计科2302王任杰 202308010212

1.1 项目名称

环境配置

1.2 实验目的

1) 安装、配置实验环境

2) 创建一个独立于操作系统的可执行程序（裸机程序）

3) 学会利用构建系统构建工程

4) 了解并掌握 GDB 简单调试方法

5) 为后续实验做好准备

1.3 实验资源

1）实验指导书 https://os2024lab.readthedocs.io

2）Linux 64 位操作系统——Ubuntu20.04

2 实验任务

1 安装工具链

一、安装交叉编译工具链 (aarch64)

1、下载工具链，以下载工具链版本为 11.2，宿主机为 x86 64 位 Linux 机器为例

2、解压工具链

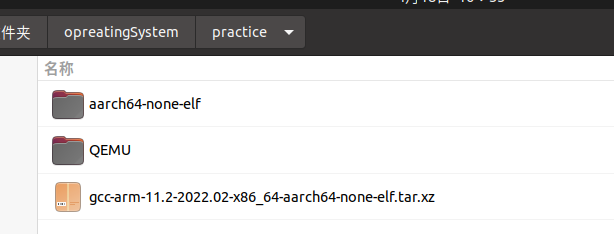
3、重命名工具链目录

$ wget

https://developer.arm.com/-/media/Files/downloads/gnu/11.2-2022.02/binrel/gcc-arm-11.2-2022.02-x86\_64-aarch64-none-elf.tar.xz

