计算机组成原理实验报告

19373106 裴宝琦

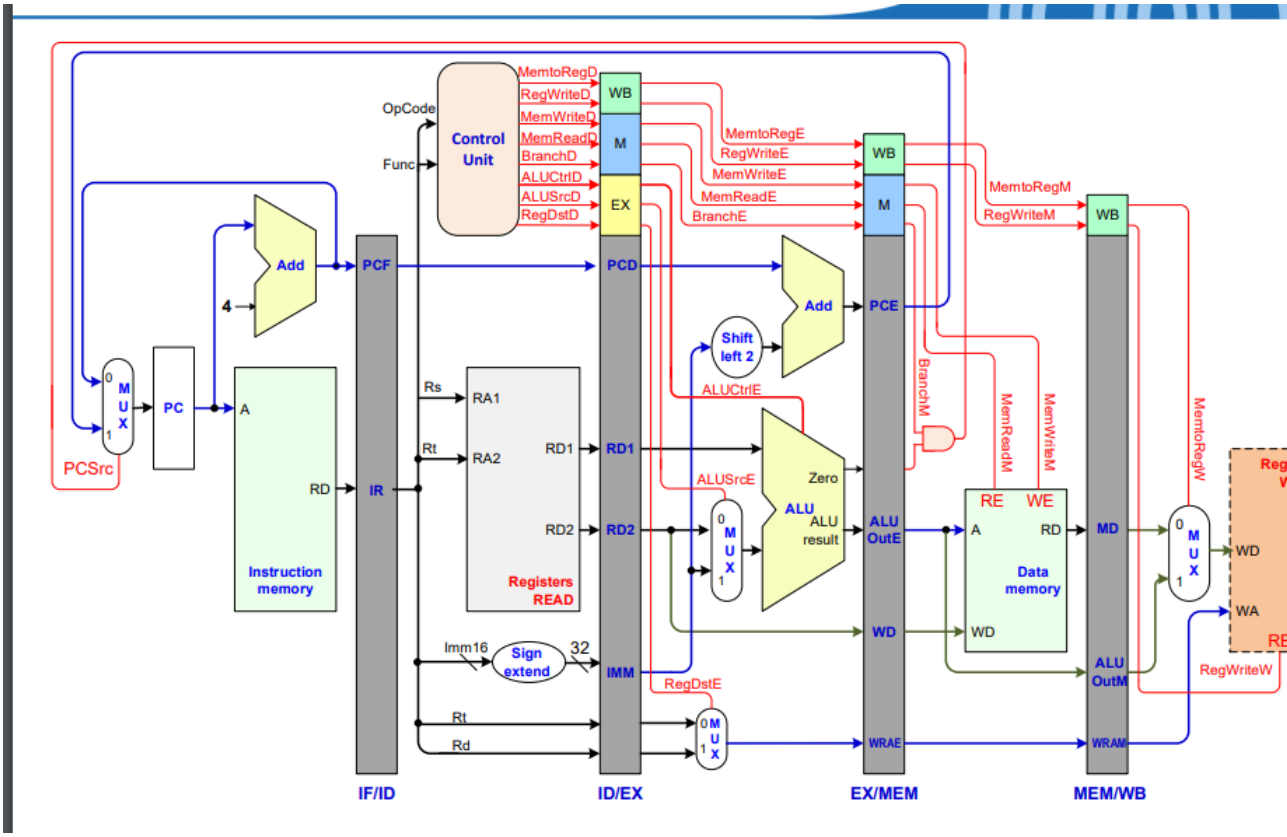
一、CPU 设计方案综述

本 CPU 为 Verilog 实现的流水线 MIPS – CPU。

第一阶段：先构造出来不考虑转发暂停的流水线主控制器。

		lw	sw	add	sub	ori	beq	j	jal	jr
PC		ADD4	ADD4	ADD4	ADD4	ADD4	ADD4|NPC	NPC	NPC	RF.RD1
IM		PC	PC	PC	PC	PC	PC	PC	PC	PC
ADD4		PC	PC	PC	PC	PC	PC	PC	PC	
D级	IR	IM	IM	IM	IM	IM	IM	IM	IM	IM
	NPC	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	
RF	A1	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D			IR[rs]@D
	A2		IR[rt]@D	IR[rt]@D	IR[rt]@D		IR[rt]@D			
EXT		IR[i16]@D	IR[i16]@D			IR[i16]@D				
NPC	PC4						PC4@D	PC4@D	PC4@D	
	I26						IR[i16]@D	IR[i26]@D	IR[i26]@D	
CMP	D1						RF.RD1			
	D2						RF.RD2			
E级	V1	RF.RD1	RF.RD1	RF.RD1	RF.RD1	RF.RD1				
	V2		RF.RD2	RF.RD2	RF.RD2					
	A1	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D				
	A2		IR[rt]@D	IR[rt]@D	IR[rt]@D					
	A3	IR[rt]@D		IR[rd]@D	IR[rd]@D	IR[rt]@D			31	
	E32	EXT	EXT							
	PC4								PC4@D	
ALU	A	V1@E	V1@E	V1@E	V1@E	V1@E				
	B	E32@E	E32@E	V2@E	V2@E	E32@E				
M级	V2		V2@E							
	A2		RD2@E							
	AO	ALU	ALU	ALU	ALU	ALU				
	A3	A3@E		A3@E	A3@E	A3@E				
	PC4								PC4@E	
