



UNIVERSIDADE FEDERAL
DE SANTA CATARINA

RISC-V: An approach for learning the architecture

RISC-V: An approach for learning the architecture
Universidade Federal de Santa Catarina, Florianópolis - Brazil

May, 2023

RISC-V: An approach for learning the architecture
May, 2023

Project Chief:

Eduardo Augusto Bezerra <eduardo.bezerra@spacelab.ufsc.br>

Authors:

João Cláudio Elsen Barcellos <joaoclaudiobarcellos@gmail.com>

Rebecca Quintino Do Ó <rebeccaquintino@gmail.com>

Yunior Alcantra Guevara <yunior.alcantra@posgrad.ufsc.br>

Contributing Authors:

Revision Control:

Version	Author	Changes	Date
0.0	J. C. E. Barcellos, Rebecca Q. Do Ó, Yunior A. Guevara	Document creation	2023/05/10



© 2023 by UFSC. RISC-V: An approach for learning the architecture. This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/>.

List of Figures

2.1	Indication that the installation was successful.	4
2.2	Some useful commands to use.	4

List of Tables

Contents

List of Figures	v
List of Tables	vii
Nomenclature	vii
1 Introduction	1
2 Compiling and executing your first program	3
2.1 Downloading and installing the toolchain	3
2.1.1 Some important commands	3
References	5

CHAPTER 1

Introduction

[1].

CHAPTER 2

Compiling and executing you first program

2.1 Downloading and installing the toolchain

First, you should access <https://github.com/stnolting/riscv-gcc-prebuilt>, to download the most recent available toolchains (today, 21/05/2023, **rv32i-4.0.0**). You should have now access to a .tar.gz file. Then, the next step, is to create the folder that the toolchain is going to be installed. You can open a terminal and type:

```
$ sudo mkdir /opt/riscv
```

Now, you need to navigate to the folder where the .tar.gz file was downloaded, like:

```
$ cd Downloads/
```

And then you have to extract the .tar.gz file to the folder previously created:

```
$ sudo tar -xzf <toolchain_version>.tar.gz -C /opt/riscv/
```

Finally, you should add the toolchain's bin folder to your system's PATH environment variable. You can open the .bashrc file:

```
$ sudo nano .bashrc
```

And then add the following line in the end of the .bashrc file:

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

To make sure everything works fine, navigate to the folder with the application examples and execute the following command:

```
$ make check
```

If everything is working fine you should see an "OK" appearing at the end, like in the [Figure 2.1](#).

2.1.1 Some important commands

Now that the toolchain was installed, you should know some commands to generate the .hex, .bin, .vhd, as well as other types of files. As you can see in the [Figure 2.2](#), you could use **hex** to generate the .hex file, which represents the machine language. You could use, for instance, the **image** to generate the .vhd file, which is the *neor32_application_image.vhd* used to store the main program that will run in the microprocessor.

```
joaac@bobbybook: ~/Documents/neorv32/sw/example/dem...
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/11/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none:amdgc-nvptx
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 11.3.0-1ubuntu1-22.04.1' --with-bugurl=file:///usr/share/doc/gcc-11/README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++,m2 --prefix=/usr --with-gcc-major-version-only --program-suffix=-11 --program-prefix=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-clocale=gnu --enable-libstdc++-debug --enable-libstdc++-time=yes --with-default-libstdc++-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-plugin --enable-default-pie --with-system-zlib --enable-libphobos-checking=release --with-target-system-zlib=auto --enable-objc-gc=auto --enable-multiarch --disable-werror --enable-cet --with-arch=32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-offload-targets=nvptx-none=/build/gcc-11-ayXV0E/gcc-11-11.3.0/debian/tmp-nvptx/usr,amdgc-nvptx=/build/gcc-11-ayXV0E/gcc-11-11.3.0/debian/tmp-gcn/usr --without-cuda-driver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu --with-build-config=bootstrap-lto-lean --enable-link-serialization=2
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 11.3.0 (Ubuntu 11.3.0-1ubuntu1-22.04.1)

Toolchain check OK
joaac@bobbybook: ~/Documents/neorv32/sw/example/dem...$
```

Figure 2.1: Indication that the installation was successful.

```
joaac@bobbybook: ~/Documents/neorv32/sw/example/dem...$ make
<<< NEORV32 SW Application Makefile >>>
Make sure to add the bin folder of RISC-V GCC to your PATH variable.

=== Targets ===
help      - show this text
check     - check toolchain
info      - show makefile/toolchain configuration
gdb       - run GNU debugger
asm       - compile and generate <main.asm> assembly listing file for manual debugging
elf       - compile and generate <main.elf> ELF file
exe       - compile and generate <neorv32_exe.bin> executable for upload via default bootloader (binary file, with header)
bin       - compile and generate <neorv32_raw_exe.bin> RAW executable file (binary file, no header)
hex       - compile and generate <neorv32_raw_exe.hex> RAW executable file (hex char file, no header)
image     - compile and generate VHDL IMEM boot image (for application, no header) in local folder
install   - compile, generate and install VHDL IMEM boot image (for application, no header)
sim       - in-console simulation using default/simple testbench and GHDL
all       - exe + install + hex + bin + asm
elf-info  - show ELF layout info
clean     - clean up project home folder
clean_all - clean up whole project, core libraries and image generator
bl_image  - compile and generate VHDL BOOTROM boot image (for bootloader only, no header) in local folder
bootloader - compile, generate and install VHDL BOOTROM boot image (for bootloader only, no header)

=== Variables ===
USER_FLAGS - Custom toolchain flags [append only]: ""
USER_LIBS  - Custom libraries [append only]: ""
EFFORT     - Optimization level: "-O2"
MARCH      - Machine architecture: "rv32i_zicsr"
MABI       - Machine binary interface: "ilp32"
APP_INC    - C include folder(s) [append only]: "-I ."
ASM_INC    - ASM include folder(s) [append only]: "-I ."
RISC_V_PREFIX - Toolchain prefix: "riscv32-unknown-elf-"
NEORV32_HOME - NEORV32 home folder: "../.."
```

Figure 2.2: Some useful commands to use.

Bibliography

- [1] J. Bouwmeester, A. Menicucci, and E.K.A. Gill. Improving CubeSat reliability: Sub-system redundancy or improved testing? *Reliability Engineering & System Safety*, 220:108288, 2022.