Assembly Language for x86 Processors

7th Edition

Kip R. Irvine

Chapter 14: 16-Bit MS-DOS Programming

Slide show prepared by the author

Revision date: 1/15/2014

Chapter Overview

- **MS-DOS and the IBM-PC**
- MS-DOS Function Calls (INT 21h)
- Standard MS-DOS File I/O Services

MS-DOS and the IBM-PC

- Real-Address Mode
- MS-DOS Memory Organization
- MS-DOS Memory Map
- Redirecting Input-Output
- Software Interrupts
- INT Instruction
- Interrupt Vectoring Process
- Common Interrupts

Real-Address Mode

- Real-address mode (16-bit mode) programs have the following characteristics:
  - Max 1 megabyte addressable RAM
  - Single tasking
  - No memory boundary protection
  - Offsets are 16 bits
- IBM PC-DOS: first Real-address OS for IBM-PC
  - Has roots in Gary Kildall's highly successful Digital Research CP/M
  - Later renamed to MS-DOS, owned by Microsoft

MS-DOS Memory Organization

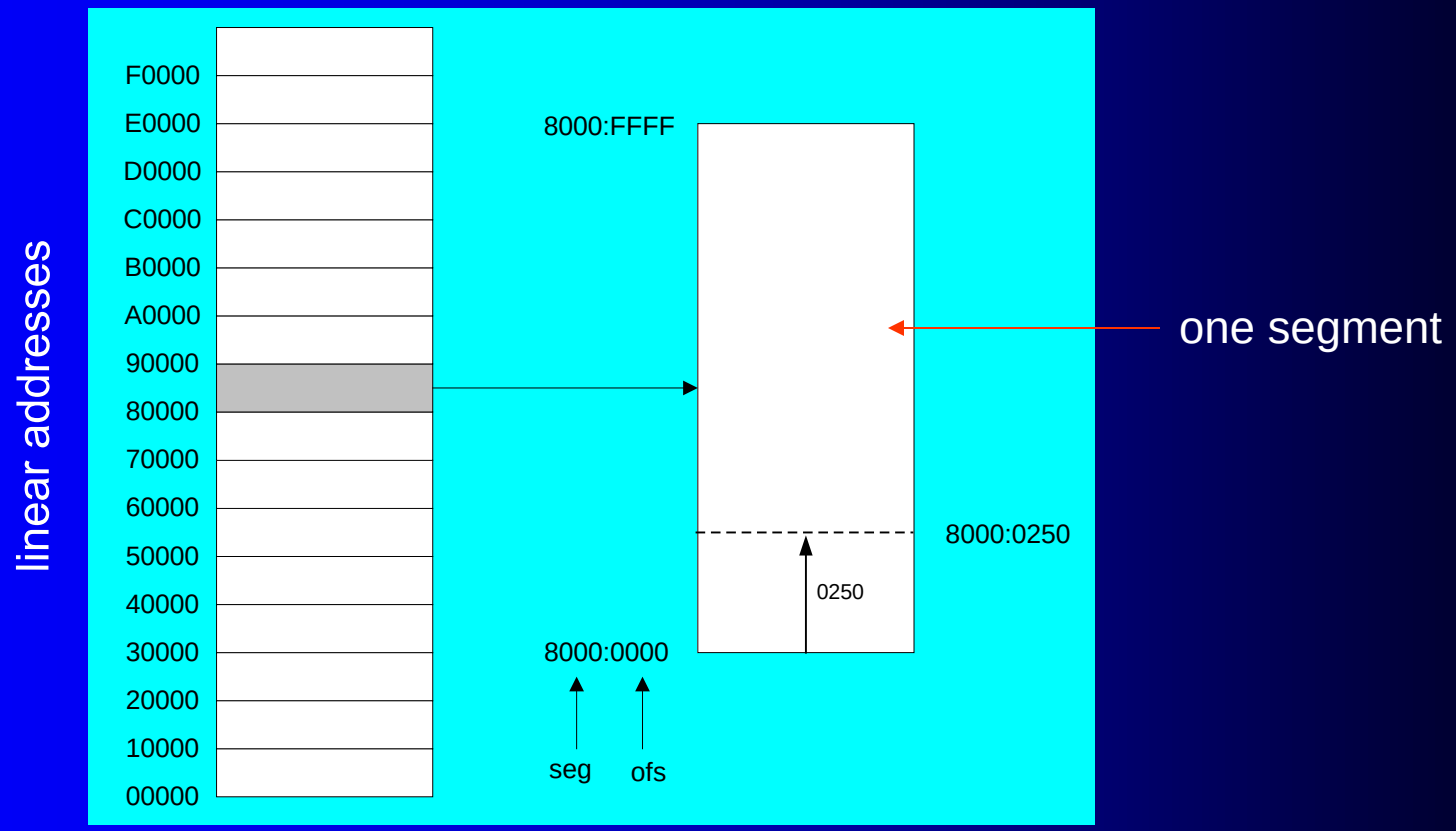
- Interrupt Vector Table
- BIOS & DOS data
- Software BIOS
- MS-DOS kernel
- Resident command processor
- Transient programs
- Video graphics & text
- Reserved (device controllers)
- ROM BIOS

