BUAA - 计算机组成实验 (/archive/?order=series&tag=BUAA+-+%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%BB%84%E6%88%90%E5%AE%9E%E9%AA%8C%40Courses%40Series)

北航 (/archive/?order=tags&tag=%E5%8C%97%E8%88%AA%40Tags%40Tags)

计算机组成 (/archive/?order=tags&tag=%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%BB%84%E6%88%90%40Tags%40Tags)

Verilog-HDL (/archive/?order=tags&tag=Verilog-HDL%40Languages%40Tags)

数字电路 (/archive/?order=tags&tag=%E6%95%B0%E5%AD%97%E7%94%B5%E8%B7%AF%40Tags%40Tags)

「BUAA-CO-Lab」 P5

流水线 CPU-lite

简单流水线 CPU

Posted by roife on November 23, 2020

上机总结 1

> 失败了

- 第一题：cco

$$
\begin{aligned}
&temp \leftarrow 0 \\
&\text{for } i \text{ in } 31 \cdots 0 \\
&\quad \text{if } GPR[rs]_i == 1 \text{ and } GPR[rt]_i == 1 \text{ then} \\
&\quad\quad temp \leftarrow temp + 1 \\
&GPR[rd] \leftarrow temp
\end{aligned}
$$

  改一下 ALU 就好了。注意如果在 ALU 里面用的是 `assign`，那么建议使用 `function`。

- 第二题：bgezall

$$
\begin{aligned}
\mathrm{I}: & target\_offset \leftarrow \mathrm{signed\_ext}(offset||0^2) \\
& condition \leftarrow GPR[rs] \geq 0 \\
& GPR[31] \leftarrow PC + 8 \\
\mathrm{I+1}: & \mathrm{if}\ condition\ \mathrm{then} \\
& \quad PC \leftarrow PC + 4 + target\_offset \\
& \mathrm{else} \\
& \quad \mathrm{NullifyCurrentInstruction}()
\end{aligned}
$$

$\mathrm{NullifyCurrentInstruction}()$ 的意思是清空延迟槽。

也就是说仅当跳转时延迟槽生效。

要考虑当 CPU stall 的时候不能清空延迟槽。

还有一种方法是监测到 `D_b_jump` 时，F 级的指令插入 `nop` （这个在 P7 很常用）。

```
1    F_instr = (D_b_jump && D_bgezall) ? 32'd0 : im[pc[13:2] - 12'hc00];
```

- 第三题：lwso

$$
\begin{aligned}
& mem\_data \leftarrow mem[GPR[rs] + offset] \\
& temp \leftarrow (GPR[rt]_{31}||GPR[rt]) + (mem\_data_{31}||mem\_data) \\
& \mathrm{if}\ temp_{32} == temp_{31}\ \mathrm{then} \\
& \quad GPR[rt] \leftarrow GPR[rt] + mem\_data
\end{aligned}
$$

实际上就是：如果第二步相加没有溢出就赋值，否则不赋值。

上机总结 2

爬了，又没过。

- 第一题：cmco

$$
\begin{aligned}
&temp \leftarrow 0\\
&count \leftarrow 0\\
&\text{for } i \text{ in } 0\cdots 31\\
&\quad \text{if } GPR[rs]_i == 1 \text{ then}\\
&\quad\quad count \leftarrow count + 1\\
&\quad\quad \text{if } count > temp \text{ then}\\
&\quad\quad\quad temp \leftarrow count\\
&\quad \text{else}\\
&\quad\quad count \leftarrow 0\\
&GPR[rd] \leftarrow temp
\end{aligned}
$$

同样改一下 ALU 就好了。就是统计二进制最长有几个连续的 1。

- 第二题：blezalc

$$
\begin{aligned}
&target\_offset \leftarrow \text{signed\_ext}(offset||0^2)\\
&condition \leftarrow GPR[rs] \leq 0\\
&\text{if } condition \text{ then}\\
&\quad PC \leftarrow PC + 4 + target\_offset\\
&\quad GPR[31] \leftarrow PC + 8
\end{aligned}
$$

这个题目课上的 RTL 表述其实有问题……更正后应该是这样的，即如果跳转则链接寄存器。

- 第三题：lrm

$$
\begin{aligned}
&vAddr \leftarrow GPR[base] + \text{sign\_extend}(offset)\\
&mem\_reg \leftarrow Memory[vAddr]_{4..0}\\
&GPR[mem\_reg] \leftarrow GPR[rt]
\end{aligned}
$$

