6. Subroutines, Number I/O, Linking to External Procedures

Background

A good programming practice in any language is to package commonly needed operations into modules that can be reused by other programs. Various names are used for these modules: subroutines, procedures, functions, etc. We will use the terms procedure and subroutine interchangeably to refer to these reusable code modules. This lab will introduce you to the subroutine call/return capability of the x86 instruction set in the context of ASCII number I/O. You will also learn how to use the external procedures that are available in the Irvine link library.

Objectives:

A. Learn how to write subroutines in x86 assembly language.

B. Understand the problems/solutions involved with ASCII Number input/output.

C. Make use of the external procedures in the Irvine link library.

Pre-Lab

1. Read Section 4.7 in the Irvine textbook about how to use the Irvine link library.
2. Read Sections 5.1 through 5.3 in the Irvine textbook about procedures.
3. Explain the difference between a NEAR subroutine call and a FAR subroutine call.

Lab

A. Number Output

The previous lab had a program that would output an 8-bit number to the screen in binary format. For example, the 8-bit value E6h was displayed as "11100110". The program had to look at each bit in the 8-bit number, and send the ASCII equivalent of a '1' (31h) or '0' (30h) to the screen using the DOS single character printing function. For the 8-bit value E6h, this meant that the following ASCII codes were sent to the console: 31h, 31h, 31h, 30h, 30h, 31h, 31h, 30h.

What if we wanted to print the 8-bit value E6h to screen as a HEX number? We would need to send the ASCII codes for 'E' and '6' to the console, or 45h and 36h.

What if we wanted to print the 8-bit value E6h to the screen as an unsigned decimal number? The value E6h as an unsigned decimal number is 14*16 + 6 = 230. The ASCII codes for '2', '3', '0' would have to be sent to the console: 32h, 33h, 30h.

What if we wanted to print the 8-bit value E6h to the screen as a signed decimal number? The value of E6h as a signed decimal number (2's complement representation) is a "-26". The ASCII codes for a minus sign "-', '2', and '6' would be sent to the console: 2Dh, 32h, 36h.

Converting a number to its HEX , BINARY or signed/unsigned DECIMAL representation in ASCII digits for display purposes is a common problem. Writing subroutines to handle different parts of this process is a good method for solving this problem.

The program below is an example of HEX output.

```
.model small
.586
.stack 100h
.data
.code
main          proc
              mov    ax,@data
              mov    ds,ax
              xor    al,al          ;clear ax
              mov    cx,16          ;print all 16 hex digits
lp1:          push   ax
              call   out1hex
              call   pcrlf
              pop    ax
              inc    al
              loop   lp1
              Mov    ax, 4c00h
              Int    21h
Main          endp

Pcrlf         proc                  ;print carriage return/line feed
              Mov    dl,0ah ;line feed
              Mov    ah,2
              Int    21h            ;print it
              Mov    dl,0dh ;carriage return
              Mov    ah,2
              Int    21h
              Ret
Pcrlf  endp

Out1hex       proc                  ;output lower 4-bits of AL as Hex char
              And    al,0fh ;make sure AL value is 0 to F
              Cmp    al,9           ;is 4 bit value above 9?
              Ja     ischar
              Add    al,30h ;convert to ascii digit '0' to '9'
              Jmp    printit
Ischar:       add    al,37h ;convert to ascii digit 'A' to 'F'
Printit:      Mov    dl,al
              Mov    ah,2
              Int    21h            ;print it using DOS single char output
              Ret
Out1hex       endp
End main
```

Assemble this program, execute it and observe this program. How does this program work?

1. The program consists of a main program that calls two subroutines named 'PCRLF' and 'OUT1HEX'.
2. The PCRLF subroutine prints out a carriage return, line feed in order to advance the cursor one line and move it back to the left hand side of the screen.

3. The OUT1HEX subroutine will output the lower 4 bits of register AL as a hex digit '0' to 'F'. If AL is 9 or lower then the value 30h is added to AL to covert it to an ASCII digit '0' to '9' (30h to 39h). To do this check, AL is compared to 9 (cmp al, 9) and then a jump is made to the instruction at 'ischar' if AL is above 9 (ja ischar --- 'ja' stands for 'Jump if Above'). If AL is above 9, the value 37h is added to AL to convert this to the ASCII character 'A' through 'F' (note that if AL is 10 or 0Ah, that 0Ah + 37h = 41h which is ASCII for 'A').
4. The main program tests the OUT1HEX subroutine by calling it for all values of 0 to Fh via a loop that sets AL to 0 and then increments AL by 1 each time through the loop for 16 times. Note that the value of AL is saved on the stack between calls to OUT1HEX via the PUSH/POP instructions because the OUT1HEX subroutine will destroy the value of AL (remember that PUSH/POP can only save 16 bit registers so PUSH AL is illegal).
5. Note that a subroutine is called via the CALL instruction and that a subroutine must have a RET instruction at the end of it in order to return from the subroutine call. To define a subroutine to MASM, it must be bracketed by 'proc' and 'endp' statements as shown. A FAR call is when both the code segment and IP is pushed on the stack (the subroutine can be in a different code segment). A NEAR call is when only the IP is pushed on the stack which means that the subroutine is in the same segment. When the statement '.model small' is used, only one code segment is allowed so all calls are NEAR calls. When the statement '.model medium' is used, multiple code segments are allowed and all calls are by default FAR calls (CS, IP pushed on the stack).

