```
Title Irvine32 Link Library Source Code         (Irvine32.asm)

Comment @
To view this file with proper indentation, set your
    editor's tab stops to columns 5, 11, 35, and 40.

Recent Updates:
06/04/05: WaitMsg simplified
06/08/05: CreateOutputFile, WriteToFile, OpenInputFile, ReadFromFile, CloseFile
06/10/05: WriteWindowsMsg
06/13/05: GetCommmandTail
06/21/05: SetTextColor, GetTextColor
06/22/05: DumpRegs
06/05/05: ReadChar
07/06/05: ReadFromFile
07/11/05: MsgBox, MsgBoxAsk
07/15/05: ParseDecimal32, ParseInteger32
07/19/05: ParseDecimal32, ParseInteger32, ReadHex
07/24/05: WriteStackFrame, WriteStackFrameName (James Brink)
06/12/08: Str_trim

This library was created exlusively for use with the book,
"Assembly Language for Intel-Based Computers", 4th Edition & 5th Edition,
by Kip R. Irvine, 2002-2008.

Copyright 2002-2008, Prentice-Hall Publishing. No part of this file may be
reproduced, in any form or by any other means, without permission in writing
from the author or publisher.

Acknowledgements:
------------------------------
Most of the code in this library was written by Kip Irvine.
Special thanks to Gerald Cahill for his many insights, suggestions, and bug fixes.
Thanks to Richard Stam for his development of Readkey-related procedures.
Thanks to James Brink for helping to test the library.

Alphabetical Listing of Public Procedures
---------------------------------
(Unless otherwise marked, all procedures are documented in Chapter 5.)

CloseFile
Clrscr
CreateOutputFile
Crlf
Delay
DumpMem
DumpRegs
GetCommandTail
GetDateTime Chapter 11
GetMaxXY
GetMseconds
GetTextColor
Gotoxy
IsDigit
MsgBox
MsgBoxAsk
OpenInputFile
ParseDecimal32
ParseInteger32
Random32
Randomize
RandomRange
ReadChar
ReadDec
ReadFromFile
ReadHex
ReadInt
ReadKey
ReadKeyFlush
ReadString
SetTextColor
Str_compare Chapter 9
Str_copy        Chapter 9
Str_length  Chapter 9
```

```
Str_trim        Chapter 9
Str_ucase       Chapter 9
WaitMsg
WriteBin
WriteBinB
WriteChar
WriteDec
WriteHex
WriteHexB
WriteInt
WriteStackFrame Chapter 8  (James Brink)
WriteStackFrameName Chapter 8  (James Brink)
WriteString
WriteToFile
WriteWindowsMsg

              Implementation Notes:
              ---------------------
1. The Windows Sleep function modifies the contents of ECX.
2. Remember to save and restore all 32-bit general purpose
   registers (except EAX) before calling MS-Windows API functions.
-------------------------------------------------------------------------@
;OPTION CASEMAP:NONE    ; optional: force case-sensitivity

INCLUDE Irvine32.inc    ; function prototypes for this library
INCLUDE Macros.inc  ; macro definitions


;************************************************************
;*                          MACROS                          *
;************************************************************

;-----------------------------------------------------------------------
ShowFlag MACRO flagName,shiftCount
         LOCAL flagStr, flagVal, L1
;
; Helper macro.
; Display a single CPU flag value
; Directly accesses the eflags variable in Irvine16.asm/Irvine32.asm
; (This macro cannot be placed in Macros.inc)
;-----------------------------------------------------------------------

.data
flagStr DB "  &flagName="
flagVal DB ?,0

.code
    push eax
    push edx

    mov  eax,eflags ; retrieve the flags
    mov  flagVal,'1'
    shr  eax,shiftCount ; shift into carry flag
    jc   L1
    mov  flagVal,'0'
L1:
    mov  edx,OFFSET flagStr ; display flag name and value
    call WriteString

    pop  edx
    pop  eax
ENDM

;-------------------------------------------------------------
CheckInit MACRO
;
; Helper macro
; Check to see if the console handles have been initialized
; If not, initialize them now.
;-------------------------------------------------------------
LOCAL exit
    cmp InitFlag,0
    jne exit
    call Initialize
```

```
exit:
ENDM

;*************************************************************
;*                      SHARED DATA                          *
;*************************************************************

MAX_DIGITS = 80

.data       ; initialized data
InitFlag DB 0   ; initialization flag
xtable BYTE "0123456789ABCDEF"

.data?      ; uninitialized data
consoleInHandle  DWORD ?        ; handle to console input device
consoleOutHandle DWORD ?        ; handle to standard output device
bytesWritten     DWORD ?        ; number of bytes written
eflags  DWORD ?
digitBuffer BYTE MAX_DIGITS DUP(?),?

buffer DB 512 DUP(?)
bufferMax = ($ - buffer)
bytesRead DD ?
sysTime SYSTEMTIME <>   ; system time structure


;*************************************************************
;*                    PUBLIC PROCEDURES                      *
;*************************************************************

.code

;-----------------------------------------------------
CloseFile PROC
;
; Closes a file using its handle as an identifier.
; Receives: EAX = file handle
; Returns: EAX = nonzero if the file is successfully
;   closed.
; Last update: 6/8/2005
;-----------------------------------------------------

    INVOKE CloseHandle, eax
    ret
CloseFile ENDP


;-------------------------------------------------------------
Clrscr PROC
    LOCAL bufInfo:CONSOLE_SCREEN_BUFFER_INFO
;
; Clear the screen by writing blanks to all positions
; Receives: nothing
; Returns: nothing
; Last update: 10/15/02
;
; The original version of this procedure incorrectly assumed  the
; console window dimensions were 80 X 25 (the default MS-DOS screen).
; This new version writes both blanks and attribute values to each
; buffer position. Restriction: Only the first 512 columns of each
; line are cleared. The name capitalization was changed to "Clrscr".
;-------------------------------------------------------------

MAX_COLS = 512
.data
blanks BYTE MAX_COLS DUP(' ')           ; one screen line
attribs WORD MAX_COLS DUP(0)
lineLength DWORD 0

cursorLoc COORD <0,0>
count DWORD ?

.code
    pushad
```

```
    CheckInit

    ; Get the console buffer size and attributes
    INVOKE GetConsoleScreenBufferInfo, consoleOutHandle, ADDR bufInfo
    mov ax,bufInfo.dwSize.X;
    mov WORD PTR lineLength,ax
    .IF lineLength > MAX_COLS
      mov lineLength,MAX_COLS
    .ENDIF

    ; Fill the attribs array
    mov ax,bufInfo.wAttributes
    mov ecx,lineLength
    mov edi,OFFSET attribs
    rep stosw

    movzx ecx,bufInfo.dwSize.Y      ; loop counter: number of lines
L1: push ecx

    ; Write a blank line to the screen buffer
    INVOKE WriteConsoleOutputCharacter,
    consoleOutHandle,
    ADDR blanks,        ; pointer to buffer
    lineLength,     ; number of blanks to write
    cursorLoc,      ; first cell coordinates
    ADDR count      ; output count

    ; Fill all buffer positions with the current attribute
    INVOKE WriteConsoleOutputAttribute,
      consoleOutHandle,
      ADDR attribs,     ; point to attribute array
      lineLength,       ; number of attributes to write
      cursorLoc,        ; first cell coordinates
      ADDR count        ; output count

    add cursorLoc.Y, 1      ; point to the next buffer line
    pop ecx
    Loop L1

    ; Move cursor to 0,0
    mov cursorLoc.Y,0
    INVOKE SetConsoleCursorPosition, consoleOutHandle, cursorLoc

    popad
    ret
Clrscr ENDP


;-----------------------------------------------------
CreateOutputFile PROC
;
; Creates a new file and opens it in output mode.
; Receives: EDX points to the filename.
; Returns: If the file was created successfully, EAX
;   contains a valid file handle. Otherwise, EAX
;   equals INVALID_HANDLE_VALUE.
;-----------------------------------------------------
    INVOKE CreateFile,
      edx, GENERIC_WRITE, DO_NOT_SHARE, NULL,
      CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0
    ret
CreateOutputFile ENDP


;-----------------------------------------------------
Crlf PROC
;
; Writes a carriage return / linefeed
; sequence (0Dh,0Ah) to standard output.
;-----------------------------------------------------
    CheckInit
    mWrite <0dh,0ah>    ; invoke a macro
    ret
Crlf ENDP
```

```
;-----------------------------------------------------
Delay PROC
;
; THIS FUNCTION IS NOT IN THE IRVINE16 LIBRARY
; Delay (pause) the current process for a given number
; of milliseconds.
; Receives: EAX = number of milliseconds
; Returns: nothing
; Last update: 7/11/01
;-----------------------------------------------------

    pushad
    INVOKE Sleep,eax
    popad
    ret

Delay ENDP


;-----------------------------------------------------
DumpMem PROC
         LOCAL unitsize:dword, byteCount:word
;
; Writes a range of memory to standard output
; in hexadecimal.
; Receives: ESI = starting offset, ECX = number of units,
;           EBX = unit size (1=byte, 2=word, or 4=doubleword)
; Returns:  nothing
; Last update: 7/11/01
;-----------------------------------------------------
.data
oneSpace   DB ' ',0

dumpPrompt DB 13,10,"Dump of offset ",0
dashLine   DB "--------------------------------",13,10,0

.code
    pushad

    mov  edx,OFFSET dumpPrompt
    call WriteString
    mov  eax,esi    ; get memory offset to dump
    call  WriteHex
    call    Crlf
    mov  edx,OFFSET dashLine
    call WriteString

    mov  byteCount,0
    mov  unitsize,ebx
    cmp  ebx,4  ; select output size
    je   L1
    cmp  ebx,2
    je   L2
    jmp  L3

    ; 32-bit doubleword output
L1:
    mov  eax,[esi]
    call WriteHex
    mWriteSpace 2
    add  esi,ebx
    Loop L1
    jmp  L4

    ; 16-bit word output
L2:
    mov  ax,[esi]   ; get a word from memory
    ror  ax,8   ; display high byte
    call HexByte
    ror  ax,8   ; display low byte
    call HexByte
    mWriteSpace 1   ; display 1 space
```

```
    add  esi,unitsize   ; point to next word
    Loop L2
    jmp  L4

    ; 8-bit byte output, 16 bytes per line
L3:
    mov  al,[esi]
    call HexByte
    inc  byteCount
    mWriteSpace 1
    inc  esi

    ; if( byteCount mod 16 == 0 ) call Crlf

    mov  dx,0
    mov  ax,byteCount
    mov  bx,16
    div  bx
    cmp  dx,0
    jne  L3B
    call    Crlf
L3B:
    Loop L3
    jmp  L4

L4:
    call    Crlf
    popad
    ret
DumpMem ENDP


;-----------------------------------------------------
DumpRegs PROC
;
; Displays EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP in
; hexadecimal. Also displays the Zero, Sign, Carry, and
; Overflow flags.
