RISC-V Basic Assembly Programming

Sandeep Chandran

23-Aug 2024

1 RISC-V Programming Guide

1.1 Files and Folders (conventions followed)

1. We will use the following conventions:

   ***.s** The `.s` (and not `.S`) file extension is used for the RISC-V source code (assembly code)

   ***.r5o** The file extension `.r5o` is used to indicate a RISC-V ELF

   ***.r5o.dump** `.r5o.dump` extension is used for the output of objdump on a RISC-V ELF

   ***.iss** `.iss` extension is used for output of the Spike run.

2. The folder structure (of our `pr5`) so far is as follows:

   pr5/ Top-level or Project Root folder. All the code and its associated files will reside inside this folder.

   pr5/programs Folder where all the RISC-V assembly programs we write will reside.

   pr5/programs/asms/ Folder that contains all the RISC-V assembly programs that we will write (as practice and as assignment). All files should end with the extension `.s` (and not `.S`)

   pr5/programs/custom/ Folder that contains some additional files that are to generate RISC-V ELF (linker scripts, headers, etc).

   pr5/programs/bins This folder is auto-generated. This folder contains all the RISC-V executables (ELFs) with the `.r5o` extension.

   pr5/programs/dumps This folder is auto-generated. This folder contains the objdump (generated using `riscv-none-elf-objdump`) of the executables. The generated dump files will have only the `.text` and the `.data` sections of the objdump (other sections are not dumped for brevity).

   pr5/programs/runs This folder is auto-generated. This folder contains the output produced when the executables are run on Spike. The output shows the sequence of instructions produced when it is run on a RISC-V core.

3. The files (in our pr5) so far is as follows:

   pr5/programs/Makefile Contains commands to compile the assembly programs using `riscv-none-elf-gcc`, generate the objdump, and run the generated executable on spike. NOTE that it expects these programs are available on your `$PATH`. So follow the Setup instructions carefully.

pr5/programs/asms/endless$_{loop.s}$ A simple loop that does not terminate. We will such an infinite loop to halt the system after computing our program. This is a hack to avoid linking against a lot of low-level libraries to handle program termination.

pr5/programs/asms/data$_{vars.s}$ A simple RISC-V assembly code that declares two (global) variables, `myvar1` and `myvar2` and initializes it to `0xc001` (49,153) and `0xc0de` (49,374) respectively. It also declares a (global) array called `myarr`. `myarr` is of 16 bytes (4 words). These 4 words, corresponding to `myarr[0]`, `myarr[1]`, `myarr[2]` and `myarr[3]` is initialized to `0xfeed`, `0xdeed`, `0xdeaf`, `0xd00d` respectively. The main function loads the address of these variables into registers `x1`, `x2` and `x3` respectively, and then uses the addresses to access the memory and load the contents into registers `x11`, `x12`, `x13` (`myarr[0]`) and `x14` (`myarr[1]`).

1.2 Compiling (or assembling) programs

1. Write your assembly programs in the pr5/programs/asms. Make sure that the file extension is `.s` (and not `.S`). Also make sure that `riscv-none-elf-gcc` is in your `$PATH`.
2. Type the following commands.

```
$ cd pr5/programs
$ make
```

3. Issuing the make command will compile the programs (*.s) in the asms folder and create two new folders, bins/ and dumps/. The executables are kept in the bins folder and their corresponding objdumps are kept in the dumps folder.

1.3 Running RISC-V programs

1. Make sure that spike is in your `$PATH`.
2. The following command runs all the programs in the asms folder.

```
$ cd pr5/programs
$ make run_all
```

3. The run_all target of the make, will run the executable on Spike and create the corresponding `*.iss` files. The `.iss` files will show the sequence of instructions produced when the executable is run. The execution starts at the PC `0x00010000` (which is in bootrom of Spike), and finally reaches the main (label) which is mostly at `0x80002000` (the exact address is known only from objdump). The output also indicates which memory address was accessed and what is the value written by the instruction to the destination register of the instruction.

1.4 Additional Help and Reading Materials

There is a lot of help available on the internet on RISC-V assembly. The following are recommended sources.

1. Chapter 2 of the prescribed textbook (Computer Organization and Design - RISCV edition) is a fantastic primer on RISC-V assembly. Note that the text book covers RISC-V 64-bit (RV64I) ISA, but we are using the 32-bit version (RV32I) for the labs.

2. You can use the standard gcc compiler (`riscv-none-elf-gcc`) with optimizations turned off (`-O0` flag) to emit assembly code (use `-s` flag). You may have to go through additional materials from the internet to understand some of the assembler directives emitted by the compiler. Also note that there are some conventions that GCC follows (not mandatory rules) in using the registers `x0-x31`. For example, as per the conventions, `x1` is used to store return addresses (and hence is also called `ra`), `x2` is used to store the stack pointer (and hence is also called `sp`), etc. More details are available in the textbook (refer to the point above).

3. You can generate the ELF and then use objdump (`riscv-none-elf-objdump`) with the `-d` (or `-D`) option to see the contents of the resulting executable in a human-readable format. As always, you may have to go through additional materials (as suggested in the point above).

2 Practice Problems

1. Write a simple RISC-V assembly program that initializes the register `x10` to `0x1234abcd`.

2. Write a simple RISC-V assembly program to find the maximum of three numbers. Assume that these numbers are available in the registers `x1`, `x2` and `x3`.

3. Write a simple RISC-V assembly program that adds the values stored in two registers, `x1` and `x2`, and stores the result in a global variable named `sum`.

4. Write a simple RISC-V assembly program that initializes all the elements of a global array named `arr` (which contains 100 elements) to the value `0xc001` using a simple loop.

3 Lab Assignment 1 (Graded)

1. Write a RISC-V assembly program to `count` the number of positive (>= 0), even numbers in a given array `l`. The size of the array is `n` (`n > 0`). Place the result in register `x10`. Use the template in `asms/1-even.s`.

2. Write a program to check if a given number is prime. If yes, place 1 in `x10`. If not, place `-1` in `x10`. Use the template in `asms/2-prime.s`.

3. Write a program to sort an array of numbers in the descending order. The sorted array should be placed in the same addresses in memory as the initial array. Use the template in `asms/3-descending.s`.

4. Write a program to find the histogram of marks obtained by `n` students in a course. The histogram of marks should be in the `count` array. Use the template in `asms/4-histogram.s`.