$ tar -xf gcc-arm-(按 Tab 键补全

$ mv gcc-arm-(按 Tab 键补全) aarch64-none-elf



4、将目录 /.../aarch64-none-elf/bin （可用 pwd 查看）加入到环境变量 PATH 中

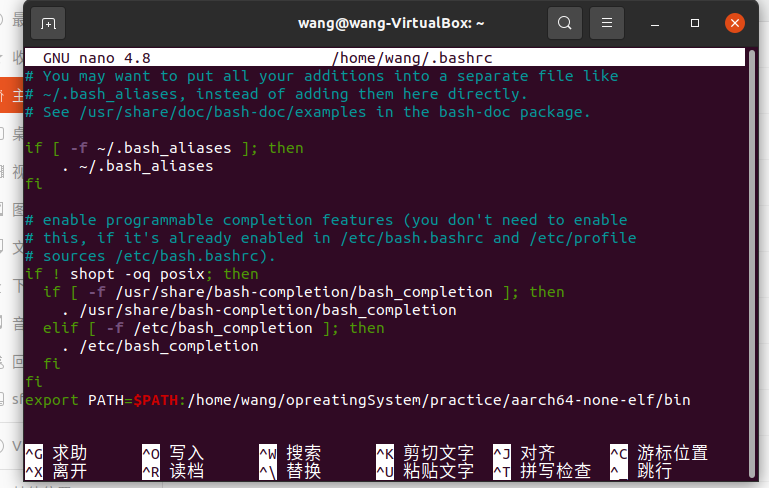
1 打开 ~/.bashrc 或 ~/.bash\_profile 文件：

nano ~/.bashrc

2 在文件末尾添加以下内容：

export PATH=$PATH:/path/to/your/aarch64-none-elf/bin

这里我的路径是/home/wang/opreatingSystem/practice/aarch64-none-elf/bin

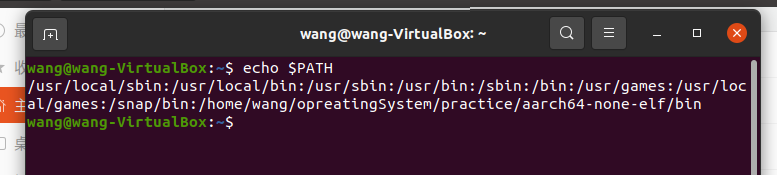


3 使修改后的文件生效：

source ~/.bashrc

4 你可以使用以下命令验证 PATH 环境变量是否已包含指定目录：

echo $PATH



二、安装 QEMU 模拟器

$ sudo apt-get update

$ sudo apt-get install qemu

$ sudo apt-get install qemu-system

三、安装 CMake

$ sudo apt-get install

2 创建裸机(Bare Metal)程序

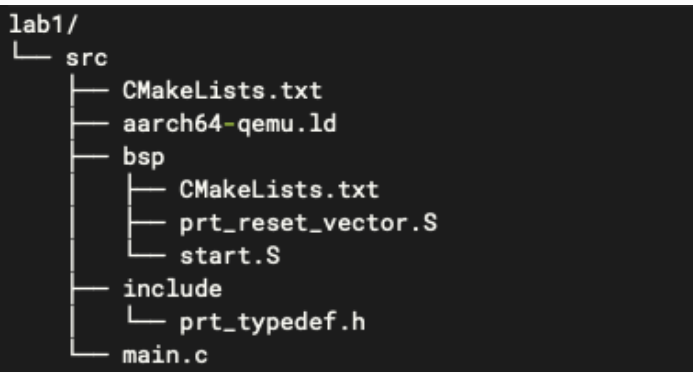
一、创建项目

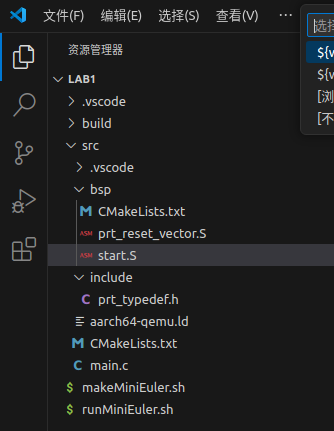
【目录结构】

src 目录：所有源代码均放在此处；

bsp 目录：存放与硬件紧密相关的代码；

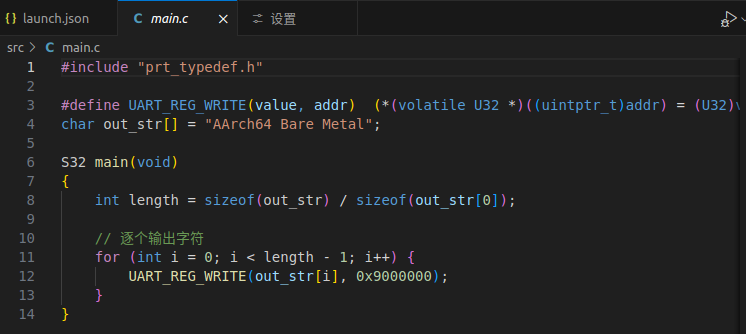
include 目录：中存放项目的大部分头文件



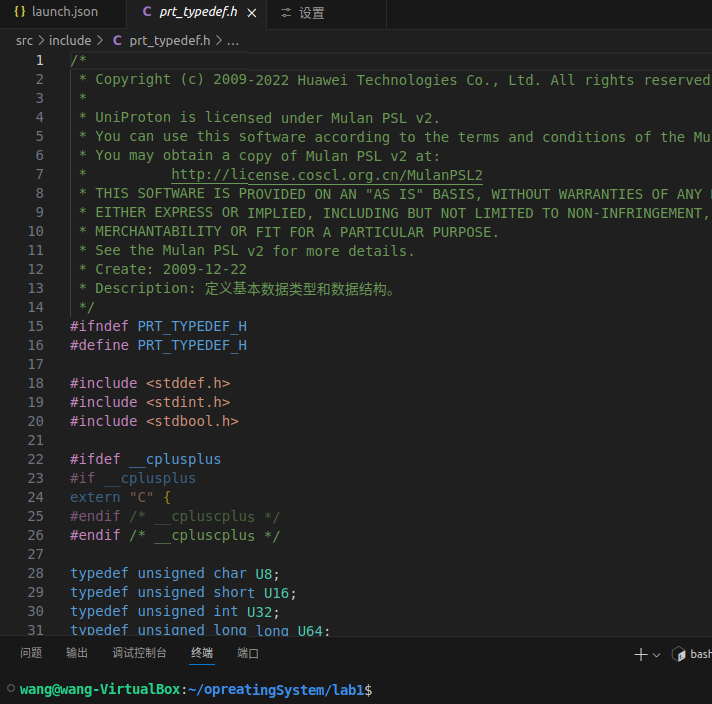


