4. More work with Memory Addressing, Simple Conditional Jumps, Looping

Background

The programs that we have examined so far have executed code from start to finish in a straight line. Most programs require some kind of loop to repetitively execute a section of code or test a condition and jump to some other location. This lab will introduce you to the following instructions: CMP, JNE/JNZ, LOOP, INC, DEC.

Objective

Learn how to write simple counting loops.

Pre-Lab

Read the descriptions of the following instructions in the back of the Irvine textbook: CMP, JNE/JNZ, LOOP, INC, DEC. Read sections 4.3 of the Irvine textbook and the portions of Chapter 6 in Irvine concerning the CMP and JNE/JNZ instructions.

Lab

A. A Simple Counting Loop (Example 1)

The program below prints out the letter 'A' to the screen 5 times. Enter this program and execute it.

```
.model small
.stack 100h
.data
.code
main  proc
      mov   ax,@data
      mov   ds,ax
      mov   cx,5
LP1:  mov   dl, 41h
      Mov   ah, 2
      Int   21h
      Dec   cx
      Jnz   LP1
      Mov   ax, 4c00h
      Int   21h
Main  endp
End main
```

How does this program work?

1. The program forms a loop that starts with the instruction "LP1: mov dl, 41h" and ends with the instruction "JNZ LP1".

2. The code that is in the loop prints the letter 'A' to the screen. The INT 21H DOS call with AH=2 will print the value of register 'DL' to the screen. The instruction 'mov dl, 41h' loads the value of DL with the ASCII value of the letter 'A'.
3. The CX register is being used to control how many time the loop is executed. The instruction 'mov cx, 5' loads CX with the value 5. The instruction 'dec cx' decrements CX (subtracts one from CX) each time though the loop.
4. The end of the loop is formed by the instruction 'JNZ LP1' which is called a *conditional jump*. The JNZ instruction (Jump on Not Zero) will jump to the instruction at the label 'LP1' if the ZERO Flag = 0 (cleared). If the Zero Flag = '1' (set), then the Jump will not be taken and instruction after the JNZ will be executed.
5. How do we exit the loop? Each time the 'dec cx' instruction is executed, the Flag register is affected. If the CX register is not equal to Zero after the 'dec cx' instruction, then the Zero flag will be '0'. This mean that the JNZ instruction will jump to the instruction at label 'LP1'. When CX finally reaches 0, the the Zero flag will be '1', and the jump WILL NOT BE TAKEN -- this will cause the program to exit the loop.
6. You can also use 'JNE' (jump not equal') as an alternate Mnemonic for JZ.
7. A conditional jump instruction must specify a label name to jump to. The label name must be present in the code somewhere in front of some x86 instruction. The label "LP1" was used in this program. Note that a colon ':' is used with the label in front of the x86 instruction "LP1: mov dl,41h". You cannot use the same label name to label different instructions.

Lab Question 1: Use Codeview to single step through this program. Change the program such that CX is loaded with 300 instead of 5, re-assemble the program and re-execute it to see what happens. Add comments to each X86 program line and explain what that line does. Include the LISTING file of this program in your lab report.

B. The LOOP instruction (Example 2)

Example 1 illustrated a counting loop in which a section of code was executed a fixed number of times. This is very common, and a special instruction called the LOOP instruction can be used as another way of accomplishing this. The program below is a modification of Example 1 in which the LOOP instruction has replaced the DEC and JNZ instructions. The LOOP instruction combines the DEC and JNZ instruction into one instruction - LOOP will decrement CX by 1, and then jump to the labeled instruction if CX is not equal to 0.

```
.model small
.stack 100h
.data
.code
main  proc
      mov   ax,@data
      mov   ds,ax
      mov   cx,5
LP1:  mov   dl, 41h
      Mov   ah, 2
      Int   21h
      Loop  LP1
      Mov   ax, 4c00h
      Int   21h
Main  endp
End main
```

Lab Question 2: Use Codeview to single step through this program using the F8 (TRACE) command. Exit codeview, and now step through using the F10 (STEP) command. What happens? You need to be aware of this difference between TRACE and STEP in codeview. Add comments to each X86 program line and explain what that line does. Include the LISTING file of this program in your lab report.

