CS 3843 Computer Organization, Spring 2013 Assignment 4

Due Monday, November 4, 2013

You must do this assignment using one of the Linux machines in our lab to make sure that everyone is using the same version of the compiler.

In this assignment you are asked to annotate assembly language code to illustrate that you understand it.

Look at the examples in the lecture notes for section 3.4, especially Example 3, and all of Examples 4, 5, and 6.

Your comments should indicate what the purpose of the code is in the context of the problem being solved, not just what an individual assembly language instruction does.

As an example, in Example 6 from the notes, the line:

```
       movl 12(%ebp), %edx    // move i into %edx
```

should not be annotated as:

```
       movl 12(%ebp), %edx    // move 4 bytes from memory address %(ebp+12)
                                 into %edx
```

Part 1:
Create a file called `calc.c` that contains the following:

```
int calc(int x, int y, int z) {
   return 3*x + 2*y + 15*z;
}
```

Compile the program to produce assembly code. Use:

```
cc -O1 -S calc.c
```

to produce `calc.s`.
Copy `calc.s` into `calc.s.txt` and annotate `calc.s.txt` with comments to illustrate that you understand how the calculation is implemented.
Make sure you identify each parameter.

Print out the annotated code and turn it in.

Part 2:
Write a main program, `testcalc.c`, that will test `calc.c`.
The main program should declare three `int` variables, `x`, `y`, and `z`, and initialize them to 2, 6, and 11, respectively.

It should then call `calc` with these parameters and print x, y, z, and the result using the format specification:
`"x=%d, y=%d, z=%d, result=%d\n"`.

Before running the program, answer the following question:
1) What output should be generated by this program?

The program will need a prototype of the function `calc`. Compile your program with:
```
cc -O1 -o testcalc testcalc.c calc.s
```
Test your program to make sure that it works correctly.

Compile `testcalc.c` produce `testcalc.s`. Use:
```
cc -O1 -S testcalc.c
```
Copy `testcalc.s` into `testcalc.s.txt`, annotate `testcalc.s.txt` and turn it in.

Part 3:
Copy the unannotated `calc.s` into `calcnew.s` and modify it so that it returns 3*`x + 7*y + 14*z`.
Do not use the `leal` instruction.
Use the 2-operand `imull` instruction with immediate addressing.
In addition to changing the code, you should modify all occurrences of `calc` to `calcnew`.
After making your modifications, make sure your file compiles using:
```
cc -c calcnew.s
```

Copy the unannotated `testcalc.c` into `testcalcnew.c` and modify it to call `calcnew` instead of `calc`.
Compile and run this program using:
```
cc -O1 -o testcalcnew testcalcnew.c calcnew.s
```

Before running the program, answer the following question:
2) What output should be generated by this program?

Copy `calcnew.s` into `calcnew.s.txt` and annotate it.

Part 4
Copy `calcnew.s` into `calcnew1.s` and modify it so that it does not use `imull` or `leal` but instead uses shift and add (or subtract).
You do not need to modify `testcalcnew` to test this program, just compile it with:
```
cc -O1 -o testcalcnew1 testcalcnew.c calcnew1.s
```
Test the program and make sure it produces the correct answer.
Copy `calcnew1.s` into `calcnew1.s.txt` and annotate it.

Answer the following question:
3) How many bytes is the compiled code for each of the programs: `calc.s`, `calcnew.s`, `calcnew1.s`?

You can find out the answer by producing a `.o` file for each and running `objdump`, for example with:

```
cc -c calc.s
objdump -d calc.o
```

Note that this is not the same as the size of the `calc.o` file.

Implementation Notes

- `calc.s` will use `%eax`, `%ecx`, and `%edx`.
  These are designated as caller-save registers. If you need to use additional registers, such as `%ebx`, this is a callee-save register, and you should save and restore it (on the stack). Note that pushing anything on the stack changes the location of the function parameters relative to the stack pointer, but not necessarily relative to the `%ebp` register.
- A small mistake in your code (such as using the wrong register) can produce a segmentation fault and it might be difficult for you to find your error.
  If this happens, start with `calc.s` and make one small change at a time, running your program after each change.
- You can compile the annotated versions of your programs directly if your annotations are in the form of comments surrounded by /* and */.

Handing in your assignment:
Use the cover sheet provided.
Include copies of the documents listed on the cover sheet along with the answers to the three questions.