RUNNING LINUX ON RISC-V ON QEMU

This documentation is for the purpose of booting LINUX on RISC-V QEMU and running a command to print “Hello World!” on the console.

PREREQUISITES:-

Before starting run these commands to ensure that we have the dependencies that are needed.

```
sudo apt install autoconf automake autotools-dev curl libmpc-dev libmpfr-dev libgmp-dev \
                 gawk build-essential bison flex texinfo gperf libtool patchutils bc \
                 zlib1g-dev libexpat-dev git
```

GETTING THE SOURCES

Next, we make the directory where all the sources will be downloaded and built.

```
mkdir riscv64-linux
cd riscv64-linux
```

We need 3 sources, namely

1) QEMU
2) LINUX
3) BUSYBOX

To obtain them, run the following commands one by one.

```
git clone https://github.com/qemu/qemu
git clone https://github.com/torvalds/linux
git clone https://git.busybox.net/busybox
```

SETTING UP GCC

Due to compatibility issues, gcc-8 is required. To check if you already have the required version,

```
gcc --version
```

If it shows any version except 8.X.X,

```
sudo apt-get install gcc-8
sudo ln -fs gcc-8 /usr/bin/gcc
```

Check the gcc version again and you should see it changed to 8.X.X.

INSTALLING THE RISC-V TOOLCHAIN

Before we proceed, the risc-v toolchain is crucial. Clone the repo.

```
git clone https://github.com/riscv/riscv-gnu-toolchain
```

To make sure we have toolchain compatibility, run the command,

```
sudo apt-get install autoconf automake autotools-dev curl python3 libmpc-dev libmpfr-dev
libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-
dev libexpat-dev
```

Next, cd into the toolchain directory.

To build the Newlib cross-compiler, we choose a path, for example */opt/riscv*.

```
./configure --prefix=/opt/riscv
make
```

To build the LINUX cross-compiler, we do the same (i.e. choose a path, for example */opt/riscv*).

```
./configure --prefix=/opt/riscv
make linux
```

To add the *bin* folder to the **PATH**, on the terminal:

```
nano ~/.bashrc
```

At the end of the file, type

```
export PATH=”/opt/riscv/bin:$PATH”
```

Save the modifications to the file and proceed to execute:

```
nano ~/.profile
```

At the end of the file, type

```
# set PATH so it includes riscv gnu bin if it exists

if [ -d "opt/riscv/bin" ] ; then

    PATH="/opt/riscv/bin:$PATH"

fi
```

Save the modifications to the file and switch back to our original working terminal.

Just verify, run

```
echo $PATH
```

And see if /opt/riscv/bin is present. If not, restart your pc.

QEMU

Now we start building QEMU.

```
cd qemu

./configure --target-list=riscv64-softmmu

make -j $(nproc)
sudo make install
```

LINUX

For LINUX build, cd out of the qemu directory, then

```
cd linux
make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu- defconfig
```

Compile the kernel

```
make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu- -j $(nproc)
```

BUSYBOX

After having done with LINUX, cd out and,

```
cd busybox
CROSS_COMPILE=riscv{{bits}}-unknown-linux-gnu- make defconfig
CROSS_COMPILE=riscv{{bits}}-unknown-linux-gnu- make -j $(nproc)
```

where {{bits}} is a placeholder for the desired the risc-v bitset, i.e. 64, 32

RUNNING

Before running QEMU, we need an image file for LINUX which can be downloaded from
https://wdc.app.box.com/s/ihywc2xap5m4mflyngjtndf0sy62zha3

Go ahead and download the linux_rootfs.img file and place it in our working directory (i.e. the directory containing the qemu, linux and busybox sub-directories).

Then, run

```
sudo qemu-system-riscv64 -nographic -machine virt \
      -kernel linux/arch/riscv/boot/Image -append "root=/dev/vda ro console=ttyS0" \
      -drive file=linux_rootfs.img,format=raw,id=hd0 \
      -device virtio-blk-device,drive=hd0
```

Running the aforementioned command boots LINUX on our RISC-V QEMU. In the buildroot input, enter “root” (without quotes) and type

```
echo ‘Hello World!’
```

TROUBLESHOOTING

If you get any error, restart your pc. Some commands require a restart to take effect.