ECE391: Computer Systems Engineering **Fall 2024**
Machine Problem 0 **Due: 11:59PM CDT Friday September 6th**

Preparing your Environment

This assignment will help you to prepare your work environment for the class and will introduce you to the tools that you will be using. As mentioned in class, much of your work will be done in the context of RISCV code being executing on a virtual machine (VM) within an Apptainer container. Most of the actual programming and development will be done within the VM. You will demo all of your work to a staff member on one of the lab machines in ECEB.

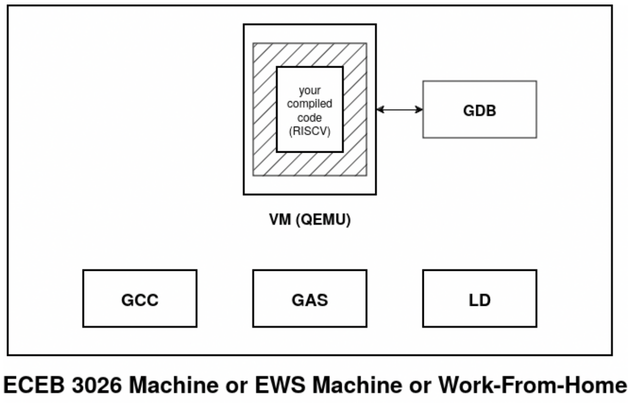

your compiled code (RISCV)
GDB
VM (QEMU)
GCC
GAS
LD
ECEB 3026 Machine or EWS Machine or Work-From-Home

Figure 1: ECE391 Setup Diagram

You may want to read the documentation on the tools—available on the ECE391 web page—to familiarize yourself with them before starting this assignment. The tools used in this lab include:

1. QEMU, (qemu-system-riscv64) emulates a RISC-V computer system, consisting of a RISC-V CPU and several peripherals. Your code will run inside QEMU
2. GNU C Compiler, (riscv64-unknown-elf-gcc) compiles C code to RISC-V executables. You will write your code (mostly) in C and compile to RISC-V.
3. GNU Assembler, (riscv64-unknown-elf-gas) assembles RISC-V assembly language to RISC-V executables. You will write some code in RISC-V assembly.
4. GNU Make, (make) manages the build process. You may need to modify some makefiles (build instructions) for your assignments.
5. GNU Debugger, (riscv64-unknown-elf-gdb) attaches to QEMU to debug programs running inside the emulator. You will use it to debug your code.
6. Git, a modern version control tool

If you would like to read up on Apptainer, please refer to here

In addition to the reference materials provided on the web page, you can find information online and can get hands-on experience by logging into one of the EWS Linux machines in ECEB or through WSL (Windows), or any command shell (Mac) on your own machines. We encourage you to learn to use the editor that you feel the most comfortable with, such as `vi(m)`,`emacs`,`nano`,`VSCode`, inside the VM. VSCode has been popular with previous students, so you may want to check it out (it is readily available in ECEB 3026 and EWS machines).

Keep in mind that familiarity with Git's command-line interface will be required both in the demo for MP0 and in general in the course.

Step by Step Instructions

Please read through this whole section before you begin, and follow the instructions carefully. Done correctly, on an unloaded network (which will not be the case near the deadline!), the entire machine problem takes less than two hours. Note that if anything gets corrupted, you will have to start over, so start early.

Accessing Your Work Environment

There are multiple ways of accessing your work environment for ECE391. The officially supported methods are

1. In-person access at ECEB 3026
2. Remote access through FastX
3. SSH access through `eceb-3026.ews.illinois.edu`

As always, any Linux-based EWS desktops located in ECEB, Grainger Library, etc. will be synced with the filesystem on a lab machine, and can be used to work on the MPs.

For a work-from-home setup, please skip to the next section to setup your environment. For all other methods read the section below.

Getting a Terminal:

Based on your method of access you'll have different ways of getting a terminal to run your development environment

1. **In-person access at ECEB 3026** : Open the Terminal application
2. **FastX** : Follow the instructions found here to reach a virtual machine through FastX. Then, open the Terminal application
3. **SSH** Run the command below to get SSH access to a lab machine. Use your NetID as your username and your NetID password as your password when logging in. Running

   ```
   ssh <your_netid>@eceb-3026.ews.illinois.edu
   ```

   should prompt you for your password. Enter your password to get a shell.

