Assembly Language for Intel-Based Computers, 4th Edition

Kip R. Irvine

Chapter 5: Procedures

Lecture 18 Linking to External Library
The Book's Link Library
Stack Operations

Slides prepared by Kip R. Irvine

Revision date: 08/22/2002

Modified by Dr. Nikolay Metodiev Sirakov

October 25, 2009

- Chapter corrections (Web) Assembly language sources (Web)

The Book's Link Library

- Link Library Overview
- Calling a Library Procedure
- Linking to a Library
- Library Procedures – Overview
- Six Examples

Link Library Overview

- 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

Take a quick look at Irvine32.asm by clicking on Examples at the bottom of this screen.

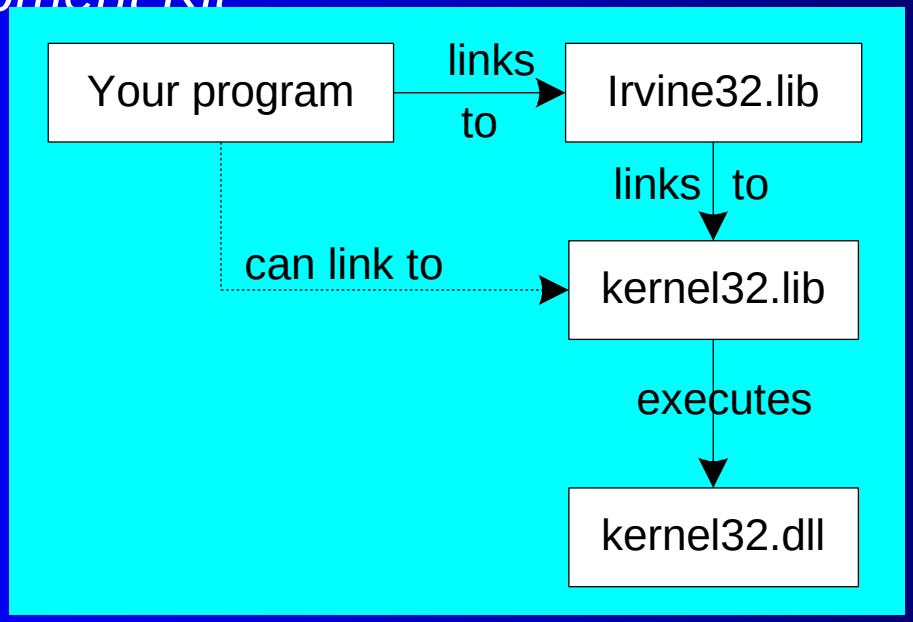
Calling a Library Procedure

- Call a library 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
```

Linking to a Library

- 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 Devlopment Kit*



Library Procedures - Overview (1 of 3)

Clrscr - Clears the console and locates the cursor at the upper left corner.

Crlf - Writes an end of line sequence to standard output.

Delay - Pauses the program execution for a specified *n* millisecond interval.

DumpMem - Writes a block of memory to standard output in hexadecimal.

DumpRegs - Displays the EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EFLAGS, and EIP registers in hexadecimal. Also displays the Carry, Sign, Zero, and Overflow flags.

GetCommandtail - Copies the program's command-line arguments (called the *command tail*) into an array of bytes.

GetMseconds - Returns the number of milliseconds that have elapsed since midnight.

Library Procedures - Overview (2 of 3)

Gotoxy - Locates cursor at row and column on the console.

Random32 - Generates a 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.

ReadHex - Reads a 32-bit hexadecimal integer from standard input, terminated by the Enter key.

ReadInt - Reads a 32-bit signed decimal integer from standard input, terminated by the Enter key.

ReadString - Reads a string from standard input, terminated by the Enter key.

Library Procedures - Overview (3 of 3)

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

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

WriteBin - Writes an unsigned 32-bit integer to standard output in ASCII binary format.

WriteChar - Writes a single character to standard output.

WriteDec - Writes an unsigned 32-bit integer to standard output in decimal format.

WriteHex - Writes an unsigned 32-bit integer to standard output in hexadecimal format.

WriteInt - Writes a signed 32-bit integer to standard output in decimal format.

WriteString - Writes a null-terminated string to standard output.

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 3

Display the same unsigned integer in binary, decimal, and hexadecimal. Each number is displayed on a separate line.

