SD1

        popa
        ret
Set_Display endp

end main

```

7.8.3 Prime Numbers

Ex 1: Prime Number Program - 1

```

Title  Chapt 7 (7.8.3) Ex 1: Prime Number Program - I

; This program prompts the user for an integer, then asserts
; whether or not the number is prime.

.model small
.386
.stack 100h
.data
prompt      db "Enter an integer between 2 and 4294967295, ",0dh,0ah
            db "          or enter 0 to quit: ",0
msgPrime    db "This number is prime.",0dh,0ah,0
msgNotPrime db "This number is not prime.",0dh,0ah,0

.code
include library.inc

main proc
    mov  ax,@data          ; init data segment
    mov  ds,ax
    call ClrScr

M1:
    call Crlf
    mov  dx,offset prompt
    call Writestring
    call Readlong         ; read value into EAX
    call Crlf
    cmp  eax,0            ; does user want to quit?
    je   M5               ; yes

    call IsPrime          ; check value in EAX

```

```

    jnz M2
    mov dx,offset msgPrime
    call Writestring
    jmp M1

M2:
    mov dx,offset msgNotPrime
    call Writestring
    jmp M1

M5:
    mov ax,4c00h          ; end program
    int 21h
main endp

; Check the integer in EAX to see if it is prime. If so,
; return with ZF = 1; otherwise, clear the Zero flag.

IsPrime proc
    pusha
    mov ecx,eax          ; get number for loop count
    shr ecx,1           ; divide by 2
    mov ebx,2           ; first divisor

IP1:
    call IsDivisible    ; EAX divisble by EBX?
    jz IP4              ; yes: quit
    inc ebx             ; next divisor
    loopd IP1

IP3:    ; is prime, so set the Zero flag.
    test eax,0
    jmp IP5

IP4:    ; not prime, so clear the Zero flag.
    or  eax,0          ; ZF = 0
    jmp IP5

IP5:
    popa
    ret
IsPrime endp

; Set the Zero flag if EAX is evenly divisible by EBX.
; All registers are preserved.

IsDivisible proc
    push eax
    push edx
    mov  edx,0
    div  ebx
    cmp  edx,0
    pop  edx
    pop  eax

```

```

    ret
IsDivisible endp
end main

```

Ex 2: Prime Number Program - 3

```

Title  Chapt 7 (7.8.3) Ex 3: Prime Number Program - III

; Produce a list of all prime numbers between 2 and 65000
; using the Sieve of Eratosthenes method.

; This implementation does not use the library.inc file,
; because it tends to use more memory in the data segment.
; Instead, EXTRN directives for specific procedures were used.

.model small
.286
.stack 100h

; In the array, a 0 indicates that the number represented
; by that position is prime, and a 1 indicates that the
; number is not prime.

.data
num dw ?
delimiter db ", ", 0

FIRST_PRIME = 2
COUNT = 65000
array db COUNT dup(0)      ; boolean array

.code
extrn Writeint:proc, Writestring:proc, Crlf:proc, Clrscr:proc

main proc
    mov  ax,@data           ; init data segment
    mov  ds,ax

    mov  num,FIRST_PRIME

L1:
    mov  si,num

L2:
    add  si,num             ; point to next multiple of num
    cmp  si,COUNT          ; end of array reached?
    jae  L3                 ; yes
    mov  array[si],1       ; mark position as not prime
    jmp  L2                 ; no: continue

    ; find the next prime

L3: inc  num                ; look for next prime
    mov  si,num
    cmp  array[si],0       ; is it marked as prime?

```

```

    jne  L3                ; no: continue loop

    cmp  num,COUNT        ; yes: check for end of array
    jb  L1                ; if not reached, repeat outer loop

; Display the list of prime numbers

    call Clrscr
    mov  si,FIRST_PRIME

L5:
    or   array[si],0      ; array[si] = 0?
    jnz  L6                ; if not, skip next lines
    mov  ax,si
    mov  bx,10            ; display counter in AX
    call Writeint         ; in decimal
    mov  dx,offset delimiter
    call Writestring

L6:
    inc  si                ; go to next array position
    cmp  si,COUNT
    jb  L5

    mov  ax,4c00h         ; end program
    int  21h
main endp

end main

```

7.8.4 Arithmetic with Large Numbers

Ex 1: Quadword Addition Program

```

title Ch 7 (7.8.4), Ex 1: QuadWord Addition Program

; Add two 256-bit integers (16 bytes, 8 words).
; Optional: display the sum in hexadecimal.

.model small
.stack 100h
.386

.data
op1    dd  000C2384h,074ABC49Dh,00234324h,0234CA40h,0234787Ah,3453C300h,
        dd  0A2B2A406h,0B7C62938h
op2    dd  0108700h,0A64938D2h,00023240h,883593DAh,002AB649h,
        dd  0C234255h,00028977Ah,099843C4Dh
result dd  9 dup(0)

.code
extrn Writelong:proc, Clrscr:proc, Crlf:proc

```

```

main proc
    mov     ax,@data
    mov     ds,ax
    call   ClrScr

    mov     si,offset op1
    mov     di,offset op2
    mov     bx,offset result
    mov     cx,8             ; counter
    call   Multi32_Add
    call   Display_Result
    call   Crlf

    mov     ax,4C00h        ; exit program
    int    21h
main endp

; Multi32_Add
;
; Add two integers, consisting of multiple 32-bit
; doublewords. Input parameters: SI and DI point
; to the two operands, BX points to the destination
; operand, and CX contains the number of
; doublewords to be added.

.code
Multi32_Add proc
    pusha
    cld                     ; clear the Carry flag

L1: mov     eax,[si]        ; get the first operand
    adc     eax,[di]        ; add the second operand
    pushf                    ; save the Carry flag
    mov     [bx],eax        ; store the result
    add     si,4            ; advance all 3 pointers
    add     di,4
    add     bx,4
    popf                     ; restore the Carry flag
    loop   L1              ; repeat count

    mov     dword ptr [bx],0
    adc     dword ptr [bx],0 ; add leftover carry
    popa
    ret
Multi32_Add endp

Display_result proc
    mov     cx,9            ; display 9 doublewords
    mov     si,cx           ; get last index
    dec     si
    shl     si,2            ; SI * 4

DISPRES1:
    mov     eax,result[si]
    mov     bx,16
    call   Writelong

```

```

    sub    si,4
    loop  DISPRES1

    ret
Display_result endp
end main

```

Ex 2: Adding 10-Digit Numbers

```

title Ch 7 (7.8.4), Ex 3: Adding 10-Digit Numbers

; Solution program by William Dever, Miami-Dade Community College
; (wDever@mdcc.edu)

.model small
.stack 100h

; BIOS 10h Function Constants:
F_SETCUR          equ    2
F_WRITECHAR      equ    9

; BIOS 16h Function Constants:
F_WAITKEY        equ    10h

; DOS Function Constants:
F_DOS_STROUT     equ    9
F_DOS_CON_INP    equ    8

; Extended ASCII codes
HLINEGRAPH       equ    0fah
VLINEGRAPH       equ    0fah

; ASCII codes
K_ENTER          equ    0dh
BEEP             equ    07h
SPACE            equ    020h
BACKSPACE        equ    08h
K_BACKSPACE      equ    01bh
C_RIGHTARROW     equ    01ah
C_LEFTARROW      equ    01bh

; SCAN codes
K_LEFTARROW      equ    04bh
K_RIGHTARROW     equ    04dh

; BIOS attributes
; 4eh red background, yellow foreground
; 7bh white back, cyan fore
; 9h black back, light blue fore
; 18h blue back, grey fore
; 24h green back, red fore
; 0fh bright
; 07eh reverse video, yellow fore
; 07h normal
; 0cfh flash, red back, white fore

```

```

FLASH                                equ      0ffh

.data

; toprow, leftcol, bottomrow, rightcol, attrib, eol
; Define ASCII Addition Screen
bx_add      db      12,20,22,62,9h,'$'
            db      '$'                ; eof

; Define messages
msgwait db 'Press any key to continue', '$'

; Define labels
labels      db      'ASCII Addition Program',          '$'
            db      'First Number ==>',                '$'
            db      'Second Number ==>',               '$'
            db      'Result ==>',                      '$'
            db      'Action: Enter ',K_BACKSPACE, 'Backspace','$'
            db      '$'                                ; eof

