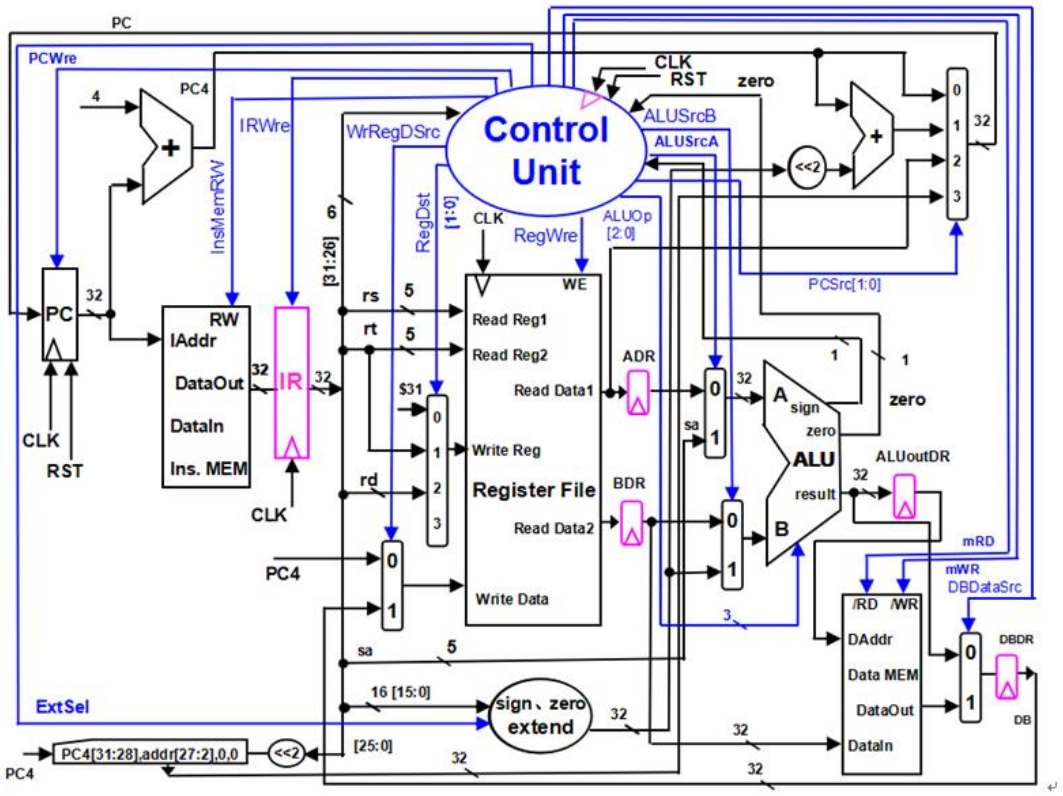
附加实验 多周期 CPU

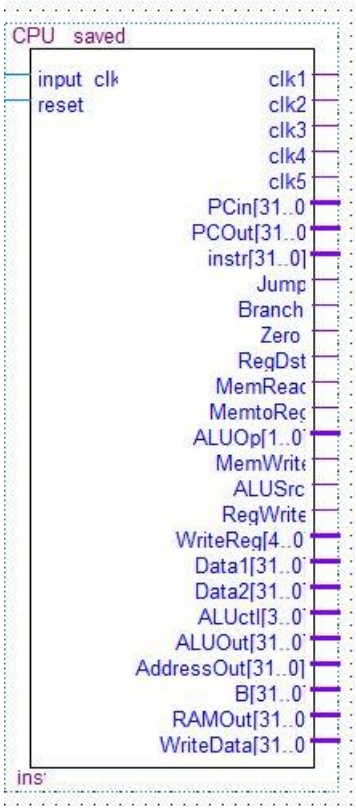
自 83 杜奇修 2018011505

一、通路绘制



二、模块分析

1.CPU 模块

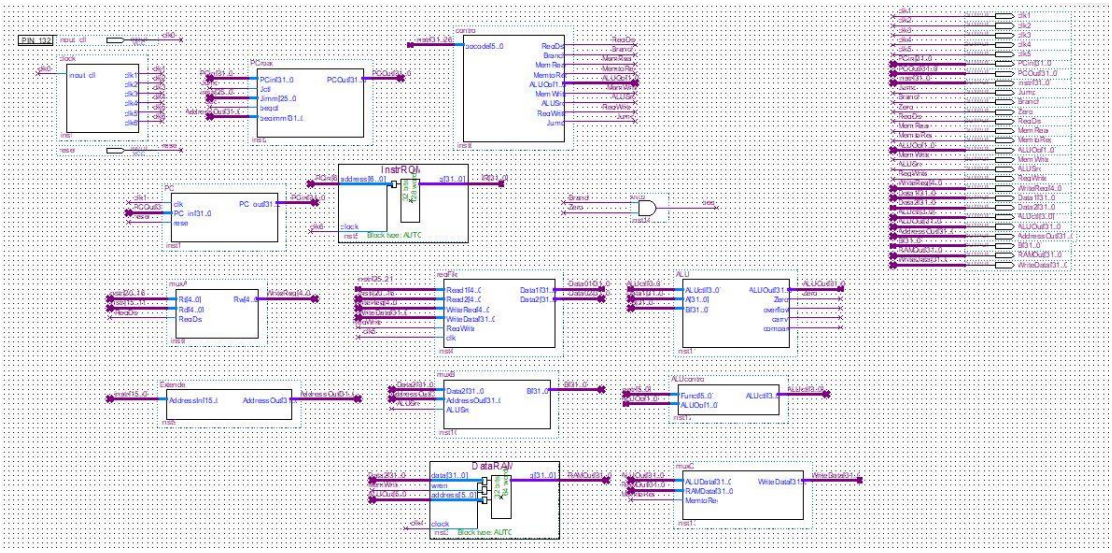


顶层模块有一个输入端$input_clk$和一个置 0 端$reset$，只需要在最开始将$reset$置零，就能保证 PC 初始值为$0x00000000$，输出端用于监视和仿真。

输出端口	功能
$clk1, clk2, clk3, clk4, clk5$	$clk1$到$clk5$依次对应多周期 CPU 的 5 个时钟信号，$clk1$上升沿控制PC更新，$clk2$上升沿控制指令寄存器的更新，$clk3$上升沿控制数据寄存器的更新，$clk4$上升沿控制$DataRAM$的读取，$clk5$上升沿控制写回时钟。
$PCin[31:0]$	PC寄存器的输出，即$PCroad$的输入。
$PCOut[31:0]$	PC寄存器的输入，即$PCroad$的输出。
$instr[31:0]$	取出的指令 2 进制表示
$Jump$	$Jump = 1$表示执行j语句，否则$Jump = 0$
$Branch$	表示是否分支，用于监测beq
$Zero$	来自ALU输出，表示运算结果是否为 0
$RegDst$	$RegDst = 1$表示写寄存器的目标寄存器号来自rd字段，$RegDst = 0$表示写寄存器的目标寄存器号来自rt字段。
$MemRead$	$MemRead = 1$表示数据存储器读使能有效，此处不使用这个控制信号。
$MemtoReg$	$MemtoReg = 1$表示写回的数据来自数据存储器，$MemtoReg = 0$表示写回的数据来自ALU
$ALUOp[1:0]$	通过输入信号的$opcode$给出，和$funct$字段结合通过$ALUcontrol$模块得到ALU的控制信号。
$MemWrite$	$MemWrite = 1$表示将写入数据输入端的数据写入到用地址指定的存储单元中去。
$ALUSrc$	$ALUSrc = 1$表示第二个ALU操作数为指令低 16 位的符号扩展，

