

# [BUAA-CO-Lab] P6 流水线 CPU-lite2

Posted by roife on Wed, Dec 2, 2020

## 上机总结

- 第一题: bgezalr

$$\begin{aligned} condition &\leftarrow GPR[rs] \geq 0 \\ GPR[rd] &\leftarrow PC + 8 \\ \text{if } condition &\text{ then} \\ PC &\leftarrow GPR[rt] \end{aligned}$$

甚至比 blezalr 还简单一点。

- 第二题: msub

$$\begin{aligned} temp &\leftarrow (HI \parallel LO) - (GPR[rs] * GPR[rt]) \\ HI &\leftarrow temp_{63..32} \\ LO &\leftarrow temp_{31..0} \end{aligned}$$

标准指令集里面有这个指令。

- 第三题: lhs

$$\begin{aligned} Addr &\leftarrow GPR[base] + \text{signed\_ext}(offset) \\ memword &\leftarrow memory[Addr] \\ byte &\leftarrow Addr_{1..0} \\ \text{if } byte = 0 &\text{ then} \\ GPR[rt] &\leftarrow \text{signed\_ext}(memword_{7..0}) \\ \text{else if } byte = 2 &\text{ then} \\ GPR[rt] &\leftarrow \text{signed\_ext}(memword_{23..16}) \end{aligned}$$

应该是这样的，记不太清了。

可以把它看作是条件写的 1h 或者是 1b。

## 分析

跳转和存储没什么好说的，都在 P5 里面说过了。所以这里说一下 P6 的计算。

P6 的计算一般是和乘除部件有关，但是其实也不难，这里建议课下做一下 madd、maddu、msub、msubu。

一个坑点在于 madd（或者 msub）里面。如果你的写法是

```
{temp_hi, temp_lo} <= {hi, lo} + $signed(rs) * $signed(rt);
```

好像就会出锅，这个具体的原因在 P1 里面讲过了，和 signedness 有关。另外，如果你写的是

```
{temp_hi, temp_lo} <= {hi, lo} + $signed($signed(rs) * $signed(rt));
```

也不对，具体也是 P1 里面有讲。正确的写法是：

```
{temp_hi, temp_lo} <= {hi, lo} + $signed($signed(64'd0) + $signed(rs) * $signed(rt));  
// 或者  
{temp_hi, temp_lo} <= {hi, lo} + $signed({{32{rs[31]}}, rs[31:0]} * $signed({{32{rt[31]}}
```

总之一句话：小心 \$signed()。

## 课下总结

P6 似乎没啥好说的，因为大部分指令我在 P5 里面一起做了。所以这里简单说一下 P6 和 P5 的区别。

做 P6 最重要的还是指令分类，做好指令分类能帮很多忙。我将指令分为以下几类。

- load：读取 DM
- save：写入 DM
- branch：b 类指令

- `calc_r`: r类指令, 除了 `jr` & `jalr` & 乘除法
- `calc_i`: i类指令, 除了 `lui`
- `md`: `mult`, `multu`, `div`, `divu`
- `mt`: `mtlo`, `mthi`
- `mf`: `mflo`, `mfhi`
- `shiftS`: `sll`, `srl`, `sra`
- `shiftV`: `sllv`, `srlv`, `srav`
- `j_r`: `jr`, `jalr`
- `j_addr`: `j`, `jal`
- `j_l`: `j`, `jalr`

然后对于新增的指令有这几种特殊的:

- 位移指令: 需要修改 ALU 的输入信号, 注意看清楚那个 RTL 的描述
- 跳转指令: 直接在 CMP 里面添加跳转方式即可
- DM 指令: 参见 P4 的那篇 blog (与今年的要求不大相同, 需要稍微修改)
- 乘除指令: 做一个乘除槽

需要注意的一点是我把 DM 的操作都整合到一个部件里面去了, 但是教程里是在 W 级寄存器后面添加了一个 BE 部件进行扩展, 这样的好处是关键路径更短, 频率可以加快。但是我懒得重写了.....

这里着重讲一下乘除指令吧。

## 乘除槽

乘除指令运算非常慢, 所以我们的 CPU 要模拟这种时延 (两种乘法延时 5 个周期, 两种除法延时 10 个周期)。为了不让乘除法拖慢速度, 我们遇到相关指令时让他们进入一个特殊的乘除部件进行运算, 其他指令继续执行。如果乘除法正在计算, 而我们遇到了新的要用到乘除部件的指令, 那么就 stall。

遇到了分析指令我们可以发现, 只有 `mflo` 和 `mfhi` 是读取乘除部件的指令。而剩下的六个 (`mult`、`multu`、`div`、`divu`、`mtlo`、`mthi`) 都是要向乘除部件写入数据的指令。而其中又只有 (`mult`、`multu`、`div`、`divu`) 会有时延, 另外两个直接向 `lo` 和 `hi` 写入数据。

在乘除部件中我们有两种信号表示正在运算:

- `busy`: 部件正在运算乘除法, 在乘除部件里面用一个寄存器记录

- `start`：部件即将要运算，即当前 E 级的信号是否是四种乘除信号之一

其中 `start` 是用来应对这种情况：

D	E	M	W
mult	mult	xxx	xxx

此时指令进入 E 级，下一个时钟上升沿乘除部件将要进行运算，并且将 `busy` 设置为 1。但是此时显然需要 `stall`。所以我们用一个 `start` 指令指示要开始乘除运算了，让 E 级的指令被清空，防止下一个时钟上升沿进入下一级。

`stall` 的条件是 `start | busy`。

```
wire stall_HILO = E_HILObusy & (D_md | D_mt | D_mf);  
  
assign stall = stall_rs | stall_rt | stall_HILO;
```

#### PREVIOUS

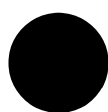
[BUAA-CO-LAB] P5 流水线 CPU-LITE  
([/POSTS/BUAA-CO-LAB-P5/](#))

#### NEXT

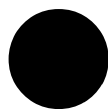
[BUAA-CO-LAB] P7 MIPS 微体系  
([/POSTS/BUAA-CO-LAB-P7/](#))



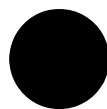
(<https://github.com/roife>)



(<https://twitter.com/roifex>)



(<mailto:roifewu@gmail.com>)



([/index.xml](#))