; Receives: nothing.
; Returns: nothing.
; Last update: 6/22/2005
;
; Warning: do not create any local variables or stack
; parameters, because they will alter the EBP register.
;-----------------------------------------------------
.data
saveIP  DWORD ?
saveESP DWORD ?
.code
    pop saveIP  ; get current EIP
    mov saveESP,esp ; save ESP's value at entry
    push saveIP ; replace it on stack
    push eax    ; save EAX (restore on exit)

    pushfd  ; push extended flags

    pushfd  ; push flags again, and
    pop  eflags ; save them in a variable

    call    Crlf
    mShowRegister EAX,EAX
    mShowRegister EBX,EBX
    mShowRegister ECX,ECX
    mShowRegister EDX,EDX
    call    Crlf
    mShowRegister ESI,ESI
    mShowRegister EDI,EDI

    mShowRegister EBP,EBP

    mov eax,saveESP
    mShowRegister ESP,EAX
    call    Crlf
```

```
    mov eax,saveIP
    mShowRegister EIP,EAX
    mov eax,eflags
    mShowRegister EFL,EAX

; Show the flags (using the eflags variable). The integer parameter indicates
; how many times EFLAGS must be shifted right to shift the selected flag
; into the Carry flag.

    ShowFlag CF,1
    ShowFlag SF,8
    ShowFlag ZF,7
    ShowFlag OF,12
    ShowFlag AF,5
    ShowFlag PF,3

    call    Crlf
    call    Crlf

    popfd
    pop eax
    ret
DumpRegs ENDP


;-----------------------------------------------------------
GetCommandTail PROC
;
; Copies the tail of the program command line into a buffer
; (after stripping off the first argument - the program's name)
; Receives: EDX points to a 129-byte buffer that will receive
; the data.
; Returns: Carry Flag = 1 if no command tail, otherwise CF=0
;
; Calls the WIN API function GetCommandLine, and scan_for_quote,
; a private helper procedure. Each argument in the command line tail
; is followed by a space except for the last argument which is
; followed only by null.
;
; Implementation notes:
;
; Running in a console window:
; When the command line is blank, GetCommandLine under Windows 95/98
; returns the program name followed by a space and a null. Windows 2000/XP
; returns the program name followed by only null (the space is omitted).
;
; Running from an IDE such as TextPad or JCreator:
; When the command line is blank, GetCommandLine returns the program
; name followed by a space and a null for all versions of Windows.
;
; Contributed by Gerald Cahill, 9/26/2002
; Modified by Kip Irvine, 6/13/2005.
;-----------------------------------------------------------

QUOTE_MARK = 22h

    pushad
    INVOKE GetCommandLine       ; returns pointer in EAX

; Initialize first byte of user's buffer to null, in case the
; buffer already contains text.

    mov BYTE PTR [edx],0

; Copy the command-line string to the array. Read past the program's
; EXE filename (may include the path). This code will not work correctly
; if the path contains an embedded space.

    mov esi,eax
L0: mov al,[esi]        ; strip off first argument
    inc esi
    .IF al == QUOTE_MARK    ; quotation mark found?
    call    scan_for_quote  ; scan until next quote mark
    jmp LB  ; and get the rest of the line
```

```
    .ENDIF
    cmp al,' '          ; look for blank
    je  LB  ; found it
    cmp al,1    ; look for null
    jc  L2  ; found it (set CF=1)
    jmp L0  ; not found yet

; Check if the rest of the tail is empty.

LB: cmp BYTE PTR [esi],1    ; first byte in tail < 1?
    jc  L2  ; the tail is empty (CF=1)

; Copy all bytes from the command tail to the buffer.

L1: mov al,[esi]    ; get byte from cmd tail
    mov [edx],al    ; copy to buffer
    inc esi
    inc edx
    cmp al,0        ; null byte found?
    jne L1              ; no, loop

    clc     ; CF=0 means a tail was found

L2: popad
    ret
GetCommandTail ENDP


;-----------------------------------------------------------
scan_for_quote PROC PRIVATE
;
; Helper procedure that looks for a closing quotation mark. This
; procedure lets us handle path names with embedded spaces.
; Called by: GetCommandTail
;
; Receives: ESI points to the current position in the command tail.
; Returns: ESI points one position beyond the quotation mark.
;-----------------------------------------------------------

L0: mov al,[esi]        ; get a byte
    inc esi ; point beyond it
    cmp al,QUOTE_MARK   ; quotation mark found?
    jne L0  ; not found yet

    ret
scan_for_quote ENDP


;-------------------------------------------------
GetDateTime PROC,
    pDateTime:PTR QWORD
    LOCAL flTime:FILETIME
;
; Gets the current local date and time, storing it as a
; 64-bit integer (Win32 FILETIME format) in memory at
; the address specified by the input parameter.
; Receives: pointer to a QWORD variable (inout parameter)
; Returns: nothing
; Updated 10/20/2002
;-------------------------------------------------
    pushad

; Get the system local time.
    INVOKE GetLocalTime,
      ADDR sysTime

; Convert the SYSTEMTIME to FILETIME.
    INVOKE SystemTimeToFileTime,
      ADDR sysTime,
      ADDR flTime

; Copy the FILETIME to a Quadword.
    mov esi,pDateTime
    mov eax,flTime.loDateTime
```

```
    mov DWORD PTR [esi],eax
    mov eax,flTime.hiDateTime
    mov DWORD PTR [esi+4],eax

    popad
    ret
GetDateTime ENDP


;-----------------------------------------------------------------
GetMaxXY PROC
    LOCAL bufInfo:CONSOLE_SCREEN_BUFFER_INFO
;
; Returns the current columns (X) and rows (Y) of the console
; window buffer. These values can change while a program is running
; if the user modifies the properties of the application window.
; Receives: nothing
; Returns: DH = rows (Y); DL = columns (X)
; (range of each is 1-255)
;
; Added to the library on 10/20/2002, on the suggestion of Ben Schwartz.
;-----------------------------------------------------------------
    push eax
    CheckInit

    ; Get the console buffer size and attributes
    pushad
    INVOKE GetConsoleScreenBufferInfo, consoleOutHandle, ADDR bufInfo
    popad

    mov dx,bufInfo.dwSize.X
    mov ax,bufInfo.dwSize.Y
    mov dh,al

    pop eax
    ret
GetMaxXY ENDP


;-----------------------------------------------------------------
GetMseconds PROC USES ebx edx
    LOCAL hours:DWORD, min:DWORD, sec:DWORD
;
Comment !
Returns the number of milliseconds that have elapsed past midnight.
Receives: nothing; Returns: milliseconds
Implementation Notes:
Calculation: ((hours * 3600) + (minutes * 60) + seconds)) * 1000 + milliseconds
Under Win NT/ 2000/ XT, the resolution is 10ms.  Under Win 98/ ME/ or any
DOS-based version, the resolution is 55ms (average).

Last update: 1/30/03
-----------------------------------------------------------------!
    pushad
    INVOKE GetLocalTime,OFFSET sysTime
    ; convert hours to seconds
    popad
    movzx eax,sysTime.wHour
    mov   ebx,3600
    mul   ebx
    mov   hours,eax

    ; convert minutes to seconds
    movzx eax,sysTime.wMinute
    mov   ebx,60
    mul   ebx
    mov   min,eax

    ; add seconds to total seconds
    movzx eax,sysTime.wSecond
    mov  sec,eax

    ; multiply seconds by 1000
    mov  eax,hours
```

```
    add  eax,min
    add  eax,sec
    mov  ebx,1000
    mul  ebx

    ; add milliseconds to total
    movzx ebx,sysTime.wMilliseconds
    add  eax,ebx

    ret
GetMseconds ENDP


;-----------------------------------------------------
GetTextColor PROC
    LOCAL bufInfo:CONSOLE_SCREEN_BUFFER_INFO
;
;
; Get the console window's color attributes.
; Receives: nothing
; Returns: AH = background color, AL = foreground
;   color
;-----------------------------------------------------

    pushad
    CheckInit

    ; Get the console buffer size and attributes
    INVOKE GetConsoleScreenBufferInfo, consoleOutHandle, ADDR bufInfo
    popad

    mov  ax,bufInfo.wAttributes
    ret
GetTextColor ENDP


;-----------------------------------------------------
Gotoxy PROC
;
; Locate the cursor
; Receives: DH = screen row, DL = screen column
; Last update: 7/11/01
;-----------------------------------------------------
.data
_cursorPosition COORD <>
.code
    pushad

    CheckInit   ; was console initialized?
    movzx ax,dl
    mov _cursorPosition.X, ax
    movzx ax,dh
    mov _cursorPosition.Y, ax
    INVOKE SetConsoleCursorPosition, consoleOutHandle, _cursorPosition

    popad
    ret
Gotoxy ENDP


;-----------------------------------------------------
Initialize PROC private
;
; Get the standard console handles for input and output,
; and set a flag indicating that it has been done.
; Updated 03/17/2003
;-----------------------------------------------------
    pushad

    INVOKE GetStdHandle, STD_INPUT_HANDLE
    mov [consoleInHandle],eax

    INVOKE GetStdHandle, STD_OUTPUT_HANDLE
    mov [consoleOutHandle],eax
```

```
    mov InitFlag,1

    popad
    ret
Initialize ENDP


;-----------------------------------------------
IsDigit PROC
;
; Determines whether the character in AL is a
; valid decimal digit.
; Receives: AL = character
; Returns: ZF=1 if AL contains a valid decimal
;   digit; otherwise, ZF=0.
;-----------------------------------------------
     cmp   al,'0'
     jb    ID1
     cmp   al,'9'
     ja    ID1
     test  ax,0             ; set ZF = 1
ID1: ret
IsDigit ENDP


;-----------------------------------------------
MsgBox PROC
;
; Displays a popup message box.
; Receives: EDX = offset of message, EBX =
;   offset of caption (or 0 if no caption)
; Returns: nothing
;-----------------------------------------------
.data
@zx02abc_def_caption BYTE " ",0
.code
    pushad

    .IF ebx == 0
      mov  ebx,OFFSET @zx02abc_def_caption
    .ENDIF
    INVOKE MessageBox, 0, edx, ebx, 0

    popad
    ret
MsgBox ENDP


;-------------------------------------------------
MsgBoxAsk PROC uses ebx ecx edx esi edi
;
; Displays a message box with a question icon and
;    Yes/No buttons.
; Receives: EDX = offset of message. For a blank
;   caption, set EBX to NULL; otherwise, EBX = offset
;   of the caption string.
; Returns: EAX equals IDYES (6) or IDNO (7).