Real-Address mode

- 1 MB RAM maximum addressable
- Application programs can access any area of memory
- Single tasking
- Supported by MS-DOS operating system

Segmented Memory

Segmented memory addressing: absolute (linear) address is a combination of a 16-bit segment value added to a 16-bit offset



Calculating Linear Addresses

- Given a segment address, multiply it by 16 (add a hexadecimal zero), and add it to the offset
- Example: convert 08F1:0100 to a linear address

```
Adjusted Segment value:  0 8 F 1 0
Add the offset:            0 1 0 0
Linear address:          0 9 0 1 0
```

Your turn . . .

What linear address corresponds to the segment/offset address 028F:0030?

028F0 + 0030 = 02920

Always use hexadecimal notation for addresses.

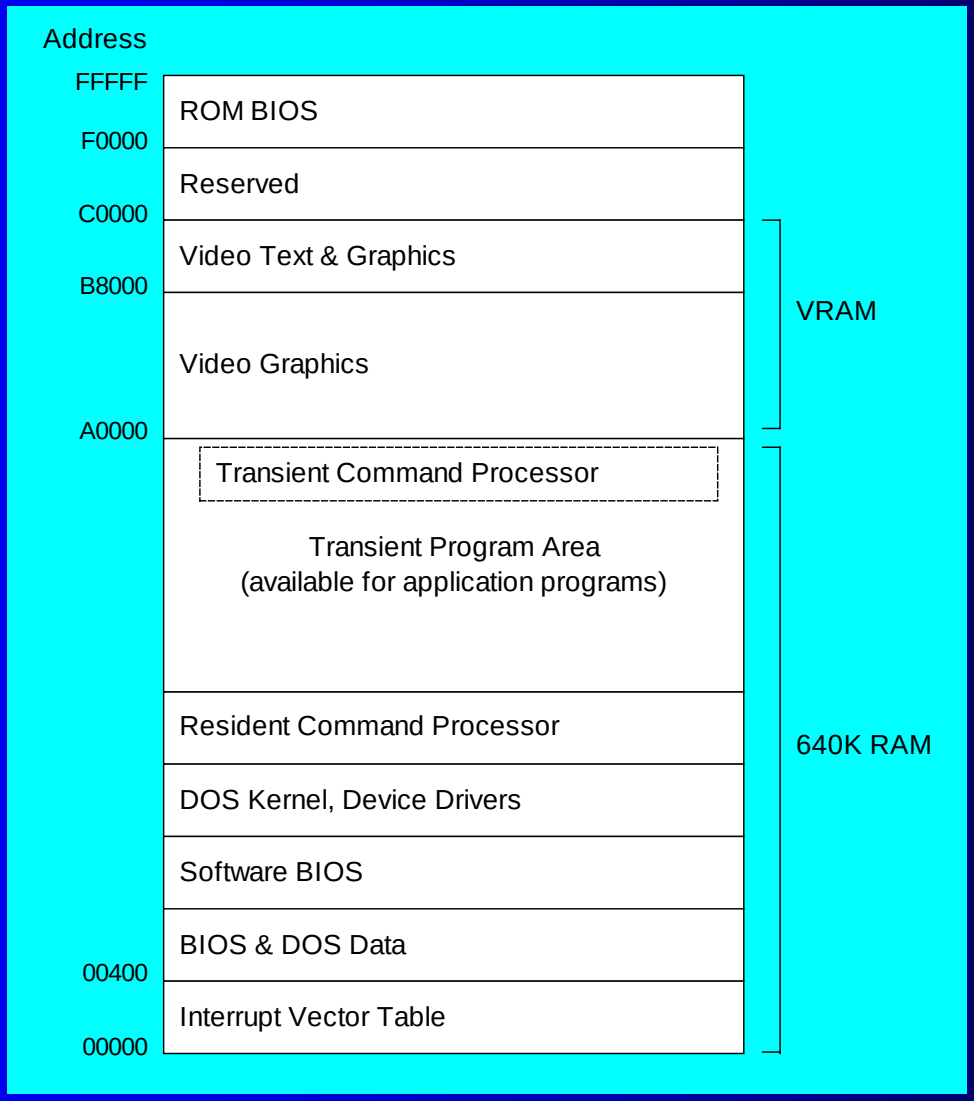
Your turn . . .

