

# Installation Guide for Spike

This guide covers a step-by-step installation process of the most common workflows for RISC-V development. We require three basic submodules:

Submodule	Contents
riscv-gnu-toolchain	binutils, gcc, newlib, glibc, Linux UAPI headers
riscv-isa-sim	Functional ISA simulator ("Spike")
riscv-pk	RISC-V Proxy Kernel

In the end, an implementation of a simple C program and RISC-V assembly program will be illustrated.

## Preliminaries

Open the terminal, and install git, make and device-tree-compiler by using the following commands:

```
sudo apt install git
```

```
sudo apt install make
```

```
sudo apt install device-tree-compiler
```

(Note: if you're using CentOS or RHEL use **sudo yum install dtc** to install device-tree-compiler)

Sometimes when you are using a basic version of ubuntu it is required to install additional dependencies which are given below:

```
sudo apt 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
```

(Note: You can skip the above steps if you have already installed them.)

Then make a new folder and name it **RISCV64**, in any directory you are comfortable with e.g. home directory.

```
mkdir RISCV64
```

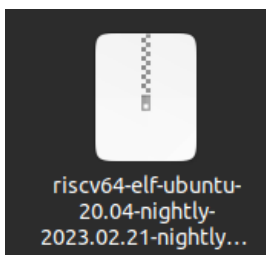
Now we will install the required submodules in a continuous manner.

## RISC-V GNU Toolchain

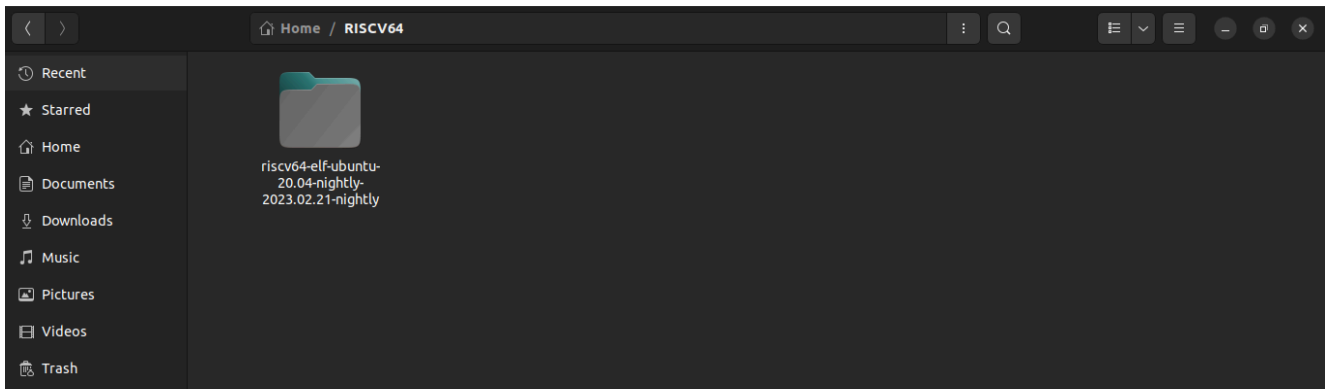
Download the latest GNU toolchain release from the link given below:

[Releases · riscv-collab/riscv-gnu-toolchain](#)

(Note: As we are working on Ubuntu. Make sure to download the correct version. You can check the version in **settings -> about**.)



Unzip, or Extract the file to the folder **RISCV64** we created before. After extraction, it would appear as follows;



We can rename the folder to **riscv64-unknown-elf-toolchain** to avoid complexity in the following commands. It can have a different name but then you have to use commands accordingly. Also, if anything fails, try a different version of the toolchain.

(Note: The same toolchain can be used for the latest versions of RHEL and CentOS. Otherwise you have to build it using this github repository [GNU toolchain for RISC-V, including GCC](#) ).

## RISC-V ISA Simulator

- 1) Now, we need to clone the riscv-isa-sim file from the link given below, in the same directory, i.e. **RISCV64**.

