# Compiling and running the Cortex-M Test

## Step 1: Toolchain

**To download the basic toolchains needed**

1. Run sudo apt-get update
2. sudo apt-get install -y apt-utils build-essential git curl wget jq iproute2 iputils-ping host htop python-is-python3 python3-dev python3-pip openssh-server qemu-user qemu-system-arm gcc-arm-none-eabi

## Step 2: Sample code file for Compilation

**flash.s**

```
.thumb
.thumb_func
.global _start
_start:
stacktop: .word 0x20001000
.word reset
.word hang

.thumb_func
reset:
    bl notmain
    b hang

.thumb_func
hang:   b .

.thumb_func
.globl PUT32
PUT32:
    str r1,[r0]
    bx lr
```

**notmain.c**

```
void PUT32 ( unsigned int, unsigned int );
#define UART0BASE 0x4000C000
int notmain ( void )
{
    unsigned int rx;
    for(rx=0;rx<8;rx++)
```

```
    {
        PUT32(UART0BASE+0x00,0x30+(rx&7));
    }
    return(0);
}
```

**flash.ld**

```
ENTRY(_start)

MEMORY
{
    rom : ORIGIN = 0x00000000, LENGTH = 0x1000
    ram : ORIGIN = 0x20000000, LENGTH = 0x1000
}

SECTIONS
{
    .text : { *(.text*) } > rom
    .rodata : { *(.rodata*) } > rom
    .bss : { *(.bss*) } > ram
}
```

## Step 3: Steps for compiling sample code

1. Compilation with arm-none-eabi
   Run: `arm-none-eabi-as --warn --fatal-warnings -mcpu=cortex-m3 flash.s -o flash.o`
   - -mcpu=cortex-m3: This option specifies the target processor for which the assembly code is written.
   - arm-none-eabi-as : assembler command for arm-none-eabi
   - flash.s is an assembly language source file written for the ARM Cortex-M3 processor
   - flash.o is the compiled file
   - --warn: This option tells the assembler to generate warnings about questionable constructs
   - --fatal-warnings: any warnings that are generated will be treated as errors, causing the assembly process to stop.
2. Run: `arm-none-eabi-gcc -Wall -O2 -ffreestanding -mcpu=cortex-m3 -mthumb -c notmain.c -o notmain.o`
   - -Wall: This option enables all the compiler's warning messages.
   - -O2: This is the optimization level set by GCC.

- -ffreestanding: identify the compilation as a freestanding environment where a standard library may not exist
- -mthumb: This option tells the compiler to generate code for the Thumb instruction set, a more compact version of the ARM instruction set, balancing code density and performance.
- arm-none-eabi-gcc : gcc compiler command for arm-none-eabi

3. Run: `arm-none-eabi-ld -nostdlib -nostartfiles -T flash.ld flash.o notmain.o -o notmain.elf`
    - arm-none-eabi-ld: The linker for arm-none-eabi
    - -nostdlib: No standard libraries when linking
    - -nostartfiles: No startup files when linking
    - -T flash.ld: The linker script
4. Run: `arm-none-eabi-objdump -D notmain.elf > notmain.list`
    - arm-none-eabi-objdump: This is the command that invokes the GNU Object Dump utility displaying various information about object files.
    - -D: tells objdump to disassemble all sections of the input file
    - > notmain.list: redirects the output of the objdump command to a file notmain.list instead of displaying it on the terminal.
5. Run: `arm-none-eabi-objcopy -O binary notmain.elf notmain.bin`
    - arm-none-eabi-objcopy: copy and translate object files into other formats.
    - -O binary: -O specifies the output format.

