ECE243 Final Project Report
A RV32I Emulator

By Yikun Wang and Yangyijian Zhou

Introduction

This project is a software emulator for the RISC-V unprivileged RV32I ISA written in C. The program emulates a RV32I system, including a processor, memory, and I/O. It reads and executes machine code from an ELF executable file.

Technical Overview

The emulator reads the guest program ELF stored as a `char` array, which is included at compile time. It first performs a simple sanity check of the ELF header and finds the entry point address. Then, it loads the guest program into memory and starts executing instructions from the entry point. The memory size is fixed to 1 MiB (0x0 - 0x100000).

The emulator sequentially fetches, interprets and executes machine code instructions. It supports all RV32I base integer instructions *(Version 2.1)* except `FENCE`.

Usage

Compiling the Guest Program

Before loading the guest program, it needs to be compiled into a statically linked ELF executable. A prebuilt GCC RV32I Toolchain for x86_64 Linux can be downloaded from https://github.com/stnolting/riscv-gcc-prebuilt. A RV32I assembly program can be compiled with

```
$ riscv32-unknown-elf-gcc -s -ffreestanding -nostartfiles -Tlink.ld [FILENAME].s
-o rvelf
```

The user can also compile a C program using

```
$ riscv32-unknown-elf-gcc -s -ffreestanding -nostartfiles -Tlink.ld startup.s
[FILENAME].c -o rvelf
```

The output filename should be `rvelf` to be correctly loaded by the emulator program. In the previous examples, a custom linker script `link.ld` is used to ignore some unnecessary sections; and a startup script `startup.s` is used with the C program to setup the stack pointer and perform the exit environment call.

Dumping ELF into a C Header File

To be imported into the emulator program, the ELF file needs to be converted into a C array `unsigned char rvelf[]` with an `unsigned int rvelf_len` indicating the array length. Both should be stored in a C header file `rvelf.h`. On Linux platforms, this can be done using

```
$ xxd -i rvelf > rvelf.h
```

Starting the Emulator Program

While compiling the project, the `rvelf.h` generated in previous steps should be placed in the project directory.

Although this emulator is intended for the *Nios II processor* on a *DE1-SoC* board, it can be compiled and runned on other platforms; nevertheless, in such cases, switch and key inputs and VGA output are unavailable. The project can be consolidated into a single C file using *Quom* to be simulated on *CPUlator*.

Environment Calls and Breakpoints

The `ECALL` and `EBREAK` instructions help the guest program interact with the emulator.

The `ECALL` instruction calls a service routine of the emulator. The emulator will retrieve the ecall status number from register `a0` as an `int` specifying a service. It could also retrieve a parameter stored in or put a return value into register `a1`. Environmental calls is the only way for the guest program to access I/O.

`a0` (Decimal)	Description	`a1`
`0`	exit with status code	`int`: status code
`100`	print an integer to terminal	`int`: number
`101`	print a null-terminated string to terminal	`char*`: string address
`103`	print all registers to terminal	
`200`	read *DE1-SoC* switches	returned switches value

The `EBREAK` instruction pauses the execution of the guest program. In the provided startup script `startup.s`, a breakpoint is placed before the main function is called.

Emulator User Interface

VGA Display

The VGA display is divided into three main sections.