```
IntVal = 35                    ; constant
.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 memory area where the string will be stored and ECX specifies the maximum number of characters the user is permitted to enter +1.

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

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

Example 5

Generate and display ten pseudorandom signed integers in the range 0 – 99. Each integer is passed to WriteInt in EAX and displayed 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
```

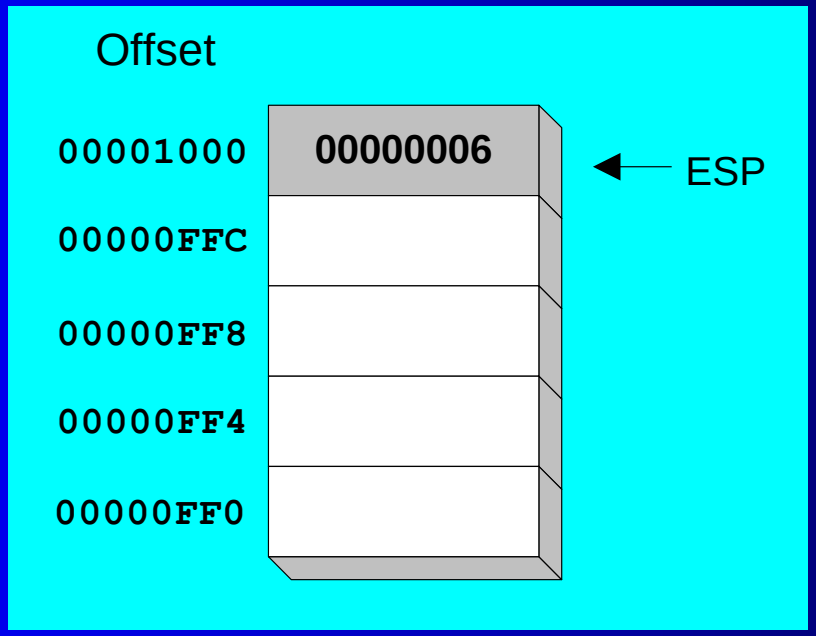
The background color must be multiplied by 16 before you add it to the foreground color.

Stack Operations

- Runtime Stack
- PUSH Operation
- POP Operation
- PUSH and POP Instructions
- Using PUSH and POP
- Example: Reversing a String
- Related Instructions

Runtime Stack

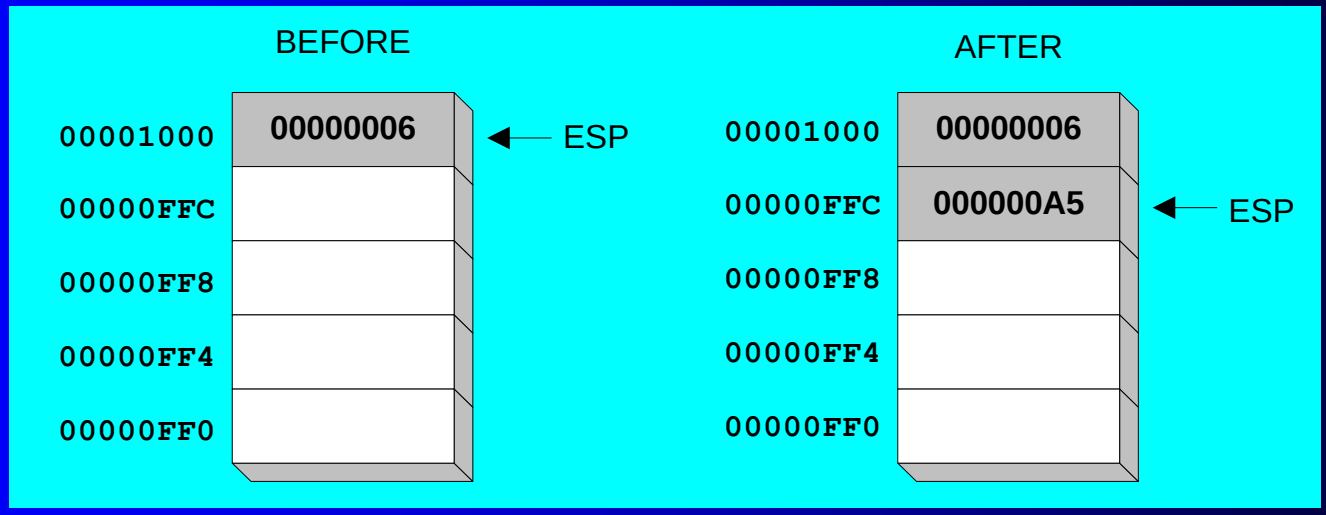
- Managed by the CPU, using two registers
  - SS (stack segment)