; Define field label screen locations
pos_fieldname db 07eh,13,30,'$'                ; attrib, row, col, eol
            db 07eh,15,21,'$'
            db 07eh,16,21,'$'
            db 07eh,18,21,'$'
            db 018h,21,21,'$'
            db '$'                                ; eof

; Define menu labels
menul       db      'Return', '$'
            db      'Help', '$'
            db      'Quit', '$'
            db      '$'                            ; eof

; Define menu action label
menuAction1 db 'Action: ', C_LEFTARROW, ' '
            db      C_RIGHTARROW , ' ', 'Enter', '$'
            db '$'                                ; eof

; Define menu action label screen location
pos_menuAction1 db 018h, 21, 21, '$'
            db '$'                                ; eof

line_action1 db 21,21,21,61,018h,'$'
            db '$'                                ; eof

; Define menu label screen locations
pos_menul   db      07h, 23, 21, '$'
            db      07h , 23, 29, '$'
            db      07h , 23, 35, '$'
            db      '$'                            ; eof

; Define edit field cursor locations
cursor_edit db 15,39,'$'                ; row, col, eol
            db 16,39,'$'

```

```

                db      '$'                ; eof

result          db      18,38,'$'
                db      '$'                ; eof

; Define field variables
field           db      10 dup(' '), '$'   ; firstnum
                db      10 dup(' '), '$'   ; secondnum
                db      11 dup(' '), '$'   ; result
                db      '$'                ; eof

; direction, toprow, rightcol, bottomrow, leftcol, attrib, eol
clearscr db 7,0,0,23,79,07h,'$'
                db '$'                    ; eof

; Global variables:          Called by Procedures:
; -----
char            db ?      ; DrawHorozontal, DrawVerticle, DispChar, DispTxt2,
                        ; AcceptKey, CountItems
item            db ?      ; Main, CountItems
stringlen       dw ?      ; Main, StrLen, DrawAcctScr
delimiter       db ?      ; DispTxt2, DrawAcctScr, ProcessName,
ProcessAcctNo,
key             db ?      ; ProcessNumber, IsValidInput, CountItemsInRange
                        ; GetKey, ProcessKeyStroke, ProcessName,
                        ; ProcessAcctNo, ProcessNumber, AcceptKey
                        ; ConvertToUpper, IsValidInput
ok_key          db ?      ; ProcessName, ProcessAcctNo, IsValidInput
case_input      db ?      ; ProcessField, ProcessCase, ProcessNumber
this_field      db ?      ; Main, ProcessField
position        db ?      ; GetItemAtPos
displacement    dw ?      ; GetItemAtPos
highlimit       db ?
lowlimit        db ?

isEmpty         db ?
num_fields      db ?      ; Main, AreFieldsEmpty
num_items       db ?      ; CountItems, ProcessDecimalPt, Process Digits,
                        ; CountItemsInRange
numchar         db ?      ; DrawHorozontal, DrawVerticle, DispChar,
DispAcctScr      ; AcceptKey

col_edit        db ?
pos_cur_1       db ?      ; Main, ProcessSign, ProcessBackSpace
max_cur_pos     db ?      ; Main, ProcessName, ProcessAcctNo
max_digit       db ?      ; ProcessNumber, ProcessDigits

direction       db ?
toprow          db ?      ; DrawBox, DrawVerticle, ScrollWindow, SetWinCenter
                        ; DrawAcctScr
leftcol         db ?      ; DrawBox, DrawHorozontal, ScrollWindow,
SetWinCenter
bottomrow       db ?      ; DrawBox, DrawVerticle, ScrollWindow, SetWinCenter

```

```

rightcol      db ?      ; DrawBox, DrawHorizontal, ScrollWindow,
SetWinCenter
attrib        db ?      ; DrawHorizontal, DispChar, ScrollWin, DispAcctScr
; ProcessName, ProcessAcctNo, ProcessNumber
row           db ?      ; Main, DrawBox, DrawVerticle, SetCur, SetWinCenter
; DispAcctScr
col           db ?      ; Main, DrawBox, DrawAcctScr, SetCur, SetWinCenter
; DispTxt2, KeyPressRequest, AcceptKey,
ProcessName
; ProcessAcctNo, ProcessSign, ProcessBackSpace

ptr_data      dw ?      ; Main, ReadParameters, DrawAcctScr
ptr_var       dw ?      ; "          "          "
ptr_string    dw ?      ; Main, KeyPressRequest, DispTxt, DispTxt2,
StrLen,
; DrawAcctScr, CountItems, CountItemsInRange
; ProcessKeyStroke
ptr_str_ok_in dw ?      ; ProcessName, ProcessAcctNo, IsValidInput
ptr_pos_save  dw ?      ; ProcessKeyStroke, AcceptKey, ProcessBackStroke
ptr_casetable dw ?      ; ProcessField, ProcessCase, ProcessNumber
ptr_leastsig_sh dw ?    ; PtrSetUp
ptr_leastsig_lg dw ?    ; PtrSetUp
ptr_leastsig_rs dw ?    ; PtrSetUp
ptr_mostsig_rs dw ?    ; AddAsci
len_short     db ?      ; PtrSetUp
len_long      db ?      ; PtrSetUp

casefield     db        2                ; first field
              dw        ProcessNumber
              db        1                ; second field
              dw        ProcessNumber
              db        '$'              ; eof

casenumber1   db        '+'
              dw        ProcessSign
              db        '-'
              dw        ProcessSign
              db        '.'
              dw        ProcessDecimalPt
              db        K_BACKSPACE
              dw        ProcessBackSpace
              db        '0'
              dw        ProcessDigits
              db        '1'
              dw        ProcessDigits
              db        '2'
              dw        ProcessDigits
              db        '3'
              dw        ProcessDigits
              db        '4'
              dw        ProcessDigits
              db        '5'
              dw        ProcessDigits
              db        '6'
              dw        ProcessDigits

```

```

        db      '7'
        dw      ProcessDigits
        db      '8'
        dw      ProcessDigits
        db      '9'
        dw      ProcessDigits
        db      '$'                ; eof

fStateCase  db      0                ; 0 fields empty
            dw      ProcessAllFields
            db      1                ; 1 field empty
            dw      ProcessOneField
            db      2                ; 2 fields empty
            dw      ProcessNofields
            db      '$'                ; eof

CoreCase    db      21               ; cursor position
            dw      Core
            db      29               ; cursor position
            dw      Help
            db      35               ; cursor position
            dw      Terminate
            db      '$'                ; eof

MenuCase    db      K_LEFTARROW
            dw      CurMoveLeft
            db      K_RIGHTARROW
            dw      CurMoveRight
            db      K_ENTER
            dw      ProcessEnter
            db      '$'                ; eof

; *****
;                                     MACRO DEF
; *****
call_StrLen macro ptr_ToString, delimiter1
; Return the string length to 'stringlen'

push    ax

mov     ax, ptr_ToString
mov     ptr_string, ax
mov     al, delimiter1
mov     delimiter, al

call   StrLen

pop     ax
endm
; -----
call_SetCur macro row1, col1
; Set cursor position to row1, col1

push  ax

mov   al, row1

```

```
mov  row, al
mov  al, col1
mov  col, al

call SetCur

pop  ax

endm
; -----

call_ReadParameters macro data_label, var_label
; Load variables beginning with variable 'var_label' from data beginning
; at 'data_label'

push ax

lea  ax, data_label
mov  ptr_data, ax
lea  ax, var_label
mov  ptr_var, ax

call ReadParameters

pop  ax
endm

; -----

call_DispTxt macro text
; Display a string labeled 'text'

push ax

lea  ax, text
mov  ptr_string, ax

call DispTxt

pop  ax

endm

; -----

call_DispTxt2 macro ptr_string1, delimiter1
; Display a string at location 'ptr_string1' terminated by 'delimiter1'

push  ax

mov  ax, ptr_string1
mov  ptr_string, ax
mov  al, delimiter1
mov  delimiter, al

call  DispTxt2

pop  ax
endm
```

```

; -----
call_DispMenu macro menu_label, data_label, var_label, edit_pos
; Display a bar menu of 'menu_label' names at 'var_label' screen locations
; loaded from 'data_label'. 'edit_pos' = position of menu item to highlight