Starting Development Environment:

EWS computers include a convenience script to start your development environment. You can run

```
/class/ece391/app/ece391-riscv-dev
```

which should drop you into your development shell.

Warning: If you're using FastX, EWS, or an ECEB 3026 machine, you need to run this script **every time** you login

If you successfully see your development shell, please move onto **Running Star Trek 1971**

Setting up Work-From-Home

For Windows:

1. First follow the instructions here to install WSL and the Ubuntu distribution of Linux. The ECE391 Work-From-Home setup requires a modern install of WSL 2.0. If you're machine does not support WSL 2.0, please refer to other methods of accessing a suitable work environment.

2. Now that you've installed WSL with Ubuntu, you should have a new entry in your Start Menu, namely START/UBUNTU. Launching this application should show a terminal with a shell prompt. To verify that you've correctly installed WSL and Ubuntu, type into your shell prompt

   ```
   cat /etc/os-release
   ```

   You should see something along the lines of

   ```
   PRETTY_NAME="Ubuntu 24.04 LTS"
   NAME="Ubuntu"
   VERSION_ID="24.04"
   VERSION="24.04 LTS (Noble Numbat)"
   VERSION_CODENAME=noble
   ID=ubuntu
   ID_LIKE=debian
   ***
   ```

3. Now we need to update the packages on our Ubuntu install, run the command below which should bring your system up to date and install new packages

   ```
   sudo apt update && sudo apt upgrade -y && \
   sudo apt install -y autoconf automake autotools-dev curl \
   python3 python3-pip libmpc-dev libmpfr-dev libgmp-dev \
   gawk build-essential bison flex texinfo gperf libtool \
   patchutils bc zlib1g-dev libexpat-dev ninja-build \
   git cmake libglib2.0-dev libslirp-dev libpixman-1-dev libgtk-3-dev
   ```

4. Now we create a directory for our class and clone the RISCV toolchain

   ```
   cd ~ && mkdir ece391 && cd ece391
   git clone --branch 2024.04.12 https://github.com/riscv/riscv-gnu-toolchain
   cd riscv-gnu-toolchain
   ```

5. Now we need to prepare to compile the toolchain, we do so by

   (a) Creating a directory named `/opt/toolchains/` owned by your current user

   (b) Cloning the correct version of QEMU and adding our custom patch

   (c) Running the toolchain configuration script with the correct arguments

   (d) Running `make`

   Now start compiling the toolchain with, where `<user>` is the current logged in user's username. Before starting to compile, please download `qemu.patch` from the course website under `Assignments/MP0/Extra` and place it in `riscv-gnu-toolchain`

   ```
   # Create toolchain dir, and change owner
   sudo bash -c 'mkdir -p /opt/toolchains/riscv && chown -R <user>:<user>
    ↪  /opt/toolchains/riscv'
   # Clone QEMU and apply patch
   git clone --depth 1 --branch v9.0.2 https://github.com/qemu/qemu && cd
    ↪  qemu
   patch -p0 < ../qemu.patch
   # configure and make QEMU
   ./configure --prefix=/opt/toolchains/riscv
    ↪  --target-list=riscv32-softmmu,riscv64-softmmu --enable-gtk
    ↪  --enable-system --disable-werror
   ```

```
make && make install
# Leave directory, configure and make toolchain
cd ..
./configure --prefix=/opt/toolchains/riscv/ --enable-multilib
make
```

6. This should compile the toolchain and QEMU, we still need to make them available on our path so run

```
echo "PATH=/opt/toolchains/riscv/bin/:$PATH" >> ~/.bashrc
source ~/.bashrc
```

To verify that you've successfully compiled the toolchain, run

```
qemu-system-riscv64
```

in your home directory, which should pop up a QEMU window

You can now move onto the next section **Running Star Trek 1971**

For Linux:

Warning: Please make sure that you're running a modern version of a well-maintained Linux distribution. We recommend Ubuntu, but you should be able to create your work-from-home environment with any Debian-based or RedHat-based distribution.

7. Now we create a directory for our class and clone the RISCV toolchain

```
cd ~ && mkdir ece391 && cd ece391
git clone --branch 2024.04.12 https://github.com/riscv/riscv-gnu-toolchain
cd riscv-gnu-toolchain
```

(a) For RedHat-based distributions (like Fedora), run the following to install dependencies

```
sudo yum install autoconf automake python3 libmpc-devel mpfr-devel \
gmp-devel gawk  bison flex texinfo patchutils gcc gcc-c++ \
zlib-devel expat-devel libslirp-devel
```

Now start compiling the toolchain with, where `<user>` is the current logged in user's username.Before starting to compile, please download `qemu.patch` from the course website under `Assignments/MP0/Extra` and place it in `riscv-gnu-toolchain`

```
# Create toolchain dir, and change owner
sudo bash -c 'mkdir -p /opt/toolchains/riscv && chown -R <user>:<user>
 ↪  /opt/toolchains/riscv'
# Clone QEMU and apply patch
git clone --depth 1 --branch v9.0.2 https://github.com/qemu/qemu && cd
 ↪  qemu
patch -p0 < ../qemu.patch
# configure and make QEMU
./configure --prefix=/opt/toolchains/riscv
 ↪  --target-list=riscv32-softmmu,riscv64-softmmu --enable-gtk
 ↪  --enable-system --disable-werror
make && make install
# Leave directory, configure and make toolchain
cd ..
./configure --prefix=/opt/toolchains/riscv/ --enable-multilib
make
```

This should compile the toolchain and QEMU, we still need to make them available on our path so run

```
echo "PATH=/opt/toolchains/riscv/bin/:$PATH" >> ~/.bashrc
source ~/.bashrc
```

To verify that you've successfully compiled the toolchain, run

```
qemu-system-riscv64
```

in your home directory, which should pop up a QEMU window

(b) For Debian-based distributions (like Ubuntu), run the following to install dependencies

```
sudo apt install autoconf automake autotools-dev curl python3 \
python3-pip libmpc-dev libmpfr-dev libgmp-dev gawk build-essential \
bison flex texinfo gperf libtool patchutils bc zlib1g-dev \
libexpat-dev ninja-build git cmake libglib2.0-dev libslirp-dev \
libpixman-1-dev libgtk-3-dev
```

Now start compiling the toolchain with, where `<user>` is the current logged in user's username. Before starting to compile, please download `qemu.patch` from the course website under `Assignments/MP0/Extra` and place it in `riscv-gnu-toolchain`

```
# Create toolchain dir, and change owner
sudo bash -c 'mkdir -p /opt/toolchains/riscv && chown -R <user>:<user>
 ↪  /opt/toolchains/riscv'
# Clone QEMU and apply patch
git clone --depth 1 --branch v9.0.2 https://github.com/qemu/qemu && cd
 ↪  qemu
patch -p0 < ../qemu.patch
# configure and make QEMU
./configure --prefix=/opt/toolchains/riscv
 ↪  --target-list=riscv32-softmmu,riscv64-softmmu --enable-gtk
 ↪  --enable-system --disable-werror
make && make install
# Leave directory, configure and make toolchain
cd ..
./configure --prefix=/opt/toolchains/riscv/ --enable-multilib
make
```

This should compile the toolchain and QEMU, we still need to make them available on our path so run

```
echo "PATH=/opt/toolchains/riscv/bin/:$PATH" >> ~/.bashrc
source ~/.bashrc
```

To verify that you've successfully compiled the toolchain, run

```
qemu-system-riscv64
```

in your home directory, which should pop up a QEMU window

You can now move onto the next section **Running Star Trek 1971**

For Mac OS:

The native setup for Mac OS has demonstrated issues affecting the setup process. If you would like to work remotely with Mac OS, it is recommended to either use FastX or SSH X-forwarding.

