Lab 3

Computer Architecture I @ ShanghaiTech University

Objectives

- Run and debug RISC-V assembly code.
- Write RISC-V functions using the right calling procedure.
- Get an idea of how to translate C code to RISC-V

Exercises

Download the files for Lab 3 first.

Intro to Assembly with RISC-V Simulator

So far, we have been dealing with C program files (.c file extension), and have been using the `gcc` compiler to execute these higher-level language programs. Now, we are learning about the RISC-V assembly language, which is a lower-level language much closer to machine code. For context, `gcc` takes the C code we write, first compiles this down to assembly code, and then assembles this down to machine code/binary.

In this lab, we will deal with several RISC-V assembly program files, each of which have a .s file extension. To run these, we will need to use a RISC-V simulator. The simulator we will use was developed by Keyhan Vakil and improved by Stephan Kaminsky. The simulator is called **Venus** and can be found online here. We have also deployed Venus on our Autolab server (link).

Assembly/Venus Basics

- Enter your RISC-V code in the “Editor” tab
- Programs start at the first line regardless of the label. That means that the `main` function (entrance of the function, the line labeled `main` in the provided .s file) must be put first.
- Programs end with an `ecall` with argument value 10. This signals for the program to exit. The ecall instructions are analogous to System Calls and allow us to do things such as print to the console or request chunks of memory from the heap.
- Labels end with a colon (`:`).
- Comments start with a pound sign (`#`).
- You CANNOT put more than one instruction per line.
- When you have done editing, click the “Simulator” tab to prepare for execution.

For the following exercises, please save your completed code in a file on your local machine. This is crucial for the checkoff portion to work.

Exercise 1: Familiarizing yourself with Venus

Getting started:

1. Paste the contents of `lab3_ex1.s` into the editor.
2. Click the “Simulator” tab. This will prepare the code you wrote for execution.
3. In the simulator, click “Assemble & Simulate from Editor”
4. In the simulator, to execute the next instruction, click the “step” button.
5. To undo an instruction, click the “prev” button.
6. To run the program to completion, click the “run” button.
7. To reset the program from the start, click the “reset” button.
8. The contents of all 32 registers are on the right-hand side, and the console output is at the bottom
9. To view the contents of memory, click the “Memory” tab on the right. You can navigate to different portions of your memory using the dropdown menu at the bottom.

Action Item

Record your answers to the following questions in a text file. Some of the questions will require you to run the RISC-V code using Venus’ simulator tab.

1. What do the `.data`, `.word`, `.text` directives mean (i.e. what do you use them for)? **Hint**: think about the 4 sections of memory.
2. What basic instruction(s) are the pseudo instructions like `la`, `mv` and `j` converted to by the assembler?
3. Run the program to completion. What number did the program output? What does this number represent?
4. At what address is `n` stored in memory? **Hint**: Look at the contents of the registers.
5. Without using the “Edit” tab, have the program calculate the 13th fib number (0-indexed) by *manually* modifying the value of a register. You may find it helpful to first step through the code. If you prefer to look at decimal values, change the “Display Settings” option at the bottom.

Check-off

Show your TA that you are able to run through the above steps and provide answers to the questions.

Exercise 2.1: Translating from C to RISC-V

Open the files `lab3_ex2.c` and `lab3_ex2.s`. The assembly code provided (.s file) is a translation of the given C program into RISC-V.

Action Item

Find/explain the following components of the assembly file and put your answers in a text file.

- The register representing the variable `k`.
- The registers acting as pointers to the `source` and `dest` arrays.
- The assembly code for the loop found in the C code.
- How the pointers are manipulated in the assembly code.

After you’ve explained the above components, edit `lab3_ex2.s` so that `dest` satisfies the following conditions.

- `dest[i] = 2 * source[i]` for even `i`
- `dest[i] = 1` for odd `i`

Hint: This can be done by adding one line of code and modifying another (in other words, you only need to make 2 changes). Look at the initial values of dest; how does this help you implement this modification?

Verify that your changes work for the given `source` and `dest` arrays by running your code in a new Venus tab and check that the output looks like:

```
3 6 4 2 5 9

6 1 8 1 10 1
```

Check-off

Show `lab3_ex2_1.s` to your TA, and run it in Venus, which should give the correct result.

Exercise 2.2: Going a Step Further

You can continue using the file `lab3_ex2.s` as a template.

Action Item

Not modifying the initial values of dest, edit `lab3_ex2.s` so that `dest` satisfies the following conditions.

- `dest[i] = 2 * source[i]` for even `i`

- `dest[i] = i` for odd `i`

Verify that your changes work for the given `source` and `dest` arrays by running your code in a new Venus tab and check that the output looks like:

```
3 6 4 2 5 9

6 1 8 3 10 5
```

Check-off

Show `lab3_ex2_2.s` to your TA, and run it in Venus, which should give the correct result.

Exercise 3: SeriesOp

In this exercise, you will be implementing a function `seriesOp` in RISC-V that calculates the summation and subtraction of series of positive integers `n` in turn and returns `ans`. A stub of this function can be found in the file `lab3_ex3.s`. You will only need to add instructions under the `seriesOp` label, and the arguments that are passed into the function are defined at the top of the file. You must solve this problem using either recursion or iteration.

The formula for the operation is as follows:

- `ans = n - (n-1) + (n-2) - ... 1`

The series begins with n and ends with 1. The sign of each term alternates according to its position: odd-positioned terms are positive (+), and even-positioned terms are negative (-). e.g.

For n = 4, ans = 4 - 3 + 2 - 1.

For n = 7, ans = 7 - 6 + 5 - 4 + 3 - 2 + 1.

Action Item

Implement `seriesOp` and make sure that the program can correctly output the answers `3`, `2`, `3`, `6` when the inputs are `6`, `3`, `5`, and `12`.

Check-off

Show `lab3_ex3.s` to your TA, and run it in Venus, which should give the correct result.