DM	A	AO@M	AO@M	AO@M	AO@M	AO@M				
	WD		RD2@M							
W级	A3	A3@M		A3@M	A3@M	A3@M				
	PC4								PC4@M	
	AO	AO@M		AO@M	AO@M	AO@M				
	DR	DM								
RF	A3	A3@W		A3@W	A3@W	A3@W				
	WD	DR@W		AO@W	AO@W	AO@W			PC4@W	

完全按照高小鹏老师的 PPT 进行，同时借鉴 P4 的数据通路。



改动 1：把 branch 前移到第二级，和 j 类指令一起决定 nextPC

改动 2： jr 指令把 GPR[rs]写入 PC 在 D 级完成

改动 3：转发-处理 ALU 的两个输入,更新寄存器的值

F_RS3: 1:4 级 A3=RS3 2（更高优先级）：5 级 A3=RS

F_RT3: 1:4 级 A3=RT3 2（更高优先级）：5 级 A3=RT3

改动 4：load 指令的导致的暂停（普通 lw）

Tnew：供给者的最早时间 Tuse：需求者的最晚时间

Tnew>R 类指令的 Tuse

改动 5：处理 sw 在第四级的转发问题

改动 6：简单处理 beq 指令

进行简单的运行 调 bug

改动 7：处理 J 类指令和 beq 的暂停转发问题

1：加一个 4 级 ALUout 到 beq 的转发/5 级 WD_5 到 beq 的转发，更新关于 beq 指令的暂停

改动 8：jal/jalr 的回写

Debug

改动 9：刚知道有延迟槽这玩意（没看懂，就嗯加了一个 PC+8

改动 10：jal/jalr 对 31 号寄存器的转发

备注：当前 aluin1 和 aluin2 的 mux 组合：

1：F_RT3/F_RS3 的冒险

2：jal 的冒险

3：aluin2 的 sel

被一个 sel 电线卡了半天 第一次通过课上 OJ

改动 11：同周期读和写一个寄存器

改动 12：完善 jal/jalr，针对 jal 在 5 级 jr 在 2 级时的处理

改动 13： jal 和 R 类指令转发的先后问题 发现 jal 和转发分开再处理会极度麻烦 所以直接用 AT 法。。

冒险表格：

	Tnew	Tusers	Tusert
add	1	1	1
sub	1	1	1
ori	1	1	3
lw	2	1	3
sw		1	2
beq		0	0
lui	1	1	3
j			
jal			
jr			
nop			