Running Star Trek 1971

Star Trek 1971 is a classic text-based strategy video game. In game, you control the USS Enterprise and you're on a mission to hunt down and destry enemy ships. If you'd like to know more about this hidden gem of a game, check out the Wikipedia page

To run Star Trek 1971 we need to:

1. Create a repository through the repo-creator bot
2. Pull and merge the MP0 branch from the .release repo
3. Run the included binary

We achieve this as follows

1. First follow the instructions on `https://edu.cs.illinois.edu/create-gh-repo/fa24_ece391` to create your class repo, and clone it with

   ```
   git clone
    ↪  https://github.com/illinois-cs-coursework/fa24_ece391_<your_netid_here>.git
   cd fa24_ece391_<your_netid_here>
   ```

2. Next, add the .release repo to your local repository as a remote. To do this run

   ```
   git remote add release
    ↪  https://github.com/illinois-cs-coursework/fa24_ece391_.release.git
   ```

3. Now we need to pull the MP0 branch from the .release repository into our own MP0 branch. To do so you can run

   ```
   git branch mp0
   git switch mp0
   git config pull.rebase false
   git pull release mp0 --allow-unrelated-histories
   ```

   After this, you should have a new branch on your repository named MP0,

4. Now you can demonstrate that you've setup for the class by running

   ```
   make
   qemu-system-riscv64 -m 128M -machine virt -bios none -kernel
    ↪  kernel.elf -nographic
   ```

   which should launch Star Trek 1971 in your terminal. Make sure to play the game and confirm that all functionality is working since this is what you'll be asked to demonstrate to course staff during demo.

Handin

For handin, find any TA in office hours and show your working Trek game.

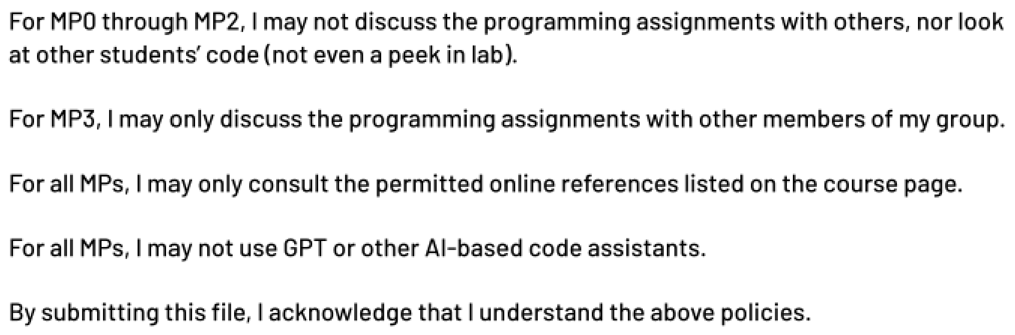
In addition, **you will be asked to demonstrate basic knowledge of Git, GDB, and version control concepts**. Being able to use GDB to debug your code is a critical skill for success in this class.

For Git, you may want to read the documentation on the course web page in preparation, particularly if you have not already used Git in previous classes (version control systems are all solving the same problem, so don't panic).

Version control, in the context of this course, is a tool to manage and track changes to your program code. When there are multiple collaborators working on the same project, it becomes a nearly indispensable tool.

For all ECE 391 assignments, **you will be required to learn and use Git**. All your repositories will be stored on Engineering IT's instance of Github. **You may log in using your NetID and password.** At the beginning of the semester and before each of the remaining assignments, you'll have to fetch the release repo and merge.

In addition to handing in to a TA at office hours you also need to submit the text below, typed into a file called `academic_integrity.txt` into the `MP0` branch of your class repository.



For MP0 through MP2, I may not discuss the programming assignments with others, nor look at other students' code (not even a peek in lab).

For MP3, I may only discuss the programming assignments with other members of my group.

For all MPs, I may only consult the permitted online references listed on the course page.

For all MPs, I may not use GPT or other AI-based code assistants.

By submitting this file, I acknowledge that I understand the above policies.

Figure 2: Academic Integrity Policy

Git Commands To Be Tested :

1. `git init`
2. `git clone`
3. `git branch`
4. `git remote`
5. `git commit`
6. `git pull`
7. `git push`
8. `git merge`
9. `git reset/revert`
10. `git diff`
11. `git stash`

A Couple of Tips

Your work directory will be deleted at the end of the semester. Be sure to save any source code that you wish to retain by moving it to your home directory or to your own personal data storage.

If you would like to understand the virtual machine organization and how the environment works, ask a TA.