实验报告

组号：3

组员：谢志康、俞浩珅

贡献度与分工情况：贡献度大致各50%，我们项目是两人协力完成的，从最初的思路架构到代码编写，大约各写了一半的函数。最后整合时碰到许多问题，这个耗费了我们比较多的时间，最后也是一起在debug解决的。

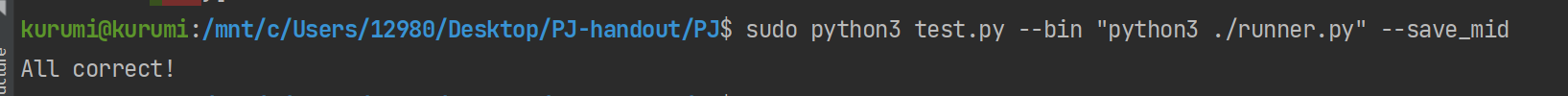
Note:

压缩文件中 CPU.py 是我们cpu的所有实现逻辑， runner.py 是最终运行程序。

\_debug后缀程序可以算作简陋的前端（显示每条指令cpu的内部状态）

使用命令 python3 test.py --bin "python3 ./runner.py" --save\_mid 在linux下运行

## 0. 功能正确性截图



## 1. CPU架构概览

参考文章——

CSAPP：第四章——处理器体系结构(上)\_csapp 第四章-CSDN博客

Y86-64的顺序实现\_取指译码执行访存写回-CSDN博客

简要分析——

在Y86-64模拟器中，CPU的主要组成部分包括：

寄存器 (Register)\*\*: 用于存储临时数据和指令执行的中间结果。寄存器可以快速地被CPU访问，以支持高效的数据处理和计算。

内存 (Memory)\*\*: 用于存储程序的指令和数据。CPU通过读取和写入内存中的数据来执行程序。

条件代码 (Condition Code)\*\*: 存储关于最近执行的运算的状态信息，例如运算结果是否为零（Zero Flag, ZF）、是否为负（Sign Flag, SF）以及是否溢出（Overflow Flag, OF）。

程序计数器 (Program Counter, PC)\*\*: 存储当前正在执行或即将执行的指令的地址。

此外还应有：

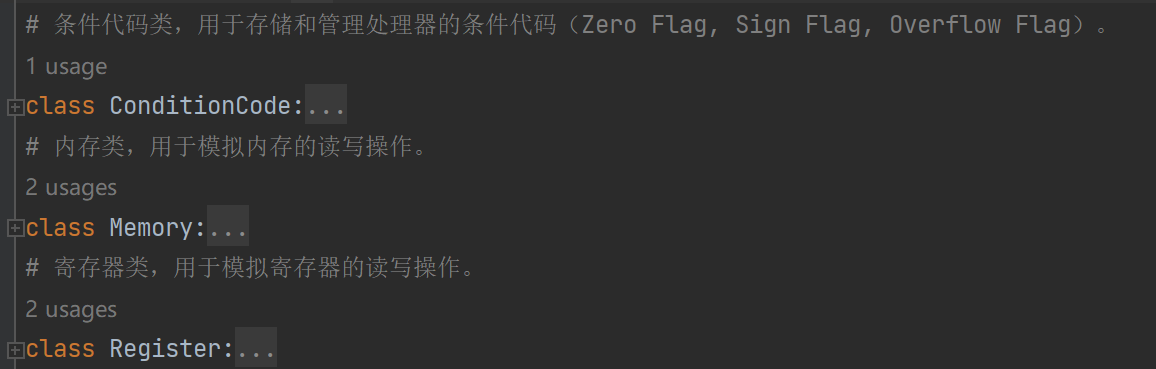
状态（STATE）

Y86-64状态码可以取以下值，1表示执行正常，2表示执行一条halt指令，3遇到非法读写，4表示遇到非法指令代码，其中2、3、4则为异常状态码。Y86-64的状态码为异常时，程序会停止（没有异常处理），一般完整的指令集定义都会有异常处理程序.