What segment addresses correspond to the linear address 28F30h?

Many different segment-offset addresses can produce the linear address 28F30h. For example:

28F0:0030, 28F3:0000, 28B0:0430, . . .

MS-DOS Memory Map



Redirecting Input-Output (1 of 2)

- Input-output devices and files are interchangeable
- Three primary types of I/O:
  - Standard input (console, keyboard)
  - Standard output (console, display)
  - Standard error (console, display)
- Symbols borrowed from Unix:
  - < symbol: *get input from*
  - > symbol: *send output to*
  - | symbol: pipe output from one process to another
- Predefined device names:
  - PRN, CON, LPT1, LPT2, NUL, COM1, COM2

Redirecting Input-Output (2 of 2)

- Standard input, standard output can both be redirected
- Standard error cannot be redirected
- Suppose we have created a program named myprog.exe that reads from standard input and writes to standard output. Following are MS-DOS commands that demonstrate various types of redirection:

```
myprog < infile.txt

myprog > outfile.txt

myprog < infile.txt > outfile.txt
```

INT Instruction

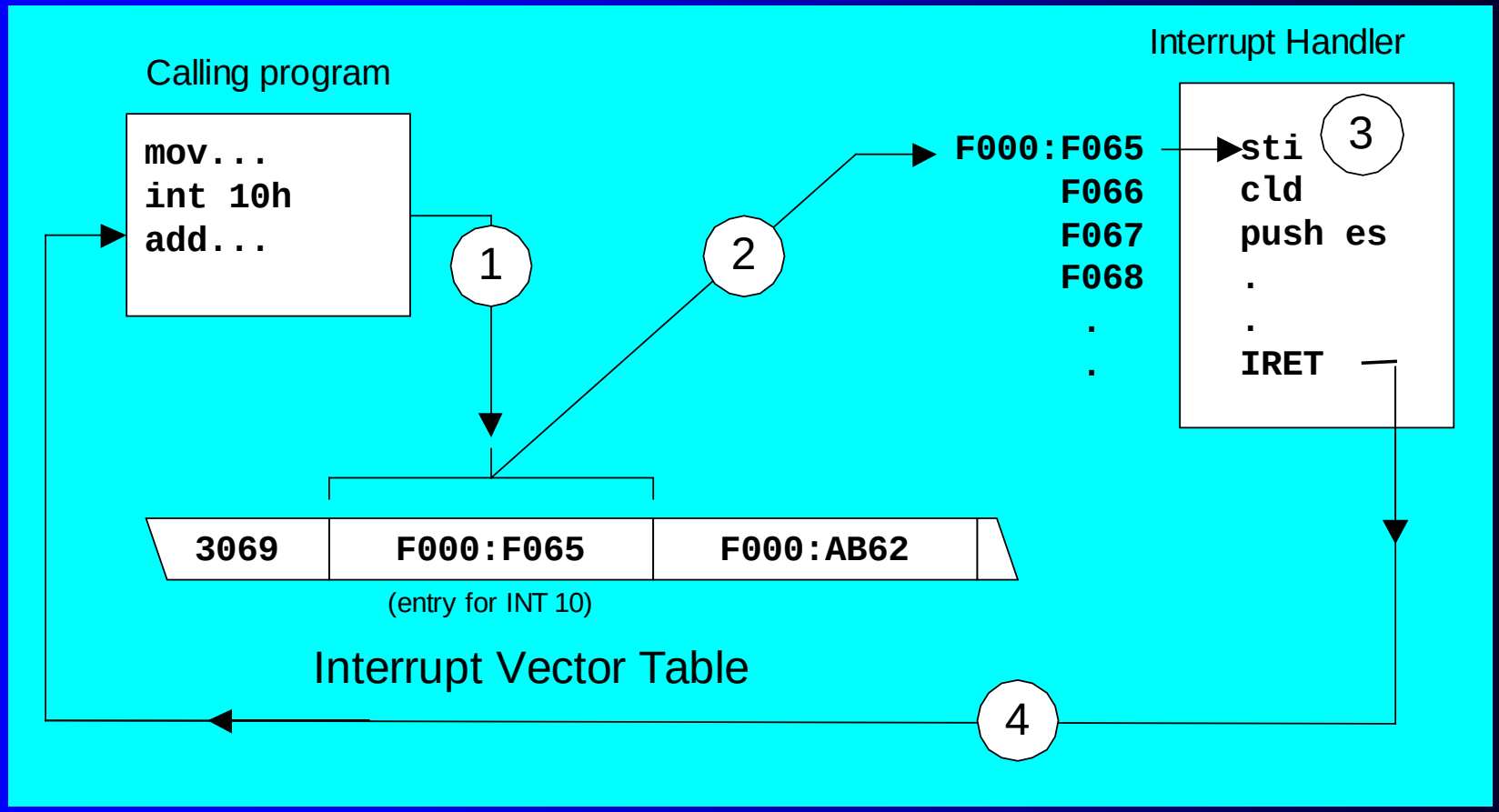
- The INT instruction executes a software interrupt.
- The code that handles the interrupt is called an interrupt handler.
- Syntax:

```
INT number
(number = 0..FFh)
```

The Interrupt Vector Table (IVT) holds a 32-bit segment-offset address for each possible interrupt handler.