存储地址为 DM 输出的低位。

上机总结 3 & 分析

终于过了……这次的指令是前两次的混合体，所以就不展开说了。

这里就简单讲讲 P5 的套路吧。

首先课上的三条指令：计算 + 跳转 + 存储。

1. 计算绝对是最简单的，重点在于你的写法。如果你用 `always@(*)` 那么问题不大。但是如果你用的是 `assign` 那么最好使用 `function` 来实现计算指令。
2. 跳转一般也不难，一般是条件跳转 + 条件写。 跳转指令一个好处在于它是在 D 级决定是否跳转的，也就是说在 D 级你可以获得全部的正确信息（相反如果是类似于 `lwso` 这种，你必须要读出 DM 的值才能决定怎么做）。所以我们的方案是 D 级生成一个 `D_check` 信号然后流水它。然后每一级根据这个信号判断写入地址/写入值之类的。

```
// 检测信号
D_check = D_bgezalc & D_b_jump;
// CU
assign RFDst = // ...
               bgezalc ? (check ? 5'd31 : 5'd0) :
               5'd0;
```

3. 条件存储一般是最难的。 但是掌握了套路之后也还好。条件存储的特点是必须要到 M 级才知道要写啥，这就给转发之类的造成了困难，所以我们的策略是如果 D 级要读寄存器，而且新指令**可能**要写这个寄存器，那么就 stall。具体来说是这样的：

```
// lwso
wire stall_rs_e = (TuseRS < TnewE) && D_rs_addr && (D_lwso ? D_rs_addr ==
// lrm
wire stall_rs_e = (TuseRS < TnewE) && D_rs_addr && (D_lrm ? D_rs_addr : D_
```

在 CU 中的写法则与条件跳转类似。

```
// lwso
M_check = D_lwso && condition;
// CU
assign RFDst = // ...
               lwso ? (check===1'd1 ? 5'd31 : 5'd0) : // 注意不是直接一个 che
               5'd0;
```

这里要注意的一点是，如果你用的是统一 CU 的写法，那么会出现一个问题： `check` 信号只有在 M 级传入。这个时候在 E 级的 CU 里面这个信号是不定态 x，这样会导致 RFDst 信号出锅。所以我们这里的写法是 `check===1'd1` ，这样可以排除 `x` 的情况。

课下总结

P5 做得真是艰难，一方面是流水线这个东西本身比较复杂，包括转发/暂停等一系列机制；另一方面是数据通路由于流水线寄存器的加入，代码量大大上升，个人经验大概比 P4 增加了 1/2 到 1 倍左右。而且 P5 时流水起迭代开发的起点，所以一个好的架构非常重要，所以做的时候会非常想要不停地重构，这又浪费了很多时间。所以建议在做之前先做好设计工作，然后做的过程中不要临时起意去修改。

总的来说流水线的难点在于三方面：

1. 数据通路和控制器架构
2. 转发
3. 暂停

下面我会分别介绍这三方面的内容（主要是教程内容，我会做一些个人理解和细节补充）。

注意：做 P5 前请确保自己熟悉 Verilog 和 P4，并且看完了教程内容和高老板的课件。

流水线要点

开发时你需要注意以下几个方面：

1. 命名风格：建议使用 流水级_名称 ，如 `D_instr`
2. 代码排布：建议同一级放在一起，同样的功能放在一起，独立的部分可以适当抽出独立成模块，即遵循“低耦合，高内聚”的原则
3. 多使用宏和常量来简化代码，如推荐使用 `ALU_add` 代替 `3'b001`

数据通路及控制架构

架构方案选择

数据通路架构即 FDEMW 五个部分的数据通路，以及数据通路中的流水线寄存器。

P5 是 CPU 迭代开发的起点，因此一个好的架构是很好的起点。总的来说架构包括这几部分：

1. 译码方式：分为集中式译码和分布式译码。

   集中式译码即在 `F` 或 `D` 级进行译码，然后将控制信号流水传递，即 P3/P4 采用的译码方式；分布式译码则只流水传递指令，控制信号在每一级单独译码。

   集中式译码的好处在于速度更快，关键路径更短；分布式译码关键路径更长，速度较慢（差不了很多）但是译码信息模块化，不需要流水传递控制信号，更适合应试和学习。

   这里我们采用分布式译码。

2. 译码语句：

   可以用 `always @(*) + case` 或者 `assign` 进行译码。这里使用后者（因为后者代码更短）。

3. 控制器设计：

   为每一级单独写一个 CU, 或者写一个大的 CU 每一级实例化一次，取不同的信号使用。

   前者可以节省晶体管，但是实际操作会遇到很多问题：比如指令一多（P6）那样译码信息写起来很麻烦。如果你像我一样，写了一个 `interpreter.v` 来复用解码代码，那样会发现还不如直接单写一个 CU，而且一些信号会重复使用，如寄存器写使能信号在阻塞模块中会重复使用。出于面向对象的工程思想，我们倾向于把信号集中在一起。