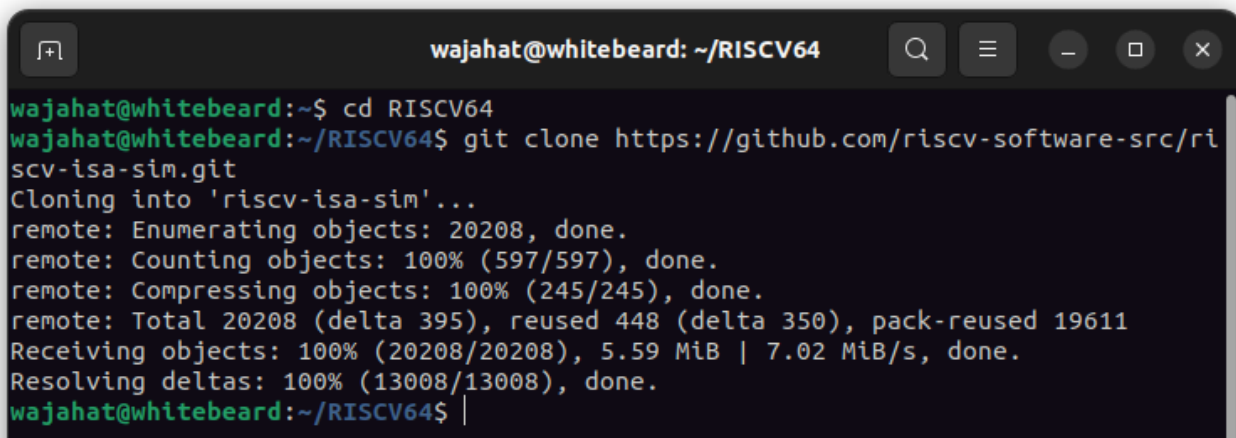
Nov 7, 2022

[GitHub - riscv-software-src/riscv-isa-sim: Spike, a RISC-V ISA Simulator](https://github.com/riscv-software-src/riscv-isa-sim)

To clone this file we would use the following terminal command.

```
git clone https://github.com/riscv-software-src/riscv-isa-sim.git
```

Once you're done cloning the *riscv-isa-sim*, the terminal window will show you the following message:

A terminal window titled 'wajahat@whitebeard: ~/RISCV64' showing the execution of 'git clone https://github.com/riscv-software-src/riscv-isa-sim.git'. The output shows the cloning process: 'Cloning into 'riscv-isa-sim'...', 'remote: Enumerating objects: 20208, done.', 'remote: Counting objects: 100% (597/597), done.', 'remote: Compressing objects: 100% (245/245), done.', 'remote: Total 20208 (delta 395), reused 448 (delta 350), pack-reused 19611', 'Receiving objects: 100% (20208/20208), 5.59 MiB | 7.02 MiB/s, done.', and 'Resolving deltas: 100% (13008/13008), done.' The prompt returns to 'wajahat@whitebeard:~/RISCV64\$'.

Now we would use the following commands one by one to build the spike simulator in the *riscv-isa-sim* directory.

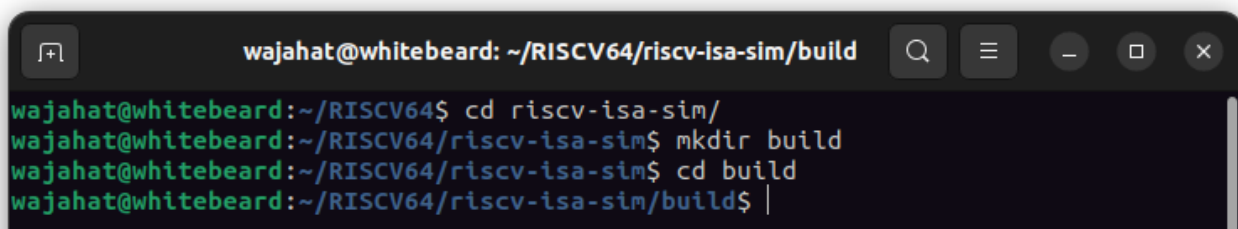
Afterwards, we will create a folder within the *riscv-isa-sim* directory by running the following commands:

```
cd riscv-isa-sim
```

```
mkdir build
```

Now we will move within this folder using the following command:

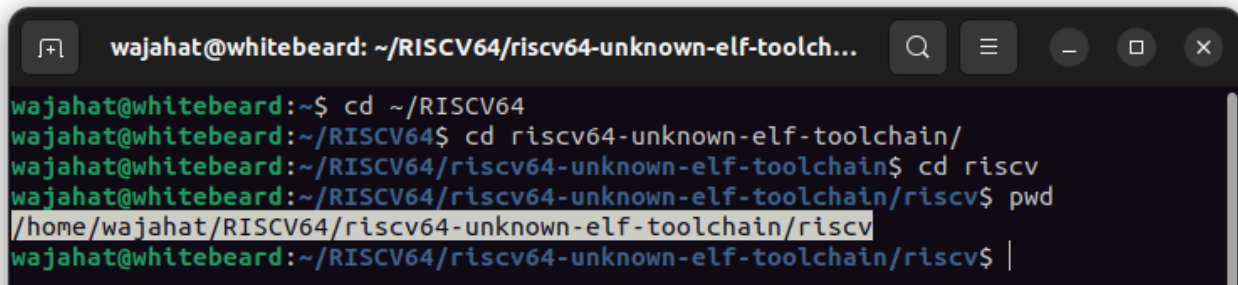
```
cd build
```

A terminal window titled 'wajahat@whitebeard: ~/RISCV64/riscv-isa-sim/build' showing the execution of 'cd riscv-isa-sim/', 'mkdir build', 'cd build', and the final prompt 'wajahat@whitebeard:~/RISCV64/riscv-isa-sim/build\$'.

Then we would link the path of the toolchain directory, which would then be used in the

next command.

To obtain the path, we will open the **riscv64-unknown-elf-toolchain** folder present in the **RISCV64** folder. Then we will open the folder named **riscv** present in the **riscv64-unknown-elf-toolchain** folder. Then we would open the terminal from within this folder and find the complete path using the command **pwd** command. If you created the **RISCV64** folder in home directory, then the path would appear as

A terminal window titled 'wajahat@whitebeard: ~/RISCV64/riscv64-unknown-elf-toolch...' shows the following commands and output:

```
wajahat@whitebeard:~$ cd ~/RISCV64
wajahat@whitebeard:~/RISCV64$ cd riscv64-unknown-elf-toolchain/
wajahat@whitebeard:~/RISCV64/riscv64-unknown-elf-toolchain$ cd riscv
wajahat@whitebeard:~/RISCV64/riscv64-unknown-elf-toolchain/riscv$ pwd
/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv
wajahat@whitebeard:~/RISCV64/riscv64-unknown-elf-toolchain/riscv$ |
```

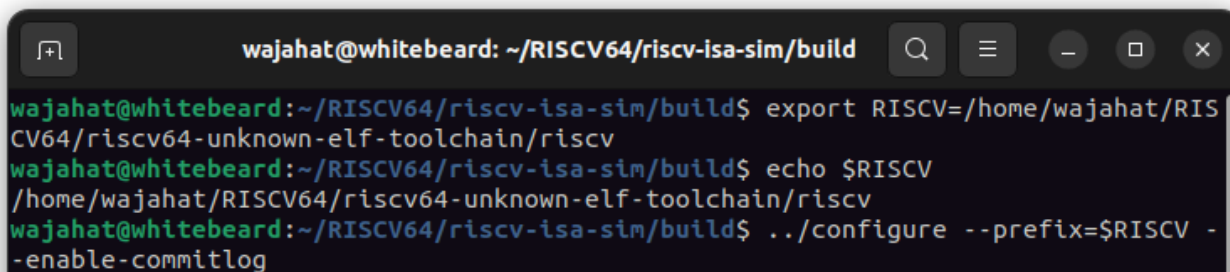
***/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv***

Now enter the following command in the build directory we created previously, for linking the path is as follows:

***export RISCV=/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv***

You can then use the following command to check whether that path was assigned to RISCV, or not.

***echo \$RISCV***

A terminal window titled 'wajahat@whitebeard: ~/RISCV64/riscv-isa-sim/build' shows the following commands and output:

```
wajahat@whitebeard:~/RISCV64/riscv-isa-sim/build$ export RISCV=/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv
wajahat@whitebeard:~/RISCV64/riscv-isa-sim/build$ echo $RISCV
/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv
wajahat@whitebeard:~/RISCV64/riscv-isa-sim/build$ ../configure --prefix=$RISCV -enable-commitlog
```

Then we will run the following command:

***../configure --prefix=\$RISCV --enable-commitlog***

(***--enable-commitlog*** flag is optional. This is used if you want to generate a log file)