C. The CMP instruction (Example 3)

Both of the previous examples were 'Count Down' loops in which we counted down to zero by decrementing a register. What if we wanted to count up? We would need some method of checking to see if CX had reached a particular value. The program below is an example of a count UP loop.

```
.model small
.stack 100h
.data
.code
main  proc
      mov   ax,@data
      mov   ds,ax
      mov   cx,0
LP1:  mov   dl, 41h
      Mov   ah, 2
      Int   21h
      inc   cx
      cmp   cx, 5
      Jnz   LP1
      Mov   ax, 4c00h
      Int   21h
Main  endp
End main
```

How does this program work?

1. The CX register is being initialized to 0 via the instruction 'mov cx,0'.
2. Each time through the loop, the CX is being incremented by 1 via the 'inc cx' instruction.
3. We want to exit the loop after it has been executed 5 times. The CMP instruction is being used for this purpose. A CMP instruction does a subtraction, but does not store the result - only the flags are affected. So 'cmp cx,5' does a 'CX - 5' operation. Note that if CX is not equal to 5, then the subtraction result is non-zero (Zero Flag = 0) which causes the 'Jnz LP1' to jump to LP1. If CX = 5, then the subtraction result is '0' (Zero Flag = '1') which cause the loop to not be taken.
4. You can also use the mnemonic 'JNE' (Jump Not Equal). This may be easier to read, since the conditional jump will be taken if CX is not equal to 5.

Lab Question 3: Edit the program to use JNE mnemonic instead of JNZ, and verify its operation by assembling and executing it. Add comments to each X86 program line and explain what that line does. Include the LISTING file of this program in your lab report.

D. Memory Copy (Example 4)

The program below prints out string #1, copies string#1 to string#2, then prints out string #2. Unfortunately, it was written by somebody from Ole Miss so it does not use loops.

```
.model small
.stack 100h
.data
string1  db "Hello","$"
string2  db 128 dup (?)
.code
main  proc
      mov   ax,@data
      mov   ds,ax
      mov   dx, offset string1
      mov   ah,9
      int   21h
      mov   bx, offset string1
      mov   si, offset string2
      mov   al,[bx]                 ;read byte from string1
      mov   [si],al                 ;write byte to string2
      mov   al,[bx+1]
      mov   [si+1],al
      mov   al,[bx+2]
      mov   [si+2],al
      mov   al,[bx+3]
      mov   [si+3],al
      mov   al,[bx+4]
      mov   [si+4],al
      mov   al,[bx+5]
      mov   [si+5],al
      mov   al,[bx+6]
      mov   [si+6],al
      mov   dx, offset string2
      mov   ah,9
      int   21h
      Mov   ax, 4c00h
      Int   21h
Main  endp
End main
```

Lab Question 4: Assemble and execute this program. Then use Codeview to single step through the program (do not proceed to Question #5 until you understand how this program operates). Use the memory window to watch the memory locations for string#2 as they are modified by the program. Include the LISTING file of this program in your lab report.

A. What are the logical addresses for the start of string#1 and string#2?

B. What values get initially loaded into DS? Into SI? Into BX? How does this correspond to the logical addresses for string#1 and string#2?

Lab Question 5: Edit this program so that a count down loop is used to copy the string (you can assume that the string is always 6 characters in length (includes the '$'). Add comments to each X86 program line and explain what that line does. Include the LISTING file of this program in your lab report. You may find the instructions 'inc bx' (add 1 to bx), 'inc si' (add one to the SI register) useful. You can choose whether or not to use the LOOP instruction.

Lab Question 6: Modify the program you did in Question 4 so that it does NOT assume a fixed length string (can handle any length string up to 128 bytes including the '$'). Add comments to each X86 program line and explain what that line does. Include the LISTING file of this program in your lab report. HINT: Your program needs to check when it is at the END of string#1. You will need to compare the byte value that you read from String#1 against the ASCII value for a '$' to see if your loop needs to be exited.

Lab Report

Include the answers for all **Lab Questions** in your report.

Applying what you learned

You must demo the programs written for Lab questions 5 and 6 to the TA.