根据以上思路，我们可以定义三个类：

class Memory class Register class cc

PC 由于操作太过简单可以无需定义类，后续定义函数实现更新即可



## 2. 指令执行流程

CPU的操作可以划分为以下几个阶段：

### 取指 (Fetch)

在这一阶段，CPU从内存中读取指令。这涉及到以下函数：

`fetch(cpu)`: 这个函数从内存中读取当前程序计数器（PC）指向的指令，并对其进行初步解析，以确定后续操作。

`handle\_fetch\_errors(cpu, instr\_valid)`: 用于处理取指阶段可能出现的错误，并根据指令的有效性更新CPU状态。

### 译码 (Decode)

在译码阶段，CPU解析指令并准备必要的操作数。这包括以下函数：

`set\_src(cpu)`: 根据当前指令设置源操作数寄存器 `srcA` 和 `srcB`。

`set\_dst(cpu)`: 根据当前指令设置目标操作数寄存器 `dstE` 和 `dstM`。

`decode(cpu)`: 解析指令并从寄存器中读取必要的值以准备执行。

### 执行 (Execute)

执行阶段是指令逻辑的核心实现。这一阶段的函数包括：

`execute(cpu)`: 执行指令的逻辑，这可能包括算术运算、逻辑运算等。该函数根据指令类型处理不同的操作逻辑。

### 访存 (Memory Access)

如果指令需要，这一阶段会对内存进行读写操作。相关函数包括：

`memory(cpu)`: 根据指令需求处理内存的读写操作。例如，加载指令会从内存中读取数据，而存储指令会向内存写入数据。

`read\_memory(cpu, mem\_addr)`: 从指定的内存地址读取数据。

`write\_memory(cpu, mem\_addr, mem\_data)`: 向指定的内存地址写入数据。

### 写回 (Write Back)

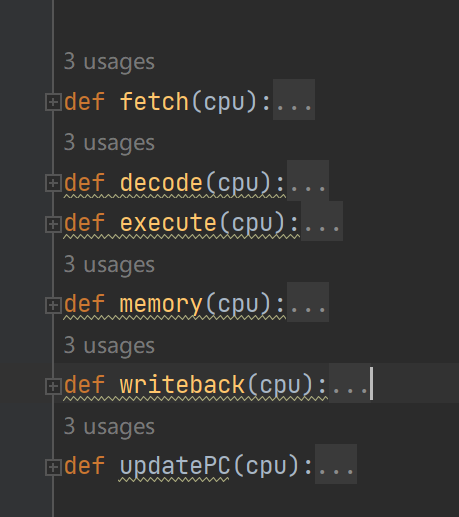
这一阶段将执行结果写回到寄存器中。相关函数：

`writeback(cpu)`: 将计算结果或内存读取的数据写回到寄存器。

### 更新程序计数器 (Update PC)

在每个指令周期的最后，更新程序计数器，以指向下一条要执行的指令。这涉及到：

`updatePC(cpu)`: 更新程序计数器（PC），以指向下一条指令或根据跳转指令更新指令地址。

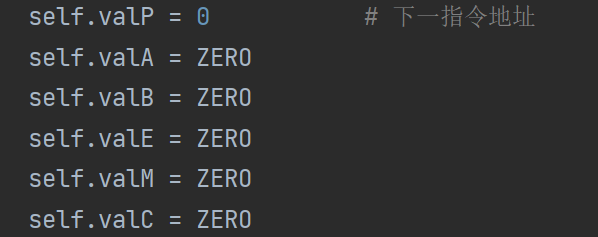


## 3. 运行

使用 `runner.py` 运行模拟器。这个脚本负责实例化CPU对象，加载指令到内存，然后按顺序执行每个指令周期。

## 4. 流水线

在每个阶段中引入流水寄存器，存储上一阶段的输出和下一阶段的输入



思路参考图——



举例说明——



我们在译码阶段就同时取出寄存器中的两个源操作数，当后续需要使用时，就可以直接获取。

## 5. 前端

没有用html或qt实现比较精美的前端，我们就在python下运行算是也写了个前端（？）

在任意输入的yo文件中，我们cpu执行时能告诉我们运行的每一条信息，并且返回执行该语句后cpu的内部所有状态。

图例如下——