The output will look like as given below:

```
wajahat@whitebeard: ~/RISCV64/riscv-isa-sim/build
checking for main in -lboost_regex... no
checking for library containing dlopen... none required
checking for pthread_create in -lpthread... (cached) yes
configure: configuring default subproject : disasm
configure: configuring default subproject : customext
configure: configuring default subproject : fdt
configure: configuring default subproject : softfloat
configure: configuring default subproject : spike_main
configure: configuring default subproject : spike_dasm
configure: creating ./config.status
config.status: creating fesvr.mk
config.status: creating riscv.mk
config.status: creating disasm.mk
config.status: creating customext.mk
config.status: creating fdt.mk
config.status: creating softfloat.mk
config.status: creating spike_main.mk
config.status: creating spike_dasm.mk
config.status: creating Makefile
config.status: creating riscv-fesvr.pc
config.status: creating riscv-disasm.pc
config.status: creating config.h
configure: WARNING: unrecognized options: --enable-commitlog
wajahat@whitebeard:~/RISCV64/riscv-isa-sim/build$
```

Then enter the following command:

***make***

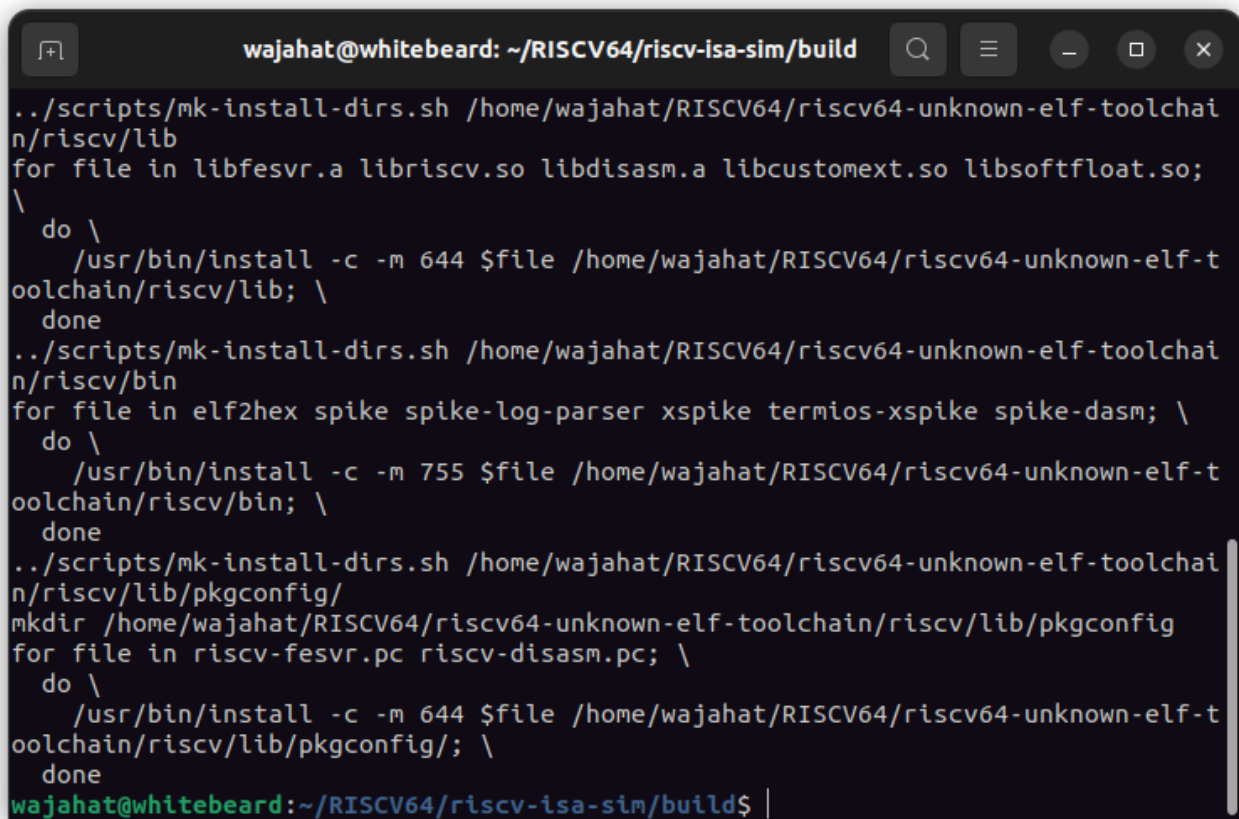
```
wajahat@whitebeard: ~/RISCV64/riscv-isa-sim/build
wajahat@whitebeard:~/RISCV64/riscv-isa-sim/build$ make
g++ -MMD -MP -DPREFIX=\"/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv\" -Wall -Wno-unused -Wno-nonportable-include-path -g -O2 -fPIC -g -O2 -DPREFIX=\"/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv\" -Wall -Wno-unused -Wno-nonportable-include-path -g -O2 -fPIC -std=c++17 -g -O2 -I. -I.. -I../fesvr -I../riscv -I../disasm -I../customext -I../fdt -I../softfloat -I../spike_main -I../spike_dasm -I/include/boost-0 -c ../fesvr/elfloader.cc
g++ -MMD -MP -DPREFIX=\"/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv\" -Wall -Wno-unused -Wno-nonportable-include-path -g -O2 -fPIC -g -O2 -DPREFIX=\"/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv\" -Wall -Wno-unused -Wno-nonportable-include-path -g -O2 -fPIC -std=c++17 -g -O2 -I. -I.. -I../fesvr -I../riscv -I../disasm -I../customext -I../fdt -I../softfloat -I../spike_main -I../spike_dasm -I/include/boost-0 -c ../fesvr/elfloader.cc
```