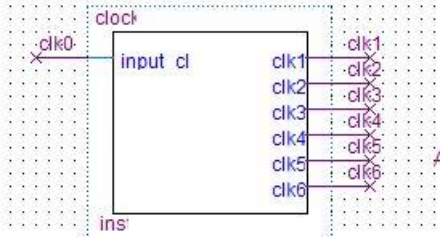
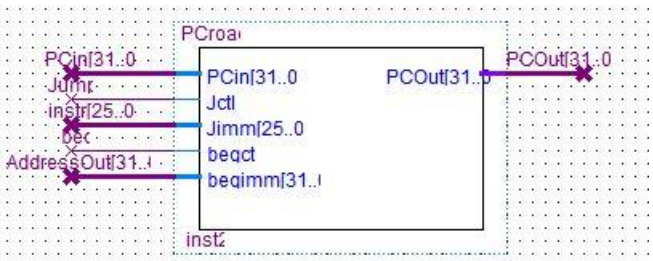
	$ALUSrc = 0$表示第二个ALU操作数为来自寄存器堆的第二个输出。
$RegWrite$	$RegWrite = 1$表示寄存器堆写使能有效。
$WriteReg[4:0]$	写寄存器堆的寄存器编号，从0~31变化。
$Data1[31:0]$	寄存器堆的第一个输出。
$Data2[31:0]$	寄存器堆的第二个输出。
$ALUctl[3:0]$	由上面的$ALUOp$指定的ALU直接控制信号。
$ALUOut[31:0]$	ALU的输出。
$AddressOut[31:0]$	符号拓展器的 32 位输出。
$B[31:0]$	$muxB$选择的ALU的第二个操作数。
$RAMOut[31:0]$	$DataRAM$的输出。
$WriteData[31:0]$	写入$DataRAM$的数据。

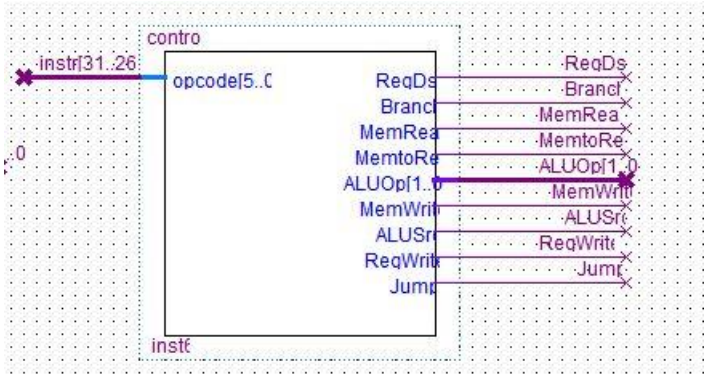
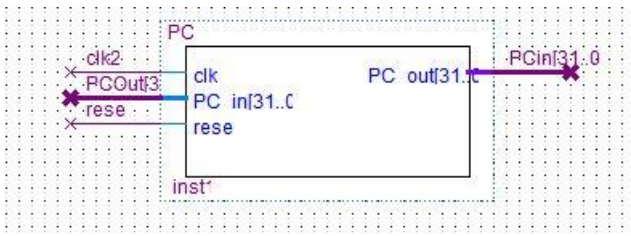
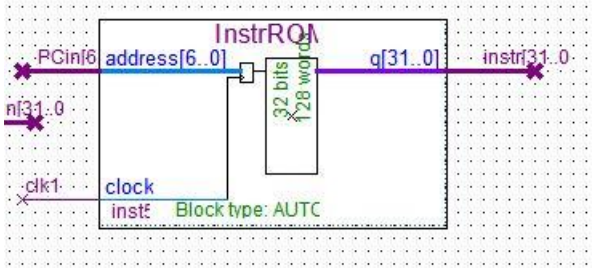
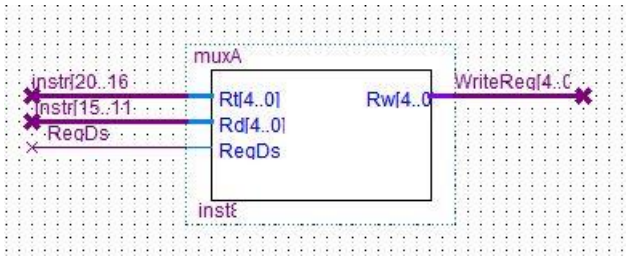
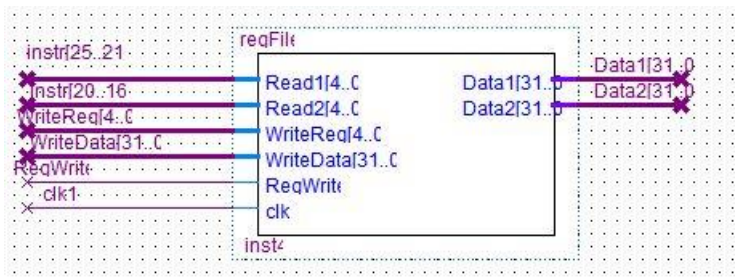
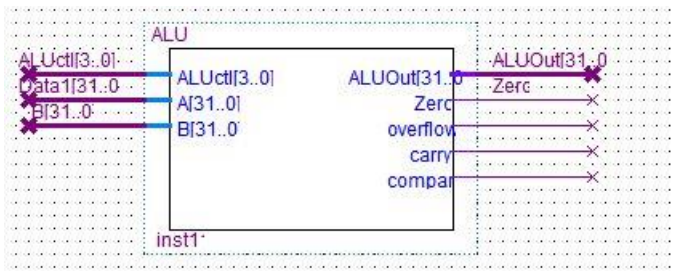
CPU 模块展开图如下：