Interrupt Service Routine (ISR) is another name for interrupt handler.

Interrupt Vectoring Process



Common Interrupts

- INT 10h Video Services
- INT 16h Keyboard Services
- INT 17h Printer Services
- INT 1Ah Time of Day
- INT 1Ch User Timer Interrupt
- INT 21h MS-DOS Services

What's Next

- MS-DOS and the IBM-PC
- **MS-DOS Function Calls (INT 21h)**
- Standard MS-DOS File I/O Services

MS-DOS Function Calls (INT 21h)

- ASCII Control Characters
- Selected Output Functions
- Selected Input Functions
- Example: String Encryption
- Date/Time Functions

INT 4Ch: Terminate Process

- Ends the current process (program), returns an optional 8-bit return code to the calling process.
- A return code of 0 usually indicates successful completion.

```
mov ah,4Ch              ; terminate process
mov al,0                ; return code
int 21h

; Same as:

.EXIT 0
```

Selected Output Functions

- ASCII control characters
- 02h, 06h - Write character to standard output
- 05h - Write character to default printer
- 09h - Write string to standard output
- 40h - Write string to file or device

ASCII Control Characters

Many INT 21h functions act upon the following control characters:

- 08h - Backspace (moves one column to the left)
- 09h - Horizontal tab (skips forward n columns)
- 0Ah - Line feed (moves to next output line)
- 0Ch - Form feed (moves to next printer page)
- 0Dh - Carriage return (moves to leftmost output column)
- 1Bh - Escape character

