# 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, sample data and output are attached to this handout. [Sat 4/20]

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.

 

## Added Sat 4/20

The packet for this handout contains:

- `FP instructions.txt` — the set of opcodes you are required to handle. (The others can be ignored.)
- `data.txt` — Some sample input data.
- `sample_output.txt` — Output for the sample data.
  - You should include the same information in your output, though the format doesn't have to match exactly.
  - Do line up the opcode mnemonics and arguments.
  - You can use upper case if you want. (BEQ $0, $AT, x0100, for example.) But be consistent. (Don't do some opcodes or hex numbers in lower case and some in upper case.)
  - For branch instructions, show both the offset and the actual target address (PC + offset*4)
    - Note the offset is a signed 16-bit field, so it can be negative.
  - For the I functions I showed the offset/immediate value in hex and then in decimal.
    - Reverse the order if you want. (beq $0, $at, 256 (hex x0100, ...), for example.)
  - For the R functions I showed the shamt/funct in hex and then in decimal.
    - Again, reversing the order is fine.
    - Since shamt and funct are 5 and 6 bits respectively, it's not hard to mentally convert them between hex and decimal, so if you want to just print them out in hex only (or decimal only), that's ok.
  - For both I and R instructions, be consistent (don't do some X instructions one way and some the other way, $X \in \{I, R\}$).

(Instructor note: I spent way too much time changing back and forth between printing (fill in arbitrary field) in decimal or hex until I realized "Should I print in decimal or hex" didn't need to use exclusive or.)

 