Ryan Bandilla
Assignment 4
CS 211
Prof. Russell

Y86 Emulator

This program can be used to execute *.y86 program files. The program can be called with either a help flag –h or a valid *.y86 file.

./y86emul <y86 program file>
./y86emul –h

This emulator works by looking for the .size directive in the y86 program file. If there is no .size directive, or more than one .size directive, the emulator prints an error message and quits.

After finding the .size directive the emulator allocates an unsigned character array as large as the size specified by the .size directive. This unsigned character array will be the simulated address space which will contain all instructions and information that is required for the y86 program file execution.

After the address space is allocated, all of the other directives and their associated arguments are stored in the address space. If there are multiple .text directives, the emulator prints an error and quits.

When the entire y86 emulator is loaded into memory, execution beings at the location specified by the first argument in the .text directive. I implemented the execution by reading byte by byte from the specified program start. There is a large switch statements that handles the execution of all of the instructions. The hardest ones to implement were movsbl, readl, call, and ret. I had numerous problems with them causing infinite loops, and debugging was difficult. Unions came in handy in my implantation. I used a union of an integer and an unsigned char array so I could decode the 4 contiguous bytes of the arguments of some of the instructions (rmmovl, mrmovl and jxx come to mind).

Y86 Disassembler

This program can be used to disassemble *.y86 program files. The program can be called with either a help flag -h or a valid *.y86 file.

./y86dis <y86 program file>

./y86dis -h

The disassembler works by finding the .text directive in the input y86 program file. The program prints an error message if there is no .text directive or more than one .text directive.

After the .text directive's arguments are stored in an unsigned character array, I just iterated through the contents in the same fashion as in the emulator, except I just printed out what the instructions and registers were instead of doing any actual work. After finishing the emulator finishing this portion was trivial. I mirrored the switch statement from the emulator and printed out the corresponding instructions.