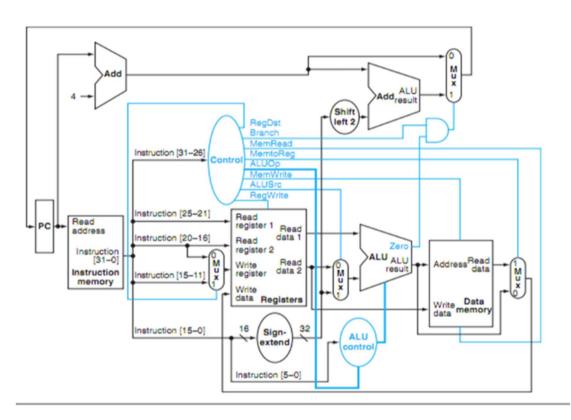
实验三 MIPS 单周期 CPU 设计

1. 实验目的

- 1. 掌握单周期 CPU 数据通路的原理与设计方法;
- 2. 掌握 MIPS 指令与 CPU 的关系
- 3. 掌握存储器映像 IO 接口设计原理
- 4. 学会观察与分析指令执行状态

2. 数据通路结构

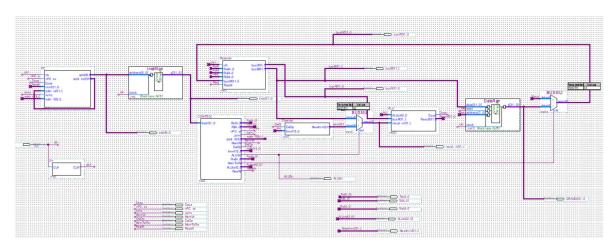
数据通路结构主要参考了老师 ppt 的这张图片:



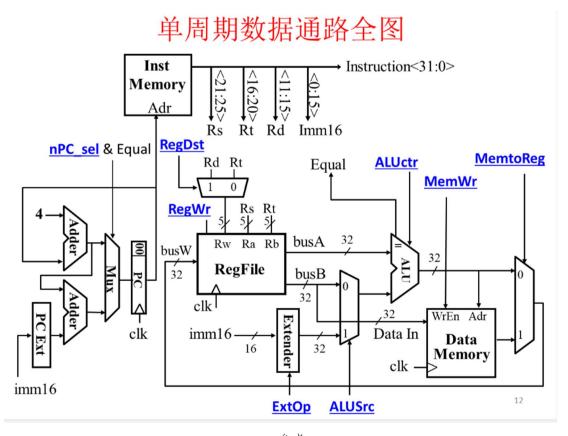
稍加修改的就是,我把整个 32 位的 instruction 都输入进 control 里头,直接在 control 中分出了 rs, rt, rw, 以及可能需要的立即数

3. 顶层模块以及各个模块功能

顶层图:



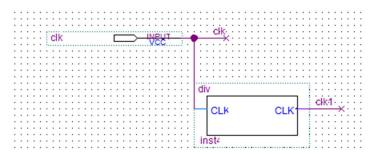
顶层



ppt 参考

ppt 参考图中的命名都很清晰,基本上输入输出的命名都是直接用 ppt 里头的命名

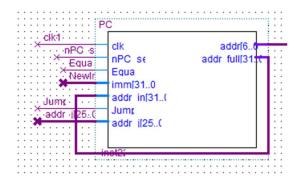
● 分频模块:



使用分频的想法很简单,就是想让 rom 和 ram 的时钟远快于 cpu 的时钟,这样 cpu 指令一发出去,两个存储模块马上就能解析出需要的数据了。

设计直接套用上次实验的就好了, 很简单没什么说的

● PC 计数器模块



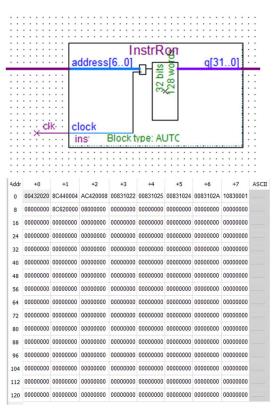
借鉴了老师的设计, pc 更新在下降沿, 别的则在上升沿, 这样也保证了能够支撑一定的保持时间。

分三级, jump 当然最优先, 确定没有 jump 的情况下就看 nPC_sel 和 Equal, 两个信号一起 决定了是否要用立即数去完成跳转, 最后的情况才是普通的+4

