

Supplementary

Install Docker

Windows

1. <https://docs.docker.com/docker-for-windows/install/>



2. Get docker desktop for windows (stable)



3. Install Docker Desktop



4. Reboot to finish the installation
5. Install WSL (Windows Subsystem for Linux)



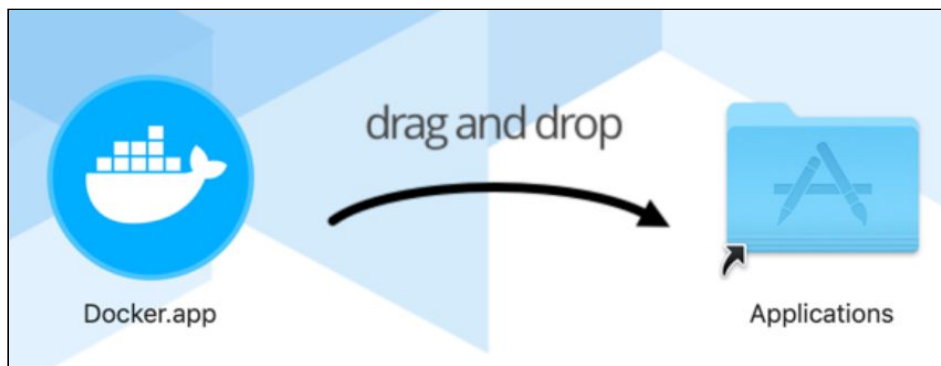
6. Open terminal and check docker

```
Usage: docker [OPTIONS] COMMAND
A self-sufficient runtime for containers

Options:
  --config string      Location of client config files (default
                        "C:\Users\User\.docker")
  -c, --context string  Name of the context to use to connect to the
                        daemon (overrides DOCKER_HOST env var and
                        default context set with "docker context use")
  -D, --debug           Enable debug mode
  -H, --host list       Daemon socket(s) to connect to
  -l, --log-level string Set the logging level
                        ("debug"|"info"|"warn"|"error"|"fatal")
                        (default "info")
  --tls                Use TLS; implied by --tlsverify
  --tlscacert string    Trust certs signed only by this CA (default
                        "C:\Users\User\.docker\ca.pem")
```

Mac

1. <https://docs.docker.com/docker-for-mac/install/>
2. Double-click Docker.dmg to open the installer, then drag the Docker icon to the Applications folder.



3. Open terminal and check docker

```
uca2020@shichengzhide-MacBook-Pro ~ % docker

Usage: docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
  --config string      Location of client config files (default
                        "/Users/uca2020/.docker")
  -D, --debug           Enable debug mode
  -H, --host list       Daemon socket(s) to connect to
  -l, --log-level string Set the logging level
```

Ubuntu

```
$ sudo apt-get install docker.io
$ docker
```

工作站上用docker

https://wslab.csie.ntu.edu.tw/docker_tutorial.html

Setup hw2 Environment using Docker

```
$ docker pull ntuca2020/hw2 (may require sudo on Linux)
$ docker run --name=test -it ntuca2020/hw2
```

```
C:\Users\User>docker pull ntuca2020/hw2
Using default tag: latest
latest: Pulling from ntuca2020/hw2
d72e567cc804: Pull complete
0f3630e5ff08: Pull complete
b6a83d81d1f4: Pull complete
7fac6f750215: Pull complete
```

```
C:\Users\User>docker run --name=test -it ntuca2020/hw2
root@6da5d12dbd44:/# cd root
root@6da5d12dbd44:~# ls
Examples Problems
root@6da5d12dbd44:~#
```

Docker basic usage

<https://docs.docker.com/engine/reference/commandline/stats/>

```
$ docker pull [image] // 抓image
$ docker images // 列出載的image
$ docker run --name=test -it [images] // 把container跑起來, 並取名
$ docker exec -it test bash // 在test跑bash起來
$ docker stop test // 把container關掉
$ docker start test // 把container打開
$ docker ps -a // 列出所有的container
$ docker rm test // 把container移除
$ docker rmi [image] // 把image移除
$ docker cp test:/root/Problems . // 把container裡面的檔案copy出來
```