Lab Question 1:

A. Use codeview to examine the machine code of the program above. What is the machine code that gets generated for the CALL OUT1HEX instruction? What is the machine code that gets generated for the RET instruction in the OUT1HEX instruction? What is the logical address of the 'call OUT1HEX' and 'call PCLF' instructions in the program?

B. Set the memory window to point to the STACK SEGMENT:SP value (the stack grows down in memory so you should modify the last line in your memory display to point to this). How does the stack memory area and stack pointer get modified after the first 'PUSH AX' instruction? How does the stack memory area and stack pointer get modified after the first 'CALL OUT1HEX' instruction? DRAW A STACK PICTURE in your lab report that shows this. WARNING: Part of the 'tracing' process by Codeview modifies memory below your current Stack Pointer value. This means that you will see highlighted areas of memory in your stack memory display that indicates changes - these changes are not being done your program but by codeview. You should only be concerned about stack memory changes that are made above or equal to your stack pointer.

C. Change the statement '.model small' to '.model medium'. Make no other changes to the program. Use codeview to look at the machine code -- what is different now about the code generated for the CALL instructions and the RET instructions (compare the machine codes)? Answer the same questions that you answered for 'A'.

D. Answer the same questions that you answered for 'B' with the new code.

Lab Question 2: Modify the program above to have a subroutine called 'OUT2HEX' that calls OUT1HEX twice to printout the 8-bit bit value in register AL as a 2-digit HEX value (see the online lecture notes for this program). Modify the main program to test OUT2HEX with all values between 00 and FFh. Include the assembled listing in your lab report.

B. Using an External Library

We will return to the problems of ASCII number conversion later in this lab. This section will explore using the procedures in the Irvine library provided with the CDROM in your Irvine textbook. Locate the file 'irvine.lib' and copy it to your local directory (can be found the CDROM with the Irvine textbook or on the PCs in the Micro I lab). The Irvine library provides many procedures that you will find useful in this course. Make sure that you read section 4.7 in Irvine.

The program below uses some procedures from the Irvine library to read an ASCII string representing a signed decimal value and display that value to the screen in binary, octal, unsigned decimal, signed decimal and hex.

```
.model small
.586
.stack 100h
.data
prompt db "Enter Signed decimal: ",0
bin    db "Binary: ",0
oct    db "Octal: ",0
udec   db "Unsigned Decimal: ",0
sdec   db "Signed Decimal: ",0
hex    db "Hex: ",0
.code
extrn  Clrscr:proc, Crlf:proc, ReadInt:proc
extrn  Writestring:proc, WriteInt:proc
extrn Writeint_signed:proc


main          proc
              mov    ax,@data
              mov    ds,ax
              call   Clrscr
              mov    dx, offset prompt
              call   Writestring
;get 16-bit signed decimal number, value returns in AX
              call   Readint
              call   Crlf
              mov    dx, offset bin
              call   Writestring
              mov    bx,2
              call   WriteInt            ;display as binary
              call   Crlf
              mov    dx, offset oct
              call   Writestring
              mov    bx, 8
              call   WriteInt            ;display as octal
              call   Crlf
              mov    dx, offset udec
              call   Writestring
              mov    bx,10
              call   Writeint            ;display as unsigned decimal
              call   Crlf
              mov    dx, offset sdec
              call   Writestring
              call   Writeint_signed     ;display as signed decimal
              call   Crlf
              mov    dx, offset hex
              call   Writestring
              mov    bx,16
              call   Writeint            ; display as hex
              Mov    ax, 4c00h
              Int    21h
Main          endp
End           main
```

After you assemble this program, you must specify the "irvine" library when the 'link' program asks you for a library name. Assuming the object filename is 'exam2.obj', you can also do the link without prompting via:

link exam2,,,irvine,,

This assumes that the *irvine.lib* file is in your current directory. Execute the program at least twice and enter values of 20 and -2 . How does this program work? (Section 4.7 of the Irvine text has a complete description of all external procedures listed in this program)

1. The external procedure *Clrscr* is used to clear the screen and put the cursor in the upper left corner.
2. All prompt/message strings that are written to the screen use the external procedure *Writestring*. Strings that are passed to this procedure must be null-terminated, i.e, the last byte has a value of 0h (these are also known as ASCIIZ strings). This is a much more common (and reasonable) way of terminating strings than using a '$' as DOS does.
3. The external procedure *Readint* is used to read an ASCII string that represents a signed decimal number and converts that number to a 16-bit value that is returned in AX.
4. The external procedure *WriteInt* is used to display the number that was entered in binary, octal, unsigned decimal, and hex. The number to be displayed is passed in AX. The value passed in BX determines the base (2 = binary, 8 = octal, 10= decimal, 16 = hex).
5. The external procedure *WriteInt_signed* is used to display the number as a signed decimal number.

