PJ: Y86-64 模拟器

字数	1252 字
阅读时间	6 分钟

项目简介

CSAPP 第四章配套课程项目。

本项目需要基于课本中 Y86-64 指令集实现一个软件层面的模拟处理器，使其能够执行 Y86-64 指令。

项目要求

- **单人或组队（不多于3人）** 完成。
- 输入输出格式正确，能够通过测试。
- 对技术栈不作限制，你可以使用 C，C++，Python 等编程语言实现模拟器。
- 对处理器架构不作限制，可以是单周期 / 多周期。
- 有兴趣的同学可以在通过测试后做一些创新，可以从课本上的其他章节寻找灵感。

> **IMPORTANT**
>
> 你需要确保提交的代码中有能够通过测试的基础部分代码。

- 按时提交实验代码与 PPT，完成期末汇报。

项目实现

处理器架构

单周期 CPU 执行一条指令一般分为以下阶段：

- **取指（Fetch）**：取指阶段从内存中读取指令的字节，使用程序计数器（PC）作为内存地址。
- **译码（Decode）**：从寄存器中读取用于运算的数据。
- **执行（Execute）**：算术/逻辑单元（ALU）执行当前指令，或执行算术运算或计算内存引用的有效地址，或增加或减少栈指针。其间可能会设置条件码。
- **访存（Memory）**：可能会将数据写入内存，或者从内存中读取数据。
- **写回（Write back）**：将最多两个数据写入寄存器。
- **更新PC（PC update）**：将PC更新为下一个指令的内存地址。

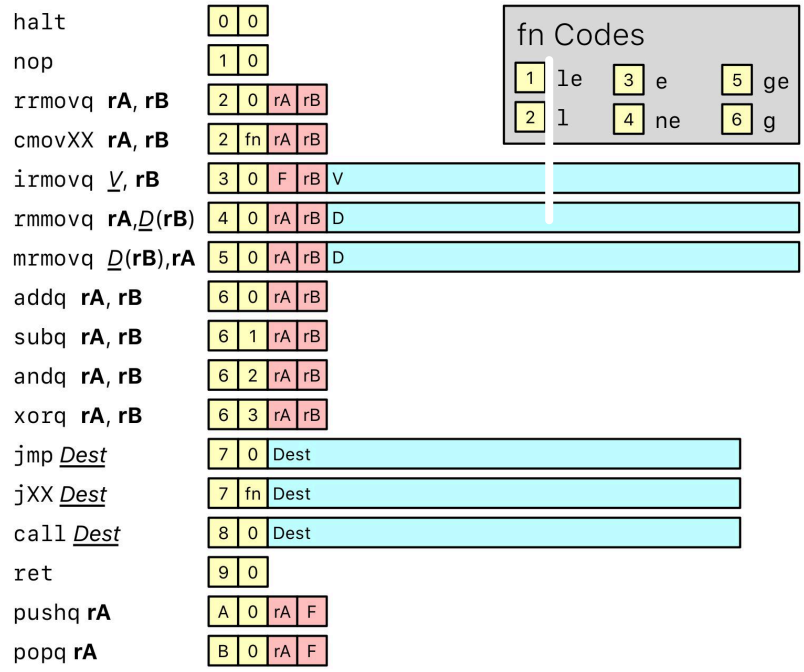
指令集

Y86-64 指令集在 CSAPP 书中第四章有详细的介绍。以下是 Y86-64 指令集的概述图。

CSCI 370: Computer Architecture

Y86-64 Reference

Instruction Format



Registers

ID	Enc	Usage	
%rdi	7	arg1	caller-saved
%rsi	6	arg2	
%rdx	2	arg3	
%rcx	1	arg4	
%r8	8	arg5	
%r9	9	arg6	
%rax	0	return	
%r10	A	general	
%r11	B	general	
%rbx	3	general	callee-saved
%r12	C	general	
%r13	D	general	
%r14	E	general	
%rsp	4	stack ptr	
%rbp	5	base ptr	
	F	no reg	

Status Conditions

AOK	1	Normal
HLT	2	Halt Encountered
ADR	3	Bad Address
INS	4	Invalid Instruction

HCL Y86-64 Hardware Registers

stage	register(s)	description
Fetch	icode,ifun	Read instruction byte
	rA,rB	Read register byte
	valC	Read constant word
	valP	Compute next PC
Decode	valA,srcA	Read operand A
	valB,srcB	Read operand B
Execute	valE	Perform ALU operation
	cnd	Set/Use Condition Code
Memory	valM	Memory Read/Write
Writeback	dstE	Write back ALU result
	dstM	Write back Mem result
PC Update	PC	Update PC

Y86-64 Data Example

```
    .align 8
Array:
    .quad 0x0000000000000001
    .quad 0x0000000000000002
    .quad 0x0000000000000003
    .quad 0x0000000000000004
```