(Note: This is going to take some time so have some coffee.)

Then run the following command to install:

***sudo make install***

Output will look like as given below:



```
wajahat@whitebeard: ~/RISCV64/riscv-isa-sim/build
../scripts/mk-install-dirs.sh /home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv/lib
for file in libfesvr.a libriscv.so libdisasm.a libcustomext.so libsoftfloat.so; \
do \
    /usr/bin/install -c -m 644 $file /home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv/lib; \
done
../scripts/mk-install-dirs.sh /home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv/bin
for file in elf2hex spike spike-log-parser xspike termios-xspike spike-dasm; \
do \
    /usr/bin/install -c -m 755 $file /home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv/bin; \
done
../scripts/mk-install-dirs.sh /home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv/lib/pkgconfig/
mkdir /home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv/lib/pkgconfig
for file in riscv-fesvr.pc riscv-disasm.pc; \
do \
    /usr/bin/install -c -m 644 $file /home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv/lib/pkgconfig/; \
done
wajahat@whitebeard:~/RISCV64/riscv-isa-sim/build$
```

Now, we are done with cloning and installing the **riscv-isa-sim** submodule, and the next step is to clone and build the **riscv-proxy** kernel.

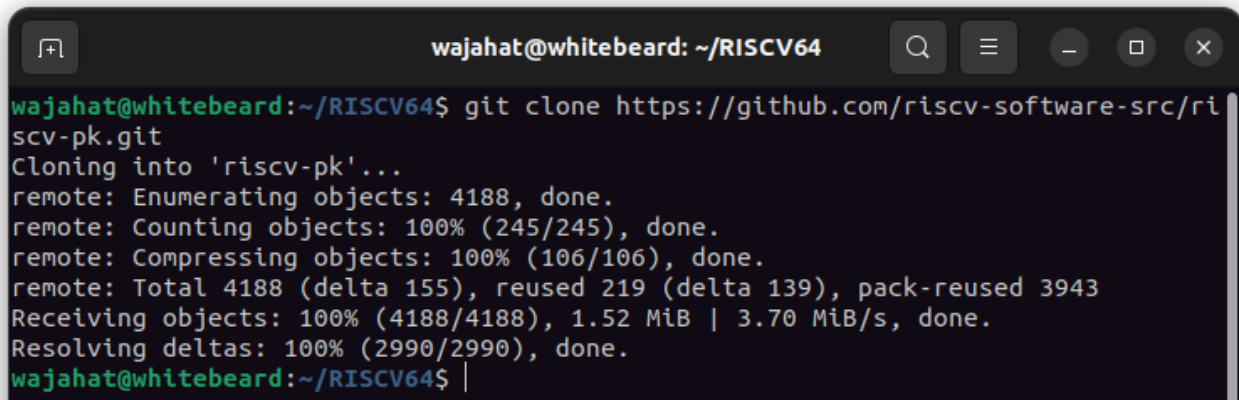
## RISC-V Proxy Kernel

Now, we need to clone the riscv-proxy kernel file from the link given below, in the **RISCV64** directory.