Lab Question 3: Included the assembled listing of this program in your lab report.

A. Run the program and enter the value 32767. Record the values that get displayed.

B. Run the program and enter the value 32768. What gets displayed and why?

C. Number Input

The external procedure *ReadInt* is doing more work than might be apparent to you at first. In this section you will write some code that duplicates what *ReadInt* accomplishes. The program below is an incomplete program that uses the Irvine external procedure *Readstring* to get two ASCII strings from a user. The ASCII strings are intended to represent two-digit, unsigned decimal numbers. The procedure calls a subroutine Dec2Hex that should convert the two digit decimal ASCII string to an 8-bit value. The program then adds these two values together and displays the result in decimal.

```
.model small
.586
.stack 100h
.data
prmpta db "Enter first 2-digit decimal string  (xx): ",0
prmptb db "Enter second 2-digit decimal string (yy): ",0
prmptc db "The sum is:  ",0
buffa  db 4 dup (?)
buffb  db 4 dup (?)
.code
extrn  Clrscr:proc, Crlf:proc, Readstring:proc
extrn  Writestring:proc, WriteInt:proc


main          proc
              mov    ax,@data
              mov    ds,ax
              call   Clrscr
              mov    dx, offset prmpta
              call   Writestring
              mov    ax,2
              mov    dx, offset buffa    ;returned characters go here
              call   Readstring
              call   crlf
              mov    dx, offset prmptb
              call   Writestring
              mov    ax,2
              mov    dx, offset buffb    ;returned characters go here
              call   Readstring
              call   crlf
              mov    bx,    offset buffa
              call   dec2hex       ;convert 2 digit ASCII decimal string
              push   ax            ;save converted value
              mov    bx,    offset buffb
              call   dec2hex
              mov    bx,ax
              pop    ax
              add    ax,bx
              push   ax                   ;save value
              mov    dx, offset prmptc
              call   Writestring
              pop    ax
              mov    bx,10
              call   Writeint
              call   Crlf
              Mov    ax, 4c00h
              Int    21h
Main          endp

Dec2hex       proc
              ;; you fill this in…
              ret
dec2hex       endp
End           main
```

The *Readstring* procedure reads characters from the keyboard and stores them in consecutive memory locations starting at the address passed in register DX. So if the string '35' was typed in, the values 30h, 35h would be stored starting at the address pointed to by DX.

Lab Question 4:

A. Assemble and execute this program (do not forget to link in the *irvine.lib* file). If you are not sure how the program works, trace through it with Codeview. The program produces the incorrect result because the 'dec2hex' subroutine is incomplete.

B. Complete the 'dec2hex' subroutine such that it takes the two digit unsigned decimal string pointed to by register 'BX' and converts to it the correct 8-bit value that is returned in AL (register AH must be returned as '0'). The first ASCII digit represents the 10's digit, so subtract 30h from this ASCII byte to get the value of the digit, then multiply it by 10. Take the result and add it to the 2^{nd} ASCII digit (after you subtract 30h from the 2^{nd} ASCII digit). You do not have to worry about any error checking -- you can assume that the ASCII string is always two digits long (if you test you program with 2+3, enter these as '02' , '03'). Comment the operation of your dec2hex subroutine and include the listing file of the complete program in your lab report. You might need to use the "Set Breakpoint" capability of Codeview to debug this program. To set a breakpoint, click on the code window, and use the arrow keys to scroll to the instruction for the breakpoint (click on the address with the mouse). Then use the "Data->Set Breakpoint" command to set a breakpoint. You might want to set a breakpoint at the first instruction of 'dec2hex', then use the F5 (Go) command in codeview to execute the program. When the breakpoint is reached, trace the program from that point and verify that your DEC2HEX subroutine is producing the correct value.

Lab Question 5: Write a new version of the program that you did for Question 4 that allows signed two digit decimal numbers to be entered (+03, -45, etc). One needed change is that your *dec2hex* subroutine can no longer always return 0h in AH -- you need to sign extend AL to a 16 bit value returned in AX (look at the description of the CBW instruction). You also need to use the *Writeint_signed* external procedure to display the sum value. Think about how the '+' and '-' sign characters need to be used by your *dec2hex* subroutine. Include the complete assembled listing of your program in your lab report.

Lab Report

A. Describing What You Have Learned

Include the answers to all "**Lab Questions**" in your report.

B. Applying What You Have Learned

Demo the programs that you wrote for lab questions 2, 4, and 5 to the TA.