;-------------------------------------------------
.data
@zq02abc_def_caption BYTE " ",0
.code
    .IF ebx == NULL
      mov  ebx,OFFSET @zq02abc_def_caption
    .ENDIF
    INVOKE MessageBox, NULL, edx, ebx,
        MB_YESNO + MB_ICONQUESTION

    ret
MsgBoxAsk ENDP


;-----------------------------------------------------
OpenInputFile PROC
```

```
;
; Opens an existing file for input.
; Receives: EDX points to the filename.
; Returns: If the file was opened successfully, EAX
; contains a valid file handle. Otherwise, EAX equals
; INVALID_HANDLE_VALUE.
; Last update: 6/8/2005
;-----------------------------------------------------

    INVOKE CreateFile,
      edx, GENERIC_READ, DO_NOT_SHARE, NULL,
      OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0
    ret
OpenInputFile ENDP


;-----------------------------------------------------
ParseDecimal32 PROC USES ebx ecx edx esi
  LOCAL saveDigit:DWORD
;
; Converts (parses) a string containing an unsigned decimal
; integer, and converts it to binary. All valid digits occurring
; before a non-numeric character are converted.
; Leading spaces are ignored.

; Receives: EDX = offset of string, ECX = length
; Returns:
;  If the integer is blank, EAX=0 and CF=1
;  If the integer contains only spaces, EAX=0 and CF=1
;  If the integer is larger than 2^32-1, EAX=0 and CF=1
;  Otherwise, EAX=converted integer, and CF=0
;
; Created 7/15/05 (from the old ReadDec procedure)
;-----------------------------------------------------

    mov   esi,edx                   ; save offset in ESI

    cmp   ecx,0                 ; length greater than zero?
    jne   L1                    ; yes: continue
    mov   eax,0                 ; no: set return value
    jmp   L5                    ; and exit with CF=1

; Skip over leading spaces, tabs

L1: mov   al,[esi]              ; get a character from buffer
    cmp   al,' '                ; space character found?
    je  L1A     ; yes: skip it
    cmp al,TAB      ; TAB found?
    je  L1A     ; yes: skip it
    jmp   L2                    ; no: goto next step

L1A:
    inc   esi                   ; yes: point to next char
    loop  L1        ; continue searching until end of string
    jmp   L5        ; exit with CF=1 if all spaces

; Replaced code (7/19/05)-----------------------------------------
;L1:mov   al,[esi]              ; get a character from buffer
;   cmp   al,' '                ; space character found?
;   jne   L2                    ; no: goto next step
;   inc   esi                   ; yes: point to next char
;   loop  L1        ; all spaces?
;   jmp   L5        ; yes: exit with CF=1
;-----------------------------------------------------------------

; Start to convert the number.

L2: mov  eax,0                  ; clear accumulator
    mov  ebx,10                 ; EBX is the divisor

; Repeat loop for each digit.

L3: mov  dl,[esi]       ; get character from buffer
    cmp  dl,'0'     ; character < '0'?
```

```
    jb   L4
    cmp  dl,'9'     ; character > '9'?
    ja   L4
    and  edx,0Fh        ; no: convert to binary

    mov  saveDigit,edx
    mul  ebx        ; EDX:EAX = EAX * EBX
    jc   L5     ; quit if Carry (EDX > 0)
    mov  edx,saveDigit
    add  eax,edx                ; add new digit to sum
    jc   L5     ; quit if Carry generated
    inc  esi                    ; point to next digit
    jmp  L3     ; get next digit

L4: clc         ; succesful completion (CF=0)
    jmp  L6

L5: mov  eax,0      ; clear result to zero
    stc         ; signal an error (CF=1)

L6: ret
ParseDecimal32 ENDP


;-----------------------------------------------------
ParseInteger32 PROC USES ebx ecx edx esi
  LOCAL Lsign:SDWORD, saveDigit:DWORD
;
; Converts a string containing a signed decimal integer to
; binary.
;
; All valid digits occurring before a non-numeric character
; are converted. Leading spaces are ignored, and an optional
; leading + or - sign is permitted. If the string is blank,
; a value of zero is returned.
;
; Receives: EDX = string offset, ECX = string length
; Returns:  If CF=0, the integer is valid, and EAX = binary value.
;   If CF=1, the integer is invalid and EAX = 0.
;
; Created 7/15/05, using Gerald Cahill's 10/10/03 corrections.
; Updated 7/19/05, to skip over tabs
;-----------------------------------------------------
.data
overflow_msgL BYTE  " <32-bit integer overflow>",0
invalid_msgL  BYTE  " <invalid integer>",0
.code

    mov   Lsign,1                   ; assume number is positive
    mov   esi,edx                   ; save offset in SI

    cmp   ecx,0                     ; length greater than zero?
    jne   L1                        ; yes: continue
    mov   eax,0                     ; no: set return value
    jmp   L10                       ; and exit

; Skip over leading spaces and tabs.

L1: mov   al,[esi]              ; get a character from buffer
    cmp   al,' '                ; space character found?
    je  L1A     ; yes: skip it
    cmp al,TAB      ; TAB found?
    je  L1A     ; yes: skip it
    jmp   L2                    ; no: goto next step

L1A:
    inc   esi                   ; yes: point to next char
    loop  L1        ; continue searching until end of string
    mov eax,0       ; all spaces?
    jmp   L10       ; return 0 as a valid value

;-- Replaced code (7/19/05)-----------------------------------
;L1:    mov   al,[esi]                 ; get a character from buffer
;   cmp   al,' '                   ; space character found?
```

```
;   jne   L2                        ; no: check for a sign
;   inc   esi                       ; yes: point to next char
;   loop  L1
;   mov   eax,0   ; all spaces?
;   jmp   L10     ; return zero as valid value
;-----------------------------------------------------------------

; Check for a leading sign.

L2: cmp   al,'-'                    ; minus sign found?
    jne   L3                        ; no: look for plus sign

    mov   Lsign,-1                  ; yes: sign is negative
    dec   ecx                       ; subtract from counter
    inc   esi                       ; point to next char
    jmp   L3A

L3: cmp   al,'+'                    ; plus sign found?
    jne   L3A                           ; no: skip
    inc   esi                       ; yes: move past the sign
    dec   ecx                       ; subtract from digit counter

; Test the first digit, and exit if nonnumeric.

L3A: mov  al,[esi]              ; get first character
    call IsDigit                ; is it a digit?
    jnz  L7A                    ; no: show error message

; Start to convert the number.

L4: mov   eax,0                     ; clear accumulator
    mov   ebx,10                  ; EBX is the divisor

; Repeat loop for each digit.

L5: mov  dl,[esi]               ; get character from buffer
    cmp  dl,'0'                 ; character < '0'?
    jb   L9
    cmp  dl,'9'                 ; character > '9'?
    ja   L9
    and  edx,0Fh                ; no: convert to binary

    mov  saveDigit,edx
    imul ebx                ; EDX:EAX = EAX * EBX
    mov  edx,saveDigit

    jo   L6                 ; quit if overflow
    add  eax,edx                ; add new digit to AX
    jo   L6                     ; quit if overflow
    inc  esi                    ; point to next digit
    jmp  L5                     ; get next digit

; Overflow has occured, unlesss EAX = 80000000h
; and the sign is negative:

L6: cmp  eax,80000000h
    jne  L7
    cmp  Lsign,-1
    jne  L7                     ; overflow occurred
    jmp  L9                     ; the integer is valid

; Choose "integer overflow" messsage.

L7: mov  edx,OFFSET overflow_msgL
    jmp  L8

; Choose "invalid integer" message.

L7A:
    mov  edx,OFFSET invalid_msgL

; Display the error message pointed to by EDX, and set the Overflow flag.

L8: call WriteString
```

```
    call Crlf
    mov al,127
    add al,1                    ; set Overflow flag
    mov  eax,0                  ; set return value to zero
    jmp  L10                    ; and exit

; IMUL leaves the Sign flag in an undeterminate state, so the OR instruction
; determines the sign of the iteger in EAX.
L9: imul Lsign                      ; EAX = EAX * sign
    or eax,eax                  ; determine the number's Sign

L10:ret
ParseInteger32 ENDP


;-----------------------------------------------------------
Random32  PROC
;
; Generates an unsigned pseudo-random 32-bit integer
;   in the range 0 - FFFFFFFFh.
; Receives: nothing
; Returns: EAX = random integer
; Last update: 7/11/01
;-----------------------------------------------------------
.data
seed  DWORD 1
.code
      push  edx
      mov   eax, 343FDh
      imul  seed
      add   eax, 269EC3h
      mov   seed, eax     ; save the seed for the next call
      ror   eax,8         ; rotate out the lowest digit (10/22/00)
      pop   edx

      ret
Random32  ENDP


;-----------------------------------------------------------
RandomRange PROC
;
; Returns an unsigned pseudo-random 32-bit integer
; in EAX, between 0 and n-1. Input parameter:
; EAX = n.
; Last update: 09/06/2002
;-----------------------------------------------------------
     push  ebx
     push  edx

     mov   ebx,eax  ; maximum value
     call  Random32 ; eax = random number
     mov   edx,0
     div   ebx      ; divide by max value
     mov   eax,edx  ; return the remainder

     pop   edx
     pop   ebx

     ret
RandomRange ENDP


;-----------------------------------------------------
Randomize PROC
;
; Re-seeds the random number generator with the current time
; in seconds.
; Receives: nothing
; Returns: nothing
; Last update: 09/06/2002
;-----------------------------------------------------
      pushad
```

```
      INVOKE GetSystemTime,OFFSET sysTime
      movzx eax,sysTime.wMilliseconds
      mov   seed,eax

      popad
      ret
Randomize ENDP


;-----------------------------------------------------------
ReadChar PROC USES ebx edx
;
; Reads one character from the keyboard. The character is
; not echoed on the screen. Waits for the character if none is
; currently in the input buffer.
; Returns:  AL = ASCII code, AH = scan code
; Last update: 7/6/05
;-----------------------------------------------------------

L1: mov  eax,10 ; give Windows 10ms to process messages
    call Delay
    call ReadKey    ; look for key in buffer
    jz   L1 ; no key in buffer if ZF=1

    ret
ReadChar ENDP


;-----------------------------------------------------------
ReadDec PROC USES ecx edx
;
; Reads a 32-bit unsigned decimal integer from the keyboard,
; stopping when the Enter key is pressed.All valid digits occurring
; before a non-numeric character are converted to the integer value.
; Leading spaces are ignored.

; Receives: nothing
; Returns:
;  If the integer is blank, EAX=0 and CF=1
;  If the integer contains only spaces, EAX=0 and CF=1
;  If the integer is larger than 2^32-1, EAX=0 and CF=1
;  Otherwise, EAX=converted integer, and CF=0
;
; Last update: 7/15/05
;-----------------------------------------------------------

    mov   edx,OFFSET digitBuffer
    mov ecx,MAX_DIGITS
    call  ReadString
    mov ecx,eax ; save length

    call    ParseDecimal32  ; returns EAX

    ret
ReadDec ENDP


;-----------------------------------------------------------
ReadFromFile PROC
;
; Reads an input file into a buffer.