对应的分别是: 直接到 addr_j, pc+4+4* imm, pc+4

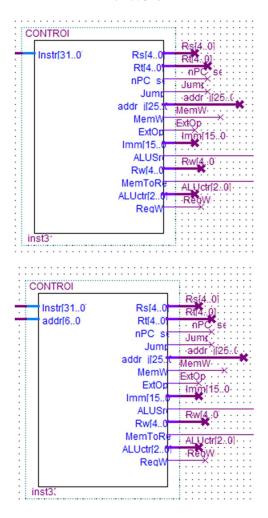
这个模块也不难

InstrRom



按照附件的来就好了, mif 文件就是把文档要求的 mips 代码转为机器码, 在附件最后都有

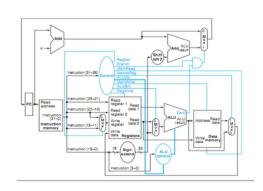
● CONTROL 控制模块



一开始我直接没弄输入时钟了,因为我想的是当 instr 变动的时候就运行 control,并且仿真的结果也很好。

但是写报告的时候一想,要是两条指令是一模一样的岂不是就只算了一条,于是我又加了 addr 地址,因为 pc 输出的不可能两次是一样的,所以 addr 变动的时候只可能是从当前指令 到下一条指令了。

Control 模块说简单也不简单, 但也算不上复杂, 不过确实是这里面最需要考虑的一个模块。



主要还是参考了这张图,通过解析 Instr 指令得到 Rs, Rt, nPC_sel, Jump, addr_j, MemWr, ExtOp, Imm;, ALUSrc, Rw, MemToReg, ALUctr, RegWr 这些输出量的值主要参考了 ppt 的表格:

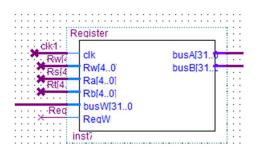
See func Appendix A op		10 0000	10 0010	We Don't Care :-)						
		00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010		
		add	sub	ori	lw	sw	beq	jump		
	RegDst	1	1	0	0	х	х	x		
	ALUSrc	0	0	1	1	1	0	x		
	MemtoReg	0	0	0	1	х	х	х		
	RegWrite	1	1	1	1	0	0	0		
	MemWrite	0	0	0	0	1	0	0		
	nPCsel	0	0	0	0	0	1	?		
	Jump	0	0	0	0	0	0	1		
	ExtOp	х	х	0	1	1	х	х		
	ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract	x		

同样的方法把 slt, and 补上即可

之后得到了这张表,我是对每个输出量,也就是对行分析,通过 if-else 以及 case 语句去完成这个模块。

注: 我设计的控制模块, Rd 在这里不需要, 这是因为我直接用控制器选好了 Write register 是 Rt 还是 Rd,同理 RegDst 也可以省略

Registers



在 registers 内需先初始化 32 个寄存器所存的值,之后就根据 regwr 选择 busa 和 busb 输出即可,无难度

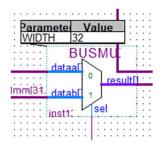
Extender



扩展器,根据 extop 选择是符号扩展还是零扩展,特别的,如果是正数,则符号扩展和零扩

展一样。这只需要在 imm16 位基础上补高位即可, 很简单。

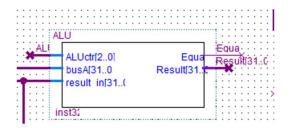
BUSMUX



我用了两个 quartus 库自带的二选一选通器,分别用来选择:

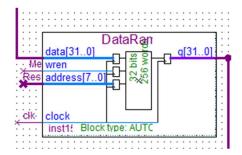
- 1. ALU 的第二个输入源是立即数还是寄存器的内容
- 2. 要写回的是主存(DataRam)的内容还是从 ALU 中运算得出的内容

