archlab Part B文档

软件52班 张迁瑞 2015013226

Seq程序描述

iaddl

对iaddl指令的描述如下：

iaddl **V rb**

fetch:

```
icode:ifun <- M1[PC]
rA:rB <- M1[PC+ 1]
valC <- M4[PC + 2]
valP <- PC + 6
```

decode:

```
valB <- R[rb]
```

execute:

```
valE <- valB + valC
set CC
```

memory: write back:

```
R[rb] <- valE
```

PC update:

```
PC <- valP
```

leave

对leave的描述如下:

leave

fetch:

```
icode:ifun <- M1[PC]
valP <- PC + 1
```

decode:

```
# 其实这里的valA最后并没有用上，只是仿照popl的写法加入了valA的设置
valA <- R[%esp]
valB <- R[%ebp]
```

execute:

```
valE <- valB + 4
```

memory:

```
valM <- M4[valB]
```

write back:

```
R[%esp] <- valE
R[%ebp] <- valM
```

PC update:

```
PC <- valP
```

Pipeline

iaddl

对pipeline的iaddl指令的描述如下：

iaddl **V rb**

PC select:

在PC选择方面，两条指令相同。当一条预测错误的分支进入访存阶段时，会从流水线寄存器M中读出该指令valP的值。当ret指令进入写回阶段时，会从流水线寄存器W中读出它的返回地址。其他情况会使用流水线寄存器F中的PC预测值valP。

```
f_predPC = f_valP
f_pc = f_predPC
```

fetch:

```
icode:ifun <- M1[PC]
rA:rB <- M1[PC+ 1]
f_valC <- M4[PC + 2]
f_valP <- PC + 6
```

decode:

```
d_valB <- R[rb]
```

execute:

```
e_valE <- E_valB + E_valC
set CC
```

memory: write back:

```
R[rb] <- W_valE
```

leave

对pipeline的leave指令的描述如下(PC_select阶段略）：

由于在pipeline中没有M_valB,所以相比leave的seq版本，我修改了两个寄存器的顺序。

fetch:

```
icode:ifun <- M1[PC]
f_valP <- PC + 1
```

decode:

```
d_valA <- R[%ebp]
d_valB <- R[%esp]
```

execute:

```
e_valE <- E_valA + 4
```

memory:

```
m_valM <- M4[M_valA]
```

write back:

```
R[%esp] <- W_valE
R[%ebp] <- W_valM
```

关于数据冒险问题

由于leave后面的指令必然是ret，并不存在Load/Use Data Hazards,所以出于效率起见，其实不用考虑这个问题。

但是毕竟这个实验不需要考虑运行时间之类的东西，我还是给了这个指令和POPL相同的待遇，这样运行时间加长了，不过处理器的稳定性也增强了。

收获与感受

没有别的感想，做完这个实验，只感觉对CSAPP的作者五体投地。

如果其他的课程也有这样既有乐趣又能学到很多知识的大作业就好了。

以及最后感谢杨老师和助教一学期的付出！