在src/下创建main.c

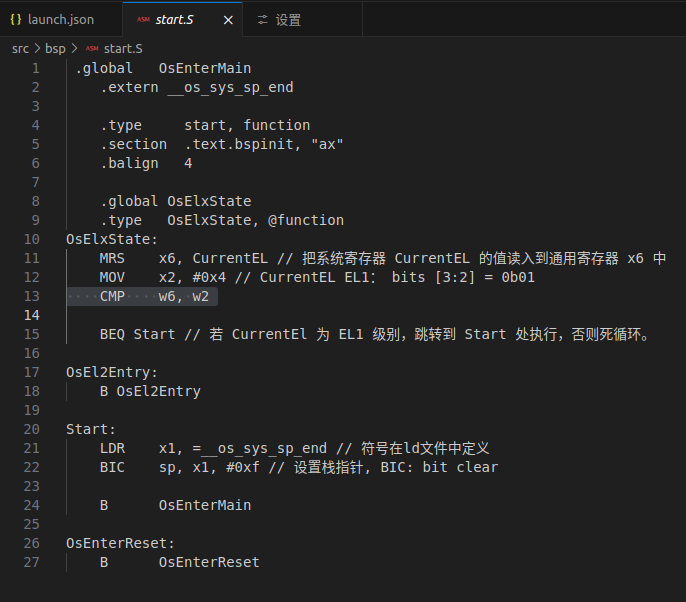
main.c源码

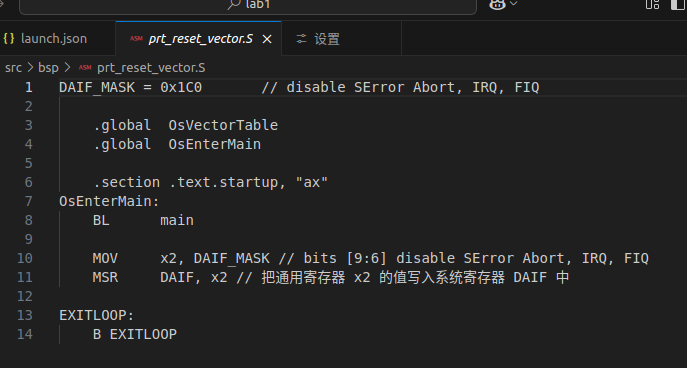


在src/include/下创建prt\_typedef.h

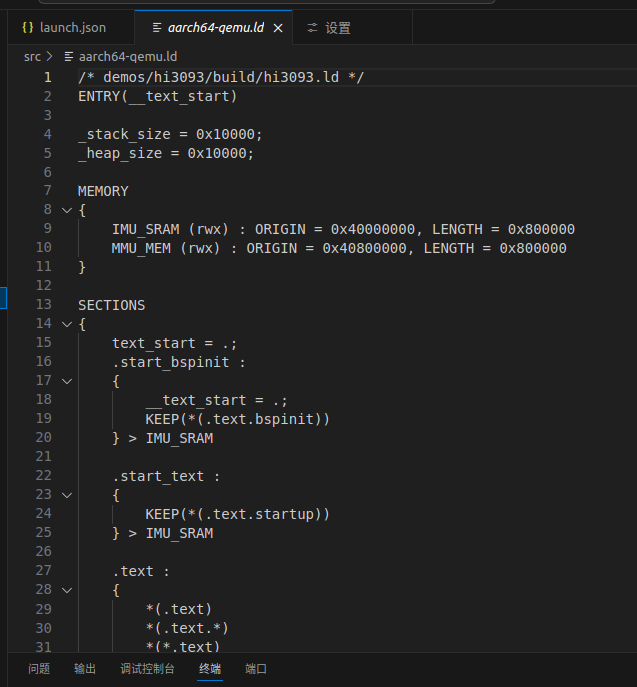


在src/bsp/下创建 start.S 和 prt\_reset\_vector.S 两个文件





在src/下创建链接文件 aarch64-qemu.ld

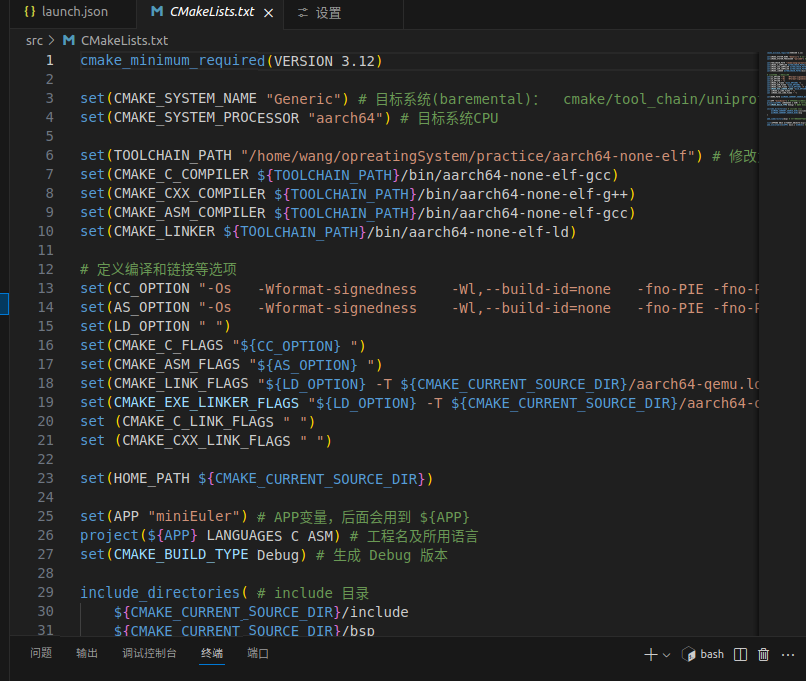


**工程构建**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab1/index.html#id8)

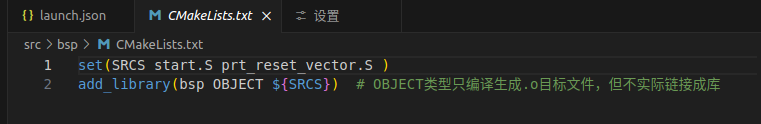
CMake 是一个跨平台的开源构建系统。CMake 通过简单的、与平台和编译器无关的配置文件来控制软件编译过程。

**CMakeLists.txt**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab1/index.html#cmakelists-txt)

src/下的CMakeLists.txt



src/bsp/下的CMakeLists.txt



src/下的 CMakeLists.txt 设置了交叉工具路径、编译和链接选项、项目名称和语言等全局环境，然后设置了需包含头文件的位置和源文件及其子目录。

编译运行

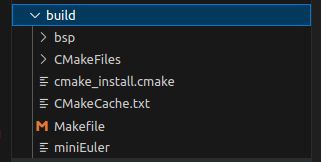
编译

首先在项目目录 lab1 下创建 build 目录用于编译生成，然后进入 build 目录执行

cmake ../src

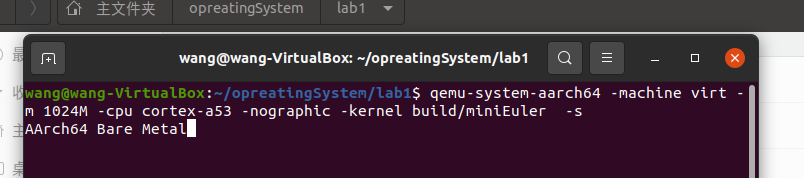
cmake --build .