INT 21h Functions 02h and 06h:
Write Character to Standard Output

Write the letter 'A' to standard output:

```
mov ah,02h
mov dl,'A'
int 21h
```

or: mov ah,2

Write a backspace to standard output:

```
mov ah,06h
mov dl,08h
int 21h
```

INT 21h Function 05h:
Write Character to Default Printer

Write the letter 'A':

```
mov ah,05h
mov dl,65
int 21h
```

Write a horizontal tab:

```
mov ah,05h
mov dl,09h
int 21h
```

INT 21h Function 09h:
Write String to Standard Output

- The string must be terminated by a '$' character.
- DS must point to the string's segment, and DX must contain the string's offset:

```
.data
string BYTE "This is a string$"

.code
mov  ah,9
mov  dx,OFFSET string
int  21h
```

INT 21h Function 40h:
Write String to File or Device

Input: BX = file or device handle (console = 1), CX = number of bytes to write, DS:DX = address of array

```
.data
message "Writing a string to the console"
bytesWritten WORD ?

.code
    mov ah,40h
    mov bx,1
    mov cx,LENGTHOF message
    mov dx,OFFSET message
    int 21h
    mov bytesWritten,ax
```

Selected Input Functions

- 01h, 06h - Read character from standard input
- 0Ah - Read array of buffered characters from standard input
- 0Bh - Get status of the standard input buffer
- 3Fh - Read from file or device

INT 21h Function 01h:
Read single character from standard input

- Echoes the input character
- Waits for input if the buffer is empty
- Checks for Ctrl-Break (^C)
- Acts on control codes such as horizontal Tab

```
.data
char BYTE ?
.code
mov ah,01h
int 21h
mov char,al
```

INT 21h Function 06h:
Read character from standard input without waiting

- Does not echo the input character
- Does not wait for input (use the Zero flag to check for an input character)
- Example: repeats loop until a character is pressed.

```
.data
char BYTE ?
.code
L1: mov  ah,06h          ; keyboard input
    mov  dl,0FFh         ; don't wait for input
    int  21h
    jz   L1              ; no character? repeat loop
    mov  char,al         ; character pressed: save it
    call DumpRegs        ; display registers
```

INT 21h Function 0Ah:
Read buffered array from standard input (1 of 2)

- Requires a predefined structure to be set up that describes the maximum input size and holds the input characters.
- Example:

```
count = 80

KEYBOARD STRUCT
   maxInput BYTE count          ; max chars to input
   inputCount BYTE ?            ; actual input count
   buffer BYTE count DUP(?)     ; holds input chars
KEYBOARD ENDS
```

INT 21h Function 0Ah (2 of 2)

Executing the interrupt:

```
.data
kybdData KEYBOARD <>

.code
    mov ah,0Ah
    mov dx,OFFSET kybdData
    int 21h
```

INT 21h Function 0Bh:
Get status of standard input buffer

- Can be interrupted by Ctrl-Break (^C)
- Example: loop until a key is pressed. Save the key in a variable:

```
L1: mov ah,0Bh       ; get buffer status
    int 21h
    cmp al,0         ; buffer empty?
    je  L1           ; yes: loop again
    mov ah,1         ; no: input the key
    int 21h
    mov char,al      ; and save it
```

Example: String Encryption

Reads from standard input, encrypts each byte, writes to standard output.

```
XORVAL = 239                ; any value between 0-255
.code
main PROC
     mov  ax,@data
     mov  ds,ax
L1:  mov  ah,6              ; direct console input
     mov  dl,0FFh           ; don't wait for character
     int  21h               ; AL = character
     jz   L2                ; quit if ZF = 1 (EOF)
     xor  al,XORVAL
     mov  ah,6              ; write to output
     mov  dl,al
     int  21h
     jmp  L1                ; repeat the loop
L2:  exit
```

INT 21h Function 3Fh:
Read from file or device

- Reads a block of bytes.
- Can be interrupted by Ctrl-Break (^C)
- Example: Read string from keyboard:

```
.data
inputBuffer BYTE 127 dup(0)
bytesRead WORD ?
.code
mov  ah,3Fh
mov  bx,0                       ; keyboard handle
mov  cx,127                     ; max bytes to read
mov  dx,OFFSET inputBuffer      ; target location
int  21h
mov  bytesRead,ax               ; save character count
```

Date/Time Functions

- 2Ah - Get system date
- 2Bh - Set system date *
- 2Ch - Get system time
- 2Dh - Set system time *

* may be restricted by your user profile if running a console window under Windows NT, 2000, and XP.

