

Y86 流水线模拟器的实现

计算机原理项目报告

游沛杰 13307130325

June 9, 2015

Contents

1	背景介绍	2
2	工作原理	2
2.1	指令集	2
2.2	PIPELINE	3
3	我的项目概况	3
4	内核具体实现	4
4.1	中间变量的实现	4
5	UI 介绍	5
6	UI 实现	5
7	其他功能	5
8	遇到的问题	5
9	项目总结	5
10	Thank You	5

1 背景介绍

在本次项目实验中，我们需要实现一个 Y86 模拟器，能够正确的流水线化模拟执行课本 (CS:APP[1]) 第 4 章介绍的 Y86 指令，并将执行过程和程序运行结果通过图形界面的方式可视化。

2 工作原理

这里分指令集，流水线阶段，流水线异常处理等部分来介绍

2.1 指令集

实现了书本上介绍的全部 Y86 指令，如下

Byte	0	1	2	3	4	5
halt	1	0				
nop	0	0				
rrmovl rA, rB	2	0	rA	rB		
irmovl V, rB	3	0	F	rB	V	
rrmovl rA, D(rB)	4	0	rA	rB	D	
rrmovl D(rB), rA	5	0	rA	rB	D	
OpI rA, rB	6	fn	rA	rB		
jXX Dest	7	fn	Dest			
cmovXX rA, rB	2	fn	rA	rB		
call Dest	8	0	Dest			
ret	9	0				
pushl rA	A	0	rA	F		
popl rA	B	0	rA	F		

其中，地址的形式为 D(rB)，而指令跳转使用绝对地址的方式（比如 jmp 0x123，那么下一条要执行的指令就是 0x123 开始的指令，而不是 pc+0x123 或者 pc+0x123+0x2）。

另外需要说明的是，书本上的 halt 和 nop 对应的机器码不明确（见第二版英文书 P338 和 P384，前后矛盾）。所以在这里，我把 10 看成是 halt 指令对应机器码，00 看成是 nop 指令。

2.2 PIPELINE

我实现的流水线分成五个阶段 (Fetch, Decode, Execute, Memory, Write back)

1. Fetch 从指令内存中取出指令以及相关信息 (rA、rB 编号, valC)
2. Decode 读取相关的寄存器的值 (rA, rB, RESP 的值)
3. Execute 进行算术/逻辑运算
4. Memory 读写内存
5. Write back 将需要更新的寄存器值放回 register file 中

使用 forward+stall/bubble 来处理异常情况。

- forward 将前面阶段中已经计算好的值直接传输到 decode 阶段
- stall 不执行更新，但是保留原来流水线寄存器中的值 (延迟到以后再更新)
- bubble 把 pipeline register 中保存的值重置置为默认值 (舍弃当前已在进行的指令操作)

3 我的项目概况

Table 1: 开发环境

主要使用语言	内核	Python 2.7.9
	GUI	PyQt 4.11.3
开发平台	IDE	PyCharm
	操作系统	Windows 8.1
辅助工具	GUI 界面绘制	Qt Creator

全部代码均作者本人独立完成，并没有抄袭和复制粘贴的成分。

项目的设计和准备阶段从 5 月 16 日开始，经历两个星期（其中有另外 2 个项目需要完成）。从 6 月 1 日左右开始投入实际开发。其中核心部分历时 4 天编写并完成调试，UI 部分做了 4-5 天。

在这个项目之前我们还完成了数据库的课程项目 (HTML+PHP 实现)，所以作者这次决定不再写 web 的 UI，试图使用不会的 Qt 来实现一个桌面端应用程序。虽然界面不一定有 HTML+CSS+JavaScript 实现的好看，但是也是一种挑战。

4 内核具体实现

4.1 中间变量的实现

这里的中间变量指的 stage output，通常以小写字母 `fmemw` 开头 (比如 `e_valE`)。

对于这些结果，在硬件上相当于一个组合电路，那么我们就用一个函数来实现，其中函数的输入就相当于硬件层面上的接线，组合电路就在函数内部实现其功能，比如下面例子：

```
def f_stat(f_icode, imem_error):  
    #     DONE  
    #     有优先级?  
    if imem_error: return SADR  
    if not instr_valid(f_icode): return SINS  
    if f_icode == IHALT: return SHLT  
    return SAOK
```

其中部分 stage output 只用了一次，我们在需要的时候再计算这个结果，以保证程序整体运行速度，而对于在不止一个地方用到的值，我们采用中间变量来保存这些值。

我们可以发现，Python 的语法与课本上的 HCL 有很多相似之处，比如有 `in` 的用法，Python 的 `list` 理解起来也比较简单，如下语句：

```
def d_srcB(D_icode, D_rB):  
    #     DONE  
    if D_icode in [IOPL, IRMMOVL, IMRMOVL]: return D_rB  
    if D_icode in [IPUSHL, IPOPL, ICALL, IRET]: return RESP;  
    return RNONE;
```

我们发现 Python 在这里看起来就和真的硬件描述语言一样，同时十分接近自然语言，给人带来愉悦感！这也是类似于 C++ 之类的语言所没有的。

5 UI 介绍

6 UI 实现

7 其他功能

8 遇到的问题

9 项目总结

10 Thank You

View this project on GitHub[2]

Report is written by L^AT_EX

References

[1] <http://www.csapp.cs.cmu.edu/>

[2] <https://github.com/kjkszpj/AE86>