运行



在项目目录 lab1 下执行

qemu-system-aarch64 -machine virt -m 1024M -cpu cortex-a53 -nographic -kernel build/miniEuler -s



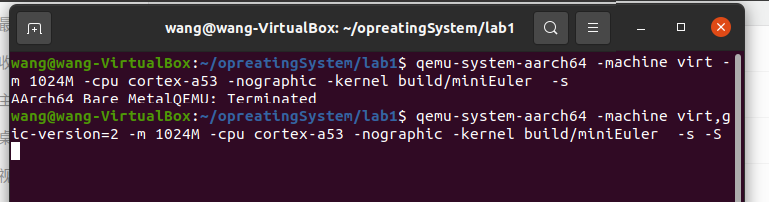
调试支持

GDB简单调试方法

编译成功后就可以运行，这需要使用前面安装的QEMU模拟器。此外，为了查找并修正bug，我们需要使用调试工具。

通过QEMU运行程序并启动调试服务器，默认端口1234

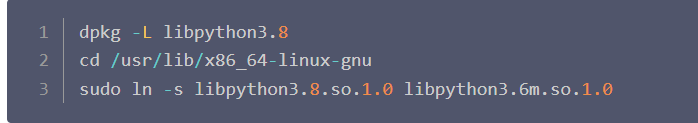
qemu-system-aarch64 -machine virt,gic-version=2 -m 1024M -cpu cortex-a53 -nographic -kernel build/miniEule



