Project 实验报告

22307110054 徐厚泽 2023/12/17

1 项目简介

跟据CSAPP中对Y86-64指令集的描述和讲解,以任意技术栈自行实现一个CPU模拟器。

2 项目内容

- CPU架构
 - 寄存器REG[15]
 - 程序计数器PC
 - 条件码CC: ZF SF OF
 - 程序状态 state
- 内存DMEM 设置为一个长度为0x800的unsigned char数组
- I/O实现
 - input 打开指定的文件infile,使用getline函数逐行处理(后续因重定向,这里只需cin即可)
 - create_output 为实现.json格式文件的输出,引入库<json.hpp>¹ 定义一个json对象data,包含了CPU与内存的信息。遍历内存,每8字节为一单元,将非零值加入json对象中。

创建一个全局json变量DATA每次create后将data push_back到DATA中

- output 输出DATA

¹来自https://github.com/nlohmann/json

2 项目内容 2

- save_instruction 将input和中每行的信息转换为指令存放于内存
 - * 读取地址addr 每行以0xXXX开头,从第三位开始读至':'结束,依次将字符'0'-'9'、'a'-'f'转换为 数字得到地址.
 - * 读取并存储指令 指令编码为十六进制,两位为一字节,按小端法存储于内存。

• 指令集的实现

- 内存处理函数

- * fetch_from_memory 参数为地址addr和偏移量offset,从地址为addr+offset处按小端法往后取8个字节。
- * save_to_memory 参数为地址addr和偏移量offset,从地址为addr + offset处按小端法往后存8个字节。

- 指令函数

- * halt 将CPU的state设为2
- * nop 空
- * 传从指令
 - rrmovq 参数需引用,将前者赋给后者
 - · irmovq 类似
 - · rmmovq 调用写内存函数save_to_memory
 - · mrmovg 调用读内存函数fetch_from_memory
 - · cmovle,cmovl,cmove,cmovne,cmovge,cmovg 为条件传送指令,根据CPU状态 决定是否传送

* 整数操作指令

- · addq 修改第二个参数为两者的和并检验0、溢出、正负,修改条件码
- · subq 修改第二个参数为两者的差条件码类似
- · andq 修改第二个参数为两者的与条件码只需判断正负和0
- · xorq 修改第二个参数为两者的异或条件码只需判断正负和0
- * 跳转指令直接跳转将PC设为传入的参数,条件跳转根据条件码判断是否修改PC
- * 栈操作
 - pushq rsp减8,并调用写内存函数将值写入rsp指向的内存位置
 - · popq rsp加8,并调用读内存函数将rsp-8出的内存读取到参数中

3 运行结果 3

- * call函数将PC+9(即call指令的下一条指令地址)压栈,PC设为call的地址
- * ret函数调用pop函数,并将PC设为pop出的值
- * 异常处理函数halt, ADR, INS分别将state设为2, 3, 4

- 功能封装实现

- * fetch函数根据当前PC值返回内存地址
- * decode_and_execute函数从内存中读取指令,进入选择结构,根据第一个字节决定往后读的字节数、将要调用的函数、PC的改变

3 运行结果

PS E:\study\ICS\PJ> python test.py --bin ./cpu.exe All correct!

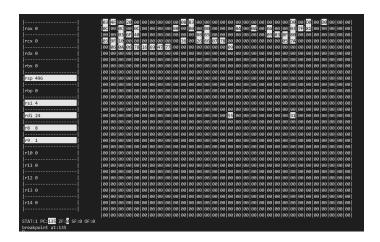
4 简易前端与项目亮点

4.1 亮点1

本人为了更好的展示cpu在运行中的状态,以及实现指令调试,引入了gdb功能,使用此功能在文件gdb.cpp中

主要功能为:

- 打印实时cpu状态,在项目要求以外,我在每个寄存器值、PC值、状态值、CC值和内存字 节改变时会显示高亮,并且将每个内存块都打印出来
- 加入断点指令指令格式为 b < address >
- 单行执行指令指令格式为s
- 运行指令指令格式为r 它的功能是执行指令直到断点处或程序结束 它的效果如下:



4.2 亮点2

引入CacheLab的内容,增设头文件cache.h,针对大小为1024的内存,引入一个8*4*8的缓存。

由于技术限制,这个缓存只对偏移量为8的整数倍的rmmovq和mrmovq指令使用 我在这里重载函数decode_and_execute,以及rmmovq,mrmovq,在cache.cpp文件中运行带 有cache的CPU,并用test.py测试,结果如下:

```
PS E:\study\ICS\PJ> python test.py --bin ./cache.exe All correct!
```

同时我在gdb.cpp文件里加入另一种带有cache的调试模式,能够实时输出hit,miss,eviction数,结果如下:

