EE126 Final Report

Harrison Kaiser and Chris Phifer

Background

Motivation

During the course of the semester we manually compiled many instructions into the binary machine code for the machine we built. This is a labor intensive process which we didn't enjoy doing anymore. So we decided to cap the semester off by making it easier for future generations. We built an assembler.

Project Definition

Desired Outcome

In order to simplify the process of translating MIPS assembly code into machine code (for use in loading programs into our VHDL processor), we sought to implement a MIPS assembler capable of handling a useful subset of the MIPS instruction set, including the `mov` pseudo-instruction and named labels. This would greatly improve the efficiency of developing programs for the processors we implemented in VHDL for testing purposes.

Design Specifications

The assembler we built met a few design specifications:

- Include at least one macro: we included a `mov` macro
- Support the instructions we used during the semester including a few others
- Support labels and targets for easily making loops
- Make an easy to use interface
- Easily convert the compiled program into VHDL so it can be tried on the VHDL machine we have built

Implementation Details

To implement the assembler, we chose the purely functional front-end web language Elm. The high-order nature of the language and strong static type system gave us confidence in code correctness from the beginning, and helps guarantee robustness of the assembler. This decision simplified parts of the process that in other languages would be quite difficult (e.g. parsing), but made difficult some things that are usually straightforward (e.g. accessing the machine representation of integers for translation into binary).

To simplify the process of translating our assembler's output machine code into VHDL initialization code, we wrote a very short Python script that transforms ASCII 0s and 1s into the appropriate initialization code, indented properly to be inserted directly into the VHDL file where instruction memory initialization occurs.

Results

Demonstration of Functionality

The functionality of our assembler can be seen through the provided example files: `program.txt` which contains ASCII machine code generated by the assembler, and `gen_program_example.txt` which contains the generated VHDL code from the Python script.

Evaluation

Our implementation does meet all the specifications we laid out above. While there are some ways to polish our assembler, like making our error messages better, this will certainly serve future students well. Our design is not very scale-able I wouldn't want to use this assembler for a very large program. This lab clearly makes it easier to compile the bits of code which was difficult to handle by hand.