• ALU



这个模块也简单,根据 ALUctr 判断这条指令要完成什么内容, case 一下然后将对应的两个输入源运算即可。

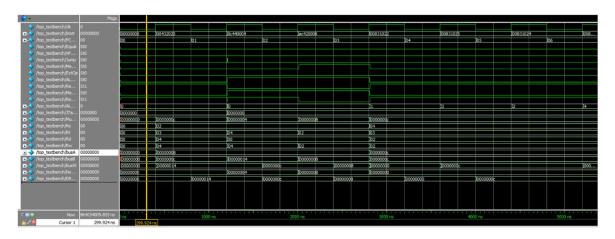
DATARam



和 rom 设置类似,用一个 mif 文件自行定义内容

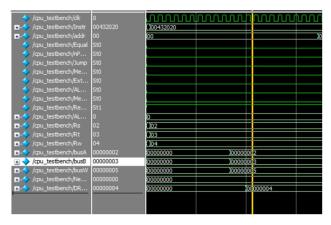
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0	4	8	12	16	20	24	28	
8	32	36	40	44	48	52	56	60	\$(,048<
16	0	4	8	12	16	20	24	28	
24	32	36	40	44	48	52	56	60	\$(,048<
32	0	4	8	12	16	20	24	28	
40	32	36	40	44	48	52	56	60	\$(,048<
48	0	4	8	12	16	20	24	28	
56	32	36	40	44	48	52	56	60	\$(,048<
64	0	4	8	12	16	20	24	28	
72	32	36	40	44	48	52	56	60	\$(,048<
80	0	4	8	12	16	20	24	28	
88	32	36	40	44	48	52	56	60	\$(,048<
96	0	4	8	12	16	20	24	28	
104	32	36	40	44	48	52	56	60	\$(,048<
112	0	4	8	12	16	20	24	28	
120	32	36	40	44	48	52	56	60	\$(,048<

4. Modelsim 仿真分析



我定义的第i个寄存器的内容就是i

这里看到, 第一条指令 add \$4,\$2,\$3 时, Rs 存的为 2, Rt 则为 3, 对应的 busW 即为 5, rw 为 4, 仿真很成功



后续波形图类似验证, 均成立

5. signalTap II 测量结果分析

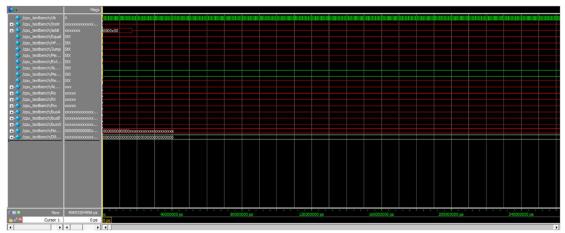
Type Alias	Name	-16 -1	5 -1	4 -1	3 -1	2 -1	1 -	-10
35	addr add	07h	(08h)	00h	(01h)	(02h)	03h	X 04h
3	■ ALUctr			1h	X 3h			
34	⊞ busW	00000	0007h	00000000h	00000004h	X 00000008h X	00000000h	00000004h
8	■ DRAMDt	00000	004h	00000000h	00000004h	00000	00000004h	
9	⊞ Instr	10830001h	08000000h	00432020h	8C440004h	AC420008h X	00831022h	00831025h
8	⊞-Rs	04h	00000004h 10h		02h			04h
3	. Rt	03h	00h	03h	(04h	(02h)		03h
OULT	. Rw	03h	(00h	04	ih)			02h

可以看到,第一条是 08000000h,为 j main,然后发现下一条便为 00432020h,为 add 指令,且对应的 rs rt rw 都是正确的;后面的验证同理可知成立。

但是可惜的是,我想验证 BusW 和 BusA BusB 之间的关系,但是当输出都放上时系统会报 引脚不够的错误,并且当 signaltap 验证,一点运行就跑了很多次,也无法确定每个寄存器的 真实存的数值了,故验证所有程序的正确与否我主要还是用的波形图去验证,signaltap 只是 作为第二个参考。

6. 总结与思考

这次实验,当我写完报告的时候,长舒一口气,同时也发现整理下来的思路特别清晰,每个模块一介绍似乎也都不难,可是就是花了我很久很久的时间去 debug 例:



这个 modelsim 仿真全是红线,看不到数据,后来琢磨,是 testbench 仿真的时候没有给(准确的说,是在 clk 出现的时候,没有得到一个有效的数据)导致的,于是我在控制模块附了一个初值。

还有很多其他零碎的 bug,就不细举了

总结与思考:

可以说这次实验很考细心, 完成了这次实验, 成就感满满, 同时也是对前面的数据通路、单周期 cpu 的一次知识梳理,帮助很大!