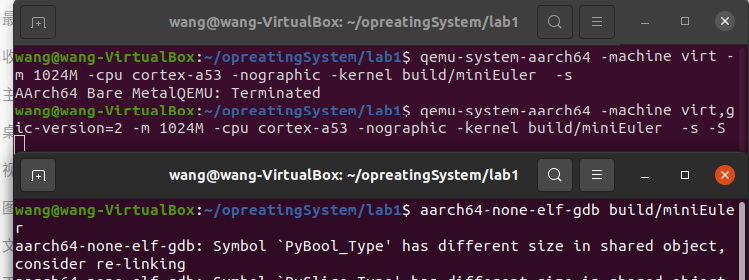
启动调试客户端

aarch64-none-elf-gdb build/miniEuler

在这里出现了一个报错，虚拟机已经安装了python3.8，报错显示需要3.6的相关文件，查资料得到下面的代码解决了问题



设置调试参数，开始调试



(gdb) target remote localhost:1234

(gdb) disassemble

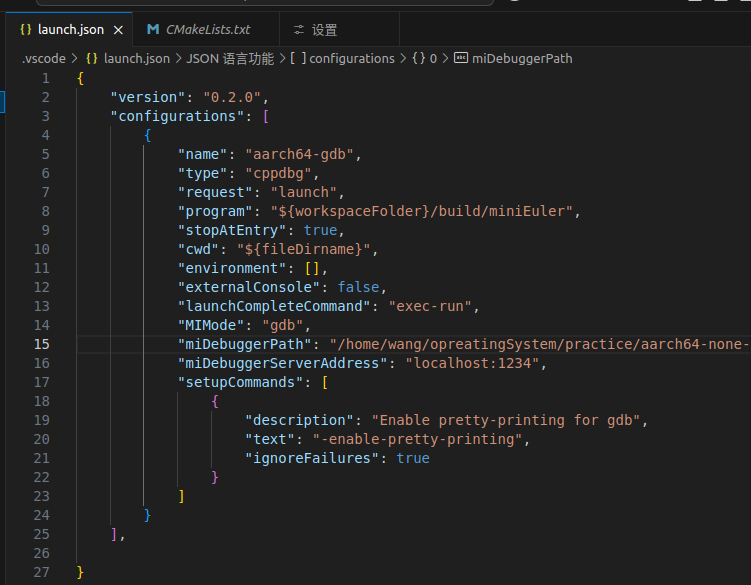
(gdb) n



将调试集成到vscode

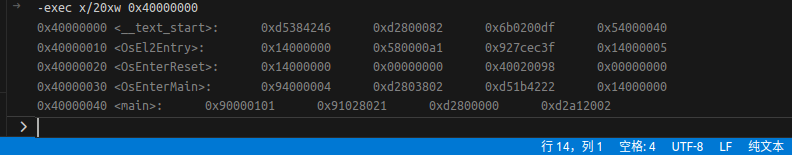
如上通过QEMU运行程序并启动调试服务器

打开 main.c 文件，点击 vscode左侧的运行和调试按钮，弹出对话框选择创建 launch.json文件，增加如下配置：



在左边面板顶部选择刚添加的 aarch64-gdb 选项，点击旁边的绿色 开始调试（F5） 按钮开始调试。

在调试时，可以在调试控制台执行gdb命令，如



查看指定地址的内存内容。在调试控制台执行 -exec x/20xw 0x40000000 即可，其中 x表示查看命令，20表示查看数量，x表示格式，可选格式包括 Format letters are o(octal), x(hex), d(decimal), u(unsigned decimal),t(binary), f(float), a(address), i(instruction), c(char) and s(string).Size letters are b(byte), h(halfword), w(word), g(giant, 8 bytes).，最后的 w表示字宽，b表示单字节，h表示双字节，w表示四字节，g表示八字节。还可以是指令：-exec x/20i 0x40000000; 字符串：-exec x/20s 0x40000000

显示所有寄存器。-exec info all-registers

查看寄存器内容。-exec p/x $pc

修改寄存器内容。-exec set $x24 = 0x5

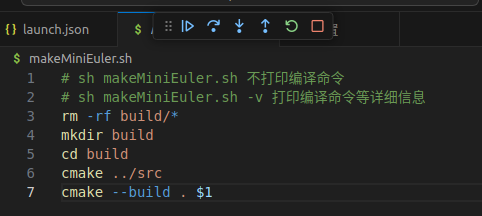
修改指定内存位置的内容。-exec set {int}0x4000000 = 0x1 或者 -exec set \*((int \*) 0x4000000) = 0x1

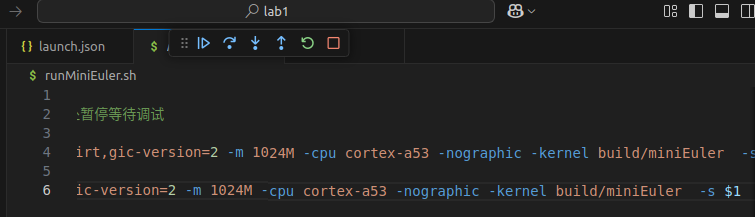
修改指定MMIO 寄存器的内容。 -exec set \*((volatile int \*) 0x08010004) = 0x1

退出调试 -exec q

自动化脚本

每次构建和运行系统都需要键入长短不一的命令。方便起见，我们可以使用 shell 脚本来简化这项工作。在项目目录 lab1 下新建 makeMiniEuler.sh 脚本来编译项目，新建 runMiniEuler.sh 脚本来运行项目。你也可以自行创建符合自己需求的脚本。

