Lab Notes. How to install a RISC-V toolchain and the QEMU emulator for RISC-V

Wuu Yang

Date: September 23, 2019.

Last update: 2019.10.11

The qooqoo/riscv_qemu image (which is a Docker image) contains both RISC-V gcc (a collection of RISC-V compiler/assembler/etc.) and QEMU (a RISC-V instruction-set emulator and many other functions).

我們將先安裝 Docker，再安裝 qooqoo/riscv_qemu image。

在 linux-base system 上執行 docker：

A. 安裝 docker，執行下列指令
   1. sudo apt-get purge docker lxc-docker docker-engine docker.io (若之前有使用過 docker 和上述的套件，才需執行。)
   2. sudo apt-get install curl apt-transport-https ca-certificates software-properties-common
   3. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
   4. sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
   5. sudo apt-get update
   6. sudo apt-get install docker-ce
   7. sudo systemctl status docker

   參考網址：https://tecadmin.net/install-docker-on-ubuntu/

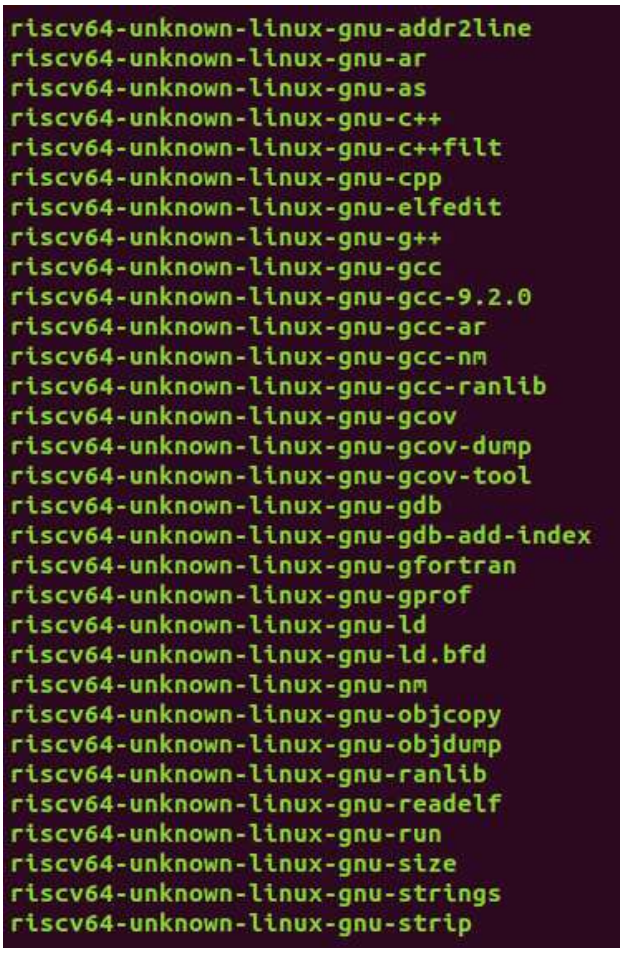
B. 取得 docker image” qooqoo/riscv_qemu” 並執行
   1. sudo docker pull qooqoo/riscv_qemu
      i. qooqoo/riscv_qemu is a [docker-image-name] . [docker-image-name]可在 https://hub.docker.com/ 上尋找所要的 docker image name
      ii. 接下來以產生 riscv 環境的 docker image—“ qooqoo/riscv_qemu ”為例
      iii. 所以我們首先執行 sudo docker pull qooqoo/riscv_qemu
      iv. You may check the downloaded image as follows:

sudo docker images

2. 接著執行 sudo docker run -it qooqoo/riscv_qemu /bin/bash
   i. 這樣就會執行該 image 並會進入建置的環境中
3. 要離開便在該環境中執行 exit 即可

參考網址：https://blog.longwin.com.tw/2017/01/docker-learn-initial-command-cheat-sheet-2017/

C. 進入該 image 產生的 container 後，便可在任意目錄下執行和編譯
   1. 執行和編譯
      a. 透過 vim 撰寫 C code
      b. 用 riscv toolchain 來編譯： riscv64-unknown-linux-gnu-gcc XXX.c
      c. 使用 qemu 來執行： qemu-riscv64 a.out
      d. 可在/opt/riscv/bin/ 中看到這個 riscv toolchain 有提供哪些工具



Some commands for controlling Docker:

2. sudo docker ps -l -q
3. sudo docker logs hash-id
4. sudo docker rm hash -id

Reference for Docker usage:

https://blog.longwin.com.tw/2017/01/docker-learn-initial-command-cheat-sheet-2017/

Example 1. Compile and run a C program.