   后者写起来简单，但是要使用更多的晶体管。

   这里使用后者。（我们要对编译器抱有信心，相信他会帮我们优化代码的 (。･ω･。)）

总而言之，我们采用的是更工程化，更符合“低耦合高内聚”原则的方案，效率和成本不是我们考虑的首要因素。

数据通路架构

数据通路部分的元件基本上和 P4 一样，我们将它们分在不同的层级。

1. F 级：PC，IM
2. D 级：GRF，EXT，CMP，NPC
3. E 级：ALU
4. M 级：DM
5. W 级：无

连线参考教程和 P4。需要注意的有两点：

1. 新元件 CMP 用于 b 类指令

2. NPC 位于 D 级，他的输入中既需要 `D_pc` （b 类，j 类），又需要 `F_pc` （PC+4）。

控制器架构

控制器我分为三部分。第一部分分割指令（取出 rs，imm 等），第二部分解析指令（`wire add = (opcode == `OP_add) && (func == `FU_add)`），第三部分分类指令（`load`， `calc_r`，`calc_i` 之类），第四部分建模控制信号。

采用统一控制器的好处是，在数据通路、转发、阻塞中你都可以复用这些代码。

转发

只有 RS 和 RT 会被转发。有四个位点是转发的接受端：

1. NPC 的 RS 输入端
2. CMP 的两个输入端
3. ALU 的输入端
4. DM 的输入端

可以的话最好实先画一个图。

转发接受
NPC
D_CMP_A/B
ALU_A/B
DM_in

转发输出	**（就近优先）**
E_reg	j_l / lui
M_reg	calc_r/calc_i

转发输出	**（就近优先）**
W_reg	load
寄存器内部转发	

注意这里我把 `lui` 特殊处理了，并没有归入到 `calc_i` 里面去。

然后是接受端选择数据，选择的顺序按照**就近原则**，优先选择下一级的数据，不行就下两级，如果都不行就采用本级寄存器的数据。接受端可以接受数据的条件为：

1. 供给端的寄存器地址与当前的相同
2. 当前需要的地址不为 0
3. 供给端可以转发（我们使用无脑转发，所以不需要考虑）
4. 供给端的指令会写数据（如果不写数据我们令写入地址为 0，所以不需要考虑）

即：

```
1    wire [31:0] FWD_E_RS =  (E_rs_addr == 5'b0) ? 0 :
2                            (E_rs_addr == M_RFDst) ? M_RFWD :
3                            // ...
4                            E_rs;
```

这里的 A-T 法有一个很妙的地方。注意到我们的判断条件仅仅是 `E_rs_addr == M_RFDst` 而不用加其他信号，相当于暴力转发。假如 D 级可以用 E/M/W 级的数据，但是 E 级还没算好，为什么不需要判断 `E` 的指令能否被转发呢？我们考虑两种情况：

1. 需要暂停 比如 `beq-add-add-add`，那么直接 stall，下一个周期 `E` 变成 `nop` 了，那么 `E_RFDst` 不符合 `E_rs_addr == M_RFDst`，自然就使用 `M` 了。
2. 不需要暂停 比如 `add-add-add-add`，那么这个时候 `D` 的 `add` 会从 `E` 得到一个错误的数据，但是没关系，现在还不用。下一个周期要用到这个数据的时候，我们进行了第二次转发，即 `M`（原先的 `E`）转发过来的数据，此时这个数据经过 `ALU` 已经算完了，所以我们可以直接利用。

