计算机体系结构大作业报告

5130309059　　李佳骏

taringlee@sjtu.edu.cn

5130309052　　陆嘉辉

warandpeace@sjtu.edu.cn

2015.6.18

一、简介

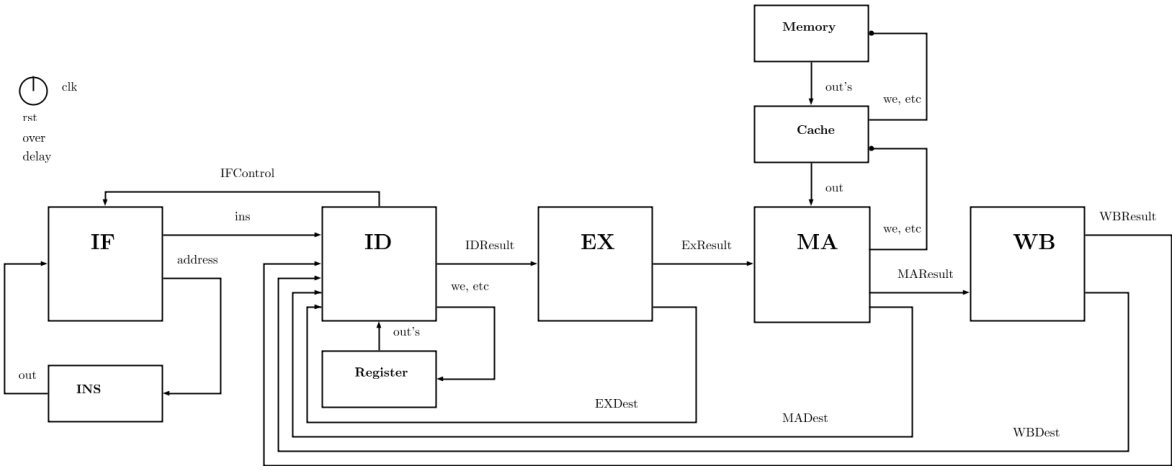
1. 设计图

大致上是遵循了标准的五级pipeline的构造，在一些细节上做了改动。

本设计中，指令和数据的内存是分开存储的；而因为，本设计中由于输入的指令很少，因此就把他们全部放在了缓存中，而数据全部放在内存中，再通过缓存进行读写。

此外，设置了全局的时钟clk，确保所有部件依照同一个时钟运作。同时我们有设置了复位reset，在特定的时候（例如程序结束），将所有的原件全部恢复为零，且用一个变量over判断指令是否已经结束。另外为了实现200 cycle 的内存延迟，设置了delay用于记录延时。

由于自身能力和经验限制以及其它原因，最后delay没有成功实现，十分抱歉。



2. Instruction Fetch模块

指令抓取是五级pipeline的第一步。

IF模块的输出端比较简单，只需要将读取到的指令inst输出即可。

IF模块需要获取stall信号、delay信号，和branch predictor的结果（包括是否跳转和跳转的地址），因此需要对应的数据进行输入。

由此可见，需要将IF模块与指令缓存模块相连。

3. Instruction Cache模块

此模块相对容易实现，由于只需要读取，而不需要向其中写入指令。我们设计输入为指令地址，而输出则是对应地址上的指令。

4. Instruction Decode模块

指令解码过程在五级pipeline中其相当重要的作用，因为在这一阶段会处理hazadests以及决定branch是否条跳转。

对于stall的处理如设计图所示，ID模块除了接收IF 模块发出的指令，还要接受EX,MA,WB阶段的相关信息（主要是dest）与指令进行比对。如果该指令需要用到的寄存器在前面的三条指令中被使用到了，就需要stall。

而对于branch predictor，本设计直接在ID中实现，使用一个bit：当上一个branch是taken的时候，预测下次也为taken；反之，预测下次为not taken。

此外，ID模块还应获取EX，MA，WB阶段返回的信息，实现bypassing。

ID模块的输出有两种可能，一是进行ALU运算，二是在内存中读写数据，因此输出两条bus，IFControl连入IF模块，IDResult连入EX模块。

IFControl是ID模块控制IF模块所使用的组件，包含了如下三个数据：[34]处是valid的信号（valid=1表示可以执行，反之则必须stall）；[33]处是branch predict的值（1为跳转，0位不跳转）；[32]处是branch taken的实际结果（1为跳转，0位不跳转）；[31:0]处是跳转的地址。

IDResult是ID模块控制EX模块所使用的组件，包含了如下六个数据：[105]处是valid信号；[104:101]为操作符的值；[100:96]为dest，[95:64]为src1，[63:32]为src2，[31:0]为立即数。