[GitHub - riscv-software-src/riscv-pk: RISC-V Proxy Kernel](https://github.com/riscv-software-src/riscv-pk)

We will clone the repository using the following command:

```
git clone https://github.com/riscv-software-src/riscv-pk.git
```

A terminal window titled 'wajahat@whitebeard: ~/RISCV64' showing the execution of 'git clone https://github.com/riscv-software-src/riscv-pk.git'. The output shows the cloning process, including enumerating, counting, and compressing objects, and receiving the repository data.

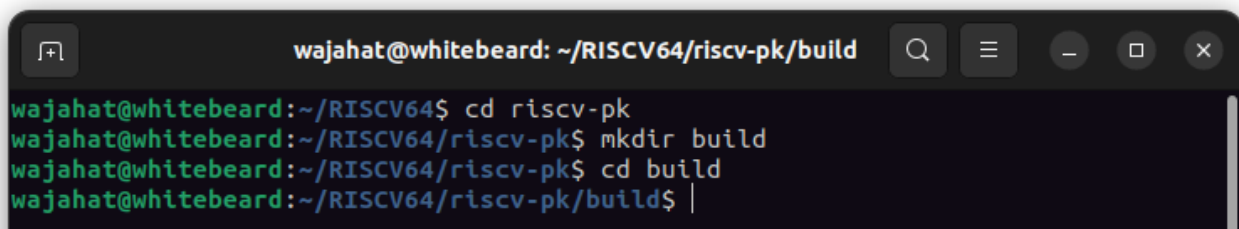
```
wajahat@whitebeard: ~/RISCV64
wajahat@whitebeard:~/RISCV64$ git clone https://github.com/riscv-software-src/riscv-pk.git
Cloning into 'riscv-pk'...
remote: Enumerating objects: 4188, done.
remote: Counting objects: 100% (245/245), done.
remote: Compressing objects: 100% (106/106), done.
remote: Total 4188 (delta 155), reused 219 (delta 139), pack-reused 3943
Receiving objects: 100% (4188/4188), 1.52 MiB | 3.70 MiB/s, done.
Resolving deltas: 100% (2990/2990), done.
wajahat@whitebeard:~/RISCV64$
```

After cloning we will create a directory within the **riscv-pk** directory. Once you're done cloning the pk, you need to follow the following steps.

**`cd riscv-pk`**

**`mkdir build`**

**`cd build`**

A terminal window titled 'wajahat@whitebeard: ~/RISCV64/riscv-pk/build' showing the navigation to the 'riscv-pk' directory, creating a 'build' directory, and then entering the 'build' directory.

```
wajahat@whitebeard: ~/RISCV64/riscv-pk/build
wajahat@whitebeard:~/RISCV64$ cd riscv-pk
wajahat@whitebeard:~/RISCV64/riscv-pk$ mkdir build
wajahat@whitebeard:~/RISCV64/riscv-pk$ cd build
wajahat@whitebeard:~/RISCV64/riscv-pk/build$
```

In the next step, we need to link the directory, as we did before but there is one small twist, which most of you will find confusing at times. Sometimes when you end a session or open a new session, the paths get lost. Therefore, we need to be sure that our paths are correct.

Having said that, this time we need to append the link of the bin directory of the toolchain with PATH, which is a system defined directory. Kindly note that we would not be overwriting the value of PATH with the one we want to use, instead, we would append the path of the bin directory with the one already present. We would use the following linux command.

**`export RISCV=/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv`**

**`export PATH="${PATH}:${RISCV/bin}"`**



```
wajahat@whitebeard: ~/RISCV64/riscv-pk/build
wajahat@whitebeard:~/RISCV64/riscv-pk/build$ export RISCV=/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv
wajahat@whitebeard:~/RISCV64/riscv-pk/build$ export PATH="${PATH}:${RISCV}/bin"
wajahat@whitebeard:~/RISCV64/riscv-pk/build$ |
```

