关于 P4-P6 一些反思

2015/12/20

写在开头：

1.我的计组之旅是一个比较坎坷的过程吧，看似听懂了理解了原理，但实际上理解的并不全面和到位。但是，这个时候不要害怕或者停滞不前，努力通过看大黑书，ppt 等多种思考方式并且一定要紧跟手动敲代码，只有理论和实践同时进行，你才会有所突破。

2.听从了各路大神给我的建议和指导，改变了自己的想法，一次次的修改代码是一件很浪费时间并且打击自己已完成工程的事情。如果你也有幸有各路大神给你建议，这是好事也是坏事吧。因为，每个人的想法其实的都不一样，你可以去听，去比较和思考哪一种框架和写法更好来优化自己的部分代码，但是如果你试图按照别人的模式去构建自己的 CPU，抱歉，一个拄着拐杖的人是不太可能走远的。而计组实验想带给我们的是独立完成工程的能力。想起马院长说过的一句话，“知识是可以获取的，但能力需要自己一次次的磨练才能增长”计组末了我想起了这句话，希望能给自己和看到这份反思的读者一些思考。

3.由于之前的一些耽误，没有能够在 P7 测试的时候完成 P7。但是我想我在进步，因为由开始的完全不会到在 P6 测试前三天独立写完了两个 P，这本身就是一种进步。虽然计组进程不等人，错过了测试 p7、p8 的机会，但是我会继续写，因为我们不完全是要获得那部分学分，而是要让自己真正的掌握技能，不是吗？所以，继续向前。

正文

首先，P4/P5P6 分成两大类。明确 P4 是开发单周期处理器、P5P6 是开发流水线处理器。

无论哪个 P，很重要的一点是你要明确你的指令，它要干什么，要怎么一步步经过数据通路中的部件。数据通路顾名思义即当某条指令执行时，顺序经过的部件。那么我们就想并不是所有的指令都要经过所有的部件，进而需要控制器的控制信号。

Project4 反思：

P4 实际上是是将 P3 用代码的实现，容易出错的地方有：

1.指令理解是否完全正确。例如 jal 指令，是先将当前 pc+4 保存到 31 号寄存器中。然后再将{pc[31:28],instr[25:0],2’b00}作为下一条指令的 pc。

2.不同情况下不同的数据进入同一端口。所以需要多选器并加以控制信号，实例化各模块时，把各控制信号考虑完备。并且实例化时建议采用 .clk(clk) (括号中的内容可自行定义)的形式，该方法不会限制输入输出端口的个数，顺序。说到问题的考虑完备与否这件事，并结合后期的工程的复杂性，强烈建议码代码之前列好表格并写实验报告，报告中尽可能的把你的各个模块描述清楚，这既是对自己思路的整理，也方便后期调试。或者说，与其报告之后写，作为总结不如之前去写作为指导。调整一下做事的顺序会有很大的改观。

3.起初会犯一些初级错误，比如指令的 op 和 func 写错某位。尽量减少此类错误，以减少后期 debug 时间。说到 bug 这件事，遇到 bug 是一件好事，不要惧怕或者烦 debug，应当摆正心态，出现问题才能很好的解决问题。

Project5&6 反思：

前言：流水线相比于单周期是一种更有趣的阶段。起初我沿用了 p4 的架构即 mips 下的 datapath 和 ctrl（如下图 1 所示）。但是你会发现对于一个简单的新加的指令，或许就需要在数据通路中重新来一个多选器以及其控制信号。并且这个时候的调试等于火葬场。本身代码功力不强再加上繁多指令一个个袭来，且不说暂停和转发的控制，一不留神写错一个字母，工程就会崩溃。在痛苦的用这种结构写完 p5.1p5.2。看 p6 的 50+指令时，果断放弃了这种结构。之后的三天就是用高老师工程化的方法写了 p5.1,p5.2,p6。当然，有同学会不喜欢这种工程化的方法，有自己更好的设计架构。但是这种方法重点二字是“工程”。