; Receives: EAX = file handle, EDX = buffer offset,
;    ECX = number of bytes to read
; Returns: If CF = 0, EAX = number of bytes read; if
;    CF = 1, EAX contains the system error code returned
;    by the GetLastError Win32 API function.
; Last update: 7/6/2005
;-----------------------------------------------------------

    INVOKE ReadFile,
        eax,    ; file handle
        edx,    ; buffer pointer
        ecx,    ; max bytes to read
        ADDR bytesRead, ; number of bytes read
```

```
        0       ; overlapped execution flag
    cmp eax,0   ; failed?
    jne L1  ; no: return bytesRead
    INVOKE GetLastError ; yes: EAX = error code
    stc     ; set Carry flag
    jmp L2

L1: mov eax,bytesRead   ; success
    clc     ; clear Carry flag

L2: ret
ReadFromFile ENDP


;-----------------------------------------------------
ReadHex PROC USES ebx ecx edx esi
;
; Reads a 32-bit hexadecimal integer from the keyboard,
; stopping when the Enter key is pressed.
; Receives: nothing
; Returns: EAX = binary integer value
; Returns:
;  If the integer is blank, EAX=0 and CF=1
;  If the integer contains only spaces, EAX=0 and CF=1
;  Otherwise, EAX=converted integer, and CF=0

; Remarks: No error checking performed for bad digits
; or excess digits.
; Last update: 7/19/05 (skip leading spaces and tabs)
;-----------------------------------------------------
.data
xbtable     BYTE 0,1,2,3,4,5,6,7,8,9,7 DUP(0FFh),10,11,12,13,14,15
numVal      DWORD ?
charVal     BYTE ?

.code
    mov   edx,OFFSET digitBuffer
    mov   esi,edx       ; save in ESI also
    mov   ecx,MAX_DIGITS
    call  ReadString        ; input the string
    mov   ecx,eax                   ; save length in ECX
    cmp   ecx,0                 ; greater than zero?
    jne   B1                    ; yes: continue
    jmp   B8                    ; no: exit with CF=1

; Skip over leading spaces and tabs.

B1: mov   al,[esi]              ; get a character from buffer
    cmp   al,' '                ; space character found?
    je  B1A     ; yes: skip it
    cmp al,TAB      ; TAB found?
    je  B1A     ; yes: skip it
    jmp   B4                    ; no: goto next step

B1A:
    inc   esi                   ; yes: point to next char
    loop  B1        ; all spaces?
    jmp   B8        ; yes: exit with CF=1

;--- Replaced code (7/19/05)-----------------------------------
;B1:    mov   al,[esi]              ; get a character from buffer
;   cmp   al,' '                ; space character found?
;   jne   B4                    ; no: goto next step
;   inc   esi                   ; yes: point to next char
;   loop  B1        ; all spaces?
;   jmp   B8        ; yes: exit with CF=1
;--------------------------------------------------------------

    ; Start to convert the number.

B4: mov  numVal,0       ; clear accumulator
    mov  ebx,OFFSET xbtable     ; translate table

    ; Repeat loop for each digit.
```

```
B5: mov  al,[esi]   ; get character from buffer
    cmp  al,'F' ; lowercase letter?
    jbe  B6 ; no
    and  al,11011111b   ; yes: convert to uppercase

B6: sub  al,30h ; adjust for table
    xlat    ; translate to binary
    mov  charVal,al
    mov  eax,16 ; numVal *= 16
    mul  numVal
    mov  numVal,eax
    movzx eax,charVal   ; numVal += charVal
    add  numVal,eax
    inc  esi    ; point to next digit
    loop B5 ; repeat, decrement counter

B7: mov  eax,numVal ; return valid value
    clc ; CF=0
    jmp  B9

B8: mov  eax,0  ; error: return 0
    stc ; CF=1

B9: ret
ReadHex ENDP


;-----------------------------------------------------
ReadInt PROC USES ecx edx
;
; Reads a 32-bit signed decimal integer from standard
; input, stopping when the Enter key is pressed.
; All valid digits occurring before a non-numeric character
; are converted to the integer value. Leading spaces are
; ignored, and an optional leading + or - sign is permitted.
; All spaces return a valid integer, value zero.

; Receives: nothing
; Returns:  If CF=0, the integer is valid, and EAX = binary value.
;   If CF=1, the integer is invalid and EAX = 0.
;
; Updated: 7/15/05
;-----------------------------------------------------

; Input a signed decimal string.

    mov   edx,OFFSET digitBuffer
    mov   ecx,MAX_DIGITS
    call  ReadString
    mov   ecx,eax   ; save length in ECX

; Convert to binary (EDX -> string, ECX = length)

    call    ParseInteger32  ; returns EAX, CF

    ret
ReadInt ENDP


;-------------------------------------------------------------------------------
ReadKey PROC USES ecx
    LOCAL evEvents:DWORD, saveFlags:DWORD
;
; Performs a no-wait keyboard check and single character read if available.
; If Ascii is zero, special keys can be processed by checking scans and VKeys
; Receives: nothing
; Returns:  ZF is set if no keys are available, clear if we have read the key
;   al  = key Ascii code (is set to zero for special extended codes)
;   ah  = Keyboard scan code (as in inside cover of book)
;   dx  = Virtual key code
;   ebx = Keyboard flags (Alt,Ctrl,Caps,etc.)
; Upper halves of EAX and EDX are overwritten
;
```

```
; ** Note: calling ReadKey prevents Ctrl-C from being used to terminate a program.
;
; Written by Richard Stam, used by permission.
; Modified 4/6/03 by Irvine; modified 4/16/03 by Jerry Cahill
; ; 6/21/05, Irvine: changed evEvents from WORD to DWORD
;-----------------------------------------------------------------------------
.data
evBuffer INPUT_RECORD <>    ; Buffers our key "INPUT_RECORD"
evRepeat WORD  0    ; Controls key repeat counting

.code
    CheckInit   ; call Inititialize, if not already called

    ; Save console flags
    INVOKE GetConsoleMode,consoleInHandle,ADDR saveFlags

    ; Clear console flags, making it possible to detect Ctrl-C and Ctrl-S.
    INVOKE SetConsoleMode,consoleInHandle,0

    cmp evRepeat,0  ; key already processed by previous call to this function?
    ja  HaveKey ; if so, process the key

Peek:
    ; Peek to see if we have a pending event. If so, read it.
    INVOKE PeekConsoleInput, consoleInHandle, ADDR evBuffer, 1, ADDR evEvents
    test evEvents,0FFFFh
    jz   NoKey                      ; No pending events, so done.

    INVOKE ReadConsoleInput, consoleInHandle, ADDR evBuffer, 1, ADDR evEvents

    test evEvents,0FFFFh
    jz   NoKey                      ; No pending events, so done.

    cmp  evBuffer.eventType,KEY_EVENT                   ; Is it a key event?
    jne  Peek                       ; No -> Peek for next event
    TEST evBuffer.Event.bKeyDown, KBDOWN_FLAG       ; is it a key down event?
    jz   Peek                       ; No -> Peek for next event

    mov  ax,evBuffer.Event.wRepeatCount             ; Set our internal repeat counter
    mov  evRepeat,ax

HaveKey:
    mov  al,evBuffer.Event.uChar.AsciiChar              ; copy Ascii char to al
    mov  ah,BYTE PTR evBuffer.Event.wVirtualScanCode    ; copy Scan code to ah
    mov  dx,evBuffer.Event.wVirtualKeyCode              ; copy Virtual key code to dx
    mov  ebx,evBuffer.Event.dwControlKeyState           ; copy keyboard flags to ebx

    ; Ignore the key press events for Shift, Ctrl, Alt, etc.
    ; Don't process them unless used in combination with another key
    .IF dx == VK_SHIFT || dx == VK_CONTROL || dx == VK_MENU || \
      dx == VK_CAPITAL || dx == VK_NUMLOCK || dx == VK_SCROLL
      jmp Peek                  ; Don't process -> Peek for next event
    .ENDIF

    call  ReadKeyTranslate                  ; Translate scan code compatability

    dec  evRepeat                   ; Decrement our repeat counter
    or   dx,dx                  ; Have key: clear the Zero flag
    jmp  Done

NoKey:
    mov  evRepeat,0                 ; Reset our repeat counter
    test eax,0                  ; No key: set ZF=1 and quit

Done:
    pushfd                  ; save Zero flag
    pushad
                            ; Restore Console mode
    INVOKE SetConsoleMode,consoleInHandle,saveFlags

    ;Unless we call ReadKeyFlush in Windows 98, the key we just read
    ;reappears the next time ReadString is called! We don't know why.
    call ReadKeyFlush
```

```
    popad
    popfd                   ; restore Zero flag
    ret
ReadKey ENDP


;-----------------------------------------------------------------------------
ReadKeyFlush PROC
; Flushes the console input buffer and clears our internal repeat counter.
; Can be used to get faster keyboard reponse in arcade-style games, where
; we don't want to processes accumulated keyboard data that would slow down
; the program's response time.
; Receives: nothing
; Returns: nothing
; By Richard Stam, used by permission.
; Modified 4/5/03 by Irvine
;-----------------------------------------------------------------------------
    INVOKE FlushConsoleInputBuffer, consoleInHandle     ; Flush the buffer
    mov    evRepeat,0                           ; Reset our repeat counter
    ret
ReadKeyFlush ENDP


;-----------------------------------------------------------------------------
ReadKeyTranslate PROC PRIVATE USES ebx ecx edx esi
; Translates special scan codes to be compatible with DOS/BIOS return values.
; Called directly by ReadKey.
; Receives:
;   al  = key Ascii code
;   ah  = Virtual scan code
;   dx  = Virtual key code
;   ebx = Keyboard flags (Alt,Ctrl,Caps,etc.)
; Returns:
;   ah  = Updated scan code (for Alt/Ctrl/Shift & special cases)
;   al  = Updated key Ascii code (set to 0 for special keys)
; Written by Richard Stam, used by permission.