push    ax

lea     ax, menu_label
mov     ptr_string, ax          ; ptr_string := addr. of menu labels
lea     ax, data_label
mov     ptr_data, ax          ; ptr_data := addr. of screen coordinates
lea     ax, var_label
mov     ptr_var, ax           ; ptr_var := addr. of loaded variables
mov     al, edit_pos
mov     col_edit, al          ; col_edit := screen loc. of highlighted menu
item

call    DispMenu

pop     ax
endm

; -----
call_DispChar macro char1, numchar1, attrib1
; Display a character 'char1' 'numchar1' times with attribute 'attrib1'

push    ax

mov     al, char1
mov     char, al
mov     al, numchar1
mov     numchar, al
mov     al, attrib1
mov     attrib, al

call    DispChar

pop     ax

endm

; -----
call_DrawHorizontal macro row, col1, col2
; Displays a horizontal line to the screen at 'row' from 'col1' to 'col2'

push    ax                    ; preserve preentry state

mov     al, col2
sub     al, col1
inc     al
mov     numchar, al          ; numchar = (col2 - col1) + 1

mov     char, HLINEGRAPH    ; Set char

call_SetCur    row, col1
call_DispChar char, numchar, attrib

pop     ax                    ; restore preentry state

```

```

endm
; -----
call_DrawVertical macro coll, row1, row2
; Displays a verticle line on screen at 'coll' from 'row1' to 'row2'

push ax ; preserve preentry state

mov al, coll
mov col, al ; col := coll

mov al, row1
mov toprow, al

mov al, row2
mov bottomrow, al

call DrawVerticle

pop ax ; restore preentry state

endm
; -----
call_GetAddrOccupied macro string1
; Return the address of the first field occupied in record 'string1'
; to 'ptr_string'

push ax

lea ax, string1
mov ptr_string, ax

call GetAddrOccupied

pop ax

endm
; -----
call_GetAddrNextField macro ptr_string1
; Returns the address of the next field from the current field at address
; 'ptr_string1' to 'ptr_string'

push ax

call_StrLen ptr_string1, '$'

mov ax, ptr_string
add ax, stringlen
inc ax
mov ptr_string, ax ; ptr_string := ptr_string + stringlen + 1

pop ax

endm
; -----

```

```

call_CountItemsInRange macro ptr_string1, lowlimit1, highlimit1, delimit-
iter1
; Returns to 'num_items' the count of items of specified range 'lowlimit1'
to
; 'highlimit1' in string at address 'ptr_string1' terminated by 'delimit-
iter1'

push ax

mov ax, ptr_string1
mov ptr_string, ax
mov al, lowlimit1
mov lowlimit, al
mov al, highlimit1
mov highlimit, al
mov al, delimiter1
mov delimiter, al

call CountItemsInRange

pop ax
endm
; -----
call_ProcessCase macro casetable, caseitem
; Invokes a case table of label 'casetable' for item 'caseitem'

push ax

lea ax, casetable
mov ptr_casetable, ax
mov al, caseitem
mov case_input, al

call ProcessCase

pop ax

endm

; *****
.code

main proc

mov ah, 0 ; set video mode
mov al, 3 ; color text
int 10h

mov ax, @data ; assign data seg address to ds
mov ds, ax

call Core

quit:
mov ax, 4c00h

```

```

int    21h

main endp
;
*****

Core proc
; Main Procedure

push   ax                ; preserve preentry state

call   ClearScreen

; clear field memory
lea   ax, field
mov   ptr_string, ax
mov   num_items, 3
call  ClearFields

call  DrawAddScr

call  EnterNumbers

call  PostResult

; Clear a line for the Action Menu
call_ReadParameters line_action1, toprow
call  ClearLines        ; clear the line

; Display the Action Menu
call_ReadParameters pos_menuAction1, attrib

lea   ax, menuAction1
mov   ptr_string, ax
call_DispTxt2 ptr_string, '$'

; Menu Setup
mov   col_edit, 21        ; init. menu col edit position

call_DispMenu  menu1, pos_menu1, attrib, col_edit

call  GetKey

call_ProcessCase  MenuCase, key

pop   ax                ; restore preentry state

ret
Core endp
; *****

AddAscii proc
; Adds ASCII numbers
; Requires: 'ptr_leastsig_lg', 'ptr_leastsig_sh', 'ptr_leastsig_rs',
; 'len_long', 'len_short' are assigned
; Returns: 'result'

```

```

push ax                ; preserve preentry state
push bx
push cx
push di
push si

mov  cx, 0              ; init.
mov  ax, 0              ; init.

mov  si, ptr_leastsig_sh ; addr. of least significant digit of shortest
num
mov  di, ptr_leastsig_lg ; addr. of least significant digit of longest
num
mov  bx, ptr_leastsig_rs ; addr. of least significant digit of result
mov  cl, len_short      ; counter = length of shortest digit

inc  si                ; init. ptrs. to preceding position
inc  di
inc  bx

; Add digits place by cooresponding place to most significant position of
; shortest digit
AA2digits:
dec  si                ; position pointers
dec  di
dec  bx

mov  al, byte ptr [si] ; al := digit of shortest number
add  al, ah            ; add carry
mov  ah, 0
add  al, byte ptr [di] ; add same place digit from longest number
aaa                                ; adjust
or   al, 30h           ; convert to ASCII
mov  [bx], al          ; save result
loop AA2digits

mov  al, len_long
cmp  al, len_short
je   CARRY             ; if length of 2 numbers are not equal then

mov  cx, 0              ; init. cx
mov  cl, len_long
sub  cl, len_short     ; loop counter := (length of longest number -
; length of shortest number)

; Carry remaining digits of long number to cooresponding place in result
AAldigit:
dec  di                ; position ptrs.
dec  bx

mov  al, [di]          ; move digit to register
add  al, ah            ; add carry

```

```

mov  ah, 0                ; clear carry
aaa                                ; adjust
or   al, 30h              ; convert to ASCII
mov  [bx], al
loop AAldigit

                                ; else
; last carry
CARRY:
cmp  ah, 0
je  AAquit                ; if carry then

dec  di                    ; position ptrs.
dec  bx

mov  al, ah
aaa                                ; adjust
or   al, 30h              ; convert to ASCII
mov  [bx], al             ; save carry
                                ; else

AAquit:
mov  ptr_mostsig_rs, bx    ; save ptr to result
                                ; endif
                                ; endif

pop  si                    ; restore preentry state
pop  di
pop  cx
pop  bx
pop  ax

ret
AddAscii endp

; *****

ProcessNumber proc
; Set up for 'ProcessCase'

push ax                    ; preserve preentry state

mov  attrib, 04eh         ; red back, yellow fore

mov  delimiter, '$'
mov  max_digit, 8

call_ProcessCase casenumber1, key

pop  ax                    ; restore preentry state

ret
ProcessNumber endp
; *****

```

```

ProcessSign proc
; Validity check for '+'/'-' position in numeric field

push  ax                      ; preserve preentry state

mov   al, pos_cur_1
cmp   al, col
jne   NOTCUR1                 ; if col position = first cursor pos then

call  AcceptKey               ; accept and process key
jmp   OKSIGN                  ; else

NOTCUR1:
call  SoundWarning            ; give user a sound beeping!!
; endif

OKSIGN:
pop   ax                      ; restore preentry state

ret
ProcessSign endp
; *****

ProcessDecimalPt proc
; Validity check for number of decimal points entered in one numeric field

push  ax                      ; preserve preentry state

call  CountItems              ; count decimal points entered

mov   al, num_items
cmp   al, 1
je    DEC_ERROR               ; if num of dec. pts <= 1 then

call  AcceptKey               ; accept and process key
jmp   OKDEC                   ; else
DEC_ERROR:
call  SoundWarning            ; give user a sound beeping!!
; endif

OKDEC:
pop   ax                      ; restore preentry state

ret
ProcessDecimalPt endp
; *****

ProcessBackSpace proc

push  ax                      ; preserve preentry state
push  bx

mov   al, pos_cur_1
cmp   col, al
je    FIRSTCURPOS             ; if col pos <> min cursor pos then

dec   col                      ; move cursor back one space
call  _SetCur row, col

```