Example1

<https://github.com/riscv/riscv-asm-manual/blob/master/riscv-asm.md>

<https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html>

用inline方式寫assembly, 這裡簡單介紹一下。

```
int main() {
    int a = 123, b = 456, c = 789, d;
    asm volatile(
        "add %[a_], %[a_], %[b_]\n\t" : /* a = a + b */
        "add %[d_], %[a_], %[c_]\n\t" : /* d = a + c */
        : [d_] : "r" (d)
        : [a_] : "r" (a), [b_] : "r" (b), [c_] : "r" (c)
    );
    printf("%d + %d + %d = %d\n", a, b, c, d);
}
```

: [d_] "r" (d) → d_是inline assembly裡面register的名稱, 對應到的是c output的d。

: [a_] "r" (a) → a_一樣是register的名稱, 而a對應到的是c input的a。

跑出來的結果 :

```
root@6eaa5765abec:~/Examples/Example1# make
riscv64-unknown-elf-gcc -o sum sum.c
root@6eaa5765abec:~/Examples/Example1# make test
spike pk sum
bbl loader
123 + 456 + 789 = 1368
```

Example2

把assembly寫在另一個檔案：

可能會需要看一下[calling convention](#):

```
int main() {  
    int a = 123, b = 456, c = 789;  
    printf("%d + %d + %d = %d\n", a, b, c, sum(a, b, c));  
}
```

```
.global sum  
  
sum:  
    add    a0, a0, a1      # a0 = a0 + a1  
    add    a0, a0, a2      # a0 = a0 + a2  
    ret                     # return
```

a0-a7是argument，再多就放memory傳；a0和a1可以當作return value。

Example3

這裡簡單介紹一下，在寫assembly時如何使用gdb做debugging。(參考[此連結](#))

要debug的program，用wait先擋起來：

```
volatile int wait = 1;  
  
int main() {  
    while(wait);  
  
    int a = 123, b = 456, c = 789;  
    printf("%d + %d + %d = %d\n", a, b, c, sum(a, b, c));  
    while(!wait);  
}
```

照下面的順序去執行，就可以開始使用gdb debug了。

```
# In first shell  
make  
spike --rbb-port=9824 pk ./sum  
  
# In second shell  
openocd -f spike.cfg  
  
# In third shell  
riscv64-unknown-elf-gdb ./sum  
(gdb) target remote :3333
```

(如果要copy到Problems裡面去做debug時，要稍微看一下Makefile, spike.cfg, spike.lds, sum.c這些寫法，和是怎麼link在一起的。或是參考上面的連結。)