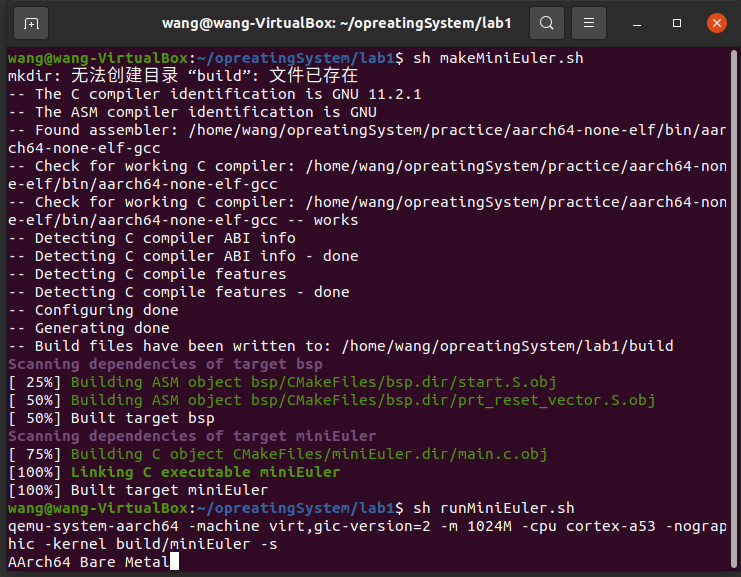




之后编译及运行程序只需要执行：

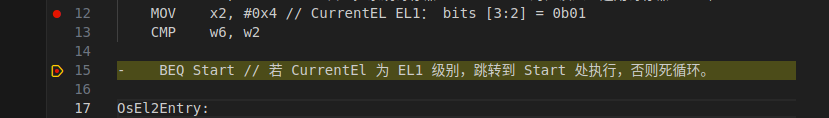
sh makeMiniEuler.sh

sh runMiniEuler.sh

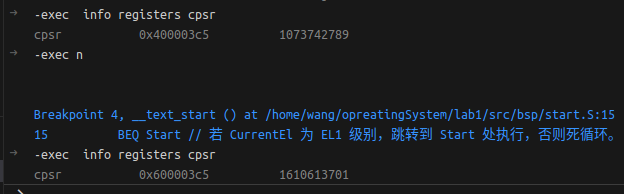


**作业1**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab1/index.html#id22)

请操作 NZCV 寄存器获取 start.S 中执行 *CMP w6, w2* 前后 NZCV 寄存器的变化。



在cmp前后打上断点



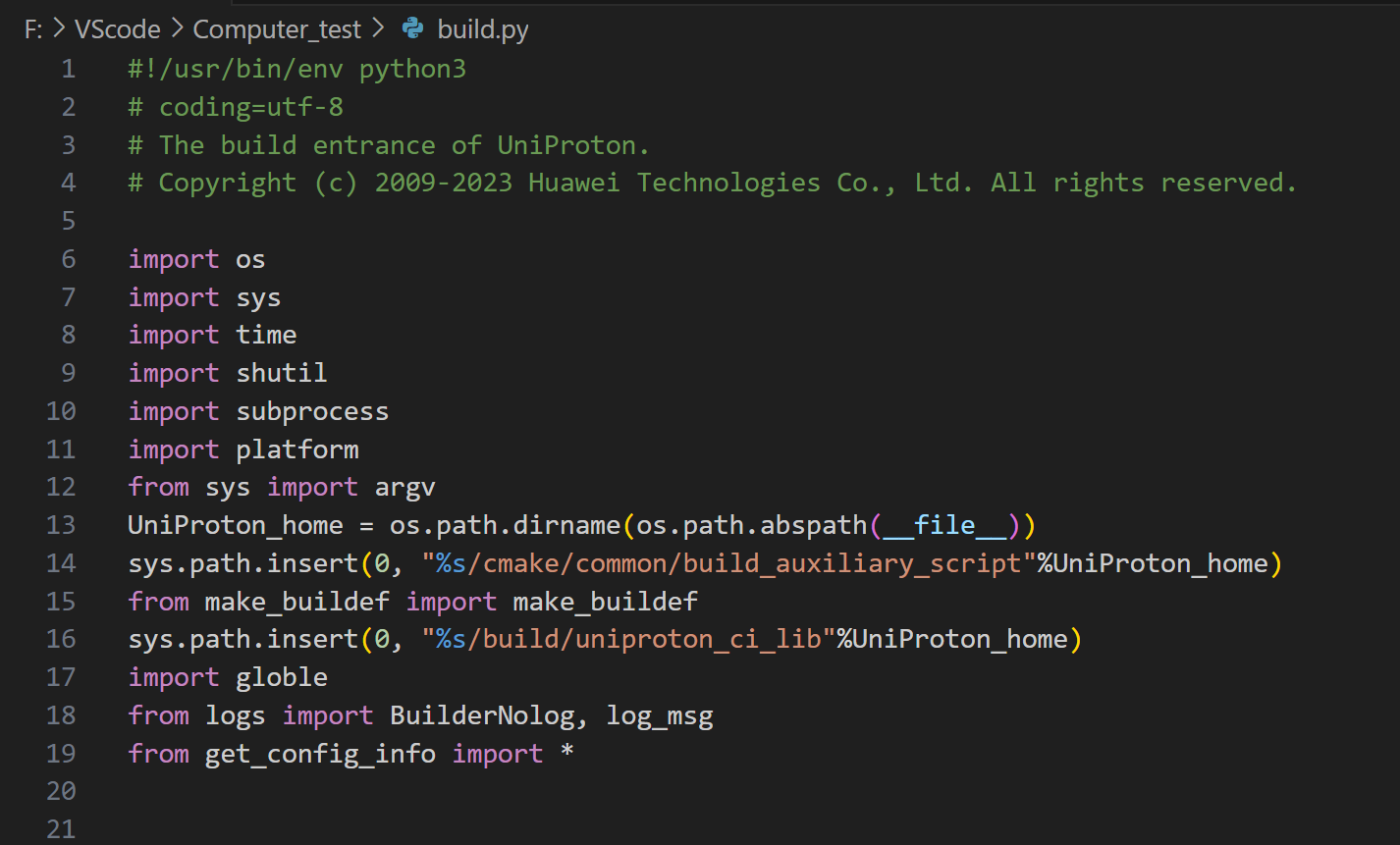
NZCV由0100到0110，C由0到1，Z为1，即运算结果为0，且无符号运算有进位

