Week 6
Chapter 5: Stack and Procedures

Class 16

- Stack Operations
- Defining and Using Procedures
- **Linking to an External Library**
- The Irvine32 Library
- 64-Bit Assembly Programming

From Book's Page No 153 to 181

Linking to an External Library

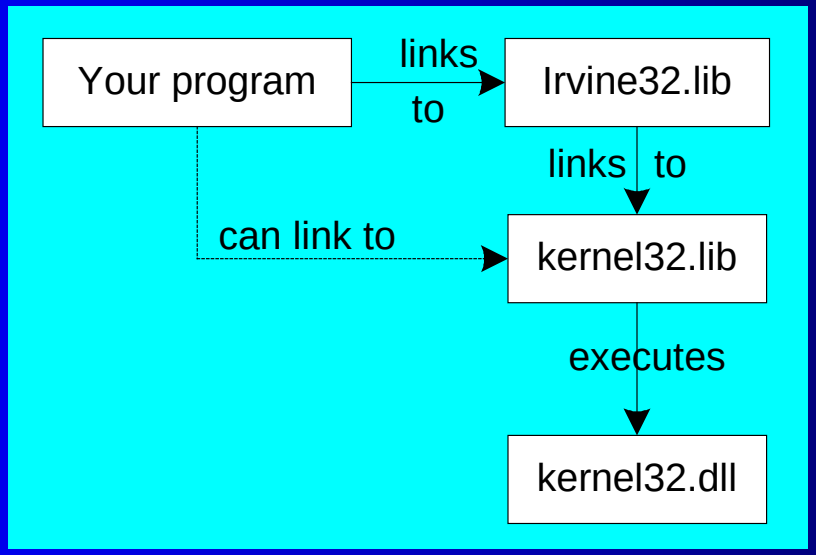
- What is a Link Library?
- How the Linker Works

What is a Link Library?

- A file containing procedures that have been compiled into machine code
  - constructed from one or more OBJ files
- To build a library, . . .
  - start with one or more ASM source files
  - assemble each into an OBJ file
  - create an empty library file (extension .LIB)
  - add the OBJ file(s) to the library file, using the Microsoft LIB utility

How The Linker Works

- Your programs link to Irvine32.lib using the linker command inside a batch file named make32.bat.
- Notice the two LIB files: Irvine32.lib, and kernel32.lib
  - the latter is part of the Microsoft *Win32 Software Development Kit (SDK)*



What's Next

- Stack Operations
- Defining and Using Procedures
- Linking to an External Library
- **The Irvine32 Library**
- 64-Bit Assembly Programming

Calling Irvine32 Library Procedures

- Call each procedure using the CALL instruction. Some procedures require input arguments. The INCLUDE directive copies in the procedure prototypes (declarations).
- The following example displays "1234" on the console:

```
INCLUDE Irvine32.inc
.code
   mov  eax,1234h          ; input argument
   call WriteHex           ; show hex number
   call Crlf               ; end of line
```

Library Procedures - Overview (1 of 4)

CloseFile – Closes an open disk file

Clrscr - Clears console, locates cursor at upper left corner

CreateOutputFile - Creates new disk file for writing in output mode

Crlf - Writes end of line sequence to standard output

Delay - Pauses program execution for *n* millisecond interval

DumpMem - Writes block of memory to standard output in hex

DumpRegs – Displays general-purpose registers and flags (hex)

GetCommandtail - Copies command-line args into array of bytes

GetDateTime – Gets the current date and time from the system

GetMaxXY - Gets number of cols, rows in console window buffer

GetMseconds - Returns milliseconds elapsed since midnight

Library Procedures - Overview (2 of 4)

GetTextColor - Returns active foreground and background text colors in the console window

Gotoxy - Locates cursor at row and column on the console

IsDigit - Sets Zero flag if AL contains ASCII code for decimal digit (0–9)

MsgBox, MsgBoxAsk – Display popup message boxes

OpenInputFile – Opens existing file for input

ParseDecimal32 – Converts unsigned integer string to binary

ParseInteger32 - Converts signed integer string to binary

Random32 - Generates 32-bit pseudorandom integer in the range 0 to FFFFFFFFh

Randomize - Seeds the random number generator

RandomRange - Generates a pseudorandom integer within a specified range

ReadChar - Reads a single character from standard input

Library Procedures - Overview (3 of 4)

ReadDec - Reads 32-bit unsigned decimal integer from keyboard

ReadFromFile – Reads input disk file into buffer

ReadHex - Reads 32-bit hexadecimal integer from keyboard

ReadInt - Reads 32-bit signed decimal integer from keyboard

ReadKey – Reads character from keyboard input buffer

ReadString - Reads string from stdin, terminated by [Enter]

SetTextColor - Sets foreground/background colors of all subsequent text output to the console

Str_compare – Compares two strings

Str_copy – Copies a source string to a destination string

Str_length – Returns the length of a string in EAX

Str_trim - Removes unwanted characters from a string.

Library Procedures - Overview (4 of 4)

Str_ucase - Converts a string to uppercase letters.

WaitMsg - Displays message, waits for Enter key to be pressed

WriteBin - Writes unsigned 32-bit integer in ASCII binary format.

