CS 429H, Spring 2014
Adding instructions to Y86
Assigned: March 20, Due: March 27, 11:59PM

1 Introduction

In this lab, you will learn how an instruction flows through the various pipeline stages of a Y86 processor and gain familiarity with HCL. Your task is to augment the Y86 processor to support three new instructions.

2 Logistics

You will work on this lab alone.

Any clarifications and revisions to the assignment will be posted on the course Web page.

3 Handout Instructions

Download the `datapath_lab-handout.tar` file from the class webpage.

1. Start by copying the file `datapath_lab-handout.tar` to a (protected) directory in which you plan to do your work.
2. Then give the command: `tar xvf datapath_lab-handout.tar`. This will cause the following files to be unpacked into the directory: `README`, `Makefile`, `sim.tar`, `archlab.ps`, `archlab.pdf`, and `simguide.pdf`.
3. Next, give the command `tar xvf sim.tar`. This will create the directory `sim`, which contains your personal copy of the Y86 tools. You will be doing all of your work inside this directory. Do not use the sim directory from the previous labs, as this one includes a few changes which are needed for the purpose of this lab.
4. Finally, change to the `sim` directory and build the Y86 tools:

```
unix>  cd sim
unix>  make clean; make
```

4 Adding instructions to the Y86 processor

You will be working in directory `sim/seq`.

Your task is to extend the SEQ processor to support three new instructions: `iaddl` (described in Homework problems 4.47 and 4.49), `leave` (described in Homework problems 4.48 and 4.50) and `mrimovl` (described below). To add these instructions, you will modify the file `seq-full.hcl`, which implements the version of SEQ described in the CS:APP2e textbook. In addition, it contains declarations of some constants that you will need for your solution.

Your HCL file must begin with a header comment containing the following information:

- Your name and ID.
- A description of the computations required for the `iaddl` instruction. Use the descriptions of `irmovl` and `OPl` in Figure 4.18 in the CS:APP2e text as a guide.
- A description of the computations required for the `leave` instruction. Use the description of `popl` in Figure 4.20 in the CS:APP2e text as a guide.
- A description of the computations required for the `mrimovl` instruction. Use the description of `popl` in Figure 4.20 in the CS:APP2e text as a guide.

Instruction `mrimovl`

We often encounter cases where we need to load some value from an address in memory and then increment the address by 4 to access the consecutive location. This requires first using an `mrmovl` instruction to load address A, and then using an `iaddl` to increment the address A. The `mrimovl` instruction will do both these operations in a single instruction - load the value at address rB into the register rA, and then increment rB by 4. This is especially useful for array based accesses. Unlike, `mrmovl`, the load from `mrimovl` does not take a constant offset field when computing the address. For instance - `mrimovl` (%ecx) %eax will load the value which is stored at the address in %ecx into register %eax. It will also increment %ecx by 4. The instruction `mrimovl` 4(%ecx) %eax will merely ignore the constant offset field and do the same operation as `mrimovl` (%ecx) %eax. This is because the Y86 processor can only do a single arithmetic operation in the execute stage.

Building and Testing Your Solution

Once you have finished modifying the `seq-full.hcl` file, then you will need to build a new instance of the SEQ simulator (`ssim`) based on this HCL file, and then test it:

- *Building a new simulator.* You can use `make` to build a new SEQ simulator:

```
unix>  make VERSION=full
```

This builds a version of `ssim` that uses the control logic you specified in `seq-full.hcl`. To save typing, you can assign `VERSION=full` in the Makefile.

- *Testing your solution on a simple Y86 program.* For your initial testing, we recommend running simple programs such as `asumi.yo` (testing `iaddl`) and `asuml.yo` (testing `leave`) in TTY mode, comparing the results against the ISA simulation:

```
unix>  ./ssim -t ../y86-code/asumi.yo
unix>  ./ssim -t ../y86-code/asuml.yo
unix>  ./ssim -t ../y86-code/asummri.yo
```

  If the ISA test fails, then you should debug your implementation by single stepping the simulator in GUI mode:

```
unix>  ./ssim -g ../y86-code/asumi.yo
unix>  ./ssim -g ../y86-code/asuml.yo
unix>  ./ssim -g ../y86-code/asummri.yo
```

- *Retesting your solution using the benchmark programs.* Once your simulator is able to correctly execute small programs, then you can automatically test it on the Y86 benchmark programs in `../y86-code`:

```
unix>  (cd ../y86-code; make testssim)
```

  This will run `ssim` on the benchmark programs and check for correctness by comparing the resulting processor state with the state from a high-level ISA simulation. Note that none of these programs test the added instructions. You are simply making sure that your solution did not inject errors for the original instructions. See file `../y86-code/README` file for more details.

- *Performing regression tests.* Once you can execute the benchmark programs correctly, then you should run the extensive set of regression tests in `../ptest`. To test everything except `iaddl` and `leave`:

```
unix>  (cd ../ptest; make SIM=../seq/ssim)
```

  To test your implementation of `iaddl`:

```
unix>  (cd ../ptest; make SIM=../seq/ssim TFLAGS=-i)
```

  To test your implementation of `leave`:

```
unix>  (cd ../ptest; make SIM=../seq/ssim TFLAGS=-l)
```

  To test your implementation of `mrimovl`:

```
unix>  (cd ../ptest; make SIM=../seq/ssim TFLAGS=-M)
```

  To test `iaddl`, `leave` and `mrimovl`:

```
unix>  (cd ../ptest; make SIM=../seq/ssim TFLAGS=-ilM)
```

For more information on the SEQ simulator refer to the handout *CS:APP2e Guide to Y86 Processor Simulators* (`simguide.pdf`).

5 Evaluation

This lab is worth 85 points:

- 10 points for your description of the computations required for the `iaddl` instruction.
- 10 points for your description of the computations required for the `leave` instruction.
- 10 points for your description of the computations required for the `mrimovl` instruction.
- 10 points for passing the benchmark regression tests in `y86-code`, to verify that your simulator still correctly executes the benchmark suite.
- 15 points for passing the regression tests in `ptest` for `iaddl`.
- 15 points for passing the regression tests in `ptest` for `leave`.
- 15 points for passing the regression tests in `ptest` for `mrimovl`.

6 Handin Instructions

- You need to submit only `seq-full.hcl`.
- Make sure you have included your name and ID in a comment at the top of the file.
- Turnin your seq-full.hcl file through Canvas.

7 Hints

- If you running in GUI mode on a Unix server, make sure that you have initialized the DISPLAY environment variable:

```
unix>  setenv DISPLAY myhost.edu:0
```

- With some X servers, the "Program Code" window begins life as a closed icon when you run `psim` or `ssim` in GUI mode. Simply click on the icon to expand the window.
- With some Microsoft Windows-based X servers, the "Memory Contents" window will not automatically resize itself. You'll need to resize the window by hand.
- The `psim` and `ssim` simulators terminate with a segmentation fault if you ask them to execute a file that is not a valid Y86 object file.