

Project 1: w75 Disassembler

Prof. Loh

CS3220 - Processor Design - Spring 2005

Handed Out: 15 Feb 2005

Due: 03 Mar 2005

This first project lets you get your hands dirty with the architecture that we will be using for all of the projects in this course. Please see the accompanying document `w75-isa.doc` for a full specification of the ISA. Also, download the gzip'd tar file `w75-package.tar.gz` from the course website. I recommend doing all of your development and testing in a Linux/x86 environment.

The goal of this project is to write a disassembler for w75 binaries. The `w75-package.tar.gz` file contains an assembler and an example w75 program. You can use the assembler to generate your own w75 binaries. The tarball also contains a binary for a functional simulator that you can use to execute the binaries. Both the assembler and the functional simulator has verbose modes (`-v` on the command line) that will result in a dump of a *lot* of information about the instructions being assembled or executed. The tarball also include a file that describes the format of the w75 binaries.

Your program should have the following command line format:

```
$ ./w75-dis file.bin
```

Your program should then open up the w75 binary file specified, read in the data and the code, and print out all of the contents. More specifically, your disassembler's output should be compatible with the assembler. That is, starting with a binary file `file.bin`, you should be able to use your disassembler to generate a new w75 assembly file, which when assembled creates a binary that is identical to the original:

```
$ ./w75-dis file.bin > file2.w75
$ ./w75-asm file2.w75
$ diff file.bin file2.bin
$
```

Note that the assembly files may differ because the disassembled version will not contain the names of jump targets (no jump labels) and the data in the `.DATA` section may have different names. The binaries should still come out identical.

It is recommended that you define a struct that can hold all of the relevant decoded information for an instruction. This includes opcodes, register specifiers, the addressing mode, immediates, etc. You should use one function that, given 2-4 bytes, decodes the instruction and populates the instruction struct, and then returns the number of bytes for that instruction (i.e. did you decode a 2- or 4-byte instruction?). Then, implement a separate function that takes an instruction struct (or a pointer to one), and prints out the assembly equivalent of the instruction.

The first function that populates the instruction struct should then be easily adapted to serve as your decode logic in the subsequent projects. The print function will be useful for debugging purposes in your subsequent projects.

C or C++ are highly recommended. To submit your project, make a directory containing your source code, a Makefile, and a short write-up (it does not need to be more than one page) describing how you organized your code. Tar and gzip the directory, and then email the file to me at `loh@cc.gatech.edu`. Projects are due by 11:59:59 PM on the due date.

Do not hesitate to ask if you have questions or if something about this project is unclear to you. Feel free to make use of office hours, instant messaging ("gt 2005a cs3220" on AIM), or if necessary we can make other meeting arrangements.