Now, we will run the following command:

```
../configure --prefix=$RISCV --host=riscv64-unknown-elf
```

```
wajahat@whitebeard: ~/RISCV64/riscv-pk/build
wajahat@whitebeard:~/RISCV64/riscv-pk/build$ ../configure --prefix=$RISCV --host=riscv64-unknown-elf
checking build system type... x86_64-pc-linux-gnu
checking host system type... riscv64-unknown-elf
checking for riscv64-unknown-elf-gcc... riscv64-unknown-elf-gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... yes
checking for suffix of object files... o
checking whether the compiler supports GNU C... yes
checking whether riscv64-unknown-elf-gcc accepts -g... yes
checking for riscv64-unknown-elf-gcc option to enable C11 features... none needed
checking for riscv64-unknown-elf-g++... riscv64-unknown-elf-g++
checking whether the compiler supports GNU C++... yes
checking whether riscv64-unknown-elf-g++ accepts -g... yes
checking for riscv64-unknown-elf-g++ option to enable C++11 features... none needed
checking for riscv64-unknown-elf-ar... riscv64-unknown-elf-ar
checking for riscv64-unknown-elf-ranlib... riscv64-unknown-elf-ranlib
checking for riscv64-unknown-elf-readelf... riscv64-unknown-elf-readelf
checking for riscv64-unknown-elf-objcopy... riscv64-unknown-elf-objcopy
checking for a BSD-compatible install... /usr/bin/install -c
```

If anything fails try a different version of the toolchain. Now we will build and install using the following command:

```
make
```



```
wajahat@whitebeard: ~/RISCV64/riscv-pk/build
ftffloat -I../dummy_payload -I../machine -I../util -c ../util/string.c
riscv64-unknown-elf-ar rcv libutil.a snprintf.o string.o
a - snprintf.o
a - string.o
riscv64-unknown-elf-ranlib libutil.a
riscv64-unknown-elf-gcc -MMD -MP -Wall -Werror -D__NO_INLINE__ -mcmodel=medany -
O2 -std=gnu99 -Wno-unused -Wno-attributes -fno-delete-null-pointer-checks -fno-P
IE -DBBL_LOGO_FILE=\"bbl_logo_file\" -DMEM_START=0x80000000 -fno-stack-protec
tor -U_FORTIFY_SOURCE -DBBL_PAYLOAD=\"bbl_payload\" -I. -I../pk -I../bbl -I../so
ftffloat -I../dummy_payload -I../machine -I../util -c ../pk/pk.c
riscv64-unknown-elf-gcc -Wl,--build-id=none -nostartfiles -nostdlib -static -
fno-stack-protector -o pk pk.o -L. -lpk -lmachine -lsoftfloat -lutil -lgcc -
Wl,--defsym=MEM_START=0x80000000,-T,../pk/pk.lds
riscv64-unknown-elf-gcc -MMD -MP -Wall -Werror -D__NO_INLINE__ -mcmodel=medany -
O2 -std=gnu99 -Wno-unused -Wno-attributes -fno-delete-null-pointer-checks -fno-P
IE -DBBL_LOGO_FILE=\"bbl_logo_file\" -DMEM_START=0x80000000 -fno-stack-protec
tor -U_FORTIFY_SOURCE -DBBL_PAYLOAD=\"bbl_payload\" -I. -I../pk -I../bbl -I../so
ftffloat -I../dummy_payload -I../machine -I../util -c ../bbl/bbl.c
riscv64-unknown-elf-gcc -Wl,--build-id=none -nostartfiles -nostdlib -static -
fno-stack-protector -o bbl bbl.o -L. -lbbl -ldummy_payload -lmachine -lsoftf
loat -lutil -lgcc -Wl,--defsym=MEM_START=0x80000000,-T,../bbl/bbl.lds
riscv64-unknown-elf-objcopy -S -O binary --change-addresses -0x80000000 bbl bbl.
bin
wajahat@whitebeard:~/RISCV64/riscv-pk/build$ |
```

### ***make install***

This would give us the following output:

```
wajahat@whitebeard: ~/RISCV64/riscv-pk/build
toolchain/riscv/riscv64-unknown-elf/lib/riscv-pk; \
done
../scripts/mk-install-dirs.sh //home/wajahat/RISCV64/riscv64-unknown-elf-toolcha
in/riscv/riscv64-unknown-elf/bin
for file in pk bbl dummy_payload ; \
do \
/usr/bin/install -c -m 755 $file //home/wajahat/RISCV64/riscv64-unknown-elf-
toolchain/riscv/riscv64-unknown-elf/bin; \
done
wajahat@whitebeard:~/RISCV64/riscv-pk/build$ |
```

Running a little sanity check to see if spike is working by entering the following command:

### ***spike pk***

**Note:** As discussed before you have added a temporary path and it goes away after a session. You might have to add it permanently by appending the paths in the **.bashrc** file.

Now, we are done with cloning and installing the **riscv-pk** submodule.