这里把 i 型指令都调到 3 了 防止误判暂停

最后的项目总结：基本的模块都是对应相应的功能 主要是根据 5 个转发位

点冒险来建立对应的 MUX，我这里有的地方用了五级流水线的 12345，而不是 FDEMW，其实无区别，针对 jal/jalr 的值、R 类指令 ALUOUT 值的转发，解决了先后顺序的问题，然后根据 Tnew 和 Tuse 来触发暂停，暂停操作包括封锁 PC 的变化、IF/ID(1 级)的变化、清空 ID/EX(2 级)。

然后延迟槽的行为？有点迷惑

暂停最苛刻的情况 lw 接 beq 暂停两次

MULT、 MULTU、 DIV、 DIVU、MFHI、MFLO、MTHI、MTLO}

P6 接 P5 今天做简单的计算指令 ADD SUB SLL SLLV SRL SRA SRLV SRAV AND OR XOR NOR ADDI ADDIU ANDI XORI SLT SLTI SLTIU SLTU

1.增加 BED BEDop LED 扩展 DM

BEDop: 0:SW 1:SH 2:SB LB LH LBU

b) sw 指令：GPR[rt]写入对应的字。

地址[1:0]	BE[3:0]	用途
XX	1111	WD[31:24]写入 byte3 WD[23:16]写入 byte2 WD[15:8]写入 byte1 WD[7:0]写入 byte0

c) sh 指令：GPR[rt]$_{15:0}$写入对应的半字。

地址[1:0]	BE[3:0]	用途
0X	0011	WD[15:8]写入 byte1 WD[7:0]写入 byte0
1X	1100	WD[15:8]写入 byte3 WD[7:0]写入 byte2

d) sb 指令：GPR[rt]$_{7:0}$写入对应的字节。

地址[1:0]	BE[3:0]	用途
00	0001	WD[7:0]写入 byte0
01	0010	WD[7:0]写入 byte1
10	0100	WD[7:0]写入 byte2
11	1000	WD[7:0]写入 byte3

参考的接口定义如下：

信号名	方向	描述
A[1:0]	I	最低 2 位地址。
Din[31:0]	I	输入 32 位数据
Op[2:0]	I	数据扩展控制码。 000：无扩展 001： 无符号字节数据扩展 010：符号字节数据扩展 011：无符号半字数据扩展 100：符号半字数据扩展
DOut[31:0]	O	扩展后的 32 位数据

BE 模块和 LE 模块

增加判断溢出的操作（等待完善）

二、测试方案

三、思考题

- 为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

  因为计算乘除法需要模拟延迟，这和其他指令是是不同的，需要单独处理乘除法。

  因为 32 位的乘除法结果为 64 位，需要两个 32 位寄存器保存和提取。

- 参照你对延迟槽的理解，试解释“乘除槽”。

  乘除槽的出现是由于乘除计算指令有延迟，出现冲突时不得不暂停而形成的。

- 举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑 C 语言中字符串的情况）

  在指令中 sb/sh/lb/lh 指令出现的比例更多的时候，，按字节访问会更有优势

- 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

  如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证**覆盖**了所有需要测试的情况；如果你是**完全随机**生成的测试样例，请思考完全随机

的测试程序有何不足之处；如果你在生成测试样例时采用了**特殊的策略**，比如构造连续数据冒险序列，请你描述一下你使用的策略如何**结合了随机性**达到强测的效果。

此思考题请同学们结合自己测试 CPU 使用的具体手段，按照自己的实际情况进行回答

见各个测试文件

- 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

1.增加 ALU 的译码信号，把各种 R 类指令抽象成一种数据通路。

2.把各种 b 类跳转指令规范在一个 mux 内，使各种跳转的条件一目了然

3.非常不抽象的把每一个乘除指令都放进了乘除单元里，简单粗暴，思路清晰。