友情提示：1.遇事先别急着做，比较且选择更具备后续扩充的框架结构。

2.写代码和调试能力同样重要，**bug 要自己去调**，这是细化指令流程和代码能力的很好的方式。工欲善其事必先利其器，善于运用仿真调试工具。

3.工程化方法的特点是，做好前期表格准备，后期码代码是体力活动。

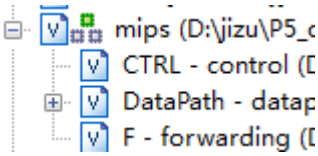

图 1

于是，接下来我要讲的是在接触到工程化方法后一步步的摸索，很关键的一点是进行指令的分类。

A: 确定采用分布式译码还是集中式译码？

- 集中式控制器

  控制器只在 ID 阶段

  一个控制器，产生流水线全部的译码信号

  根据每一级的需要，考虑是否要将谋控制信号继续传下去。

- 分布式式控制器

  每一个流水线阶段都有各自的控制器，且每一级只产生该级功能部件相关的译码信号。

看似无脑，实际当你新增加特殊情况时，相比较于集中式控制器会不需要增加控制信号的级别传输。也就是前言中所说的，具有后续扩充升级的优点。

B：数据通路、暂停、转发的表格以及实验报告真的那么重要吗？

开始，我认为这些东西水一水就够了，关键是代码写出来。但实际上磨刀不误砍柴工，不仅是不断修正自己理解不深刻知识点的过程，也便于后期调试。很多事情，转换一下先后顺序会有不同的效果。

那么多指令怎么列表呢？一条条指令一级级的来。

首先，将指令的流程按照 5 级依次列表。从 F 级功能部件开始-D 级流水线寄存器-D 级功能部件-D 级更新 pc-E 级流水线寄存器-E 级功能部件-M 级流水线寄存器-M 级功能部件-W 级流

水线寄存器-W 级功能部件。在每一级中中有哪些功能部件，其输入端口的名字是什么依次列在表中。然后拿一个简单的指令开始对表格进行填充。在每一级每一个部件中该指令的输入是什么。当再一条指令当与之前的输入来源产生冲突的时候，只需在输入来源该列某部件这一行添加即可，如下图 ALU 部件 A 的输入来源可能是多个，由此我们增加相应的控制器以及控制信号。

级别	部件	输入	输入来源			MUX	控制信号
	ALU	A	RS_E	IR_E[10:6]		MUX_ALU_A	ALUasel

把基础的 load,store,beq,add,addi,j 族指令写好后，会发现其它新增加的 50 多条指令，目前除乘除法之外都是一样的数据通路，即在控制器归类即可，相比较于之前的结构有很大的改观。

部件	输入	LW
PC		
ADD4		PC
IM		PC
PC		ADD4
IR@D		IM
PC4@D		
RF	A1	IR@D[rs]
	A2	
EXT		IR@D[i16]
NPC	PC4	
	I26	
PC		
IR@E		IR@D
PC4@E		
RS@E		RF.RD1
RT@E		
EXT@E		EXT
ALU	A	RS@E
	B	EXT@E
IR@M		IR@E
PC4@M		
AO@M		ALU
RT@M		
DM	A	AO@M
	WD	
IR@W		IR@M
PC4@W		
AO@W		
DR@W		DM
RF	A3	IR@W[rt]
	WD	DR@W

C：流水线冲突覆盖性分析

在和同学的交流中，我发现有两种流水线冲突分析的方法。

首先，冲突产生是因为供应者不能满足需求者的需要而产生的。如果供应者虽然没有到达最终目的地，但是可以提前通过流水线寄存器为需求者提供需要的寄存器值，那么可以通过转发实现，转发包括 W->E,W->D,M->E,M->D。然而如果不能通过转发保证供应，那么就需要暂停以等待转发或者其它方式的数据提供。

