CISC 340 – Simulator

James Allender
Mitchell Nelson

October 17, 2018

1 Description:

The behavioral simulator is a simple simulator to simulate the UST-3400 (Rip Van saWinkle) machine code. The program uses a single procedural C program to read in a given file containing machine code and simulate the machine code with the correct behavior. The program will print the output to the screen.

2 Overview:

The program takes one argument: an input file. Any deviation from this will lead to a failure and exit from the program.

One input example:

```
$./simulator -i [machine code file]
```

The program can also have its output redirected to a file with:

```
$./simulator -i [machine code file] > [out file]
```

The program requires that an input file is given directly after a "-i" flag, or it will fail and close. When an input file is provided, the program checks that the input file exists in main, and if so, it will start processing the file.

The file is read once by the program. The program iterates over each line and stores each line of machine code into the simulated memory.

The program uses bit masking and bit shifting to retrieve opcodes from each line of the machine code. It does the same to retrieve values for registerA, registerB, the destination register, and the immediate value. While the destination register and the immediate value never get used at the same time, both are still calculated immediately, despite the type of instruction. The calculated values for each line then move on to the actual instruction portion of the program. The opcodes determine what the simulator should do with each line.

The simulator keeps track of simulated memory, a program counter, and 8 simulated registers. This is all part of the state for the UST-3400 and this information is printed before each instruction is executed and once before the program is complete. The simulator will keep executing instructions until a halt instruction is found. The simulator assumes that functional code is written and it will not check for code that contains no halt instructions or other types of invalid programs.

The simulator uses a program counter to keep track of what line is executing. The instructions "beq" and "jalr" modify the program counter which allows for looping and jumping.

3 Challenges:

Some challenges that were encountered when implementing the simulator were with the "lw", "beq", and "sw" instructions. The main part of the challenge was first getting a good understanding of how these instructions should behave. The way that we combated this challenge was to study the provided example and class notes which gave us clarification.