Assembly Translation Example

```
/* find number of elements in null-terminated list */
long len(long* a) {
  long len;
  for (len = 0; a[len]; ++len)
    ;
  return len;
}
```

```
len:
    irmovq $1, %r8          # Constant 1
    irmovq $8, %r9          # Constant 8
    irmovq $0, %rax         # len = 0
    mrmovq (%rdi), %rdx     # val = *a
    and %rdx, %rdx          # Test val
    je Done                 # If zero, goto Done
Loop:
    addq %r8, %rax          # len++
    addq %r9, %rdi          # a++
    mrmovq (%rdi), %rdx     # val = *a
    andq %rdx, %rdx         # Test val
    jne Loop                # If !0, goto Loop
Done:
    ret
```

项目中测试用的指令序列在 `/test` 目录下，具体格式示例：

yaml

```
                        | # prog1: Pad with 3 nop's
0x000: 30f20a00000000000000 |   irmovq $10,%rdx
0x00a: 30f00300000000000000 |   irmovq  $3,%rax
0x014: 10                   |   nop
```

```
0x015: 10                  |    nop
0x016: 10                  |    nop
0x017: 6020                |    addq %rdx,%rax
0x019: 00                  |    halt
```

每个测试 `/test/{test_name}.yo` 对应的期望输出在 `/answer/{test_name}.json` ，具体格式示例：

json


```
[
    {
        "CC": {
            "OF": 0,
            "SF": 0,
            "ZF": 1
        },
        "MEM": {
            "0": 717360,
            "16": 6922050288173973504,
            "24": 32,
            "8": 16914579456
        },
        "PC": 10,
        "REG": {
            "r10": 0,
            "r11": 0,
            "r12": 0,
            "r13": 0,
            "r14": 0,
            "r8": 0,
            "r9": 0,
            "rax": 0,
            "rbp": 0,
            "rbx": 0,
            "rcx": 0,
            "rdi": 0,
            "rdx": 10,
            "rsi": 0,
            "rsp": 0
        },
        "STAT": 1
```

```
    }, // 第一条指令执行完之后的状态码、内存、寄存器、程序计数器状态
]
```

TIP

1. 你的输出格式应当严格遵循 `/answer` 下的示例。
   - 要求在每条指令执行完毕后输出：完整的寄存器信息和内存非零值(八字节对齐，按小端法解释为十进制有符号整数)。内存非零值指{(内存地址,内存值)|内存值 ≠ 0}，即所有非零内存值的内存地址-值键值对。
   - 所有输出(含内存地址、寄存器值、内存值)均以十进制输出。
   - 不用关心每次 log 内 key-value 的排列顺序，但要确保列表内 log 的顺序与程序执行顺序一致。
2. 如果你使用 C / C++ 编程，可以参考第三方库，例如 nlohmann/json。
3. 你最终实现的程序应当以 **标准输入流（stdin）** 为输入，**标准输出流（stdout）** 为输出。可以使用 **重定向** 实现读写文件：

```bash
# 读取 test/prog1.yo，写入 answer/prog1.json
./cpu < test/prog1.yo > answer/prog1.json
python cpu.py < test/prog1.yo > answer/prog1.json
```

项目开发与提交

环境配置

- 本次项目不同于以往实验，可以在非 Linux 环境下进行开发。
- 你需要根据你的技术栈安装必要的环境，请自行上网搜索。

代码下载

仓库地址为 ICS-25Fall-FDU/PJ-Y86-64-Simulator，在你的开发环境终端执行以下命令以获取项目初始代码。

```bash
git clone git@github.com:ICS-25Fall-FDU/PJ-Y86-64-Simulator.git
```

> **INFO**
>
> - `cpu.h` 、 `cpu.c` 、 `cpu.py` 、 `Makefile` 文件仅供参考，你可以将他们删去。

测试

参考项目仓库中的 `README.md` 文件。

提交

将你的代码打包为 `.zip` 或 `.tar.gz` 文件，命名为 `code-姓名1-姓名2-姓名3.zip` 或 `code-姓名1-姓名2-姓名3.tar.gz` ，PPT 命名为 `pre-姓名1-姓名2-姓名3.pptx` 提交到 elearning。**小组内每位同学都需要提交。**

评分标准

项目得分计算公式为：

$$总分 = 得分上限 * (\frac{5}{6}基础功能得分 + \frac{1}{6}汇报得分)$$

> **NOTE**
>
> **选课代码为 CS10005.02 的同学** 请阅读以下内容：
>
> 1. 多人组队的得分上限为总分的 **90%**。
> 2. 创新设计可能会给你的汇报带来更高的分数，但不鼓励在实现前端上花费太多时间。

fduTristin