INT 21h Function 2Ah:
Get system date

- Returns year in CX, month in DH, day in DL, and day of week in AL

```
mov  ah,2Ah
int  21h
mov  year,cx
mov  month,dh
mov  day,dl
mov  dayOfWeek,al
```

INT 21h Function 2Bh:
Set system date

- Sets the system date. AL = 0 if the function was not successful in modifying the date.

```
mov  ah,2Bh
mov  cx,year
mov  dh,month
mov  dl,day
int  21h
cmp  al,0
jne  failed
```

INT 21h Function 2Ch:
Get system time

- Returns hours (0-23) in CH, minutes (0-59) in CL, and seconds (0-59) in DH, and hundredths (0-99) in DL.

```
mov   ah,2Ch
int   21h
mov   hours,ch
mov   minutes,cl
mov   seconds,dh
```

INT 21h Function 2Dh:
Set system time

- Sets the system date. AL = 0 if the function was not successful in modifying the time.

```
mov  ah,2Dh
mov  ch,hours
mov  cl,minutes
mov  dh,seconds
int  21h
cmp  al,0
jne  failed
```

Example: Displaying the Date and Time

- Displays the system date and time, using INT 21h Functions 2Ah and 2Ch.
- Demonstrates simple date formatting
- View the source code
- Sample output:

```
Date: 12-8-2001,   Time: 23:01:23
```

ToDo: write a procedure named ShowDate that displays any date in mm-dd-yyyy format.

What's Next

- MS-DOS and the IBM-PC
- MS-DOS Function Calls (INT 21h)
- **Standard MS-DOS File I/O Services**

Standard MS-DOS File I/O Services

- 716Ch - Create or open file
- 3Eh - Close file handle
- 42h - Move file pointer
- 5706h - Get file creation date and time
- Selected Irvine16 Library Procedures
- Example: Read and Copy a Text File
- Reading the MS-DOS Command Tail
- Example: Creating a Binary File

INT 21h Function 716Ch:
Create or open file

- AX = 716Ch
- BX = access mode (0 = read, 1 = write, 2 = read/write)
- CX = attributes (0 = normal, 1 = read only, 2 = hidden, 3 = system, 8 = volume ID, 20h = archive)
- DX = action (1 = open, 2 = truncate, 10h = create)
- DS:SI = segment/offset of filename
- DI = alias hint (optional)

Example: Create a New File

```
mov  ax,716Ch                ; extended open/create
mov  bx,2                    ; read-write
mov  cx,0                    ; normal attribute
mov  dx,10h + 02h            ; action: create + truncate
mov  si,OFFSET Filename
int  21h
jc   failed
mov  handle,ax               ; file handle
mov  actionTaken,cx          ; action taken to open file
```

Example: Open an Existing File

```
mov  ax,716Ch                ; extended open/create
mov  bx,0                    ; read-only
mov  cx,0                    ; normal attribute
mov  dx,1                    ; open existing file
mov  si,OFFSET Filename
int  21h
jc   failed
mov  handle,ax               ; file handle
mov  actionTaken,cx          ; action taken to open file
```

INT 21h Function 3Eh:
Close file handle

- Use the same file handle that was returned by INT 21h when the file was opened.
- Example:

```
.data
filehandle WORD ?
.code
    mov  ah,3Eh
    mov  bx,filehandle
    int  21h
    jc   failed
```

INT 21h Function 42h:
Move file pointer

Permits random access to a file (text or binary).

```
mov   ah,42h
mov   al,0            ; offset from beginning
mov   bx,handle
mov   cx,offsetHi
mov   dx,offsetLo
int   21h
```

AL indicates how the pointer's offset is calculated:

0: Offset from the beginning of the file
1: Offset from the current pointer location
2: Offset from the end of the file