WriteBinB – Writes binary integer in byte, word, or doubleword format

WriteChar - Writes a single character to standard output

WriteDec - Writes unsigned 32-bit integer in decimal format

WriteHex - Writes an unsigned 32-bit integer in hexadecimal format

WriteHexB – Writes byte, word, or doubleword in hexadecimal format

WriteInt - Writes signed 32-bit integer in decimal format

Library Procedures - Overview (5 of 4)

WriteStackFrame - Writes the current procedure's stack frame to the console.

WriteStackFrameName - Writes the current procedure's name and stack frame to the console.

WriteString - Writes null-terminated string to console window

WriteToFile - Writes buffer to output file

WriteWindowsMsg - Displays most recent error message generated by MS-Windows

Example 1

Clear the screen, delay the program for 500 milliseconds, and dump the registers and flags.

```
.code
    call Clrscr
    mov  eax,500
    call Delay
    call DumpRegs
```

Sample output:

```
EAX=00000613 EBX=00000000 ECX=000000FF EDX=00000000
ESI=00000000 EDI=00000100 EBP=0000091E ESP=000000F6
EIP=00401026 EFL=00000286 CF=0 SF=1 ZF=0 OF=0
```

Example 2

Display a null-terminated string and move the cursor to the beginning of the next screen line.

```
.data
str1 BYTE "Assembly language is easy!",0

.code
   mov  edx,OFFSET str1
   call WriteString
   call Crlf
```

Example 2a

Display a null-terminated string and move the cursor to the beginning of the next screen line (use embedded CR/LF)

```
.data
str1 BYTE "Assembly language is easy!",0Dh,0Ah,0

.code
   mov  edx,OFFSET str1
   call WriteString
```

Example 3

Display an unsigned integer in binary, decimal, and hexadecimal, each on a separate line.

```
IntVal = 35
.code
    mov  eax,IntVal
    call WriteBin           ; display binary
    call Crlf
    call WriteDec           ; display decimal
    call Crlf
    call WriteHex           ; display hexadecimal
    call Crlf
```

Sample output:

```
0000 0000 0000 0000 0000 0000 0010 0011
35
23
```

Example 4

Input a string from the user. EDX points to the string and ECX specifies the maximum number of characters the user is permitted to enter.

```
.data
fileName BYTE 80 DUP(0)

.code
   mov  edx,OFFSET fileName
   mov  ecx,SIZEOF fileName – 1
   call ReadString
```

A null byte is automatically appended to the string.

Example 5

Generate and display ten pseudorandom signed integers in the range 0 – 99. Pass each integer to WriteInt in EAX and display it on a separate line.

```
.code
    mov ecx,10                  ; loop counter

L1: mov  eax,100                ; ceiling value
    call RandomRange            ; generate random int
    call WriteInt               ; display signed int
    call Crlf                   ; goto next display line
    loop L1                     ; repeat loop
```

Example 6

Display a null-terminated string with yellow characters on a blue background.

```
.data
str1 BYTE "Color output is easy!",0

.code
   mov  eax,yellow + (blue * 16)
   call SetTextColor
   mov  edx,OFFSET str1
   call WriteString
   call Crlf
```

The background color is multiplied by 16 before being added to the foreground color.

What's Next

- Stack Operations
- Defining and Using Procedures
- Linking to an External Library
- The Irvine32 Library
- **64-Bit Assembly Programming**

64-Bit Assembly Programming

- The Irvine64 Library
- Calling 64-Bit Subroutines
- The x64 Calling Convention

The Irvine64 Library

- Crlf: Writes an end-of-line sequence to the console.
- Random64: Generates a 64-bit pseudorandom integer.
- Randomize: Seeds the random number generator with a unique value.
- ReadInt64: Reads a 64-bit signed integer from the keyboard.
- ReadString: Reads a string from the keyboard.
- Str_compare: Compares two strings in the same way as the CMP instruction.
- Str_copy: Copies a source string to a target location.
- Str_length: Returns the length of a null-terminated string in RAX.
- WriteInt64: Displays the contents in the RAX register as a 64-bit signed decimal integer.

The Irvine64 Library (cont'd)

- WriteHex64: Displays the contents of the RAX register as a 64-bit hexadecimal integer.
- WriteHexB: Displays the contents of the RAX register as an 8-bit hexadecimal integer .
- WriteString: Displays a null-terminated ASCII string.

Calling 64-Bit Subroutines

- Place the first four parameters in registers
- Add PROTO directives at the top of your program
  - examples:

```
ExitProcess PROTO   ; located in the Windows API
WriteHex64 PROTO    ; located in the Irvine64 library
```

The x64 Calling Convention

- Must use this with the 64-bit Windows API
- CALl instruction subtracts 8 from RSP
- First four parameters must be placed in RCX, RDX, R8, and R9
- Caller must allocate at least 32 bytes of shadow space on the stack
- When calling a subroutine, the stack pointer must be aligned on a 16-byte boundary.

See the CallProc_64.asm example program.

Summary

- Procedure – named block of executable code
- Runtime stack – LIFO structure
  - holds return addresses, parameters, local variables
  - PUSH – add value to stack
  - POP – remove value from stack
- Use the Irvine32 library for all standard I/O and data conversion
  - Want to learn more? Study the library source code in the c:\Irvine\Examples\Lib32 folder