```

mov  al, SPACE                ; clear the space at former cursor pos
mov  char, al
call_DispChar char, numchar, attrib

dec  ptr_pos_save             ; decrement ptr to variable space

mov  bx, ptr_pos_save         ; clear former pos in variable mem
mov  byte ptr [bx], SPACE

FIRSTCURPOS:
                                ; else
                                ; leave cursor at current pos

                                ; endif

pop  bx                       ; restore preentry state
pop  ax

ret
ProcessBackSpace endp
; *****

ProcessDigits proc

push  ax                       ; preserve preentry state

call_CountItemsInRange ptr_string, 0, 9, '$'

mov  al, num_items
cmp  al, max_digit
jb  OKDIGIT                    ; if num_items > max_digit then

call  SoundWarning            ; give the user a sound beeping!!

jmp  PDquit
OKDIGIT:                       ; else

call  AcceptKey               ; accept and process key
                                ; endif

PDquit:
pop  ax                       ; restore preentry state

ret
ProcessDigits endp
; *****

ProcessAllFields proc

push  ax                       ; preserver preentry state

lea  ax, field
mov  ptr_var, ax
call  PtrSetUp

```

```
call  AddAscii

call_ReadParameters result, row
call  SetCur

call_DispTxt2 ptr_mostsig_rs, SPACE

pop  ax          ; restore preentry state

ret
ProcessAllFields endp
; *****

ProcessOneField proc

push  ax          ; preserve preentry state

call_ReadParameters result, row
call  SetCur

; find address of occupied field
call_GetAddrOccupied field

call_DispTxt2 ptr_string, SPACE

pop  ax          ; restore preentry state

ret
ProcessOneField endp
; *****

ProcessNoFields proc

ret
ProcessNoFields endp
; *****

Terminate proc

ret
Terminate endp
; *****

Help proc

ret
Help endp
; *****

ProcessEnter proc
; Case setup

call_ProcessCase CoreCase, col

ret
```

```

ProcessEnter endp
; *****

EnterNumbers proc
; Input field loop

push  ax                ; preserve preentry state
push  cx
push  di

passes = 2
mov   cx, passes
mov   num_fields, passes
mov   ax, offset cursor_edit ; set ptr to data
mov   ptr_data, ax
mov   ax, offset row        ; set ptr to var
mov   ptr_var, ax
mov   di, offset field      ; set ptr to field
mov   item, '.'             ; item to count

; Loop through the fields
NEXTFIELD:

call  Readparameters      ; read row, col positions
call  SetCur

mov   al, col              ; minimum cursor position
mov   pos_cur_1, al

mov   ptr_string, di       ; destination index of field variable
mov   this_field, cl       ; this_field := cl
mov   al, col              ; pos_cur_1 := col (first, field cursor
pos.)
mov   pos_cur_1, al

call_StrLen ptr_string, '$'
mov   ax, stringlen
add   al, pos_cur_1
dec   al
mov   max_cur_pos, al      ; max_cur_pos := (pos_cur_1 + stringlen) - 1

call  ProcessKeyStroke

call_GetAddrNextField ptr_string
mov   di, ptr_string

loop  NEXTFIELD

pop   di                  ; restore preentry state
pop   cx
pop   ax

ret

```

```

EnterNumbers endp
; *****

PostResult proc
; Case table set up

push  ax                ; preserve preentry state

mov   num_fields, 2
lea  ax, field
mov  ptr_string, ax

call  AreFieldsEmpty    ; how many fields are empty?

call_ProcessCase  fStateCase, isEmpty

pop  ax                ; restore preentry state

ret
PostResult endp
; *****

ProcessField proc
; Set up for 'ProcessCase'

call_ProcessCase  casefield, this_field

ret
ProcessField endp
; *****

ProcessKeyStroke proc
; Process field input
; Effects:

push  ax                ; preserve preentry state

mov  ax, ptr_string     ; destination index of field var
mov  ptr_pos_save, ax   ; ptr to char in field var

NEXTKEY:

call  GetKey

mov  al, key
cmp  al, K_ENTER
je   PK                ; if enter key pressed then

call  ProcessField     ; process field
                        ; else
jmp  NEXTKEY          ; press another key
                        ; endif

PK:
pop  ax                ; restore preentry state

```

```

ret
ProcessKeyStroke endp
;*****

AcceptKey proc
; Save input char to field, display char, position cursor for next char
; Effects: col, char, numchar

push  ax                ; preserve preentry state
push  di

mov   ax, ptr_pos_save  ; save key to field var
mov   di, ax            ; di := ax := ptr_pos_save
mov   al, key
mov   [di], al         ; save key at location di(ptr_pos_save)

inc   ptr_pos_save     ; next position in field

mov   char, al         ; display char

call_DispChar char, 1, attrib

inc   col              ; position for next char
call_SetCur row, col

pop   di              ; restore preentry state
pop   ax

ret
AcceptKey endp
; *****

CurMoveLeft proc

push  ax                ; preserve preentry state

; GetLeftArrowPos proc setup
mov   displacement, 4   ; displacement := 4
lea   ax, pos_menu1
mov   ptr_data, ax
add   ptr_data, 2      ; ptr_data := addr. of pos_menu1 + 2
mov   al, col
mov   item, al         ; item := col
mov   num_items, 3     ; !!!!!!!!!!!!!!!

call  GetLeftArrowPos

; DispMenu Setup
mov   al, item
mov   col_edit, al     ; col_edit := item

call_DispMenu menu1, pos_menu1, attrib, col_edit

call  GetKey

call_ProcessCase MenuCase, key

```

```

pop    ax                ; restore preentry state

ret
CurMoveLeft endp
;*****

CurMoveRight proc

push   ax                ; preserve preentry state

; GetRightArrowPos proc setup
mov    displacement, 4    ; displacement := 4
lea    ax, pos_menu1
mov    ptr_data, ax
add    ptr_data, 2        ; ptr_data := addr. of pos_menu1 + 2
mov    al, col
mov    item, al           ; item := col
mov    num_items, 3      ; !!!!!!!!!!!!!!!!

call   GetRightArrowPos

; DispMenu Setup
mov    al, item
mov    col_edit, al      ; col_edit := item

call   DispMenu menu1, pos_menu1, attrib, col_edit

call   GetKey

call   ProcessCase MenuCase, key

pop    ax                ; restore preentry state

ret
CurMoveRight endp
;*****

GetLeftArrowPos proc
; Process a left arrow keypress
; Returns: 'item'

push   ax                ; preserve preentry state

call   GetPosOfItem

cmp    position, 1       ; if in first position then
jne    LAM1
mov    al, num_items
mov    position, al      ; position := last position
jmp    LAM2

; else
LAM1:
dec    position          ; decrement position
; endif

```

```

LAM2:
call  GetItemAtPos

pop   ax                ; restore preentry state

ret
GetLeftArrowPos endp
; *****

GetRightArrowPos proc
; Process a right arrow keypress
; Returns: next col position to 'item'

push  ax                ; preserve preentry state

call  GetPosOfItem

mov   al, num_items
cmp   position, al      ; if in last position then
jne   RAM1
mov   position, 1      ; position := first position
jmp   RAM2
; else

RAM1:
inc   position          ; increment position
; endif

RAM2:
call  GetItemAtPos

pop   ax                ; restore preentry state

ret
GetRightArrowPos endp
; *****

GetItemAtPos proc
; Returns 'item' at 'position'
; Requires: 'position', 'ptr_data', 'displacement', num_items assignment

push  ax                ; preserve preentry state
push  cx
push  si

mov   si, ptr_data      ; init
mov   cx, 0
mov   cl, num_items

GIAP:

mov   al, num_items
sub   al, cl
inc   al                ; al := num_items - (cl - 1) / iteration =
; current position

cmp   al, position
je    GIAPfound         ; If position <> current position then

```

```

add  si, displacement ; advance to next position
loop GIAP

GIAPfound:                ; else

mov  al, [si]
mov  item, al             ; item := [si]
                               ; endif

pop  si                   ; restore preentry state
pop  cx
pop  ax

ret
GetItemAtPos endp
; *****

GetPosOfItem proc
; Returns 'position' of 'item'
; Requires: 'displacement', 'item', 'num_items', 'ptr_data' assignment

push ax                   ; preserve preentry state
push cx
push si

; init.
mov  cx, 0
mov  cl, num_items
mov  si, ptr_data         ; si := ptr_data
mov  position, 0