多個shell可以用[tmux](#)，或是多個shell exec進去。

以下是實際操作一次:

```
root@Geaa5765abec:~/Examples/Example3# make
riscv64-unknown-elf-gcc -g -Og -o sum.o -c sum.c
riscv64-unknown-elf-gcc -g -Og -T spike.lds -nostartfiles -o sum sum.o sum.s
root@Geaa5765abec:~/Examples/Example3# spike --rbb-port=9824 pk
./sum
Listening for remote bitbang connection on port 9824.
bbl loader

Info : Listening on port 3333 for gdb connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
^Cshutdown command invoked
Info : remote.bitbang interface quit

root@Geaa5765abec:~/Examples/Example3# clear
root@Geaa5765abec:~/Examples/Example3# openocd -f spike.cfg
Open On-Chip Debugger 0.10.0+dev-01258-g6c1bd0508-dirty (2020-10-09-12:39)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
DEPRECATED! use 'adapter driver' not 'interface'
Info : only one transport option; autoselect 'jtag'
Info : Initializing remote bitbang driver
Info : Connecting to localhost:9824
Info : remote_bitbang driver initialized
Info : This adapter doesn't support configurable speed
Info : JTAG tap: riscv.cpu tap/device found: 0xdeadbeef (mfg: 0x777 (<unknown>), part: 0xeadb, ver: 0xd)
Warn : JTAG tap: riscv.cpu UNEXPECTED: 0xdeadbeef (mfg: 0x777 (<unknown>), part: 0xeadb, ver: 0xd)
Error: JTAG tap: riscv.cpu expected 1 of 1: 0x10e31913 (mfg: 0x489 (SiFive Inc), part: 0x0e31, ver: 0x1)
Error: Trying to use configured scan chain anyway...
Warn : Bypassing JTAG setup events due to errors
Info : datacount=2 progbufsize=2
Info : Disabling abstract command reads from CSRs.
Info : Examined RISC-V core; found 1 harts
Info : hart 0: XLEN=64, misa=0x8000000000014112d
Info : starting gdb server for riscv.cpu on 3333
Info : Listening on port 3333 for gdb connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : accepting 'gdb' connection on tcp/3333

root@Geaa5765abec:~/Examples/Example3# riscv64-unknown-elf-gdb ./sum
GNU gdb (GDB) 8.3.1
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./sum...
(gdb) target remote :3333
Remote debugging using :3333
warning: Can not parse XML target description; XML support was disabled at compile time
0x000000001001000c in main () at sum.c:8
8       while (wait);
(gdb) █
```

最右邊就停在wait那裏，有把他擋起來。等一下在gdb把wait設成0去debug。

```
Register group: general
zero      0x0      0
sp         0x7f7e9b40    0x7f7e9b40
tp         0x0      0x0
t1         0x0      0
fp         0x0      0x0
a0         0x0      0
a2         0x0      0
a4         0x0      0
ra         0x0      0x0
gp         0x0      0x0
t0         0x0      0
t2         0x0      0
s1         0x0      0
a1         0x0      0
a3         0x0      0
a5         0x0      0

sum.c
7      int main () {
8          while (wait);
9
10         int a = 123, b = 456, c = 789;
> 11         printf("%d + %d + %d = %d\n", a, b, c, sum(a, b, c));
12
13         while (!wait);
14     }

remote Remote target In: main
(gdb) lay reg
(gdb) print wait=0
$1 = 0
(gdb) n
Disabling abstract command writes to CSRs.
(gdb) █
```

目前執行到11行了，照著gdb的操作，我們就可以debug，看assembly有沒有寫錯。

gdb basic usage

```
$ riscv64-unknown-elf-gdb ./sum
(gdb) target remote :3333          // attach到remote process
(gdb) lay src                      // layout把source code叫出來
(gdb) lay reg                     // 把register叫出來
(gdb) lay asm                     // 把assembly叫出來
(gdb) p $a0                      // 把a0 register的內容print出來
(gdb) p/x $a0                    // 把a0 register的內容以hex方式print出來
(gdb) x/10 $a0                   // 在a0所指到的地方print 10個word
(gdb) n                          // 下一行
(gdb) ni                         // assembly的下一行
(gdb) s                          // step進去
(gdb) si                         // assembly的step進去
(gdb) b sum                      // 在sum這個function設breakpoint
(gdb) c                          // continue直到碰到下一個breakpoint
(gdb) quit                       // 離開
```

tmux basic usage

```
$ tmux                            // start tmux
$ ctrl + B 放開, 再按c          // 新增一個tmux session
$ ctrl + B 放開, 再按p          // 回到上一個session
$ ctrl + B 放開, 再按n          // 到下一個session
$ exit                          // 離開
$ ctrl + B 放開, 再按"          // 水平分割出一個新的視窗
$ ctrl + B 放開, 再按%          // 垂直分割出一個新的視窗
$ ctrl + B 放開, 再按方向鍵    // 在視窗間移動
```