**作业2**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab1/index.html#id23)

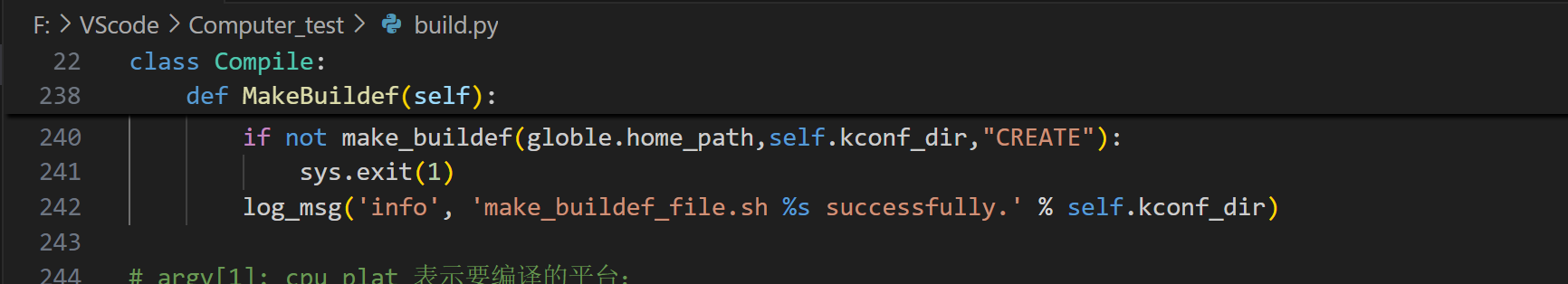
商业操作系统都有复杂的构建系统，试简要分析 UniProton 的构建系统。

**提示**

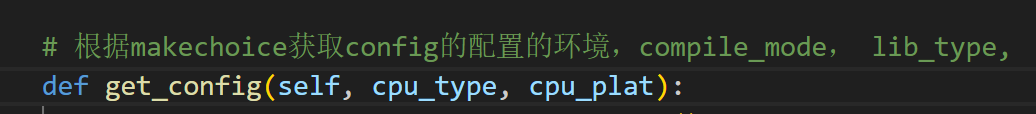
UniProton 通过在根目录下执行 python build.py m4 （m4是指目标平台，还有如hi3093等）进行构建，所以构建系统的分析可从 build.py 入手进行。