FPOI:

inc  position
mov  al, [si]
cmp  al, item
je   FPOIfound           ; if not item at position then
add  si, displacement    ; position to next item

loop FPOI

                               ; endif
FPOIfound:

pop  si                   ; restore preentry state
pop  cx
pop  ax

ret
GetPosOfItem endp
; *****

PtrSetUp proc
; Requires: ptr_var := address of first field
; Effects: returns values for 'ptr_leastsig_lg, ptr_leastsig_sh,
; ptr_leastsig_rs, len_short, len_long

```

```

push  ax                ; preserve preentry state
push  bx
push  dx

mov   ax, ptr_var       ; ptr_var := address of 1st field
mov   ptr_string, ax    ; ptr_string := ptr_var

call_StrLen ptr_string, SPACE ; stringlen := length of number in 1st
field
mov   ax, stringlen     ; ax := length of number in 1st field

; assume number in first field position has longest length
mov   dx, 0
add   dx, stringlen
add   dx, ptr_var
dec   dx
mov   ptr_leastsig_lg, dx ; ptr to least significant digit :=
                        ; address of number + (length of number -1)
; point to next number field
call_GetAddrNextField ptr_string

; assume 2nd field has shortest num

call_StrLen ptr_string, SPACE ; stringlen := length of number in 2nd
field
mov   bx, ptr_string
add   bx, stringlen
dec   bx
mov   ptr_leastsig_sh, bx ; ptr to least significant digit :=
                        ; address of field + (length of number - 1)

; determine longest of two numbers
cmp   ax, stringlen    ; compare length of first, second number
jae   GREATER          ; if length 1st num < length 2nd num then

mov   ptr_leastsig_lg, bx ; swap the two pointers
mov   ptr_leastsig_sh, dx
mov   len_short, al     ; len_short := length of first number
mov   ax, stringlen
mov   len_long, al     ; len_long := length of 2nd number
jmp   PSUnext          ; else

GREATER:
mov   len_long, al     ; len_long := length of first number
mov   ax, stringlen   ; len_short := length of 2nd number
mov   len_short, al

; endif

PSUnext:
; determine ptr to least significant digit of 'result'

call_GetAddrNextField ptr_string

```

```

; ptr_string has address of result

mov  ax, ptr_string
add  al, len_long
inc  ax
mov  ptr_leastsig_rs, ax      ; ptr to least significant digit of result :=
                               ; address of result +
                               ; (length of longest of 2 num to add) + 1

pop  dx                      ; restore preentry state
pop  bx
pop  ax

ret
PtrSetUp endp
; *****
;                               DISPLAY PROCEDURES
; *****

DrawAddScr proc

push  ax                    ; preserve preentry state
push  di

call_ReadParameters bx_add, toprow
call  DrawBox

; Display labels

mov  ax, offset pos_fieldname ; ptr_data := address of
mov  ptr_data, ax             ; pos_fieldname
mov  ax, offset attrib
mov  ptr_var, ax              ; ptr_var := address of attrib.
mov  di, offset labels       ; ptr to labels

mov  numchar, 1

LBL:

cmp  byte ptr [di], '$'      ; while not eof
je   RPeof
call Readparameters         ; parameters for field scr. loc.
mov  ptr_string, di
call_SetCur row, col

call_DispTxt2 ptr_string, '$'
call_StrLen ptr_string, '$'

add  di, stringlen          ; set pointer to next label
inc  di

jmp  LBL

; endwhile

RPeof:

```

```

pop    di                ; restore preentry state
pop    ax

ret
DrawAddScr endp
; *****

DrawBox proc
; Displays a box to the screen
; Requires: toprow, leftcol, bottomrow, rightcol assignment

; Draw top horizontal
call_DrawHorizontal toprow, leftcol, rightcol

; Draw left vertical
call_DrawVertical leftcol, toprow, bottomrow

; Draw right vertical
call_DrawVertical rightcol, toprow, bottomrow

; Draw bottom horizontal
call_DrawHorizontal bottomrow, leftcol, rightcol

ret
DrawBox endp
; *****

DrawVerticle proc

push  ax
push  cx

mov   char, VLINEGRAPH ; Set char, numchar

mov   cl, bottomrow
sub   cl, toprow
inc   cl          ; loop counter = (bottomrow - toprow) + 1

mov   al, toprow
mov   row, al    ; row := row1

DrawV1:
call_SetCur    row, col
call_DispChar  char, 1 , attrib

inc   row                ; set to next row
loop  DrawV1

pop   cx
pop   ax

ret
DrawVerticle endp
; *****

```

```

ScrollWin proc
; Scrolls window either up or down
; Effects:

push  ax                ; preserve preentry state
push  bx
push  cx
push  dx

mov   al, bottomrow    ; numlines = (bottomrow - toprow) + 1
sub   al, toprow
inc   al
mov   ah, direction
mov   ch, toprow
mov   cl, leftcol
mov   dh, bottomrow
mov   dl, rightcol
mov   bh, attrib
int   10h

pop   dx                ; restore preentry state
pop   cx
pop   bx
pop   ax

ret
ScrollWin endp
; *****

DispTxt proc
; Displays a string of text
; Requires: 'ptr_string' := address of string

push  ax                ; preserve preentry state
push  dx

mov   ah, F_DOS_STROUT
mov   dx, ptr_string
int   21h

pop   dx                ; restore preentry state
pop   ax

ret
DispTxt endp
; *****

DispTxt2 proc
; Displays a string of text using BIOS INT 10h, string length is specified
; by the variable 'delimiter'
; Requires: 'ptr_string', 'delimiter'

push  ax                ; preserve preentry state
push  bx

mov   bx, ptr_string    ; assign bx string offset

```

```

DT2:
mov  al, delimiter
cmp  byte ptr [bx], al
je   DT2quit                ; while not eol

mov  al, [bx]                ; char := [bx]
mov  char, al

call_DisPChar char, 1 , attrib ; display character

inc  col                    ; next position

call_SetCur row, col       ; set cursor position

inc  bx
jmp  DT2                    ; next ptr position
                                ; endwhile

DT2quit:

pop  bx                    ; restore preentry state
pop  ax

ret

DispTxt2 endp
; *****

DispTxt3 proc

; Display text to multiple coordinate screen locations
; Requires: 'ptr_var', 'ptr_data', 'ptr_string', set

push  ax                    ; preserve preentry state
push  bx

mov   numchar, 1
mov   delimiter, '$'

DT3:
mov   bx, ptr_string
cmp   byte ptr [bx], '$'    ; while not eof
je    DT3eof

; load parameters into variables
call  Readparameters

call  SetCur

call_DisPtxt2 ptr_string, delimiter

; point to next field address
call_GetAddrNextField ptr_string

jmp   DT3                    ; endwhile

```

```

DT3eof:

pop    bx                ; restore preentry state
pop    ax

ret
DispTxt3 endp
; *****

DispMenu proc

; Display text to multiple coordinate screen locations
; Requires: 'ptr_var', 'ptr_data', 'ptr_string'

push   ax                ; preserve preentry state
push   bx

mov    numchar, 1
mov    delimiter, '$'

DM:

mov    bx, ptr_string
cmp    byte ptr [bx], '$' ; while not eof
je     DMeof

; load parameters into variables
call   Readparameters

call   SetCur

mov    al, col
cmp    al, col_edit
jne    DM1                ; if cursor on menu then
mov    attrib, 04eh        ; assign special attribute
                        ; endif

DM1:

call_DispTxt2 ptr_string, delimiter

; point to next field address
call_GetAddrNextField ptr_string

jmp    DM                ; endwhile

DMeof:

