Y86 Emulation

Rushabh Jhaveri

Computer Architecture, Summer 2017

Professor Brian Russell

Synopsis

This project attempts to recreate a Y86 emulator. The emulator program takes in a .y86 file as an argument on the command line. It then searches for the file in the current directory, parses it, loads / sets up properly-allocated memory for the program to operate, and then finally, executes the .y86 program.

Design and Working

This program first does an argument check to ensure the correct number of arguments and the correct type of file (.y86 only) have been passed. If not, then it throws an error and exits the program.

If the “-h” flag has been passed as an argument, it prints out a message on how to use this program and then exits the program.

After doing so, the program begins to read the file in “rb” mode, the “b” being specified because it is a non-text file. It tokenizes the file by first breaking it up into lines and then further breaking up those lines by (tab) spaces. A check is made as to whether a valid directive is passed, and if not, an error is thrown. After going through the entire file, the file is closed. After this process of loading the memory is successfully completed, the fetch-execute-decode cycle begins. This consists of switch statements which call on the appropriate methods when needed.

Various structures have been made use of in this program:

struct CPU: This structure is what I have used to represent the CPU. It contains an array of ints representing the eight registers, a program counter (instruction pointer), ints representing the three flags (zero flag, overflow flag, sign flag), and an int representing the size of the CPU.

union byte: This union structure has a struct the size of a byte and a char of the same size. The struct contains two unsigned ints of size 4, and thus, both added together add up to one byte and can hold values ranging 0-15, as represented by the hexadecimal characters in the .y86 file.The char is a regular character of the size of a single byte, matching the size of the struct. The union enables the easy conversion of two sub-bytes into a byte to be stored in memory (and vice-versa), to read instructions.

union int32_to_char: This union contains a struct of size four bytes and an integer of the same size. The struct contains four chars to represent the individual bytes of the larger, little-endian integer. The int is a regular int to match the size of the struct. This union enables the easy conversion of four separately tracked bytes to an integer.

Challenges Faced

- Determining how to represent and manipulate memory.
- Debugging such a large program was an learning experience.
- Learning about and coding each individual instruction

Runtime and Spatial Analysis

This program runs in linear time and occupies linear amount of space.