1. vim test.c

```
int main(){
        printf("hello\n");
}
```

5. 編譯和執行

```
root@dd861fae99bb:/opt/riscv# riscv64-unknown-linux-gnu-gcc test.c
test.c: In function 'main':
test.c:2:2: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
    2 |  printf("hello\n");
      |  ^~~~~~
test.c:2:2: warning: incompatible implicit declaration of built-in function 'printf'
test.c:1:1: note: include '<stdio.h>' or provide a declaration of 'printf'
  +++ |+#include <stdio.h>
    1 | int main(){
root@dd861fae99bb:/opt/riscv# qemu-riscv64 a.out
hello
root@dd861fae99bb:/opt/riscv#
```

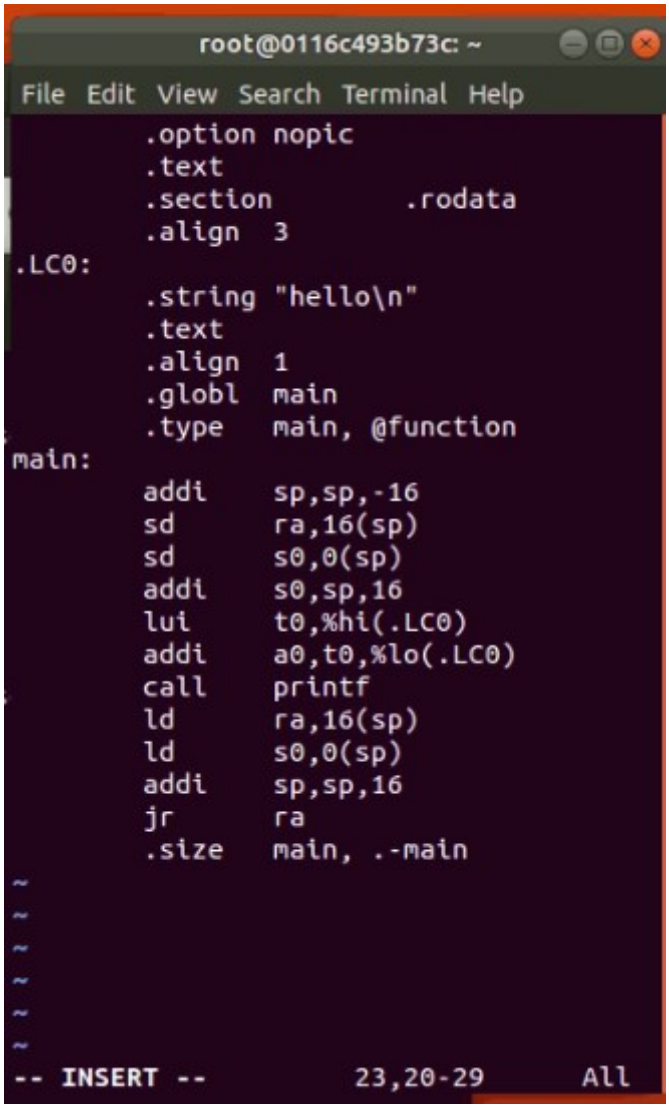
Example 2. Assemble an assembly program and run it.

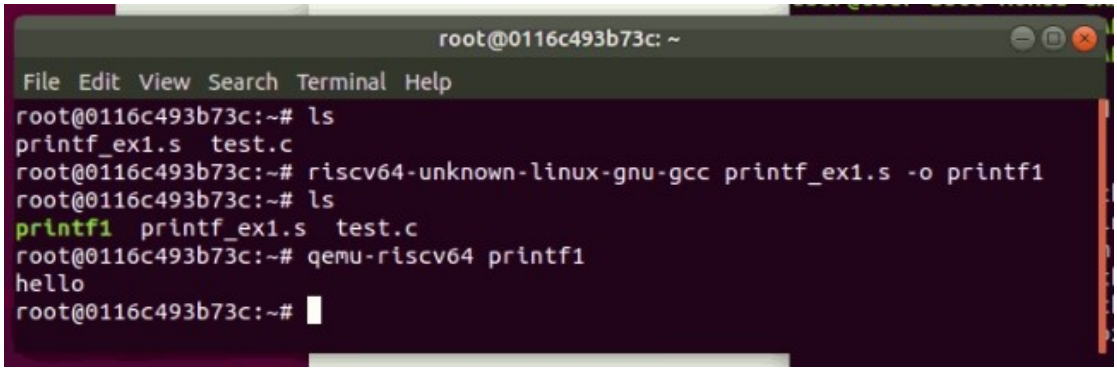
First we use vim to create the file printf_ex1.s file. Then we use the assembler to create the executable file, as follows:

riscv64-unknown-linux-gnu-gcc printf_ex1.s -o printf1

Finally we can execute the printf1 file with qemu-riscv64, as follows:

qemu-riscv64 printf1


root@0116c493b73c: ~
File Edit View Search Terminal Help
.option nopic
.text
.section .rodata
.align 3
.LC0:
.string "hello\n"
.text
.align 1
.globl main
.type main, @function
main:
addi sp,sp,-16
sd ra,16(sp)
sd s0,0(sp)
addi s0,sp,16
lui t0,%hi(.LC0)
addi a0,t0,%lo(.LC0)
call printf
ld ra,16(sp)
ld s0,0(sp)
addi sp,sp,16
jr ra
.size main, .-main
-- INSERT -- 23,20-29 All


root@0116c493b73c: ~
File Edit View Search Terminal Help
root@0116c493b73c:~# ls
printf_ex1.s test.c
root@0116c493b73c:~# riscv64-unknown-linux-gnu-gcc printf_ex1.s -o printf1
root@0116c493b73c:~# ls
printf1 printf_ex1.s test.c
root@0116c493b73c:~# qemu-riscv64 printf1
hello
root@0116c493b73c:~#