Final Project: Minimal Disassembler

CS 350: Computer Organization & Assembler Language Programming
Due Wed May 1, 11:59 pm

A disassembler takes binary input and generates an equivalent text assembler program. Your job is to write a simple disassembler: It will handle only a few instructions, and it won't try to create textual labels for things like branching to the top of a loop. (It will just show the address the branch would go to.)

Your program is to be written in C and will be tested by running it on fusion/linux1.cs.iit.edu, but as with the labs, you're welcome to develop your program on your own machine. (Just verify that it works on fusion/ linux1.)

This is a one-person project. (Sorry, no teams.) On the other hand, if there's code you can use from a lab assignment, you can use it, even if you did the lab assignment as part of a team. Examples: `printf`, `scanf`, the basic structure of a C program, bit masks and bitwise operations. The parts of the final project that are new (haven't appeared in labs) — those parts are one-person.

Your program is required to first read (from a file called `data.txt`) a sequence of hexadecimal numbers (representing 32-bit strings). As you read them, you are required to translate them into an array of `struct instruction` (the same `struct/union` combination as in Lecture 15). (You'll need to select various fields of 32-bit strings to do this.) Once you hit end-of-file, prompt the user for a hex number that will serve as the location of the first instruction, read it in, and generate your output.

You'll need to translate numeric opcodes and numeric register numbers to mnemonics like `lw` for load word and `$t0` etc. for registers. The textbook has everything you need to figure out how to do that. There's one part that's not a direct translation of a bitstring into a number or mnemonic: For a branch instruction, instead of printing the PC-offset used inside the instruction, print the actual address of the target. (You'll need to keep track of where in memory the branch instruction is so that you can do the calculation of the target location (what the PC would be at runtime plus the PC offset from the branch).

For maximum credit, make your output readable. In particular, print your output in two columns:

```
lw   $t0, 4($s0)        not        lw $t0, 4($s0)
addi $t0, $t0, 5                   addi $t0, $t0, 5
add  $t0, $t0, $s2                 add $t0, $t0, $s2
```

(The column width is up to you.) To help you, we'll post a sample solution on fusion/linux1 that you can run to see how your program should behave.

Programs that are readable, well-organized, and well-commented will earn more points than ones that aren't.

When you submit your project, don't bother including data files or test runs; just submit one big `*.c` file.

We'll discuss the project in more detail, but this should get you started.