INT 21h Function 5706h:
Get file creation date and time

- Obtains the date and time when a file was created (not necessarily the same date and time when the file was last modified or accessed.)

```
mov ax,5706h
mov bx,handle        ; handle of open file
int 21h
jc  error
mov date,dx
mov time,cx
mov milliseconds,si
```

Selected Irvine16 Library Procedures

- 16-Bit ReadString procedure
- 16-Bit WriteString procedure

ReadString Procedure

The ReadString procedure from the Irvine16 library reads a string from standard input and returns a null-terminated string. When calling it, pass a pointer to a buffer in DX. Pass a count of the maximum number of characters to input, plus 1, in CX. Writestring inputs the string from the user, returning when either of the following events occurs:

1. CX –1 characters were entered.
2. The user pressed the Enter key.

```
.data
buffer BYTE 20 DUP(?)
.code
mov  dx,OFFSET buffer
mov  cx,LENGTHOF buffer
call ReadString
```

ReadString Procedure

You can also call it using 32-bit registers:

```
.data
buffer BYTE 20 DUP(?)
.code
mov  edx,OFFSET buffer
mov  ecx,LENGTHOF buffer
call ReadString
```

ReadString returns a count of the number of characters actually read in the EAX register.

ReadString Implementation

```
ReadString PROC
    push cx                 ; save registers
    push si
    push cx                 ; save character count
    mov  si,dx              ; point to input buffer
    dec  cx                 ; save room for null byte
L1: mov  ah,1               ; function: keyboard input
    int  21h                ; returns character in AL
    cmp  al,0Dh             ; end of line?
    je   L2                 ; yes: exit
    mov  [si],al            ; no: store the character
    inc  si                 ; increment buffer pointer
    loop L1                 ; loop until CX=0
L2: mov  BYTE PTR [si],0    ; insert null byte
    pop  ax                 ; original digit count
    sub  ax,cx              ; AX = size of input string
    pop  si                 ; restore registers
    pop  cx
    ret
ReadString ENDP             ; returns AX = size of string
```

16-Bit WriteString Procedure

Receives: DX contains the offset of a null-terminated string.

```
WriteString PROC
   pusha
   INVOKE Str_length,dx  ; AX = string length
   mov  cx,ax            ; CX = number of bytes
   mov  ah,40h           ; write to file or device
   mov  bx,1             ; standard output handle
   int  21h              ; call MS-DOS
   popa
   ret
WriteString ENDP
```

(May be different from the version printed on page 482.)

Example: Read and Copy a Text File

- The *Readfile.asm* program demonstrates several INT 21h functions:
  - Function 716Ch: Create new file or open existing file
  - Function 3Fh: Read from file or device
  - Function 40h: Write to file or device
  - Function 3Eh: Close file handle

View the source code

Reading the MS-DOS Command Tail

- When a program runs, any additional text on its command line is automatically stored in the 128-byte MS-DOS command tail area, at offset 80h in the program segment prefix (PSP).
- Example: run a program named attr.exe and pass it "FILE1.DOC" as the command tail:

Offset:	80	81	82	83	84	85	86	87	88	89	8A	8B
Contents:	0A	20	46	49	4C	45	31	2E	44	4F	43	0D
			F	I	L	E	1	.	D	O	C	

View the Get_CommandTail library procedure source code.

Example: Creating a Binary File

- A binary file contains fields that are are generally not recognizable when displayed on the screen.
- Advantage: Reduces I/O processing time
  - Example: translating a 5-digit ASCII integer to binary causes approximately 100 instructions to execute.
- Disadvantage: may require more disk space
  - Example: array of 4 doublewords:
    - "795 43 1234 2" - requires 13 bytes in ASCII
    - requires 16 bytes in binary

Summary

- MS-DOS applications
  - 16-bit segments, segmented addressing, running in real-address mode
  - complete access to memory and hardware
- Software interrupts
  - processed by interrupt handlers
- INT (call to interrrupt procedure) instruction
  - pushes flags & return address on the stack
  - uses interrupt vector table to find handler
- Program Segment Prefix (PSP)
- BIOS Services (INT 10h, INT 16h, INT 17h, ...)
- MS-DOS Services (INT 21h)

The End