需要注意的有以下几点：

1. 本级的寄存器接受了转发后，一律使用转发后的值。我们可以把使用原值也看成一种转发，即我们必须使用最新的值，包括给下一级寄存器传递信息时也要用转发后的数据。

   如 `FWD_E_RS` 是转发后的值，那么在 `M_reg` 的 `rs_in` 信号就要用 `FWD_E_RS` 而不是 `E_rs`。

2. 寄存器内部转发

   其实就是 D 级接受 W 级的转发电路放在了寄存器内部

   ```
   1     assign RD1 = (A3 == A1 && A3 && WE) ? WD : grf[A1]; // 内部转发
   ```

3. 为什么转发不需要判断 `M_RFWE`？因为在 `M_RFDst` 中我们令不写寄存器的部件的写入地址为 `$0`，所以不会对结果造成影响。

阻塞

建议通读教程，看一遍高老板的 PPT，尤其要搞清楚 Tuse 和 Tnew 两个东西。

- Tuse：指令进入 D 级后，其后的某个功能部件再经过多少时钟周期就必须要使用寄存器值。对于有两个操作数的指令，其每个操作数的 Tuse 值可能不等（如 store 型指令 rs、rt 的 Tuse 分别为 1 和 2）。
- Tnew：位于 E 级及其后各级的指令，再经过多少周期就能够产生要写入寄存器的结果。在我们目前的 CPU 中，W 级的指令 Tnew 恒为 0；对于同一条指令，Tnew@M = max(Tnew@E - 1, 0)。

然后直接遵循 `Tuse < Tnew` 则阻塞的规则就可以了。建议先画一个表格分类 Tuse 和 Tnew。可以按照高老板的 PPT，用“产生输出的部件”来分类 Tnew。如 ALU 产生输出的有 R 型、I 型、sw 的地址，把他们归为一类。

阻塞时只需要判断 RS 和 RT。

Tuse			
calc_r(rs)\shiftS	1	calc_r(rt)	1
calc_i(rs)	1	shiftS(rt)	1
load(rs)	1	store(rt)	2

Tuse			
store(rs)	1	branch(rt)	0
branch(rs)	0		
jr/jalr(rs)	0		

判断的时候要注意条件：

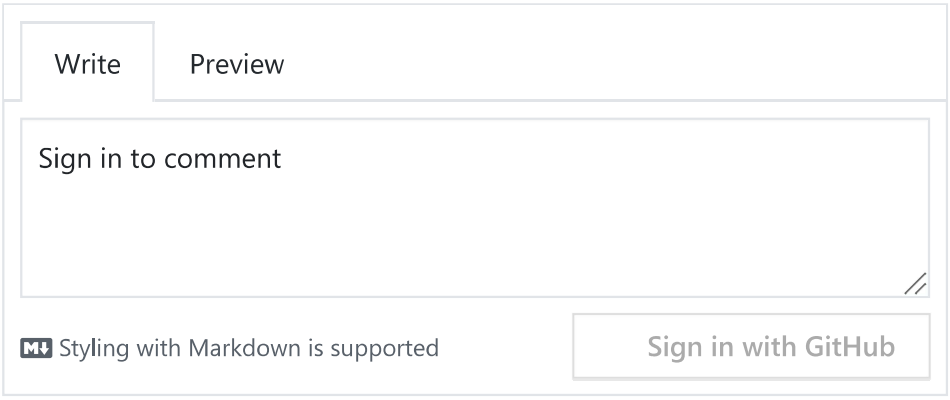
1. `Tuse < Tnew`
2. 写入地址不为 0
3. 写入地址和后面级写入地址相同
4. 后面级可写寄存器

```
wire [2:0] TuseRS =     (D_branch | D_j_r) ? 3'd0 :
                        // ...
                        3'd3; // 如果用不到就令其为无穷大，防止多余的 stall

wire [2:0] TnewE =  E_calc_r | E_calc_i ? 3'd1 :
                    // ...
                    3'd3;

wire stall_rs_e = (TuseRS < TnewE) && (D_rs_addr && D_rs_addr == E_RFDst);
// ...
wire stall_rs = stall_rs_e | stall_rs_m;

// ...

assign stall = stall_rs | stall_rt;
```

为什么这里写 `stall` 不用判断 `Wr` 信号？理由如上。

PREVIOUS
「BUAA-CO-LAB」 P4 单周期 CPU - 2
(/2020/11/19/BUAA-CO-LAB-P4/)

NEXT
「BUAA-CO-LAB」 P6 流水线 CPU-LITE2
(/2020/12/02/BUAA-CO-LAB-P6/)



(/feed.xml) (https://github.com/roife)

(mailto:roifewu@gmail.com)