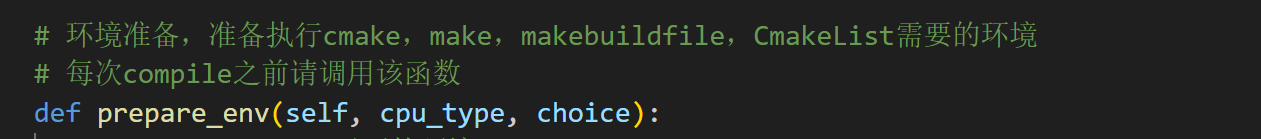


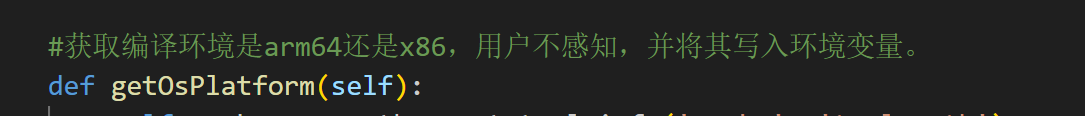
导入了一些标准库和自定义的模块，包括os、sys、time等标准库

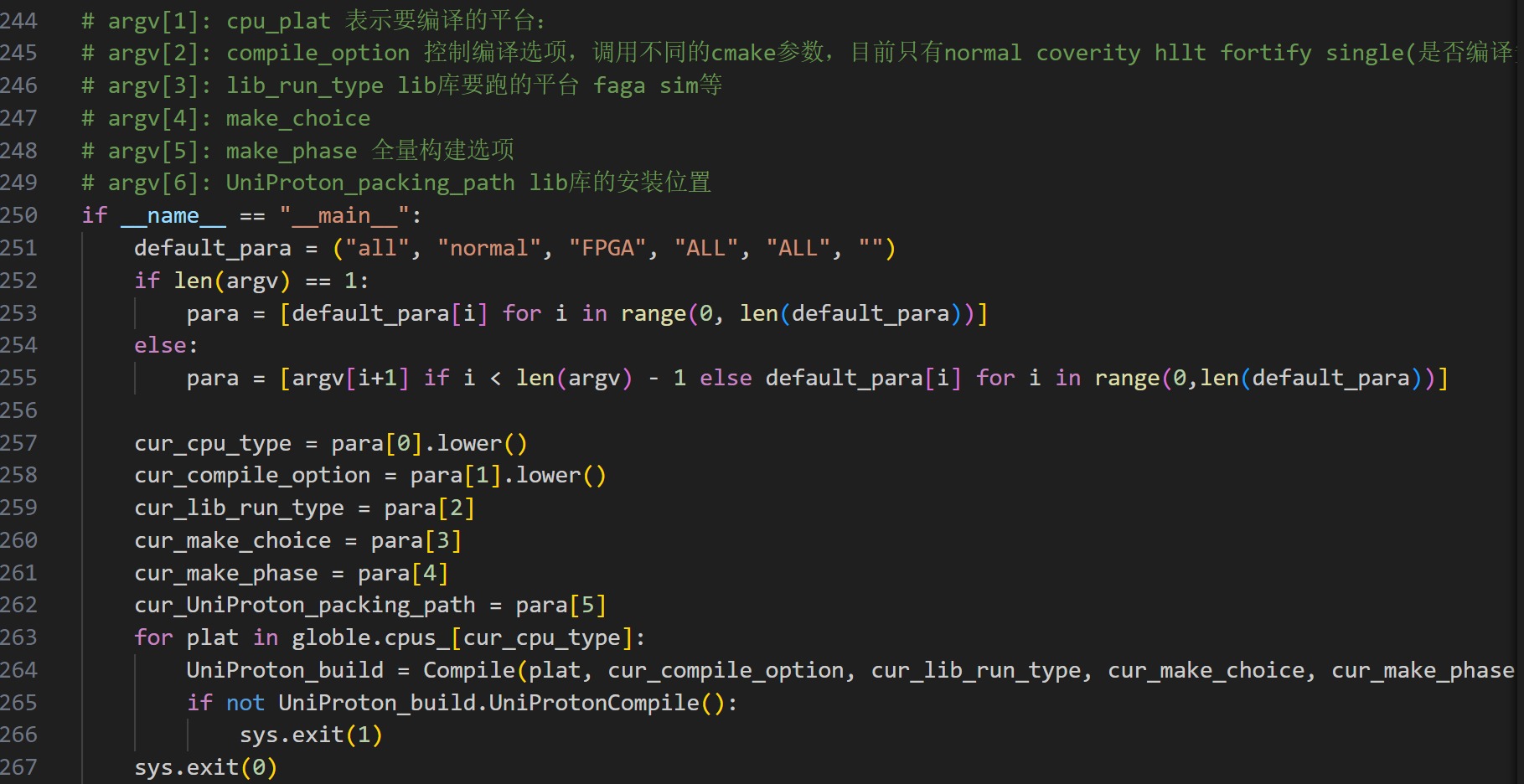


可以看到20行到243行定义了一个名为 Compile 的类，该类封装了一系列与编译相关的方法。这些方法包括获取配置信息、设置环境变量、准备环境、判断编译平台等。这些方法在后续的构建过程中会被调用，用于配置和准备构建所需的环境和参数









它根据命令行参数调用 Compile 类进行编译。根据命令行参数解析出编译的相关参数，然后根据这些参数创建 Compile 类的实例对象，并调用 UniProtonCompile 方法进行编译

**作业3**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab1/index.html#id24)

学习如何调试项目。

使用gdb进行调试

l(list) 行号/函数名 —— 显示对应的code，每次10行

r(run) —— F5【无断点直接运行、有断点从第一个断点处开始运行】

b(breakpoint) + 行号 —— 在那一行打断点

b 源文件：函数名 —— 在该函数的第一行打上断点

b 源文件：行号 —— 在该源文件中的这行加上一个断点吧

info b —— 查看断点的信息

breakpoint already hit 1 time【此断点被命中一次】

d(delete) + 当前要删除断点的编号 —— 删除一个断点【不可以d + 行号】

若当前没有跳出过gdb，则断点的编号会持续累加

d + breakpoints —— 删除所有的断点

disable b(breakpoints) —— 使所有断点无效【默认缺省】

enable b(breakpoints) —— 使所有断点有效【默认缺省】

disable b(breakpoint) + 编号 —— 使一个断点无效【禁用断点】

enable b(breakpoint) + 编号 —— 使一个断点有效【开启断点】

相当于VS中的空断点

enable breakpount —— 使一个断点有效【开启断电】

n(next) —— 逐过程【相当于F10，为了查找是哪个函数出错了】

s(step) —— 逐语句【相当于F11，】

bt —— 看到底层函数调用的过程【函数压栈】

set var —— 修改变量的值

p(print) 变量名 —— 打印变量值

display —— 跟踪查看一个变量，每次停下来都显示它的值【变量/结构体…】

undisplay + 变量名编号 —— 取消对先前设置的那些变量的跟踪

排查问题三剑客🗡

until + 行号 —— 进行指定位置跳转，执行完区间代码

finish —— 在一个函数内部，执行到当前函数返回，然后停下来等待命令

c(continue) —— 从一个断点处，直接运行至下一个断点处【VS下不断按F5】

使用vscode进行调试

查看指定地址的内存内容。在调试控制台执行 -exec x/20xw 0x40000000 即可，其中 x表示查看命令，20表示查看数量，x表示格式，可选格式包括 Format letters are o(octal), x(hex), d(decimal), u(unsigned decimal),t(binary), f(float), a(address), i(instruction), c(char) and s(string).Size letters are b(byte), h(halfword), w(word), g(giant, 8 bytes).，最后的 w表示字宽，b表示单字节，h表示双字节，w表示四字节，g表示八字节。还可以是指令：-exec x/20i 0x40000000; 字符串：-exec x/20s 0x40000000

显示所有寄存器。-exec info all-registers

查看寄存器内容。-exec p/x $pc

修改寄存器内容。-exec set $x24 = 0x5

修改指定内存位置的内容。-exec set {int}0x4000000 = 0x1 或者 -exec set \*((int \*) 0x4000000) = 0x1

修改指定MMIO 寄存器的内容。 -exec set \*((volatile int \*) 0x08010004) = 0x1

退出调试 -exec q