; set cursor to user item selected
mov    al, col_edit
mov    col, al            ; col := col_edit

call   SetCur

pop    bx                ; restore preentry state
pop    ax

```

```

ret
DispMenu endp
; *****

SetWinCenter proc
; Sets the coordinates for center window/box position
; Effects: row, col

push ax          ; preserve preentry state
push bx

mov al, bottomrow ; rowsize = (bottomrow - toprow) + 1
sub al, toprow
inc al
mov ah, 0         ; midpoint = rowsize/2
mov bl, 2
div bl
mov ah, toprow   ; row = toprow + midpoint
mov row, ah
add row, al

mov al, rightcol ; colsize = (rightcol - leftcol) + 1
sub al, leftcol
inc al
mov ah, 0         ; midpoint = colsize/2
mov bl, 2
div bl
mov ah, leftcol   ; col = leftcol + midpoint
mov col, ah
add col, al

pop bx           ; restore preentry state
pop ax

ret
SetWinCenter endp
; *****

KeyPressRequest proc
; Displays a string of text at window/box center defined at offset
'msgwait'
; and waits for user keypress
; Effects:

push ax          ; preserve preentry state
push bx

call SetWinCenter
inc col
call_SetCur row, col

; Display message
call_DispTxt msgwait

mov ah, F_WAITKEY

```

```

int    16h

dec    col                ; restore preentry state
pop    bx
pop    ax

ret
KeyPressRequest endp
; *****

DispChar proc
; Displays a character to screen 'numchar' times using BIOS INT 10h
; Effects:

push   ax                ; preserve preentry state
push   bx
push   cx

mov    ah, F_WRITECHAR   ; write char function
mov    al, char           ; write char
mov    bh, 0             ; default page num
mov    bl, attrib
mov    cl, numchar       ; display numchar times
int    10h              ; call BIOS

pop    cx                ; restore preentry state
pop    bx
pop    ax

ret
DispChar endp
; *****

SetCur proc
; Sets cursor to row, col position
; Effects:

push   ax                ; preserve preentry state
push   dx
push   bx

mov    ah, F_SETCUR     ; set cursor pos
mov    dh, row
mov    dl, col
mov    bh, 0            ; default pageno
int    10h             ; call BIOS

pop    bx                ; restore preentry state
pop    dx
pop    ax

ret
SetCur endp
; *****

ReadParameters proc

```

```

; Loads defined coordinates into variables
; Requires: ptr_data := address of data to be read
;           ptr_var  := address of first variable to be loaded
; Effects:  ptr_data := address of next data set (on ret)

push ax                ; preserve preentry state
push di
push si

mov  di, ptr_var      ; assign destination index address
; (variable)
mov  si, ptr_data     ; assign source index address (data)

; read parameters

B1:
cmp  byte ptr [si], '$'
je   RPeol           ; while not eol
mov  al, [si]        ; load values into variables
xchg al, [di]
inc  si              ; move to next data coordinate
inc  di              ; move to next variable coordinate
jmp  B1
; endwhile

RPeol:

inc  si              ; move past eol
mov  ptr_data, si    ; ptr_data := address of next data item

pop  si              ; restore preentry state
pop  di
pop  ax

ret                  ; endif
ReadParameters endp
; *****

ProcessCase proc
; Process keystroke according to specific field

push ax                ; preserve preentry state
push bx
push cx

mov  al, case_input   ; field to process
mov  bx, ptr_casetable ; pointer to casetable

PC:
cmp  byte ptr [bx], '$' ; while not eof
je   PCeof

cmp  al, byte ptr [bx] ;
jne  PC_NOTFOUND      ; if match found
call word ptr [bx + 1] ; call this procedure

```

```

    jmp    PC_QUIT                ;    exit

PC_NOTFOUND:                    ;    else
add     bx, 3                    ;    point to next item
jmp     PC                      ;    endif
; endwhile

PCeof:
call    SoundWarning            ;    give user a sound beeping!!

PC_QUIT:
pop     cx                      ;    restore preentry state
pop     bx
pop     ax

ret
ProcessCase endp
; *****

GetKey proc
; User input, unfiltered
; Effects: key

push   ax                      ;    preserve preentry state

mov    ah, F_DOS_CON_INP       ;    input keystroke, no echo
int    21h                     ;    call DOS

cmp    al, 0                    ;
jne    GK                      ;    if key = extended key then
int    21h                     ;    call DOS
; endif

GK:
mov    key, al                 ;    save input to var. 'key'

pop    ax                      ;    restore preentry state

ret
GetKey endp
; *****
;
;                               UTILITIES
; *****

IsValidInput proc
; Checks 'key' input against a defined string of acceptable input
; Requires: 'ptr_str_ok_in' := address of acceptable input string,
; 'key', and 'delimiter' assignment
; Effects: Sets 'ok_key'

push   ax                      ;    preserve preentry state
push   bx
push   dx

mov    bx, ptr_str_ok_in
mov    ok_key, 0                ;    initialize 'ok_key'
mov    dl, key                  ;    check key

```

```

mov    al, delimiter

CHECKNEXT:

cmp    byte ptr [bx], al
je     ENDSTR_IVI                ; while not end of string

cmp    byte ptr [bx], dl
je     OKITEM                    ; if not pointing to key input
inc    bx
jmp    CHECKNEXT                ; check key against next item
; endif
; endwhile

OKITEM:
mov    ok_key, 1                ; set 'ok_key' on

ENDSTR_IVI:

pop    dx                        ; restore preentry state
pop    bx
pop    ax

IsValidInput endp
; *****

CountItemsInRange proc
; Tally items in a string per specified range
; Returns sum in 'num_items'
; Requires 'ptr_string', 'delimiter', 'lowlimit', 'highlimit' assignment

push ax                        ; preserve preentry state
push bx

mov    num_items, 0             ; init. num_items
mov    bx, ptr_string          ; init. ptr_string to bx

NEXTITEM_CIIR:
mov    al, delimiter
cmp    byte ptr [bx], al
je     ENDOFSTR                ; while not eol

mov    al, lowlimit
cmp    byte ptr [bx], al
jb    OUTOFRANGE_CIIR         ; if not < lower limit

mov    al, highlimit
cmp    byte ptr [bx], al
ja    OUTOFRANGE_CIIR         ; and if not > higher limit then

inc    num_items                ; increment num of items

OUTOFRANGE_CIIR:
; else
inc    bx                        ; try next item
jmp    NEXTITEM_CIIR          ; endif
; endwhile

```

```

ENDOFSTR:

pop bx          ; restore preentry state
pop ax

ret
CountItemsInRange endp
; *****

StrLen proc
; Calculates length of a string
; Requires: 'ptr_string' := address of string
; Effects: 'stringlen'

push ax          ; preserve preentry state
push cx
push di

mov di, ptr_string ; pointer to string
mov cx, 0          ; init. counter
mov al, delimiter

SL:

cmp [di], al      ; while not end of string
je STRTERM
inc cx            ; increment counter
inc di            ; point to next char
jmp SL
; endwhile

STRTERM:

mov stringlen, cx ; save to variable

pop di           ; restore preentry state
pop cx
pop ax

ret
StrLen endp
; *****

CountItems proc
; Count specified char occurrence in a string
; Returns value to 'num_items'

push ax          ; preserve preentry state
push bx

mov bx, ptr_string
mov al, item
mov num_items, 0 ; init. num_items

CI:

cmp byte ptr [bx], '$' ; while not eol

```

```

je    CIquit

cmp   al, [bx]
jne   NOTITEM           ;   if item = item in list

inc   num_items         ;   increment num_items

NOTITEM:                ;   endif
inc   bx                ;
jmp   CI                ;   next item in list

CIquit:
pop   bx                ; restore preentry state
pop   ax

ret
CountItems endp
; *****

ConvertToUpper proc
; Convert lowercase alphabetic characters to uppercase

cmp   key, 'a'
jb    UPPER             ; If not uppercase then
sub   key, 020h         ; make uppercase

UPPER:                  ; endif

ret
ConvertToUpper endp
; *****

SoundWarning proc
; Sound Beep

push  ax                ; preserve preentry state
push  dx

mov   ah, 2             ;DOS function 2
mov   dl, BEEP
int   21h

pop   dx
pop   ax                ; restore preentry state

ret
SoundWarning endp
; *****

AreFieldsEmpty proc
; Requires: 'num_fields', 'ptr_string' assignment
; Effects: 'isEmpty' assigned the number of empty fields

push  ax                ; preserve preentry state
push  cx

```

```

mov  cx, 0                ; init.
mov  cl, num_fields      ; init.
mov  isEmpty, 0         ; init.