  - ESP (stack pointer) *



* SP in Real-address mode

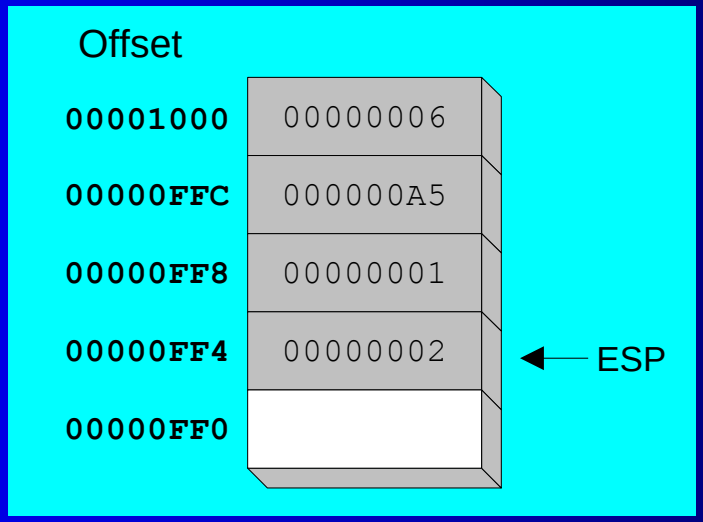
PUSH Operation (1 of 2)

- A 32-bit push operation decrements the stack pointer by 4 and copies a value into the location pointed to by the stack pointer.



PUSH Operation (2 of 2)

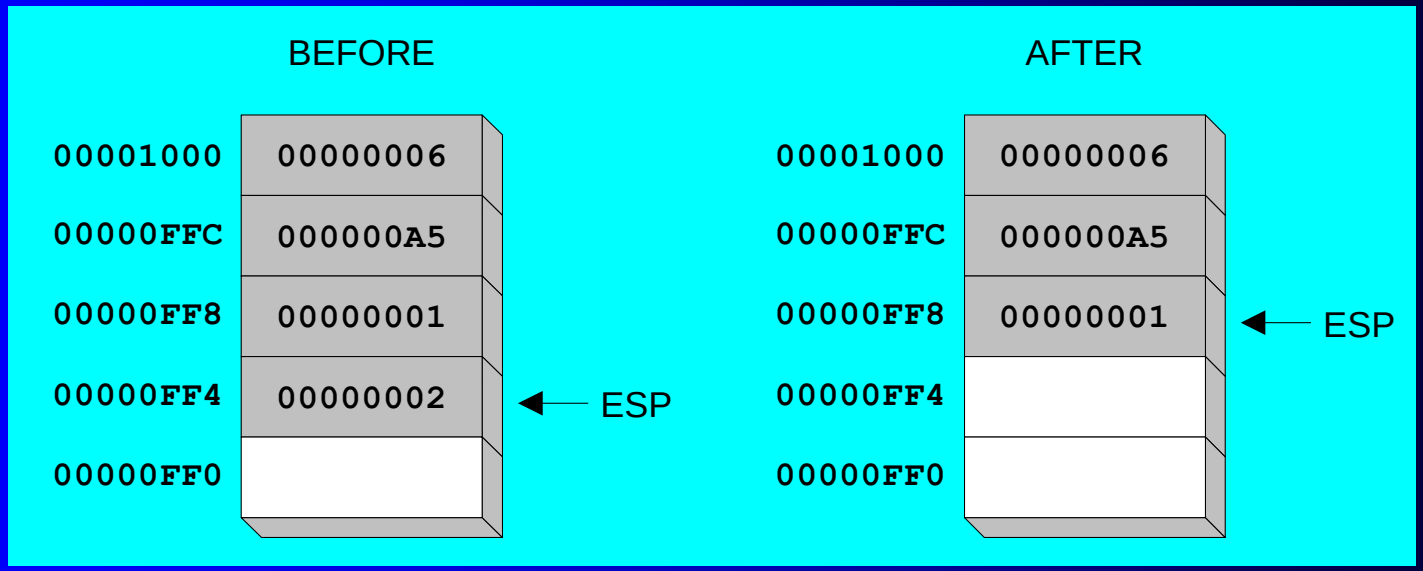
- This is the same stack, after pushing two more integers:



The stack grows downward. The area below ESP is always available (unless the stack has overflowed).

POP Operation

- Copies value at stack[ESP] into a register or variable.
- Adds *n* to ESP, where *n* is either 2 or 4.
  - depends on the attribute of the operand receiving the data



PUSH and POP Instructions

- PUSH syntax:
  - PUSH *r/m16*
  - PUSH *r/m32*
  - PUSH *imm32*
- POP syntax:
  - POP *r/m16*
  - POP *r/m32*

Using PUSH and POP

Save and restore registers when they contain important values. Note that the PUSH and POP instructions are in the opposite order:

```
push esi                          ; push registers
push ecx
push ebx

mov esi,OFFSET dwordVal           ; starting OFFSET
mov ecx,LENGTHOF dwordVal         ; number of units
mov ebx,TYPE dwordVal             ; size of a doubleword
call DumpMem                      ; display memory

pop ebx                           ; opposite order
pop ecx
pop esi
```

Example: Nested Loop

Remember the nested loop we created on page 129? It's easy to push the outer loop counter before entering the inner loop:

```
    mov ecx,100             ; set outer loop count
L1:                         ; begin the outer loop
    push ecx                ; save outer loop count

    mov ecx,20              ; set inner loop count
L2:                         ; begin the inner loop
    ;
    ;
    loop L2                 ; repeat the inner loop

    pop ecx                 ; restore outer loop count
    loop L1                 ; repeat the outer loop
```

Example: Reversing a String

- Use a loop with indexed addressing
- Push each character on the stack
- Start at the beginning of the string, pop the stack in reverse order, insert each character into the string
- Source code

- Q: Why must each character be put in EAX before it is pushed?

> Because only word (16-bit) or doubleword (32-bit) values can be pushed on the stack.

Your turn . . .

- Using the String Reverse program as a starting point,
- #1: Modify the program so the user can input a string of up to 50 characters.
- #2: Modify the program so it inputs a list of 32-bit integers from the user, and then displays the integers in reverse order.

Related Instructions

- PUSHFD and POPFD
  - push and pop the EFLAGS register
- PUSHAD pushes the 32-bit general-purpose registers on the stack
  - order: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI
- POPAD pops the same registers off the stack in reverse order
  - PUSHA and POPA do the same for 16-bit registers

Your Turn . . .

- Write a program that does the following:
  - Assigns integer values to EAX, EBX, ECX, EDX, ESI, and EDI
  - Uses PUSHAD to push the general-purpose registers on the stack
  - Using a loop, the program pops each integer from the stack and displays it on the screen