Project 1

1. About the RISC-V ISA

RISC-V (pronounced “risk-five”) is an open, free ISA enabling a new era of processor innovation through open standard collaboration. Born in academia and research, RISC-V ISA delivers a new level of free, extensible software and hardware freedom on architecture, paving the way for the next **50** years of computing design and innovation.

The final **user-level ISA specification**, a **draft compressed ISA specification**, a **draft privileged ISA specification**, and a suite of **RISC-V software tools** including a **GNU/GCC software tool chain**, **GNU/GDB debugger**, an **LLVM compiler**, a **Spike ISA simulator**, **QEMU**, and a **verification suite** is available for download now.

To sample the architecture without installing anything, try out **ANGEL**, a JavaScript ISA simulator that boots an interactive session of riscv-linux on a simulated RISC-V machine in your browser.

Key Features of the RISC-V ISA:

- Delivers a new level of software and hardware freedom on architecture in an open extensible way.
- Open ISA delivers easier support from a broad range of operating systems, software vendors and tool developers.
- The open source of hardware, RISC-V does not rely on a single supplier – offers multiple suppliers, therefore, supports unlimited potential for future growth.
- No other ISA is architected like the RISC-V ISA, allowing for user extensibility of the architecture without breaking existing extensions or incurring software fragmentation
-

2. RISC-V Tools

This repository houses a set of RISC-V simulators, compilers, and other tools, including the following projects:

- Spike, the ISA simulator
- riscv-tests, a battery of ISA-level tests
- riscv-opcodes, the enumeration of all RISC-V opcodes executable by the simulator

- riscv-pk, which contains bbl, a boot loader for Linux and similar OS kernels, and pk, a proxy kernel that services system calls for a target-machine application by forwarding them to the host machine
- riscv-fesvr, the host side of a simulation tether that services system calls on behalf of a target machine

Quickstart

```
$ git clone https://github.com/riscv/riscv-tools.git
$ cd riscv-tools
$ git submodule update --init --recursive
$ export RISCV=/path/to/install/riscv/toolchain
$ ./build.sh
```

Ubuntu packages needed:

```
$ sudo apt-get install autoconf automake autotools-dev curl libmpc-dev
libmpfr-dev libgmp-dev libusb-1.0-0-dev gawk build-essential bison flex
texinfo gperf libtool patchutils bc zlib1g-dev device-tree-compiler pkg-
config libexpat-dev
```

Fedora packages needed:

```
$ sudo dnf install autoconf automake @development-tools curl dtc libmpc-devel
mpfr-devel gmp-devel libusb-devel gawk gcc-c++ bison flex texinfo gperf
libtool patchutils bc zlib-devel expat-devel
```

Note: This requires a compiler with C++11 support (e.g. GCC >= 4.8). To use a compiler different than the default, use:

```
$ CC=gcc-5 CXX=g++-5 ./build.sh
```

Note for OS X: We recommend using Homebrew to install the dependencies (libusb dtc gawk gnu-sed gmp mpfr libmpc isl wget automake md5sha1sum) or even to install the tools directly. This repo will build with Apple's command-line developer tools (clang) in addition to gcc.

Spike RISC-V ISA Simulator

Spike, the RISC-V ISA Simulator, implements a functional model of one or more RISC-V processors.

Spike is named after the golden spike used to celebrate the completion of the US transcontinental railway.

Compiling and Running a Simple C Program

Write a short C program and name it hello.c.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
        printf("Hello RISC-V\n");
        return 0;
}
```

Then, compile it into a RISC-V ELF binary named hello:

```
$ riscv64-unknown-elf-gcc -o hello hello.c
```

Now you can simulate the program atop the proxy kernel:

```
$ spike pk hello
```

Simulating a New Instruction

Adding an instruction to the simulator requires two steps:

1. Describe the instruction's functional behavior in the file riscv/insns/<new_instruction_name>.h. Examine other instructions in that directory as a starting point.

2. Add the opcode and opcode mask to riscv/opcodes.h. Alternatively, add it to the riscv-opcodes package, and it will do so for you:

```
$ cd ../riscv-opcodes
$ vi opcodes       // add a line for the new instruction
$ make install
```

3. Rebuild the simulator.

3. Task

In this project, we are going to process an image by using risc-v processor. Before that, I'd like to introduce some concepts of the image processing. First, I would introduce the concept of convolution. An image can be converted to a matrix as shown in Fig. 1.

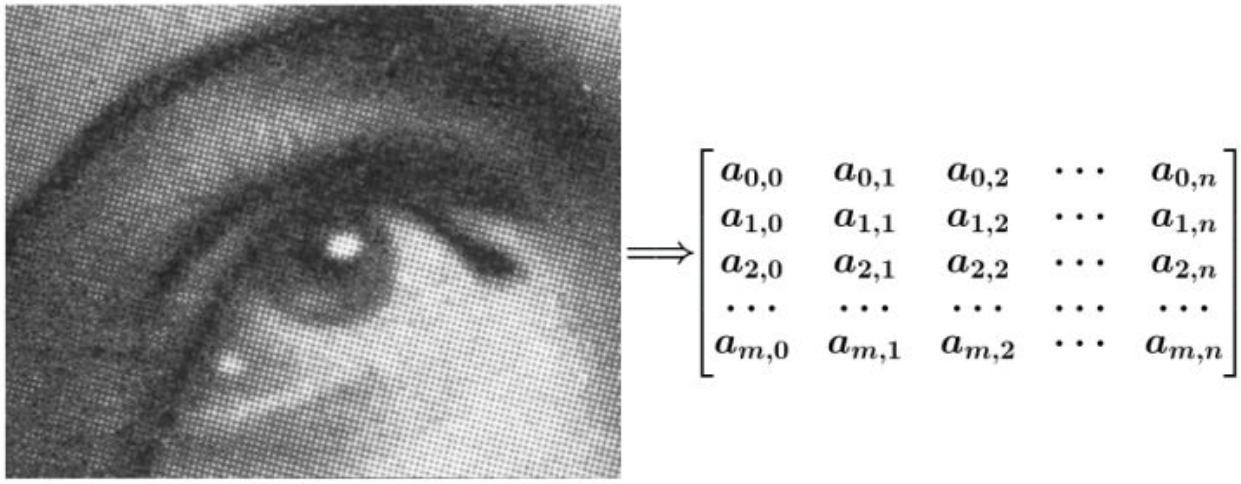

Fig. 1 An image is represented by a matrix

A convolutional filter is often used to process an image. The convolutional computing is defined as

$$(f * g)(1,1) = \sum_{k=0}^{2} \sum_{h=0}^{2} f(h,k) g(1-h, 1-k) \qquad (1)$$

Eqn. (1) can be illustrated by Fig. 2. The convolutional operation firstly rotates the filter by 180 degrees and then carries out the correlation of the rotated filter matrix with the input matrix.

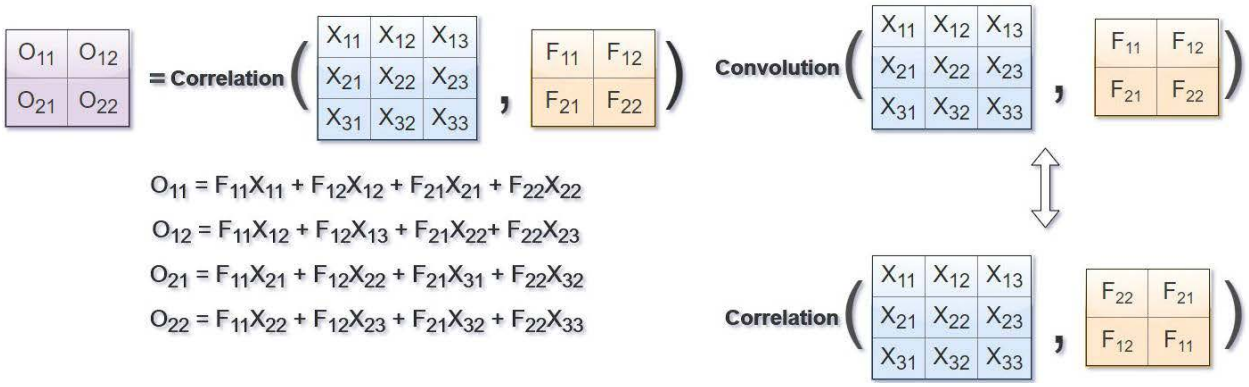

Fig. 2 Illustration of the convolutional operation

Fig. 3 A sample image

In this project, we will use Fig. 3 as the input image with the size of 2048x1024. The RGB channels of the image will be convoluted by three filters f_1, f_2, f_3, and f_4. Now we will give weights to different channels to see the effect after applying these filters to the image.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Mean Filter

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

Gaussian Filter

-1	-1	-1
-1	9	-1
-1	-1	-1

High-Pass Filter

1	1	1
1	-8	1
1	1	1

Laplacian Filter

To simplify, we use Eqn. (2) to illustrate the convolutional operations,

$$\begin{cases} y_1 = k_1 f_1 * g_r + k_2 f_1 * g_g + k_3 f_1 * g_b \\ y_2 = k_1 f_2 * g_r + k_2 f_2 * g_g + k_3 f_2 * g_b \\ y_3 = k_1 f_3 * g_r + k_2 f_3 * g_g + k_3 f_3 * g_b \\ y_4 = k_1 f_4 * g_r + k_2 f_4 * g_g + k_3 f_4 * g_b \end{cases} \quad (2)$$

where $k_1 + k_2 + k_3 = 1$.

Tasks:

1. Write a program in C to implement the above convolutional operation.

2. Compile and run the program in both RISCV32 and RISCV64 environments.
3. Please try different values of k1, k2, and k3. For example, $k1 = k2 = k3 = 1/3, k1 = k2 = 1/4, k3 = 1/2, k1 = k3 = 1/4, k2 = 1/2, k2 = k3 = 1/4, k1 = 1/2$.
4. Figure out what and how many instructions are used to run the program.
5. Find out the total execution time of the program.

Please note: the stride of the convolutional operation is 1, and zero padding should be added to the original image.

Evaluation:

Please try your best to optimize your code for both RV32 and RV64. The project will be evaluated and graded based on:

1. The efficiency of the code. For example, the number of instructions used to run the program, the execution time of the program, etc.
2. The quality of the code.