AFE:

; determine length of field occupant
call_StrLen ptr_string, SPACE
cmp  stringlen, 0
jne  OCCUPIED           ; if empty then
inc  isEmpty           ; increment isEmpty
                                ; endif

OCCUPIED:

; locate ptr to next field
call_GetAddrNextField ptr_string

loop AFE

pop  cx                ; restore preentry state
pop  ax

ret
AreFieldsEmpty endp
; *****

GetAddrOccupied proc
; Return the address of an occupied field to 'ptr_string'
; Requires: the starting addr. of the fields in 'ptr_string'
; Effects: 'ptr_string'

push ax                ; preserve preentry state

GAO:
mov  bx, ptr_string
cmp  byte ptr [bx], '$'
je   GAOeof           ; while not eof

; test field for use

call_StrLen ptr_string, SPACE
cmp  stringlen, 0
jne  GAO_Occupied     ; if field empty then

; move to next field
call_GetAddrNextField ptr_string

jmp  GAO                ; endif
                                ; endwhile
GAO_Occupied:          ; return ptr_string address
GAOeof:

pop  ax                ; restore preentry state

ret

```

```

GetAddrOccupied endp
; *****

ClearScreen proc

call_ReadParameters  clearscr, direction

call  ScrollWin

ret
ClearScreen endp
; *****

ClearFields proc
; Clears 'num_items' fields with spaces
; Requires: 'ptr_string', 'num_item' assignment

push  bx                ; preserve preentry state
push  cx

mov   bx, ptr_string
mov   cx, 0
mov   cl, num_items

L_CF:                ; for cx = num_items

CF:
cmp   byte ptr [bx], '$'
je    CFnext         ; while not eol
mov   byte ptr[bx], SPACE ; blank out location
inc   bx             ; next location
jmp   CF             ;
                                ; endwhile

CFnext:
inc   bx
loop  L_CF           ; next field

pop   cx            ; restore preentry state
pop   bx

ret
ClearFields endp
; *****

ClearLines proc
; Requires: 'bottomrow,', 'toprow', 'leftcol', 'rightcol', 'attrib'
; assignment

push  ax                ; preserve preentry state
push  cx

mov   char, SPACE      ; char := SPACE

mov   al, rightcol

```

```

sub    al, leftcol
inc    al
mov    numchar, al    ; numchar := num cols := (rightcol - leftcol) + 1

mov    al, leftcol
mov    col, al        ; col := leftcol

mov    al, toprow     ; al := toprow

CL1:

mov    row, al
call  SetCur
call  DispChar
inc    al    ; next row
cmp    al, bottomrow
ja     Cldone    ; if row <= bottomrow

jmp    CL1        ; next row
                ; else
Cldone:          ; done
                ; endif

pop    cx        ; restore preentry state
pop    ax

ret
ClearLines endp
; *****
end main

```

Ex 3: Multidigit Packed Decimal Multiplication

```

title Ch 7 (7.8.4), Ex 3: Multidigit BCD Multiplication

; This program performs unpacked BCD multiplication
; on two numbers

.radix    16
.model    small
.stack    100

cr        = 0dh
lf        = 0ah
video     = 10h
stdout    = 1

.data
factor1   db  01,02,03,04
fac1z     =  $-factor1
factor2   db  01,05
fac2z     =  $-factor2
product   db  6 dup(0)
templ     db  6 dup(0)

```

```

temp2      db  6 dup(0)

.code
main       proc
    mov     ax, @data                ; Set up
    mov     ds, ax                  ; data segment
    mov     si, offset factor1+3    ; Get pointer to the end of multiplicand
    mov     di, offset temp1+5      ; Get pointer to the of intermediate storage
    xor     dl, dl                  ; Make sure carry holder is zero
    mov     bl, 1[factor2]          ; Get last digit of multiplier
    mov     cx, 4                   ; Set counter to indicate four digits
                                           ; in multiplicand

@10:
    mov     al, [si]                ; Get a digit
    call    multiply                ; Do the multiplication
    mov     [di], al                ; Save result
    dec     di                      ; Decrement
    dec     si                      ; pointers
    loop    @10                    ; Loop until done
    mov     si, offset factor1+3    ; Get pointers to end of multiplicand
    mov     di, offset temp2+4      ; Get pointer to intermediate storage
    xor     dl, dl                  ; Make sure carry holder is zero
    mov     bl, factor2             ; Get first digit of
    mov     cx, 4                   ; Set counter to indicate four digits
                                           ; in multiplicand

@20:
    mov     al, [si]                ; Get a digit
    call    multiply                ; Do a multiplication
    mov     [di], al                ; Save the result
    dec     di                      ; Decrement
    dec     si                      ; pointers
    loop    @20                    ; Loop until done
    xor     ax, ax                  ; Set accumulator to 0
    mov     cx, 6                   ; Set counter to sweep through
                                           ; the intermediate results

    mov     bx, offset product+5    ; Point to the end of final storage
    mov     si, offset temp1+5      ; Point to the end of first result
    mov     di, offset temp2+5      ; Point to the end of second result

@30:
    mov     al, [si]                ; Get a digit
    add     al, ah                  ; Add carry from last operation if any
    xor     ah, ah                  ; Zero carry holder
    aaa                                ; Adjust for addion
    add     al, [di]                ; Get a digit from second number
    add     al, ah                  ; Add carry from last addition if any
    xor     ah, ah                  ; Zero carry holder
    aaa                                ; Adjust for addition
    mov     [bx], al                ; Move to final storage
    dec     si                      ; Decrement
    dec     di                      ; all
    dec     bx                      ; pointers
    loop    @30                    ; Loop until done
    mov     ax, 4c00h               ; Exit
    int     21h                    ; gracefully
main       endp

multiply   proc

```

```

        xor    ah, ah                ; Set high byte to 0
        mul   bl                    ; Multiply AX * BL
        aam                               ; Adjust
        add   al, dl                ; Add carry from previous
        mov   dl, ah                ; Save carry from current
        ret
multiply endp

end main

```

Ex 4: ASCII Multiplication Program

Title Chapt 7 (7.8.4), Ex 4: ASCII Multiplication Program

```

; This program demonstrates a procedure that multiplies
; two 20-digit ASCII numbers and displays the product.
; The procedure is called Multiply_Integers.

```

```

; Level: Advanced
; Solution by Kip Irvine.

```

```

DIGIT_COUNT = 20
DIGITS_PRODUCT = DIGIT_COUNT * 2

```

```

.model small
.stack 100h
.286
.data
num1 db    "65438700000000000000"    ; number is backwards
num2 db    "52260000000000000000"    ; number is backwards
product db DIGITS_PRODUCT dup(0)

msg1 db    "First number: ",0
msg2 db    "Second number: ",0
msg3 db    "Product: ",0
strTitle db "Multiplying two large integers",0dh,0ah
          db 0dh,0ah,0

.code
include library.inc

main proc
    mov ax,@data        ; init data segment
    mov ds,ax
    call OpeningScreen

    mov si,offset num1
    mov di,offset num2
    mov bx,offset product
    call Multiply_Integers ; multiply & display product

    mov ax,4c00h        ; end program
    int 21h
main endp

```

```

OpeningScreen proc
    call ClrScr
    mov dx,offset strTitle
    call Writestring
    mov dx,offset msg1
    call Writestring

    mov cx,DIGIT_COUNT
    mov bx,offset num1
    call DisplayNumber

    mov dx,offset msg2
    call Writestring

    mov bx,offset num2
    call DisplayNumber

    mov dx,offset msg3
    call Writestring
    ret
OpeningScreen endp

; ----- Multiply_Integers -----
; Multiply two multi-digit ASCII integers.
; SI & DI point to the two numbers, and BX points
; to the product.