## Step 4(Create a challenge GitHub repo):

1. Create a dojo.yml file:

**Sample:**

```
id: example
//Challenge name
name: Example Dojo

description: |
  This is an [example dojo](https://github.com/pwncollege/example-dojo).
  Fork this repository, and create your own dojo!

type: example
//module name and challenge name
modules:

  - id: main
    name: main
    description: The first module in this example dojo.
```

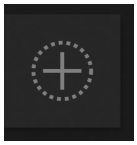
```
challenges:
  - id: main
    name: main
    description: This is main.
```

2. Create a directory folder
3. Store notmain.bin and notmain.list the directory folder
4. Push to GitHub

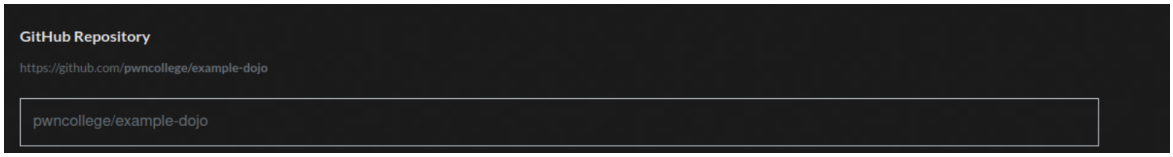
## Step 5(Adding & Running the challenges):

1. Go to [http://pwniot.cacti.academy/](http://pwniot.cacti.academy/)
2. Login as admin (username: admin password: admin)
3. Click Dojos

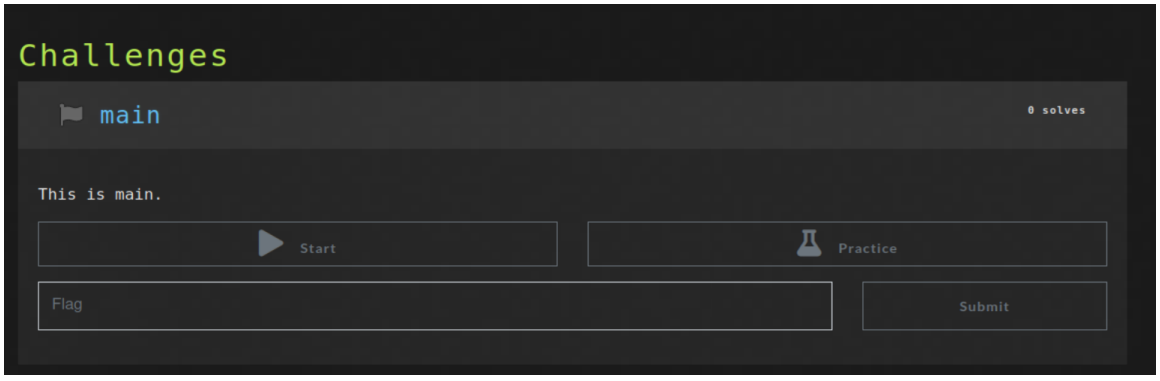

4. Click the (+) button



5. Add your github repo




6. Click the corresponding challenges and modules
7. Start the challenge

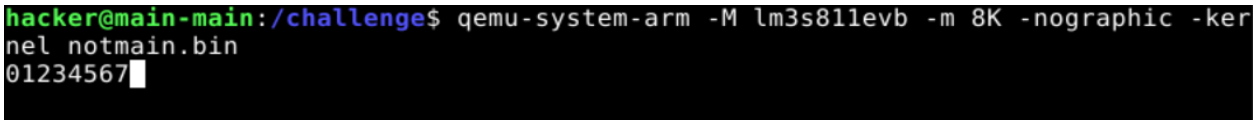



8. Go to Workspace(Vscode) or Desktop(VM). Pick the one you prefer
9. Run cd /challenge
10. Do ls to see a list of challenges
11. Run the challenge using: `qemu-system-arm -M lm3s811evb -m 8K -nographic -kernel notmain.bin`

- qemu-system-arm: This is the QEMU emulator for ARM processors
- -M lm3s811evb: -M specifies the machine type to emulate. Machine types can be found here(https://wiki.qemu.org/Documentation/Platforms/ARM)
- -m 8K: -m sets the amount of memory that the virtual machine will have. 8K means 8 kilobytes of RAM
- -nographic: This option tells QEMU to not use graphical output for the emulation.
- -kernel: The -kernel option is used to specify the binary image that will be used as the kernel.

12. Check the corresponding output

    Ex:




## Sample RISC-V Challenge Repo:

https://github.com/AnUnknownStranger/cm4