; Modified 4/5/03 by Irvine
;-----------------------------------------------------------------------------

.data  ; Special key scan code translation table
; order: VirtualKey,NormalScan,CtrlScan,AltScan
SpecialCases \
    BYTE VK_LEFT,  4Bh, 73h,  4Bh
CaseSize = ($ - SpecialCases)           ; Special case table element size
    BYTE VK_RIGHT, 4Dh, 74h,  4Dh
    BYTE VK_UP,    48h, 8Dh,  48h
    BYTE VK_DOWN,  50h, 91h,  50h
    BYTE VK_PRIOR, 49h, 84h,  49h       ; PgUp
    BYTE VK_NEXT,  51h, 76h,  51h       ; PgDn
    BYTE VK_HOME,  47h, 77h,  47h
    BYTE VK_END,   4Fh, 75h,  4Fh
    BYTE VK_INSERT,52h, 92h,  52h
    BYTE VK_DELETE,53h, 93h,  53h
    BYTE VK_ADD,   4Eh, 90h,  4Eh
    BYTE VK_SUBTRACT,4Ah,8Eh, 4Ah
    BYTE VK_F11,   85h, 85h,  85h
    BYTE VK_F12,   86h, 86h,  86h
    BYTE VK_11,    0Ch, 0Ch,  82h       ; see above
    BYTE VK_12,    0Dh, 0Dh,  83h       ; see above
    BYTE 0          ; End of Table

.code
    pushfd                  ; Push flags to save ZF of ReadKey
    mov  esi,0

    ; Search through the special cases table
Search:
    cmp  SpecialCases[esi],0                ; Check for end of search table
    je   NotFound

    cmp  dl,SpecialCases[esi]               ; Check if special case is found
    je   Found

    add  esi,CaseSize                   ; Increment our table index
```

```
    jmp  Search                ; Continue searching

Found:
    .IF ebx & CTRL_MASK
      mov  ah,SpecialCases[esi+2]                  ; Specify the Ctrl scan code
      mov  al,0                ; Updated char for special keys
    .ELSEIF ebx & ALT_MASK
      mov  ah,SpecialCases[esi+3]                  ; Specify the Alt scan code
      mov  al,0                ; Updated char for special keys
    .ELSE
      mov ah,SpecialCases[esi+1]                   ; Specify the normal scan code
    .ENDIF
    jmp  Done

NotFound:
    .IF ! (ebx & KEY_MASKS)                ; Done if not shift/ctrl/alt combo
      jmp  Done
    .ENDIF

    .IF dx >= VK_F1 && dx <= VK_F10             ; Check for F1 to F10 keys
      .IF ebx & CTRL_MASK
        add ah,23h                ; 23h = Hex diff for Ctrl/Fn keys
      .ELSEIF ebx & ALT_MASK
        add ah,2Dh                ; 2Dh = Hex diff for Alt/Fn keys
      .ELSEIF ebx & SHIFT_MASK
        add ah,19h                ; 19h = Hex diff for Shift/Fn keys
      .ENDIF
    .ELSEIF al >= '0' && al <= '9'              ; Check for Alt/1 to Alt/9
      .IF ebx & ALT_MASK
        add ah,76h                ; 76h = Hex diff for Alt/n keys
        mov al,0
      .ENDIF
    .ELSEIF dx == VK_TAB                    ; Check for Shift/Tab (backtab)
      .IF ebx & SHIFT_MASK
        mov al,0                  ; ah already has 0Fh, al=0 for special
      .ENDIF
    .ENDIF

Done:
    popfd                  ; Pop flags to restore ZF of ReadKey
    ret
ReadKeyTranslate ENDP


;-----------------------------------------------------
ReadString PROC
    LOCAL bufSize:DWORD, saveFlags:DWORD, junk:DWORD
;
; Reads a string from the keyboard and places the characters
; in a buffer.
; Receives: EDX offset of the input buffer
;           ECX = maximum characters to input (including terminal null)
; Returns:  EAX = size of the input string.
; Comments: Stops when Enter key (0Dh,0Ah) is pressed. If the user
; types more characters than (ECX-1), the excess characters
; are ignored.
; Written by Kip Irvine and Gerald Cahill
;
; Last update: 11/19/92, 03/20/2003
;-----------------------------------------------------
.data
_$$temp DWORD ?     ; added 03/20/03
.code
    pushad
    CheckInit

    mov edi,edx     ; set EDI to buffer offset
    mov bufSize,ecx     ; save buffer size

    push edx
    INVOKE ReadConsole,
      consoleInHandle,      ; console input handle
      edx,      ; buffer offset
      ecx,      ; max count
```

```
      OFFSET bytesRead,
      0
    pop edx
    cmp bytesRead,0
    jz  L5      ; skip move if zero chars input

    dec bytesRead       ; make first adjustment to bytesRead
    cld     ; search forward
    mov ecx,bufSize     ; repetition count for SCASB
    mov al,0Ah      ; scan for 0Ah (Line Feed) terminal character
    repne scasb
    jne L1      ; if not found, jump to L1

    ;if we reach this line, length of input string <= (bufsize - 2)

    dec bytesRead       ; second adjustment to bytesRead
    sub edi,2       ; 0Ah found: back up two positions
    cmp edi,edx         ; don't back up to before the user's buffer
    jae L2
    mov edi,edx         ; 0Ah must be the only byte in the buffer
    jmp L2      ; and jump to L2

L1: mov edi,edx     ; point to last byte in buffer
    add edi,bufSize
    dec edi
    mov BYTE PTR [edi],0            ; insert null byte

    ; Save the current console mode
    INVOKE GetConsoleMode,consoleInHandle,ADDR saveFlags
    ; Switch to single character mode
    INVOKE SetConsoleMode,consoleInHandle,0

    ; Clear excess characters from the buffer, 1 byte at a time
L6: INVOKE ReadConsole,consoleInHandle,ADDR junk,1,ADDR _$$temp,0
    mov al,BYTE PTR junk
    cmp al,0Ah      ; the terminal line feed character
    jne L6          ; keep looking, it must be there somewhere

    INVOKE SetConsoleMode,consoleInHandle,saveFlags ; restore console mode.
    jmp L5

L2: mov BYTE PTR [edi],0        ; insert null byte

L5: popad
    mov eax,bytesRead
    ret
ReadString ENDP


;-----------------------------------------------------------
SetTextColor PROC
;
; Change the color of all subsequent text output.
; Receives: AX = attribute. Bits 0-3 are the foreground
;   color, and bits 4-7 are the background color.
; Returns: nothing
; Last update: 6/20/05
;-----------------------------------------------------------

    pushad
    CheckInit   ; added 6/20/05

    INVOKE SetConsoleTextAttribute, consoleOutHandle, ax

    popad
    ret
SetTextColor ENDP


;-----------------------------------------------------
StrLength PROC
;
; Returns the length of a null-terminated string.
; Receives: EDX points to the string.
```

```
; Returns: EAX = string length.
; Last update: 6/9/05
;-----------------------------------------------------
    push    edx
    mov eax,0       ; character count

L1: cmp BYTE PTR [edx],0    ; end of string?
    je  L2  ; yes: quit
    inc edx ; no: point to next
    inc eax ; add 1 to count
    jmp L1

L2: pop edx
    ret
StrLength ENDP


;-----------------------------------------------------
Str_compare PROC USES eax edx esi edi,
    string1:PTR BYTE,
    string2:PTR BYTE
;
; Compare two strings.
; Returns nothing, but the Zero and Carry flags are affected
; exactly as they would be by the CMP instruction.
; Last update: 1/18/02
;-----------------------------------------------------
    mov esi,string1
    mov edi,string2

L1: mov  al,[esi]
    mov  dl,[edi]
    cmp  al,0           ; end of string1?
    jne  L2             ; no
    cmp  dl,0           ; yes: end of string2?
    jne  L2             ; no
    jmp  L3             ; yes, exit with ZF = 1

L2: inc  esi            ; point to next
    inc  edi
    cmp  al,dl          ; chars equal?
    je   L1             ; yes: continue loop
                        ; no: exit with flags set
L3: ret
Str_compare ENDP


;-----------------------------------------------------
Str_copy PROC USES eax ecx esi edi,
    source:PTR BYTE,        ; source string
    target:PTR BYTE     ; target string
;
; Copy a string from source to target.
; Requires: the target string must contain enough
;           space to hold a copy of the source string.
; Last update: 1/18/02
;-----------------------------------------------------
    INVOKE Str_length,source        ; EAX = length source
    mov ecx,eax     ; REP count
    inc ecx                 ; add 1 for null byte
    mov esi,source
    mov edi,target
    cld                     ; direction = up
    rep movsb           ; copy the string
    ret
Str_copy ENDP


;-----------------------------------------------------
Str_length PROC USES edi,
    pString:PTR BYTE    ; pointer to string
;
; Return the length of a null-terminated string.
; Receives: pString - pointer to a string
```

```
; Returns: EAX = string length
; Last update: 1/18/02
;-----------------------------------------------------
    mov edi,pString
    mov eax,0       ; character count
L1:
    cmp BYTE PTR [edi],0    ; end of string?
    je  L2  ; yes: quit
    inc edi ; no: point to next
    inc eax ; add 1 to count
    jmp L1
L2: ret
Str_length ENDP


;-----------------------------------------------------------
Str_trim PROC USES eax ecx edi,
    pString:PTR BYTE,       ; points to string
    char:BYTE       ; char to remove
;
; Remove all occurences of a given character from
; the end of a string.
; Returns: nothing
; Last update: 6/12/2008
;-----------------------------------------------------------
    pushf               ; save the flags
    mov  edi,pString
    INVOKE Str_length,edi       ; returns length in EAX
    cmp  eax,0          ; zero-length string?
    je   L2             ; yes: exit
    mov  ecx,eax        ; no: counter = string length
    dec  eax
    add  edi,eax        ; EDI points to last char
    mov  al,char        ; char to trim
    std                 ; direction = reverse
    repe scasb          ; skip past trim character
    jne  L1             ; removed first character?
    dec  edi            ; adjust EDI: ZF=1 && ECX=0
L1: mov  BYTE PTR [edi+2],0     ; insert null byte
L2: popf                ; restore the flags
    ret
Str_trim ENDP


;-----------------------------------------------
Str_ucase PROC USES eax esi,
    pString:PTR BYTE
; Convert a null-terminated string to upper case.
; Receives: pString - a pointer to the string
; Returns: nothing
; Last update: 1/18/02
;-----------------------------------------------
    mov esi,pString
L1:
    mov al,[esi]        ; get char
    cmp al,0        ; end of string?
    je  L3      ; yes: quit
    cmp al,'a'      ; below "a"?
    jb  L2
    cmp al,'z'      ; above "z"?
    ja  L2
    and BYTE PTR [esi],11011111b    ; convert the char

L2: inc esi     ; next char
    jmp L1

L3: ret
Str_ucase ENDP


;-----------------------------------------------
WaitMsg PROC
;
; Displays a prompt and waits for the user to press a key.
```

```
; Receives: nothing
; Returns: nothing
; Last update: 6/9/05
;-----------------------------------------------------
.data
waitmsgstr BYTE "Press any key to continue...",0
.code
    pushad

    mov edx,OFFSET waitmsgstr
    call    WriteString
    call    ReadChar

    popad
    ret
WaitMsg ENDP


;-----------------------------------------------------
WriteBin PROC
;
; Writes a 32-bit integer to the console window in
; binary format. Converted to a shell that calls the
; WriteBinB procedure, to be compatible with the
; library documentation in Chapter 5.
