Building the RISC-V “V” Toolchain (v0.9)

***Note: Only for Linux operating system specifically for Ubuntu/Debian-based platforms.**

As of August 2020, the "V" vector extension still has draft status with version 0.9. There is still no hardware with a RISC-V "V" CPU commercially available. Support for "V" version 0.9 for the standard development toolchain (binutils, gcc) is available, but not yet upstreamed. Meaning that one has to hunt down repositories, identify the right branches and compile those with the right flags, instead of just being able to use distro packages.

Another pitfall is that the "V" extension (similar to "F" and "D" floating point extensions) has to be enabled in the running system by setting a status register. Since the status register can only be accessed in machine-/system-mode that means that one also needs kernel support for the "V" extension.

This section details how to build the different components required for a RISC-V "V" 0.9 toolchain.

Spike:

This is an instruction set simulator (ISS) which means that it can be used to simulate RISC-V programs and the output would be exactly the same as if the program ran on an actual RISC-V core.

Open terminal in the home directory of your user account (that is: */home/your_username*). Now run these commands one by one:

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

mkdir riscvv

cd riscvv

git clone https://github.com/riscv/riscv-isa-sim.git --depth 1

cd riscv-isa-sim

mkdir build

cd build

../configure --prefix=$HOME/riscvv/riscvv09-tools/spike

make

make install
```

Note: By default Spike enables the `RV64IMAFDC` ISAs, but this default can be changed at run-time (or even configure time). So, for simulating vector instruction, we need to call spike in the terminal like this:

```
spike --isa=RV64IMAFDCV ...
spike --isa=RV64gcV     ...    # equivalent
```

GNU Toolchain:

This is the complete compiler toolchain for RISC-V based programs. You already have another GNU toolchain built on your system but that is for x86-64 based programs, given that your computer has an Intel/AMD CPU. Any such toolchain is an all-in-one package and can be used for preprocessing, compiling, assembling and linking a C program for a particular ISA.

In the same terminal, run these commands one by one:

```
cd ../..

git clone https://github.com/riscv/riscv-gnu-toolchain.git --branch rvv-0.9.x \
--single-branch --depth 1 riscv-gnu-toolchain_rvv-0.9.x

cd riscv-gnu-toolchain_rvv-0.9.x

git submodule update --init --recursive --depth 1 riscv-binutils riscv-gcc \
riscv-glibc riscv-dejagnu riscv-newlib riscv-gdb

mkdir build

cd build

../configure --prefix=$HOME/riscvv/riscvv09-tools/gnu --enable-multilib

make

make install
```

Proxy-Kernel:

The RISC-V Proxy-Kernel (pk) implements enough to get a user-space program in Spike running, i.e. including setting up some status registers in machine-mode, switching to user-mode and implementing some syscalls. That means that calling the write syscall to write to stdout then just works in Spike and the text is printed to the console.

The pk needs to be cross-compiled with the GNU Toolchain. In the same terminal, run these commands one by one:

```
cd ../..

git clone --depth 1 https://github.com/riscv/riscv-pk.git

cd riscv-pk

mkdir build

cd build

PATH=$HOME/riscvv/riscvv09-tools/gnu/bin:$PATH ../configure \
--prefix=$HOME/riscvv/riscvv09-tools/pk --host=riscv64-unknown-elf

PATH=$HOME/riscvv/riscvv09-tools/gnu/bin:$PATH make

PATH=$HOME/riscvv/riscvv09-tools/gnu/bin:$PATH make install
```

And your done! All the tools needed to work with the RISC-V vector instructions (v0.9) have been installed. Please note that we will have to set the path to the executables ourselves every time.

Getting Started:

Scenario 1: We have a C program, "main123.c" which includes the main function as well as any other function(s) that might be called in the main function. Open terminal in the location of this file and run the following commands. (Note: the first command is optional as it only outputs the assembly file for the user to inspect.)

```
$HOME/riscvv/riscvv09-tools/gnu/bin/riscv64-unknown-elf-gcc -Wall -march=rv64gcv \
-S main123.c

$HOME/riscvv/riscvv09-tools/gnu/bin/riscv64-unknown-elf-gcc -Wall -march=rv64gcv \
main123.c -o main123

$HOME/riscvv/riscvv09-tools/spike/bin/spike --isa=RV64gcV \
$HOME/riscvv/riscvv09-tools/pk/riscv64-unknown-elf/bin/pk main123
```

Scenario 2: We have a C program, "main123.c" which contains the main function and a RISC-V assembly program, "funct123.s" that contains the function(s) called by the main function. Open terminal in the location of these files and run the commands:

```
$HOME/riscvv/riscvv09-tools/gnu/bin/riscv64-unknown-elf-as -march=rv64gcv \
-o funct123.o funct123.s

$HOME/riscvv/riscvv09-tools/gnu/bin/riscv64-unknown-elf-gcc -Wall -march=rv64gcv \
main123.c -o main123 funct123.o

$HOME/riscvv/riscvv09-tools/spike/bin/spike --isa=RV64gcV \
$HOME/riscvv/riscvv09-tools/pk/riscv64-unknown-elf/bin/pk main123
```

Scenario 3: We have two RISC-V assembly programs, “main123.s” and “funct123.s”. As their names suggest, the former contains the main function and the latter contains function(s) called by the main function. Open terminal in the location of these files and run the commands:

```
$HOME/riscvv/riscvv09-tools/gnu/riscv64-unknown-elf/bin/as -march=rv64gcv -o \
funct123.o funct123.s

$HOME/riscvv/riscvv09-tools/gnu/riscv64-unknown-elf/bin/as -march=rv64gcv -o \
main123.o main123.s

$HOME/riscvv/riscvv09-tools/gnu/riscv64-unknown-elf/bin/ld main123.o funct123.o \
-o main123

$HOME/riscvv/riscvv09-tools/spike/bin/spike --isa=RV64gcV \
$HOME/riscvv/riscvv09-tools/pk/riscv64-unknown-elf/bin/pk main123
```

One more tip is to replace the last command in each scenario with the following command. This will generate a detailed log of the program run including all register/memory accesses.

```
$HOME/riscvv/riscvv09-tools/spike/bin/spike -d --isa=RV64gcV \
$HOME/riscvv/riscvv09-tools/pk/riscv64-unknown-elf/bin/pk main123 2> main123.txt
```

References:

1. Programming with RISC-V Vector Instructions by Georg Sauthoff:
   https://gms.tf/riscv-vector.html