Computer Architecture, Semester 1, 2019
Assignment — RISC-V RV32I ISS — Stage 1

Your task for this assignment is to develop an instruction set simulator (ISS) for the RV32I subset of the RISC-V instruction set. An instruction set simulator is a program used by computer architects to simulate execution of a computer's instructions. It contains representations of the computer's memory and the internal registers of the CPU. It responds to commands that specify initialization and inspection of the memories and registers, and control execution of instructions.

The RISC-V instruction set is described in *The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA,* available on the course web site. The RV32I subset is described in Chapter 2 of the Instruction Set Manual. The instruction encoding is summarized in Chapter 25. For this assignment, you should implement just the RV32I base integer instruction set, with the following exceptions:

- FENCE: These instructions should be decoded as legal instructions, but perform no operation.
- ECALL, EBREAK: These instructions should be decoded as legal instructions, but a message should be displayed indicating they are unimplemented.

For load and store intructions, you can assume that the effective address is properly aligned. For any fetched instruction word that does not represent an RV32I instruction, a message should be displayed indicating the instruction is illegal. In Stage 2, you will implement exception handling, which will be used for illegal instructions.

I have provided a skeleton program on the course web site for you to use as a starting point. The program is written in C++, and is located in the Assignment file folder (also linked in the Assignment module). You can download either rv32sim.zip or rv32sim.tgz; the content is the same in each. The skeleton program implements processing of command-line options and input commands. Your task is to implement classes to model the processor and memory. Header files are provided showing the member functions required. You can add additional classes if you need to.

The only command-line option you need to implement at this stage is the -v option to enable verbose output. Further options may be implemented in Stage 2. If the -v option is specified on the command line, your program can display debugging information. If the option is omitted, your program should display only the output required for each command. Your program must format that output exactly as specified below, since the assessment process will compare you output with expected output.

The rv324sim program reads commands from the standard input stream, one command per line. The commands are:

Command	Operation performed
x*n*	Show the content of register x*n* in hex (*n* is register number, from 0 to 31). The value is displayed as 8 hex digits with leading 0s.
x*n* = *value*	Set register x*n* to *value* (*value* in hex).
pc	Show content of PC register in hex. The value is displayed as 8 hex digits with leading 0s.
pc = *address*	Set PC register to *address* (*address* in hex).
m *address*	Show the content of memory word at *address* (*address* in hex, rv32sim rounds it down to nearest word-aligned address). The value is displayed as 8 hex digits with leading 0s.
m *address* = *value*	Set memory word at *address* to *value* (*address* in hex, rv32sim rounds it down to nearest word-aligned address; *value* in hex).
l "*filename*"	Load memory from Intel hex format file named *filename*. If the file includes a start address record, the PC is set to the start address.
.	Execute one instruction.
. *n*	Execute *n* instructions.

Command	Operation performed
b *address*	Set an execution breakpoint at address. If the simulator is executing multiple instructions (. *n* command), it stops when the PC reaches address without executing that instruction. There is only one execution breakpoint; using the b command with a different address removes any previously set breakpoint.
b	Clear the breakpoint.

Each command may be followed by a comment, starting with the '#' character and extending to the end of the line. Blank lines are permitted, as are lines containing only a comment.

The initial value of all processor general purpose registers should be 0, and the initial value of the PC should also be 0. The memory should appear to have all locations initialized to 0. Your program should count the number of instructions executed. This will be reported on completion of execution.

You can test your ISS by using the "m" command to set memory locations to the encoded value of RISC-Vinstructions, using the "pc" command to set the PC to the start of the code, then using the "." command to execute the code. Alternatively, you can use the RISC-V GNU Compiler Toolchain (C compiler, assembler, linker, binutils; available at https://github.com/riscv/riscv-gnu-toolchain) to generate hex files to load into memory. We will use both of these processes when we assess your ISS.

Performance of an ISS program is important. Computer architects typically use them to develop code for embedded system, so they must be able to execute 100s of thousands of instructions per second. You should design your ISS with performance in mind. The skeleton program provided uses the native integer data type uint32_t to represent instructions and data, rather than using a dynamically allocated class-typed object or string. When you implement the memory, you should not attempt to represent it using a large array of words. Since addresses are 32 bits, that would imply an array of 2^{32} bytes. Instead, consider a representation that allocates blocks of memory on demand (that is, on the first read or write to an address within a block).

Please keep an eye on the Questions and Answers forum on the course web site. There will no doubt be questions of clarification of requirements arising that I will answer there. I will also announce incremental releases of a test suite that you should use to test your program.

You must develop your program and check it into a subdirectory named 2019/s1/ca/rv32sim in your SVN repository. I will provide a web submission script that will check out this subdirectory, make your ISS, and run it with several test cases. Compliance with this development process will count toward the assessment of the assignment. The script will compare your output with our expected output using the "diff -iw" command (differences ignoring case and white-space).

Your work for Stage 1 will be assessed in the web submission system based on the following criteria, with points awarded out of 1500:

- Program builds and runs using web submission script — 100 points
- Correct execution of instruction, based on the number of test cases that pass — 1300 points
- Program efficiency, based on run time not exceeding a limit — 100 points

The points for this assignment will comprise 15% of your final assessment for the course.

The deadline for submission is **11:59pm Sunday 28 April 2019**.

Postgraduate requirements

If you are enrolled in the postgraduate course (COMP SCI 7026), you should implement the following additional requirements:

During simulated execution, you should count the number of simulated clock cycles, in addition to the number of instructions executed. You should use the following cycle counts for various instruction types:

- Conditional branch instruction: 2 cycles if the branch is taken, or 1 cycle if the branch is not taken.

- Unconditional branch instruction: 2 cycles.
- Load instruction: 3 cycles if the loaded data is contained within a word, or 5 cycles if the loaded data straddles two words.
- Store instruction: 2 cycles if the stored data is contained within a word, or 4 cycles if the stored data straddles two words.
- All other instruction: 1 cycle.

On completion of a simulation, if the -c option is specified on the command line, your program will report the total number of simulated clock cycles taken.