Multiply_Integers proc
.data
productPtr dw ?
tempProduct db DIGITS_PRODUCT dup(0)
.code
    mov productPtr,bx
    call ConvertToBinary ; convert all digits to binary

    mov cx,DIGIT_COUNT
    mov bx,offset tempProduct
L2:
    push cx
    call ZeroTempProduct
    call MultiplyOneDigit ; [SI] * [DI] --> [BX]
    call AddToProduct
    inc di
    inc bx
    pop cx
    Loop L2

; Display the product (pointed to by BX).
    mov bx,productPtr
    mov cx,DIGITS_PRODUCT
    call DisplayNumber

    ret
Multiply_Integers endp

ZeroTempProduct proc

```



```

    add    al,carry        ; add carry from previous loop, if any
    mov    dl,10          ; divide AX by 10
    div    dl              ; AL = quotient, AH = remainder
    mov    [bx],ah        ; store remainder
    mov    carry,al       ; save the carry value
    inc    si              ; point to next num1 digit
    inc    bx
    loop   MOD1           ; go to next digit
    popa
    ret
MultiplyOneDigit endp

; ----- DisplayNumber -----
; Display the digit string pointed to by BX,
; backwards. Size is in CX.

DisplayNumber proc
    pusha
    add    bx,cx          ; point to end of number
    dec    bx

DR1:
    mov    dl,[bx]       ; get digit
    or     dl,30h        ; convert to ASCII
    mov    ah,2          ; show on console
    int    21h
    dec    bx            ; display backwards
    Loop  DR1
    call  Crlf

    popa
    ret
DisplayNumber endp

; ----- ConvertToBinary -----
; Convert all ASCII decimal digits to binary,
; in the operands pointed to by SI and DI.

ConvertToBinary proc
    push  si
    push  di
    mov   cx,DIGIT_COUNT
L1: and  byte ptr [si],11001111b
    and  byte ptr [di],11001111b
    inc  si
    inc  di
    Loop L1
    pop  di
    pop  si
    ret
ConvertToBinary endp
end main

```

Ex 7: The DIV32 Procedure

```
title Ch 7 (7.8.4), Ex 7: The DIV32 Procedure

; Solution to Chapter 7, Exercise 10.

.model small
.286
public div32

.data
    divisor    dw 342
    dividend   dd 162134121
    remainder  dw ?

.code
div32 proc
    mov  ax, @data
    mov  ds, ax
    mov  si, offset dividend
    mov  ax, [si + 2]
    cwd
    mov  cx, divisor
    div  cx
    mov  bx, ax
    mov  ax, [si]
    div  cx
    mov  remainder, dx
    ret
div32 endp
end
```

7.8.5 Direct Video Output

Ex 1: Testing Character Output

```
Title  Chapt 7 (7.8.5), Ex 1: Testing Character Output

; Write a procedure that writes a character to every position
; on the screen, using the Writechar_direct procedure from the
; book's link library. Write another procedure that does the
; same thing using INT 10h.

.model small
.286
.stack 100h

.code
include library.inc

main proc
    mov  ax,@data           ; init data segment
    mov  ds,ax
    call ClrScr

    call FillScreenDirect
```

```
        call FillScreenBios

        mov  ax,4c00h          ; end program
        int  21h

main endp

FillScreenDirect proc
    pusha
        mov  cx,25             ; 25 rows
        mov  dh,0             ; starting row

L1:
    push cx
        mov  cx,80
        mov  dl,0             ; starting column
L2:
    mov  al,'D'               ; character
    mov  ah,0Fh               ; attribute
    call Writechar_direct
    inc  dl                    ; increment column
    loop L2

    pop  cx
    inc  dh                    ; increment row
    loop L1

    popa
    ret
FillScreenDirect endp

FillScreenBios proc
    pusha

        mov  cx,25             ; 25 rows
        mov  dh,0             ; starting row

FS1:
    push cx
        mov  cx,80
        mov  dl,0             ; starting column
FS2:
    mov  al,'B'               ; character
    mov  ah,0Fh               ; attribute
    call Writechar_bios
    inc  dl                    ; increment column
    loop FS2

    pop  cx
    inc  dh                    ; increment row
    loop FS1

    popa
    ret
```

```

FillScreenBios endp

; DH,DL = row, column. AL = character, AH = attribute

Writechar_bios proc
    pusha
    mov  char,al
    mov  attrib,ah

    mov  ah,2
    mov  bh,0
    int  10h

    mov  ah,9
    mov  al,char
    mov  bh,0           ; video page 0
    mov  bl,attrib
    mov  cx,1           ; character count
    int  10h

    popa
    ret

.data
char  db ?
attrib db ?
.code
Writechar_bios endp
end main

```

Ex 2: Optimize the Writestring_direct Procedure

```

Title  Chapt 7 (7.8.5), Ex 2: Optimize Writestring_direct

; Incorporate the Writechar_direct procedure into
; Writestring_direct to make the latter more efficient.

.model small
.stack 100h
videoSegment = 0B800h

.data
message  db "Display this string",0
attribute db 1Bh

.code
extrn Clrscr:proc

main proc
    mov  ax,@data           ; init data segment
    mov  ds,ax
    call ClrScr

    mov  ah,attribute

```

```

    mov si,offset message
    mov dh,10           ; row
    mov dl,10          ; column
    call Writestring_direct ; modified version

    mov ax,4c00h       ; end program
    int 21h
main endp

; Writestring_direct -----
;
; Write a string directly to video RAM. Input
; parameters: DS:SI points to a null-terminated
; string, AH = attribute, DH/DL = row/column on
; the screen.
;-----

Writestring_direct proc
    push ax
    push dx
    push si

    mov di,videoSegment
    mov es,di

    ; multiply the row by 160
    push ax
    mov ax,160
    mul dh
    mov di,ax

    ; multiply the column by 2, add to DI
    shl dl,1
    mov dh,0
    add di,dx
    pop ax

L1:
    mov al,[si]        ; get character
    cmp al,0           ; check for null byte
    je L2              ; quit if found

    mov es:[di],ax     ; store char/attribute

    inc si             ; next character
    add di,2           ; next screen location
    inc dl             ; next column
    jmp L1

L2:
    pop si
    pop dx
    pop ax
    ret
Writestring_direct endp

```

```

Writechar_direct proc
    push ax
    push dx
    push di
    push es

    mov di,videoSegment
    mov es,di

    ; multiply the row by 160
    push ax
    mov ax,160
    mul dh
    mov di,ax

    ; multiply the column by 2, add to DI
    shl dl,1
    mov dh,0
    add di,dx
    pop ax

    mov es:[di],ax      ; store char/attribute

    pop es
    pop di
    pop dx
    pop ax
    ret
Writechar_direct endp
end main

```

Ex 3: Calculating the Row Offset in Writechar_direct

```

Title Chapt 7 (7.8.5), Ex 1:
;Calculating the Row offset in Writechar_direct

; Implement the Writechar_direct procedure using
; shift instructions to calculate the row offset.

.model small
.286
.stack 100h

videoSegment = 0B800h

.code
include library.inc

main proc
    mov ax,@data      ; init data segment
    mov ds,ax
    call ClrScr
    mov cx,24
    mov dh,0          ; starting row

```

```

L1:
    push cx
    mov  cx,80
    mov  dl,0                ; starting column
L2:
    mov  al,'A'            ; character
    mov  ah,0Fh           ; attribute
    call WritecharDirect  ; new version
    inc  dl                ; increment column
    loop L2

    pop  cx
    inc  dh                ; increment row
    loop L1

    mov  ax,4c00h         ; end program
    int  21h
main endp

; Write a character directly to VRAM. Input
; parameters: AL = char, AH = attribute,
; DH/DL = row, column on screen (0-24, 0-79).

WritecharDirect proc
    push ax
    push dx
    push di
    push es

    mov  di,videoSegment
    mov  es,di

; Alternate method of calculating the row offset,
; using SHL with powers of 2:
; Note that (x * 160) can be expressed as
; (x * 128) + (x * 32). This
; in turn is, SHL 7 followed by SHL 5.

    push ax
    mov  al,dh            ; row
    xor  ah,ah
    shl  ax,7            ; row * 128
    mov  di,ax           ; DI = row * 128
    mov  al,dh            ; row
    xor  ah,ah
    shl  ax,5            ; row * 32
    add  di,ax           ; DI = (row * 128) + (row * 32)

; multiply the column by 2, add to DI
    shl  dl,1
    mov  dh,0
    add  di,dx
    pop  ax

    mov  es:[di],ax      ; store char/attribute

```

```
    pop  es
    pop  di
    pop  dx
    pop  ax
    ret
WritecharDirect endp
end main
```