

CS 216

Laboratory 7

Pre-Lab Activity:

x86 Assembly Language Basics¹

Notes:

See particular instructions on the course website. For Fall 2005 we are doing this in two parts over two weeks!

Objective:

The purpose of this lab is to familiarize you with the process of writing, assembling, and linking assembly language code. The purpose of the *in-lab and post-lab activities (to be posted later)* are to investigate how various C++ language features are implemented at the assembly level.

Background:

The Intel x86 assembly language is currently one of the most popular assembly languages on the planet and runs on many architectures from the x86 line through the Pentium 4. It is a CISC instruction set that has been extended multiple times (e.g. MMX) into a large instruction set.

¹ In past semesters of CS216, this was often Lab 7.

Procedure

Pre-lab

- 1. Examine the vecsum subroutine posted on the cs216 laboratories web page.
 - Use the “Tiny Guide to Programming in 32-bit x86 Assembly Language”.
 - Make sure you understand the prologue and epilogue implementation.
 - Make sure you understand the instructions used in the subroutine.
- 1. Complete the tutorial on how to create a C++/Assembly project.
 - Found at:
 - [http://www.cs.virginia.edu/~cs216/labs/lab6/Linking/MS Visual C++ and Assembly.htm](http://www.cs.virginia.edu/~cs216/labs/lab6/Linking/MS_Visual_C++_and_Assembly.htm)
 - You should follow this tutorial, with the following exception: In step 3, the command line is `c:\masm611\bin\ml /c /Cx /coff $(inputpath)`
- 1. Compile and run the vecsum program.
 - Use the tutorial as a guide.
 - You can find the assembly and C++ on the web page.
 - Use the debugger to step through the assembly code, view the register contents, and view the computer’s memory.
- 1. Write a small program of your own in C++ and use the debugger to examine the generated assembly code.
 - Use the Debug->Windows->Disassembly window.
- 1. Write an x86 assembly language routine that implements the factorial function **recursively**. That is, if you are not using the call instruction anywhere in your code, you are not doing it correctly! Follow the same procedure we used for writing IBCM code: First write out the factorial function in pseudo code, next refine that pseudo code to convert loops and conditionals to jumps, finally write your x86 code including the proper prologue and epilogue instructions. Luckily there is an assembler for the x86, so you do not need to encode your x86 assembly instructions into binary form but you DO need to be careful to use proper syntax in your input file as it will be interpreted by the assembler. As with IBCM code, you should comment copiously. In x86 assembly anything following a semicolon on a line is interpreted as a coment (much like // in C++).

Factorial is **recursively** defined as:

```
factorial(0) = 1
factorial(n) = n * factorial(n-1)
```

Your routine MUST follow the C calling convention as described in lecture and in the x86 handout – no shortcuts for efficiency. Your routine MUST use **recursion**. In addition, you should write a small C++ test program that will prompt the user for a positive integer, call your factorial routine to compute the factorial of that number and then print that result to the screen. Your factorial routine can assume it will receive an

integer greater than or equal to zero. Your C++ test program should guarantee that a proper value is passed.

- 1. Test your new factorial program. In debug mode, single step through the instructions in your assembly language function. Examine the state of the registers after each instruction. Do the register contents match your expectations? Do conditional jumps occur as expected? Use the debugger to report on the contents of ESP and EBP during the execution of your routine. What were their values immediately before the **ret** instruction? Do the contents of the stack appear to be as they should? To complete the checkoff for this lab you will be asked to show the TAs that you can identify parameters and stack frames.

In-lab

(For Fall 2005.) Come to class with prepared to demonstration you understand how to build and run the vecsum program. If you cannot, you will be asked to work through the tutorial during lab. Demonstrate and explain your pre-lab code to a TA for the check-off.

Also, you will be asked to create a simple x86 program from scratch. You will work with a partner on this activity.

Post-lab

- (For Fall 2005.) *Complete the entire Pre-lab. See the course website for details.*