1.**按照工程化的设计理念，构造 Tuse 和 Tnew 表格，进而构造暂停和转发表格**。对于不同类型的指令 Tuse 和 Tnew 是不同的值，所以在控制器中需要知道当前指令以及前一条和前前条指令是归属于哪一族的。这样构造的转发或者暂停表格以及代码量比较大。

2.**从根本上来讲转发的指令没有关系**。仅取决于当前指令所需要的寄存器的值是否和上一条指令或者上上条指令的目的寄存器是否产生冲突。故在转发单元中仅传入这些寄存器的地址，判断是否相同（当然是否是 0 寄存器也需要考虑）以及之前的写使能信号是否是 1 即可。代码量较少。

3.**出现冲突的原因是两个，数据出的晚或者用的早**。出的晚的有 lw 类指令，用的早如 beq 类，所以会产生的冲突必然是前一条是 lw 指令之后需要用到取出来的数据，或者 beq 指令比较的两个数需要来源于前期保存到寄存器中的值，或者 lw 与 beq 相遇那么需要暂停更多的周期，这样看来是不是冲突情况就分析清楚和全面了。

4.为什么乘除法模块不能添加在 ALU 中呢？

首先乘除法也是计算类指令，但其数据通路与 add 不完全一致。当执行乘除法或者 mthi/mtlo 时，到乘除法模块指令就结束了，不存在之后向 GPR 存数据的步骤。

更关键的是，假设当前是乘除法指令，例如乘法，它会执行 5 个周期，如果放在 ALU 模块中必然会影响 ALU 正常的指令执行。因为我们要求当乘除法模块为 busy 时，不能进行其他运算。但实际上不要求那么多暂停，或者说是对时间的浪费，我们把 ALU 和乘除法模块分开写。

乘除法的暂停需要独立设置单元，当 busy 信号为 1，且下一条又是乘除法类指令时，暂停。并且细究时钟的变化，刚刚 start 时 busy 还没有来得及设置为 1，但如果下一条依旧是乘除法类指令，仍需要暂停。

P5&p6　debug 过程中遇到的问题以及感受

- 代码写规整是个人问题。但还是觉得养成良好的习惯比较好，这样更方便 debug 时观察出错误。
- 首先，自己排除 wire 线没有连接好的问题。

1.通常是忘记定义连线，当线宽不为 1 时会出现 zzz 的情况。

2.定义了线却写错。这是经常出现的问题！！如 ALU_outa 和 ALUout_a 混用,建议初始时就要有自己统一的命名方式。

- 强烈建议在控制器中将指令归并，不然会在现场增加指令时需要修改很多地方，如果有遗落调试起来会很烦，并且失去工程化方法的意义。
- 当你排除了线、复制粘贴写错一行、bgtz 错写成 bltz 导致 bltz 写了两次等问题之后。如果还存在 bug，那么很有可能是有**原理性的地方**写错了。

如：

```
2'b01:
   begin
   if(cmpyes)
     npc<=$signed(pc4)+$signed({instr[15:0],2'b00});
  // npc<=$signed(pc4+{instr[15:0],2'b00});   这样并没有什么卵用
```

这是 npc 中计算 beq 应当跳转的 pc 时的代码。当 cmpyes 为 1 时，表示 beq 条件判断成立，但是此时 instr[15:0]是一个负数的补码，而 verilog 都当成无符号来算。所以就加出了一个很大的数，对于 instr 的查找目前没有问题，因为我们的 IM 比较小，但是 npc 得前几位是错误的。即 `npc<=pc4+{instr[15:0],2'b00};` 并不是完全正确的写法，正确的应该按照上图进行符号数的转化。并且当相加的两个数有一个是无符号时会当做无符号的运算。

- D 级 GPR 和 W 级 GPR 物理上是同一个寄存器，在逻辑上我们把它当做两个看待。只实例化一次。

2014 级

14061148 蔡颖婕