; Receives: EAX = the integer to write
; Returns: nothing
;
; Last update: 11/18/02
;-----------------------------------------------------

    push ebx
    mov  ebx,4  ; select doubleword format
    call WriteBinB
    pop  ebx

    ret
WriteBin ENDP


;-----------------------------------------------------
WriteBinB PROC
;
; Writes a 32-bit integer to the console window in
; binary format.
; Receives: EAX = the integer to write
;           EBX = display size (1,2,4)
; Returns: nothing
;
; Last update: 11/18/02  (added)
;-----------------------------------------------------
    pushad

    cmp   ebx,1     ; ensure EBX is 1, 2, or 4
    jz    WB0
    cmp   ebx,2
    jz    WB0
    mov   ebx,4     ; set to 4 (default) even if it was 4
WB0:
    mov   ecx,ebx
    shl   ecx,1     ; number of 4-bit groups in low end of EAX
    cmp   ebx,4
    jz    WB0A
    ror   eax,8     ; assume TYPE==1 and ROR byte
    cmp   ebx,1
    jz    WB0A      ; good assumption
    ror   eax,8     ; TYPE==2 so ROR another byte
WB0A:

    mov   esi,OFFSET buffer

WB1:
    push  ecx   ; save loop count
```

```
    mov   ecx,4 ; 4 bits in each group
WB1A:
    shl   eax,1 ; shift EAX left into Carry flag
    mov   BYTE PTR [esi],'0'    ; choose '0' as default digit
    jnc   WB2   ; if no carry, then jump to L2
    mov   BYTE PTR [esi],'1'    ; else move '1' to DL
WB2:
    inc   esi
    Loop  WB1A  ; go to next bit within group

    mov   BYTE PTR [esi],' '    ; insert a blank space
    inc   esi   ; between groups
    pop   ecx   ; restore outer loop count
    loop  WB1   ; begin next 4-bit group

    dec  esi        ; eliminate the trailing space
    mov  BYTE PTR [esi],0   ; insert null byte at end
    mov  edx,OFFSET buffer  ; display the buffer
    call WriteString

    popad
    ret
WriteBinB ENDP


;-----------------------------------------------------
WriteChar PROC
;
; Write a character to the console window
; Recevies: AL = character
; Last update: 10/30/02
; Note: WriteConole will not work unless direction flag is clear.
;-----------------------------------------------------
    pushad
    pushfd  ; save flags
    CheckInit

    mov  buffer,al

    cld ; clear direction flag
    INVOKE WriteConsole,
      consoleOutHandle, ; console output handle
      OFFSET buffer,    ; points to string
      1,    ; string length
      OFFSET bytesWritten,      ; returns number of bytes written
      0

    popfd   ; restore flags
    popad
    ret
WriteChar ENDP


;-----------------------------------------------------
WriteDec PROC
;
; Writes an unsigned 32-bit decimal number to
; the console window. Input parameters: EAX = the
; number to write.
; Last update: 6/8/2005
;-----------------------------------------------------
.data
; There will be as many as 10 digits.
WDBUFFER_SIZE = 12

bufferL BYTE WDBUFFER_SIZE DUP(?),0

.code
    pushad
    CheckInit

    mov   ecx,0           ; digit counter
    mov   edi,OFFSET bufferL
    add   edi,(WDBUFFER_SIZE - 1)
```

```
    mov   ebx,10    ; decimal number base

WI1:mov   edx,0             ; clear dividend to zero
    div   ebx               ; divide EAX by the radix

    xchg  eax,edx           ; swap quotient, remainder
    call  AsciiDigit        ; convert AL to ASCII
    mov   [edi],al          ; save the digit
    dec   edi               ; back up in buffer
    xchg  eax,edx           ; swap quotient, remainder

    inc   ecx               ; increment digit count
    or    eax,eax           ; quotient = 0?
    jnz   WI1               ; no, divide again

     ; Display the digits (CX = count)
WI3:
     inc   edi
     mov   edx,edi
     call  WriteString

WI4:
     popad  ; restore 32-bit registers
     ret
WriteDec ENDP


;-----------------------------------------------------
WriteHex PROC
;
; Writes an unsigned 32-bit hexadecimal number to
; the console window.
; Input parameters: EAX = the number to write.
; Shell interface for WriteHexB, to retain compatibility
; with the documentation in Chapter 5.
;
; Last update: 11/18/02
;-----------------------------------------------------
    push ebx
    mov  ebx,4
    call WriteHexB
    pop  ebx
    ret
WriteHex ENDP


;-----------------------------------------------------
WriteHexB PROC
    LOCAL displaySize:DWORD
;
; Writes an unsigned 32-bit hexadecimal number to
; the console window.
; Receives: EAX = the number to write. EBX = display size (1,2,4)
; Returns: nothing
;
; Last update: 11/18/02
;-----------------------------------------------------

DOUBLEWORD_BUFSIZE = 8

.data
bufferLHB BYTE DOUBLEWORD_BUFSIZE DUP(?),0

.code
    pushad                  ; save all 32-bit data registers
    mov displaySize,ebx ; save component size

; Clear unused bits from EAX to avoid a divide overflow.
; Also, verify that EBX contains either 1, 2, or 4. If any
; other value is found, default to 4.

.IF EBX == 1    ; check specified display size
    and  eax,0FFh   ; byte == 1
.ELSE
```

```
    .IF EBX == 2
      and  eax,0FFFFh    ; word == 2
    .ELSE
      mov displaySize,4 ; default (doubleword) == 4
    .ENDIF
.ENDIF

    CheckInit

    mov   edi,displaySize   ; let EDI point to the end of the buffer:
    shl   edi,1 ; multiply by 2 (2 digits per byte)
    mov   bufferLHB[edi],0  ; store null string terminator
    dec   edi    ; back up one position

    mov   ecx,0             ; digit counter
    mov   ebx,16    ; hexadecimal base (divisor)

L1:
    mov   edx,0             ; clear upper dividend
    div   ebx               ; divide EAX by the base

    xchg  eax,edx           ; swap quotient, remainder
    call  AsciiDigit        ; convert AL to ASCII
    mov   bufferLHB[edi],al       ; save the digit
    dec   edi               ; back up in buffer
    xchg  eax,edx           ; swap quotient, remainder

    inc   ecx               ; increment digit count
    or    eax,eax           ; quotient = 0?
    jnz   L1            ; no, divide again

     ; Insert leading zeros

    mov   eax,displaySize   ; set EAX to the
    shl   eax,1 ; number of digits to print
    sub   eax,ecx   ; subtract the actual digit count
    jz    L3            ; display now if no leading zeros required
    mov   ecx,eax           ; CX = number of leading zeros to insert

L2:
    mov   bufferLHB[edi],'0'    ; insert a zero
    dec   edi                   ; back up
    loop  L2                    ; continue the loop

    ; Display the digits. ECX contains the number of
    ; digits to display, and EDX points to the first digit.
L3:
    mov   ecx,displaySize   ; output format size
    shl   ecx,1             ; multiply by 2
    inc   edi
    mov   edx,OFFSET bufferLHB
    add   edx,edi
    call  WriteString

    popad   ; restore 32-bit registers
    ret
WriteHexB ENDP


;-----------------------------------------------------
WriteInt PROC
;
; Writes a 32-bit signed binary integer to the console window
; in ASCII decimal.
; Receives: EAX = the integer
; Returns:  nothing
; Comments: Displays a leading sign, no leading zeros.
; Last update: 7/11/01
;-----------------------------------------------------
WI_Bufsize = 12
true  =   1
false =   0
.data
buffer_B  BYTE  WI_Bufsize DUP(0),0  ; buffer to hold digits
```

```
neg_flag  BYTE  ?

.code
    pushad
    CheckInit

    mov   neg_flag,false    ; assume neg_flag is false
    or    eax,eax             ; is AX positive?
    jns   WIS1              ; yes: jump to B1
    neg   eax                ; no: make it positive
    mov   neg_flag,true     ; set neg_flag to true

WIS1:
    mov   ecx,0              ; digit count = 0
    mov   edi,OFFSET buffer_B
    add   edi,(WI_Bufsize-1)
    mov   ebx,10             ; will divide by 10

WIS2:
    mov   edx,0              ; set dividend to 0
    div   ebx                ; divide AX by 10
    or    dl,30h            ; convert remainder to ASCII
    dec   edi                ; reverse through the buffer
    mov   [edi],dl           ; store ASCII digit
    inc   ecx                ; increment digit count
    or    eax,eax             ; quotient > 0?
    jnz   WIS2              ; yes: divide again

    ; Insert the sign.

    dec   edi   ; back up in the buffer
    inc   ecx                   ; increment counter
    mov   BYTE PTR [edi],'+'    ; insert plus sign
    cmp   neg_flag,false        ; was the number positive?
    jz    WIS3                  ; yes
    mov   BYTE PTR [edi],'-'    ; no: insert negative sign

WIS3:   ; Display the number
    mov  edx,edi
    call WriteString

    popad
    ret
WriteInt ENDP

NoNameCode = 1;          Special nonprintable code to signal that
              ;          WriteStackFrame was called.
WriteStackFrameNameSize = 64   ; Size of WriteStackFrameName's stack frame
WriteStackFrameSize = 20       ; Size of WriteStackFrame's stack frame

.code
;-------------------------------------------------
WriteStackFrameName PROC USES EAX EBX ECX EDX ESI,
           numParam:DWORD,      ; number of parameters passed to the procedure
           numLocalVal: DWORD, ; number of DWord local variables
           numSavedReg: DWORD, ; number of saved registers
           procName: PTR BYTE  ; pointer to name of procedure
       LOCAL theReturn:  DWORD, theBase:  DWORD, \
             firstLocal: DWORD, firstSaved: DWORD, \
             specialFirstSaved: DWORD

; When called properly from a procedure with a stack frame, it prints
; out the stack frame for the procedure.  Each item is labeled with its
; purpose: parameter, return address, saved ebp, local variable or saved
; register.   The items pointed by ebp and esp are marked.

; Requires:  The procedure has a stack frame including the return address
;            and saved base pointer.
;            It is suffient that procedure's PROC statement includes either
;            at least one local variable or one parameter.  If the procedure's
;            PROC statement does not include either of these items, it is
;            sufficient if the procedure begins with
;                  push ebp
;                  mov  ebp, esp
```

```
;            and the stack frame is completed before this procedure is
;            INVOKEd providing the procedure does not have a USES clause.
;            If there is a USES clause, but no parameters or local variables,
;            the modified structure is printed
; Parameters passed on stack using STDCALL:
;            numParam:    number of parameters
;            numLocalVal: number of DWORDS of local variables
;            numSavedReg: number of saved registers
;            ptrProcName: pointer to name of procedure
; Returns:  nothing
; Sample use:
;         myProc PROC USES ebx, ecx, edx      ; saves 3 registers
;                    val:DWORD;               ; has 1 parameter
;                LOCAL a:DWORD, b:DWORD       ; has 2 local varables
;         .data
;         myProcName  BYTE "myProc", 0
;         .code
;                INVOKE writeStackFrameName, 1, 2, 3, ADDR myProcName
;  Comment:  The number parameters are ordered by the order of the
;            corresponding items in the stack frame.