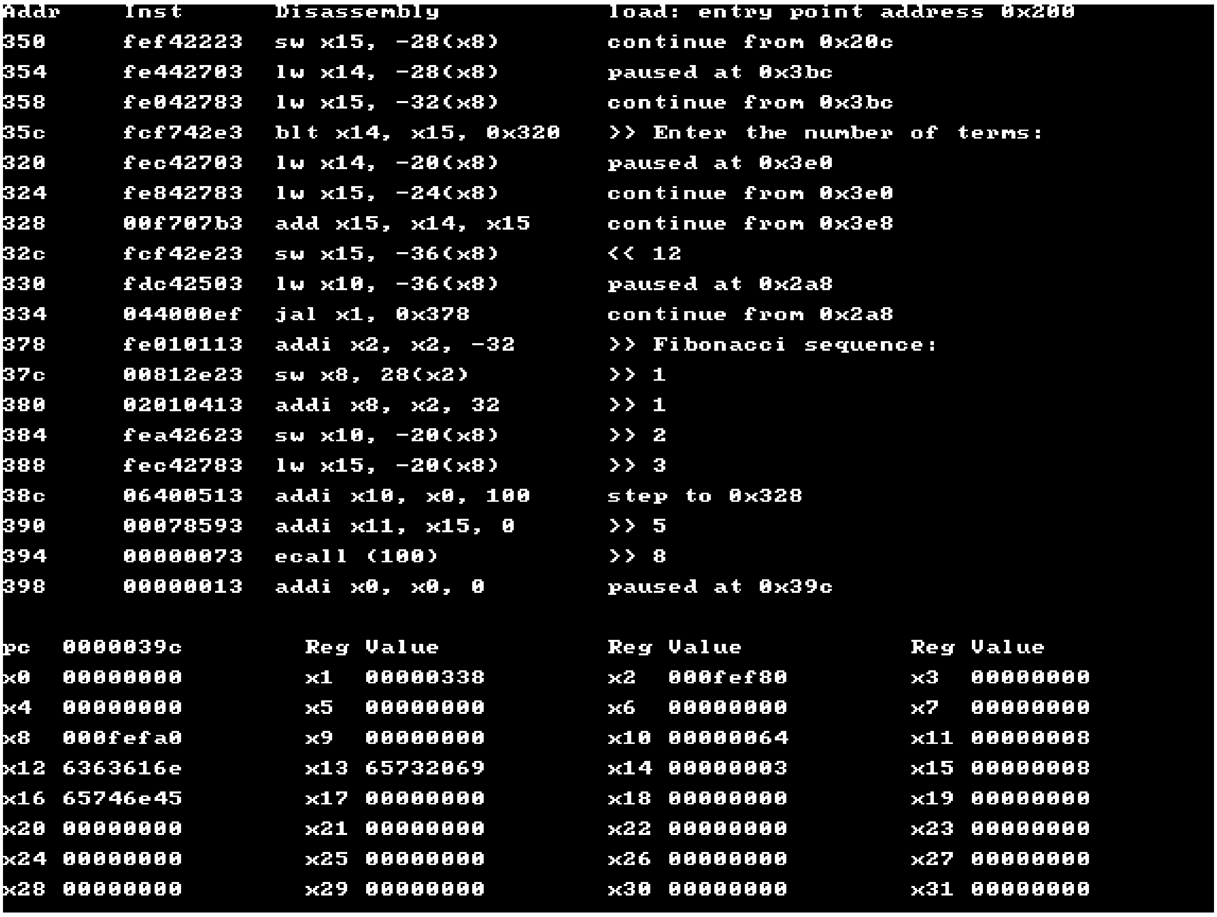
1. Upper left: Memory address, instructions in machine code, and their disassembly.
2. Upper right: Emulator messages and guest program I/O.
3. Bottom: Registers with their current values in hex.

Keys

- Press key 0 to perform “step” function, executing one instruction at a time.
- Press key 1 to pause a running program or continue from a paused state.
- Press key 3 to restart the program.

Switches

Switches are used to input numbers in binary, e.g. switch 2 indicates 8 in decimal (`0b1000`). The LEDs show the state of each switch. The guest program could read the current switches value with an environmental call. It is recommended to place an `EBREAK` before the call so that key 2 could be used as the confirm button.



```
Addr    Inst      Disassembly           load: entry point address 0x200
350     fef42223  sw x15, -28(x8)       continue from 0x20c
354     fe442703  lw x14, -28(x8)       paused at 0x3bc
358     fe042783  lw x15, -32(x8)       continue from 0x3bc
35c     fcf742e3  blt x14, x15, 0x320   >> Enter the number of terms:
320     fec42703  lw x14, -20(x8)       paused at 0x3e0
324     fe842783  lw x15, -24(x8)       continue from 0x3e0
328     00f707b3  add x15, x14, x15     continue from 0x3e8
32c     fcf42e23  sw x15, -36(x8)       << 12
330     fdc42503  lw x10, -36(x8)       paused at 0x2a8
334     044000ef  jal x1, 0x378         continue from 0x2a8
378     fe010113  addi x2, x2, -32      >> Fibonacci sequence:
37c     00812e23  sw x8, 28(x2)         >> 1
380     02010413  addi x8, x2, 32       >> 1
384     fea42623  sw x10, -20(x8)       >> 2
388     fec42783  lw x15, -20(x8)       >> 3
38c     06400513  addi x10, x0, 100     step to 0x328
390     00078593  addi x11, x15, 0      >> 5
394     00000073  ecall (100)           >> 8
398     00000013  addi x0, x0, 0        paused at 0x39c

pc  0000039c        Reg Value           Reg Value           Reg Value
x0  00000000        x1  00000338        x2  000fef80        x3  00000000
x4  00000000        x5  00000000        x6  00000000        x7  00000000
x8  000fefa0        x9  00000000        x10 00000064        x11 00000008
x12 6363616e        x13 65732069        x14 00000003        x15 00000008
x16 65746e45        x17 00000000        x18 00000000        x19 00000000
x20 00000000        x21 00000000        x22 00000000        x23 00000000
x24 00000000        x25 00000000        x26 00000000        x27 00000000
x28 00000000        x29 00000000        x30 00000000        x31 00000000
```

Figure 1: A screenshot of the VGA display simulated in *CPUlator*.

Attribution Table

Task	Person
Load & Fetch	Yikun Wang
Decode	Yangyijian Zhou
Execute	Yikun Wang: jump, load, immdiate-register, system Yangyijian Zhou: branch, store, register-register
I/O	Yikun Wang