All modules have been installed and we will illustrate a trivial compilation pipeline

## Implementation of a Simple C Program Using Spike

Now, let's try it for a simple C program. Create a file `main.c` with the following code block.

```
#include<stdio.h>
int main()
{
    printf("My name is Wajahat Riaz\n");
    return 0;
}
```

Open the terminal to compile and run this program with the help of the following commands:

Compiling the `main.c` file to create a relocatable object file which will then pass to the linker in the later stage.

```
riscv64-unknown-elf-gcc -c main.c -o main.o
```

Now, creating the final executable by linking this relocatable object file **main.o** and other necessary standard library files. (It produces a binary executable with **.out** extension).

```
riscv64-unknown-elf-gcc main.o
```

Now, running the executable using spike

```
spike pk ./a.out
```

## Running RISC-V Assembly programs on Spike

Now, let's try to run an assembly file. Firstly, create a directory for practicing assembly. Firstly, download all the required files e.g. A Linker file can be found in the google drive link: [Linkerfile](#), The header files and the test file can be found in the google drive link: [Otherfiles](#). Include all these files in your folder and run the following commands:

```
riscv32-unknown-elf-gcc -march=rv32i -mabi=ilp32 -nostdlib -T link.ld test.S -o test.elf
```

If you're using riscv64-unknown-elf-gcc (as we did previously for the C program) enter the following command instead

```
riscv64-unknown-elf-gcc -march=rv64gc -mabi=lp64 -nostdlib -T link.ld test.S -o test.elf
```

Description of above command is as **-march** flag requires the architecture specification, **-mabi** requires **abi** for **rv32** or **rv64**, **-nostdlib** includes no standard library, **link.ld** is the linker file, **test.S** is your assembly file and last is object file that is created. Run the following command to see your dis-assembly file

```
riscv32-unknown-elf-objdump -d test.elf
```

If you're using riscv64-unknown-elf-gcc enter the following command instead

```
riscv64-unknown-elf-objdump -d test.elf
```

Now run the following command to generate the log file and see how it is executed e.g. contents of registers:

```
spike --isa=RV32IMAFDC -I --log-commits test.elf 1>spike.out 2>spike.log
```

If you're using riscv64-unknown-elf-gcc enter the following command instead

```
spike --isa=RV64IMAFDC -I --log-commits test.elf 1>spike.out 2>spike.log
```

## Running Risc-V Assembly programs compatible to RiscOf on spike

Run the following command to compile your assembly file.

```
riscv32-unknown-elf-gcc -march=rv32i -static -mcmmodel=medany  
-fvisibility=hidden -nostdlib -nostartfiles -T <path-to-linker-file>/link.ld -I  
<path-to-modeltest.h> -I  
<path-to-folder-containing-archtest.h,encodings.h,test_macros.h> -mabi=ilp32  
test.S -o my.elf -Drvtest_mtrap_routine=True -DTEST_CASE_1=True -DXLEN=32
```

Sample command will for above will look like as:

```
riscv32-unknown-elf-gcc -march=rv32i -static  
-mcmmodel=medany-fvisibility=hidden -nostdlib -nostartfiles -T  
/home/user3/training10x/spike-setup-main/link.ld -I  
/home/user3/training10x/spike-setup-main/env -I  
/home/user3/training10x/spike-setup-main/env2 -mabi=ilp32  
/home/user3/training10x/spike-setup-main/a_testing/ePMP_08.S -o my.elf  
-Drvtest_mtrap_routine=True -DTEST_CASE_1=True -DXLEN=32
```

For the above commands file can be found here [link](#). Now run the following command to see your dis-assembly file.

```
riscv32-unknown-elf-objdump -d my.elf
```

Run the following command to see the log how it is executed and to see contents of register and etc

```
spike --isa=RV32IMAFDC -l --log-commits my.elf 1>spike.out 2>spike.log
```

As we have exported some paths so whenever we close our terminal these paths will no longer exist hence whenever we need to use spike or the gnu toolchain we need to export these paths. One easy solution is to export these paths in dotfile `~/.bashrc` file and we no longer need to export these paths. Steps are provided as follows:

```
[sudo] vim ~/.bashrc
```

Add these line at the end of this file

```
export RISCV=/home/wajahat/RISCV64/riscv64-unknown-elf-toolchain/riscv
```

```
export PATH="${PATH}:${RISCV/bin}"
```

Save this file and close. Now spike will run from any directory you want.

**Congratulations, you are all set!**