;
; Author:  James Brink, Pacific Lutheran University
; Last update: 4/6/2005
;-----------------------------------------------------
.data
LblStack  BYTE "Stack Frame ",  0
LblFor    Byte "for ", 0
LblEbp    BYTE "  ebp", 0             ; used for offsets from ebp
LblParam  BYTE " (parameter)", 0
LblEbpPtr BYTE " (saved ebp) <--- ebp", 0
LblSaved  BYTE " (saved register)", 0
LblLocal  BYTE " (local variable)", 0
LblReturn BYTE " (return address)", 0
LblEsp    BYTE " <--- esp", 13, 10, 0 ; adds blank line at end of stack frame
BadStackFrameMsg BYTE "The stack frame is invalid", 0
.code
        ;  register usage:
        ;  eax:  value to be printed
        ;  ebx:  offset from ebp
        ;  ecx:  item counter
        ;  edx:  location of string being printed
        ;  esi:  memory location of stack frame item

        ; print title
    mov  edx, OFFSET LblStack
    call writeString
    mov  esi, procName
              ; NOTE:  esi must not be changed until we get to
              ;        the section for calculating the location
              ;        of the caller's ebp at L0a:
    cmp  BYTE PTR [esi], 0      ; is the name string blank?
    je   L0                     ; if so, just go to a new line
    cmp  BYTE PTR [esi], NoNameCode
                                ; is the name the special code
                                ; from WriteStackFrame?
    je   L0                     ; if so, just go to a new line
    mov  edx, OFFSET LblFor     ; if not, add "for "
    call writeString
    mov  edx, procName          ; and print name
    call writeString
L0: call crlf
    call crlf

        mov  ecx, 0            ; initialize sum of items in stack frame
        mov  ebx, 0            ; initialize sum of items in stack frame
                               ;    preceding the base pointer

        ; check for special stack frame condition
        mov  eax, numLocalVal  ; Special condition:  numLocalVal = 0
        cmp  eax, 0
        ja   Normal

        mov  eax, numParam     ; Special condition:  numParm = 0
        cmp  eax, 0
```

```
        ja   Normal

        mov  eax, numSavedReg  ; Special condition:  numSaveReg > 0
        cmp  eax, 0
        ja   Special

Normal: mov  eax, numSavedReg  ; get number of parameters
    add  ecx, eax          ; add to number of items in stack frame
    mov  firstSaved, ecx   ; save item number of the first saved register
    mov  specialFirstSaved, 0
                           ; no special saved registers

    mov  eax, numLocalVal  ; get number of local variable DWords
    add  ecx, eax          ; add to number of items in stack frame
    mov  firstLocal, ecx   ; save item number of first local variable

    add  ecx, 1            ; add 1 for the saved ebp
    mov  theBase, ecx      ; save item number of the base pointer

    add  ecx, 1            ; add 1 for the return address
    add  ebx, 1            ; add 1 for items stored above ebp
                                   ; add for the return address/preceding ebp
    mov  theReturn, ecx    ; save item number of the return pointer

    mov  eax, numParam     ; get number of parameters
    add  ecx, eax          ; add to number of items in stack frame
    add  ebx, eax          ; add for the parameters/preceding ebp

    jmp  L0z

Special:
        ; MASM does not create a stack frame under these conditions:
        ;   The number of parameters is 0
        ;   The number of local variables is 0
        ;   The number of saved (USES) registers is positive.
        ;   The following assumes the procedure processed ebp manually
        ;   because MASM does not push it under these conditions.
        mov  firstSaved, ecx   ; there are no "regular" saved registers
        mov  firstLocal, ecx   ; there are no local variables

        add  ecx, 1            ; add 1 for the saved ebp
        mov  theBase, ecx      ; save item number of the base pointer

        mov  eax, numSavedReg  ; get number of saved registers
        add  ecx, eax          ; add to number of items in the stack frame
        add  ebx, eax          ; add for the items preceding ebp
        mov  specialFirstSaved, ecx

    add  ecx, 1            ; add 1 for the return address
    add  ebx, 1            ; add 1 for items stored above ebp
                                   ; add for the return address/preceding ebp
    mov  theReturn, ecx    ; save item number of the return pointer

        mov  eax, esp
        add  eax, 44
        cmp  eax, esi

L0z:
    ;  ecx now contains the number of items in the stack frame
        ;  ebx now contains the number of items preceding the base pointer

    ; determine the size of those items preceding the base pointer
    shl  ebx, 2            ; multiply by 4

        ; determine location of caller's saved ebp
L0a:    cmp  BYTE PTR [esi], NoNameCode
                           ; check for special code
L0b:    mov  esi, [ebp]        ; get the ebp (1 indirection
                               ; mov does not change flags
        jne  L0c               ; if not special code, skip the next step
        mov  esi, [esi]        ; 2nd indirection if called by WriteStackFrame
L0c:                           ; esi has pointer into caller's stack frame
;   At this point esi contains the location for the caller's saved ebp
```

```
;   Check special case to make sure ebp and esp agree.
;   Printing the stack frame cannot be printed if ebp has not been pushed
        mov  eax, specialFirstSaved ; Was this a special case?
        cmp  eax, 0             ; If so specialFirstSaved would be 0
        je   L0e                ; If not, continue normal processing
        mov  eax, esp           ; Calculate loc. of last entry before
                                ; of WriteStackFrameNames stack frame
        add  eax, WriteStackFrameNameSize
        cmp  eax, esi           ; does it equal the location of the base pointer?
        je   L0e                ; if so, continue normal processing
                                ; if not chec to see if procedure was called
                                ; by writeStackFrame
        add  eax, WriteStackFrameSize
        cmp  eax, esi           ; does it equal the location of the base pointer?
        jne  badStackFrame      ; if not, the stack frame is invalid
                                ; These are not perfect test as we haven't
                                ; checked to see which case we are in.

; Continue normal processing by calculating its stack frame size

L0e:    add  esi, ebx           ; calculate beginning of the caller's stack
                                ; frame (highest memory used)


    ; *** loop to print stack frame
    ; Note:  the order of some the following checks is important                 ↙
                                                                               ↙
                                                                  ck frame  ***
L1: ; write value and beginning offset from basepointer
        mov  eax, [esi]         ; write item in stack frame
    call writeHex
    mov  edx, OFFSET LblEbp ; write " ebp"
    call writeString
    mov  eax, ebx           ; write offset from base pointer
    call writeInt
    ; check for special labels
    cmp  ecx, theReturn     ; check for return address item
    jne  L2
    mov  edx, OFFSET LblReturn
    jmp  LPrint

L2:     cmp  ecx, theBase       ; check for base pointer
        jne  L2a
        mov  edx, OFFSET LblEbpPtr
        jmp  LPrint

L2a:    cmp  ecx, specialFirstSaved ; Check for special saved registers
    ja   L3
    mov  edx, OFFSET LblSaved
    jmp  LPrint

L3:     cmp  ecx, firstSaved    ; check for saved registers
        ja   L4
        mov  edx, OFFSET LblSaved
        jmp  LPrint
L4:     cmp  ecx, firstLocal    ; check for local variables
        ja   L5
        mov  edx, OFFSET LblLocal
        jmp  LPrint
L5:     mov  edx, OFFSET LblParam
LPrint: call writeString
        cmp  ecx, 1             ; check for last item in stack frame
        jne  LDone
        mov  edx, OFFSET LblEsp
        call writeString
LDone:  ; complete output for line
    call crlf
    ; get ready for the next line
    sub  esi, 4             ; decrement memory location by 4
    sub  ebx, 4             ; decrement offset by 4
    loop LDoneX
    jmp  Return
LDoneX: jmp  L1
Return:
```

```
    ret

; Stack frame invalid
BadStackFrame:
        lea  edx, BadStackFrameMsg
                            ; load message
        call writeString    ; write message
        call crlf
        ret             ; return without printing stack frame

WriteStackFrameName ENDP

;-------------------------------------------------

WriteStackFrame PROC,
           numParam:DWORD,     ; number of parameters passed to the procedure
           numLocalVal: DWORD, ; number of DWord local variables
           numSavedReg: DWORD  ; number of saved registers

; When called properly from a procedure with a stack frame, it prints
; out the stack frame for the procedure.  Each item is labeled with its
; purpose: parameter, return address, saved ebp, local variable or saved
; register.   The items pointed by ebp and esp are marked.

; Requires:  The procedure has a stack frame including the return address
;            and saved base pointer.
;            It is suffient that procedure's PROC statement includes either
;            at least one local variable or one parameter.  If the procedure's
;            PROC statement does not include either of these items, it is
;            sufficient if the procedure begins with
;                  push ebp
;                  mov  ebp, esp
;            and the stack frame is completed before this procedure is
;            INVOKEd providing the procedure does not have a USES clause.
;            If there is a USES clause, but no parameters or local variables,
;            the modified structure is printed
; Parameters passed on stack using STDCALL:
;            numParam:    number of parameters
;            numLocalVal: number of DWORDS of local variables
;            numSavedReg: number of saved registers
; Returns:  nothing
; Sample use:
;         myProc PROC USES ebx, ecx, edx      ; saves 3 registers
;                  val:DWORD;                 ; has 1 parameter
;               LOCAL a:DWORD, b:DWORD        ; has 2 local varables
;         .data
;         myProcName  BYTE "myProc", 0
;         .code
;               INVOKE writeStackFrame, 1, 2, 3
;
; Comments:  The parameters are ordered by the order of the corresponding
;            items in the stack frame.
;
; Author:  James Brink, Pacific Lutheran University
; Last update: 4/6/2005
;-------------------------------------------------
.data
NoName  BYTE  NoNameCode
.code
        INVOKE WriteStackFrameName, numParam, numLocalVal, \
               NumSavedReg, ADDR NoName
                      ; NoNameCode
                      ; Special signal that WriteStackFrameName
                      ; is being called from WriteStackFrame
        ret
WriteStackFrame ENDP



;-------------------------------------------------------
WriteString PROC
;
; Writes a null-terminated string to standard
; output. Input parameter: EDX points to the
```

```
; string.
; Last update: 9/7/01
;-----------------------------------------------------
    pushad

    CheckInit

    INVOKE Str_length,edx       ; return length of string in EAX
    cld ; must do this before WriteConsole

    INVOKE WriteConsole,
        consoleOutHandle,       ; console output handle
        edx,    ; points to string
        eax,    ; string length
        OFFSET bytesWritten,    ; returns number of bytes written
        0

    popad
    ret
WriteString ENDP


;-----------------------------------------------------
WriteToFile PROC
;
; Writes a buffer to an output file.
; Receives: EAX = file handle, EDX = buffer offset,
;    ECX = number of bytes to write
; Returns: EAX = number of bytes written to the file.