在ID阶段，需要获得将操作时所用到的寄存器，因此ID模块也与寄存器堆相连。

5. Register模块

寄存器堆内有32个寄存器，它们与ID模块相连。

输入可能有两种情况。如果是非写入指令，则输入接口为src1和src2，而输出只需要src1和src2的对应数据即可（rout1和rout2）；而遇到sw 指令的时候就需要写使能we，以及相应的输入接口dest 以及数据win。

6. Execution模块

执行模块主要就是ALU过程，由于要求的是行为模型，这一模块的实现就简单了很多。

输入的是ID模块发出的结果IDResult，而输出的也有两条，一条是连向MA模块的自身模块的执行结果EXResult，另一条则是为了实现bypassing而使用的EXDest，将dest的位置传回给ID模块。

EXResult是EX模块控制MA模块所用的组件：[73]处是valid信号；[72:69]是操作符对应值；[68:64]为dest；[63:32]是通过EX操作后得到的值；而[31:0] 是sw 指令对内存写入的值。

7. Memory Access模块

读写模块与cache相连。

输入线就是EX模块传出的结果，而输出也类似EX分为两条，一条连向WB（即MAResult），另一条为了实现bypassing将内存操作的dest传回给ID模块。

MAResult是MA模块控制WB模块所用的组件：[73]处是valid信号；[72:69]是操作符对应的值；[68:64]是dest；[63:32]是EX模块后得到的值；[31:0]是lw 指令从内存中读出的值。

8. 缓存Cache

本设计只采用了一级缓存，每个index（即一个row）存储四个block，共有256 个row。

cache与内存相连。

9. Data RAM模块

数据内存可以存取65536个数据，每次cache访问时，从其中取出4个block。

当内存进行读取操作时，输入端提供src1和src2的对应memory位置，内存将分别提供临近的4个block给cache。

当写使能为1时（即向内存写入数据），输入端提供需要写到的地址和数据，内存完成写入即可。

10. Write Back模块

输入端只有MA模块的MAResult，而输出线有两条，全部连向ID模块：即dest和写回操作的结果WBResult。

WBResult是WB模块对ID模块的反馈组件：[37]处是valid信号；[36:32]是rd；[31:0]是反馈给ID的数据，用于对于Register的更新以及bypassing。

二、二进制操作代码

由于mips的标准二进制码不够方便，本设计采用了自创的二进制代码来完成mips的转换，使得之后的操作更加简便：

	[31:28]	[27:23]	[22:18]	[17:13]	[12:0]
nop	0(32)				
add	0001	dest(5)	src1(5)	src2(5)	0(13)
sub	0010	dest(5)	src1(5)	imm(18)	
mul	0011	dest(5)	src1(5)	src2(5)	0(13)
addi	0100	dest(5)	src1(5)	imm(18)	
subi	0101	dest(5)	src1(5)	src2(5)	0(13)
muli	0110	dest(5)	src1(5)	imm(18)	
lw	1000	dest(5)	src	imm(18)	
sw	1001	src(5)	dest	imm(18)	
j	1010	imm(28)			
beqz	1011	src(5)	imm(23)		
bnez	1100	src(5)	imm(23)		
bltz	1101	src(5)	imm(23)		
blez	1110	src(5)	imm(23)		
over	1(32)				

三、运行方式

通过shell脚本已经能够将操作方法简单化。根目录下的plus.s文件是进行累加的MIPS代码，若没有代码改变则不需要更改；根目录下的data.in文件是输入数据文件，每次将输入数据放入该文件内即可。稍后运行run2.sh脚本文件，即可看到verilog的运行结果。

若要查看相关数据的波形。直接通过使用gtkwave查看对应数据(test.vcd)的波形即可。

四、实验结果

实验的内容是100个数的累加。这100个数放在数据内存里的前100位里，而100 个数的和将被写入到第101个数据内存位置。在这里为了简化获得答案的方式，对于写入操作(sw)，可以直接输出写入值，也就是在运行中直接通过display命令输出结果。

五、致谢

感谢梁晓峣老师提供我们这次可能是人生中唯一一次写verilog的机会，同时感谢梁晓峣老师和景乃锋老师的课程，对我们的设计起到了高屋建瓴的指导作用。

感谢李青林学长、谢佳明学长、陈皓学长与我们交流经验，对我们的工程起步和架构设计起到了良好的帮助作用。

感谢姚思秋同学、张曦虎同学、张家睿同学、姚京韬同学、翁健同学与我们一同讨论，一同熬夜奋斗。

最后感谢一起奋斗的队友，让我们深切感受到了团队合作的重要性。

六、参考资料

1. MS108 Computer System I Lectures，梁晓峣
2. Digital Circuits and Verilog HDL，景乃锋
3. Computer Architecture, John L. Hennsey & David A. Patterson