; Last update: 6/8/2005
;-----------------------------------------------------
.data
WriteToFile_1 DWORD ?       ; number of bytes written
.code
    INVOKE WriteFile,   ; write buffer to file
        eax,    ; file handle
        edx,    ; buffer pointer
        ecx,    ; number of bytes to write
        ADDR WriteToFile_1, ; number of bytes written
        0   ; overlapped execution flag
    mov eax,WriteToFile_1   ; return value
    ret
WriteToFile ENDP


;-----------------------------------------------------
WriteWindowsMsg PROC USES eax edx
;
; Displays a string containing the most recent error
; generated by MS-Windows.
; Receives: nothing
; Returns: nothing
; Last updated: 6/10/05
;-----------------------------------------------------
.data
WriteWindowsMsg_1 BYTE "Error ",0
WriteWindowsMsg_2 BYTE ": ",0
pErrorMsg DWORD ?   ; points to error message
messageId DWORD ?
.code
    call    GetLastError
    mov messageId,eax

; Display the error number.
    mov edx,OFFSET WriteWindowsMsg_1
    call    WriteString
    call    WriteDec    ; show error number
    mov edx,OFFSET WriteWindowsMsg_2
    call    WriteString

; Get the corresponding message string.
    INVOKE FormatMessage, FORMAT_MESSAGE_ALLOCATE_BUFFER + \
      FORMAT_MESSAGE_FROM_SYSTEM, NULL, messageID, NULL,
      ADDR pErrorMsg, NULL, NULL
```

```
; Display the error message generated by MS-Windows.
    mov edx,pErrorMsg
    call    WriteString

; Free the error message string.
    INVOKE LocalFree, pErrorMsg

    ret
WriteWindowsMsg ENDP


;*************************************************************
;*                    PRIVATE PROCEDURES                     *
;*************************************************************

; Convert AL to an ASCII digit. Used by WriteHex & WriteDec

AsciiDigit PROC PRIVATE
     push  ebx
     mov   ebx,OFFSET xtable
     xlat
     pop   ebx
     ret
AsciiDigit ENDP


HexByte PROC PRIVATE
; Display the byte in AL in hexadecimal

    pushad
    mov  dl,al

    rol  dl,4
    mov  al,dl
    and  al,0Fh
    mov  ebx,OFFSET xtable
    xlat
    mov  buffer,al  ; save first char
    rol  dl,4
    mov  al,dl
    and  al,0Fh
    xlat
    mov  [buffer+1],al  ; save second char
    mov  [buffer+2],0   ; null byte

    mov  edx,OFFSET buffer  ; display the buffer
    call WriteString

    popad
    ret
HexByte ENDP

END

;*************************************************************
;                           ARCHIVE AREA
;
; The following code has been 'retired', but may still be useful
; as a reference.
;*************************************************************


;-------------------------------------------------------
ReadChar PROC
;
; Retired 7/5/05
;
; Reads one character from the keyboard. The character is
; not echoed on the screen. Waits for the character if none is
; currently in the input buffer.
; Returns:  AL = ASCII code
;-------------------------------------------------------
    push ebx
```

```
    push eax

L1: mov  eax,10 ; give Windows 10ms to process messages
    call Delay
    call ReadKey    ; look for key in buffer
    jz   L1 ; no key in buffer if ZF=1

    ; Special epilogue code used here to return AL, yet
    ; preserve the high 24 bits of EAX.
    mov  bl,al  ; save ASCII code
    pop  eax
    mov  al,bl
    pop  ebx
    ret
ReadChar ENDP


;-----------------------------------------------------
ReadDec PROC USES ebx ecx edx esi
  LOCAL saveDigit:DWORD
;
; Retired 7/15/05
;
; Reads a 32-bit unsigned decimal integer from the keyboard,
; stopping when the Enter key is pressed.All valid digits occurring
; before a non-numeric character are converted to the integer value.
; Leading spaces are ignored.

; Receives: nothing
; Returns:
;  If the integer is blank, EAX=0 and CF=1
;  If the integer contains only spaces, EAX=0 and CF=1
;  If the integer is larger than 2^32-1, EAX=0 and CF=1
;  Otherwise, EAX=converted integer, and CF=0
;
; Last update: 11/11/02
;-----------------------------------------------------
; Input a string of digits using ReadString.

    mov   edx,OFFSET digitBuffer
    mov   esi,edx                   ; save offset in ESI
    mov   ecx,MAX_DIGITS
    call  ReadString
    mov   ecx,eax                   ; save length in CX
    cmp   ecx,0                 ; greater than zero?
    jne   L1                    ; yes: continue
    mov   eax,0                 ; no: set return value
    jmp   L5                    ; and exit with CF=1

; Skip over any leading spaces.

L1: mov   al,[esi]              ; get a character from buffer
    cmp   al,' '                ; space character found?
    jne   L2                    ; no: goto next step
    inc   esi                   ; yes: point to next char
    loop  L1        ; all spaces?
    jmp   L5        ; yes: exit with CF=1

; Start to convert the number.

L2: mov  eax,0                  ; clear accumulator
    mov  ebx,10                 ; EBX is the divisor

; Repeat loop for each digit.

L3: mov  dl,[esi]       ; get character from buffer
    cmp  dl,'0'     ; character < '0'?
    jb   L4
    cmp  dl,'9'     ; character > '9'?
    ja   L4
    and  edx,0Fh        ; no: convert to binary

    mov  saveDigit,edx
    mul  ebx        ; EDX:EAX = EAX * EBX
```

```
    jc   L5     ; quit if Carry (EDX > 0)
    mov  edx,saveDigit
    add  eax,edx                ; add new digit to sum
    jc   L5     ; quit if Carry generated
    inc  esi                    ; point to next digit
    jmp  L3     ; get next digit

L4: clc ; succesful completion (CF=0)
    jmp  L6

L5: mov  eax,0  ; clear result to zero
    stc ; signal an error (CF=1)
L6:
    ret
ReadDec ENDP


;-----------------------------------------------------
ReadFromFile PROC
;
; Retired 7/6/05
;
; Reads an input file into a buffer.
; Receives: EAX = file handle, EDX = buffer offset,
;    ECX = number of bytes to read
; Returns: EAX = number of bytes read.
; Last update: 6/8/2005
;-----------------------------------------------------
.data
ReadFromFile_1 DWORD ?      ; number of bytes read
.code
    INVOKE ReadFile,
        eax,    ; file handle
        edx,    ; buffer pointer
        ecx,    ; max bytes to read
        ADDR ReadFromFile_1,    ; number of bytes read
        0       ; overlapped execution flag
    mov eax,ReadFromFile_1
    ret
ReadFromFile ENDP


;-----------------------------------------------------
ReadInt PROC USES ebx ecx edx esi
  LOCAL Lsign:SDWORD, saveDigit:DWORD
;
; Retired 7/15/05
;
; Reads a 32-bit signed decimal integer from standard
; input, stopping when the Enter key is pressed.
; All valid digits occurring before a non-numeric character
; are converted to the integer value. Leading spaces are
; ignored, and an optional leading + or - sign is permitted.
; All spaces return a valid integer, value zero.

; Receives: nothing
; Returns:  If CF=0, the integer is valid, and EAX = binary value.
;   If CF=1, the integer is invalid and EAX = 0.
;
; Contains corrections by Gerald Cahill
; Updated: 10/10/2003
;-----------------------------------------------------
.data
overflow_msgL BYTE  " <32-bit integer overflow>",0
invalid_msgL  BYTE  " <invalid integer>",0
;allspace_msgL BYTE  " <all spaces input>",0
.code

; Input a string of digits using ReadString.

        mov   Lsign,1                   ; assume number is positive
        mov   edx,OFFSET digitBuffer
        mov   esi,edx                   ; save offset in SI
        mov   ecx,MAX_DIGITS
```

```
        call  ReadString
        mov   ecx,eax                   ; save length in ECX
        cmp   ecx,0                     ; length greater than zero?
        jne   L1                        ; yes: continue
        mov   eax,0                     ; no: set return value
        jmp   L10                       ; and exit

; Skip over any leading spaces.

L1:     mov   al,[esi]                  ; get a character from buffer
        cmp   al,' '                    ; space character found?
        jne   L2                        ; no: check for a sign
        inc   esi                       ; yes: point to next char
        loop  L1
        mov   eax,0      ; all spaces?
        jmp   L10        ; return zero as valid value
;       mov   edx,OFFSET allspace_msgL    (line removed)
;       jcxz  L8                          (line removed)

; Check for a leading sign.

L2:     cmp   al,'-'                    ; minus sign found?
        jne   L3                        ; no: look for plus sign

        mov   Lsign,-1                  ; yes: sign is negative
        dec   ecx                       ; subtract from counter
        inc   esi                       ; point to next char
        jmp   L3A

L3:     cmp   al,'+'                    ; plus sign found?
        jne   L3A                          ; no: skip
        inc   esi                       ; yes: move past the sign
        dec   ecx                       ; subtract from digit counter

; Test the first digit, and exit if nonnumeric.

L3A:mov  al,[esi]                       ; get first character
        call IsDigit                    ; is it a digit?
        jnz  L7A                        ; no: show error message

; Start to convert the number.

L4:     mov   eax,0                     ; clear accumulator
        mov   ebx,10                    ; EBX is the divisor

; Repeat loop for each digit.

L5:     mov  dl,[esi]           ; get character from buffer
        cmp  dl,'0'             ; character < '0'?
        jb   L9
        cmp  dl,'9'             ; character > '9'?
        ja   L9
        and  edx,0Fh            ; no: convert to binary

        mov  saveDigit,edx
        imul ebx                ; EDX:EAX = EAX * EBX
        mov  edx,saveDigit

        jo   L6                 ; quit if overflow
        add  eax,edx            ; add new digit to AX
        jo   L6                 ; quit if overflow
        inc  esi                ; point to next digit
        jmp  L5                 ; get next digit

; Overflow has occured, unlesss EAX = 80000000h
; and the sign is negative:

L6: cmp  eax,80000000h
    jne  L7
    cmp  Lsign,-1
    jne  L7                 ; overflow occurred
    jmp  L9                 ; the integer is valid

; Choose "integer overflow" messsage.
```

```
L7: mov  edx,OFFSET overflow_msgL
    jmp  L8

; Choose "invalid integer" message.

L7A:
    mov  edx,OFFSET invalid_msgL

; Display the error message pointed to by EDX, and set the Overflow flag.

L8: call WriteString
    call Crlf
    mov al,127
    add al,1                ; set Overflow flag
    mov  eax,0              ; set return value to zero
    jmp  L10                ; and exit

; IMUL leaves the Sign flag in an undeterminate state, so the OR instruction
; determines the sign of the iteger in EAX.
L9: imul Lsign                  ; EAX = EAX * sign
    or eax,eax              ; determine the number's Sign

L10:ret
ReadInt ENDP
```