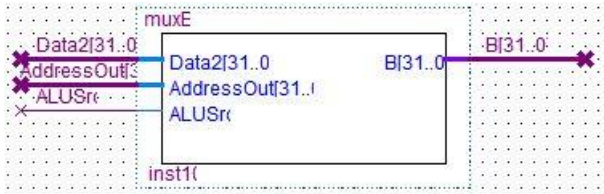
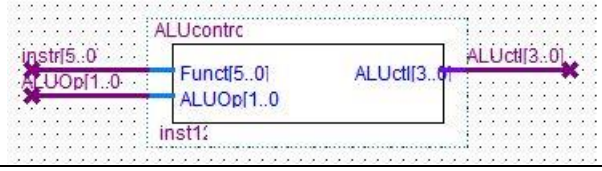
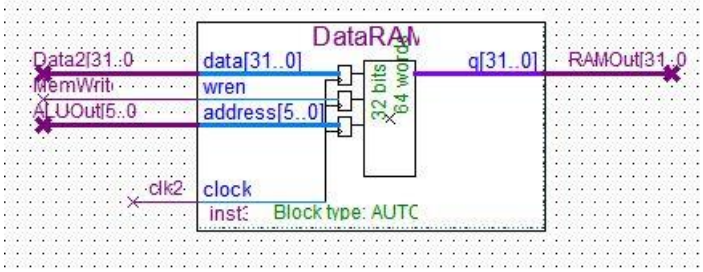
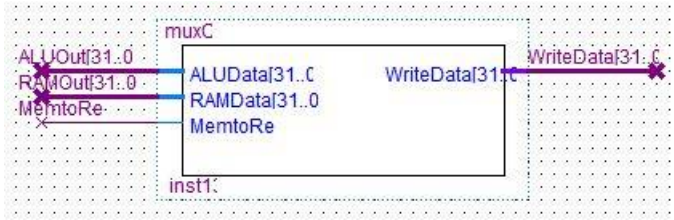
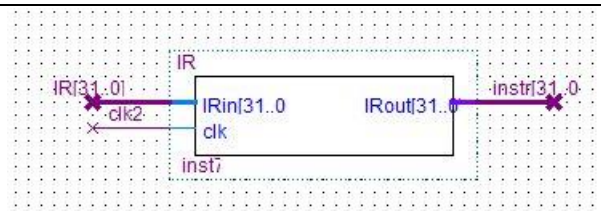


2.其他模块

A.图示

模块名	图示
$clock$	
$PCroad$	

control	
PC	
InstrROM	
muxA	
regFile	
ALU	

$Extender$	
$muxB$	
$ALUcontrol$	
$DataRAM$	
$muxC$	
IR	
DR	

B.功能简介：

$clock$	分频器，产生$clk1$,$clk2$
$PCroad$	和PC相关的数据通路，能处理PC自增和beq,j指令带来的跳转
$control$	根据$opcode$生成主要控制指令
PC	指令计数器
$InstrROM$	指令存储器
$muxA$	用于选择寄存器堆的写寄存器号

$regFile$	寄存器堆
ALU	算术逻辑单元
$Extender$	符号扩展器，进行无符号扩展
$muxB$	用于选择ALU的第二个操作数
$ALUcontrol$	用于直接控制ALU的运算方式
$DataRAM$	数据存储器，用于模拟内存
$muxC$	用于选择写回寄存器堆数据来源
IR	用于暂存指令
DR	用于暂存寄存器堆读取的数据

三、$MIPS$处理器仿真结果

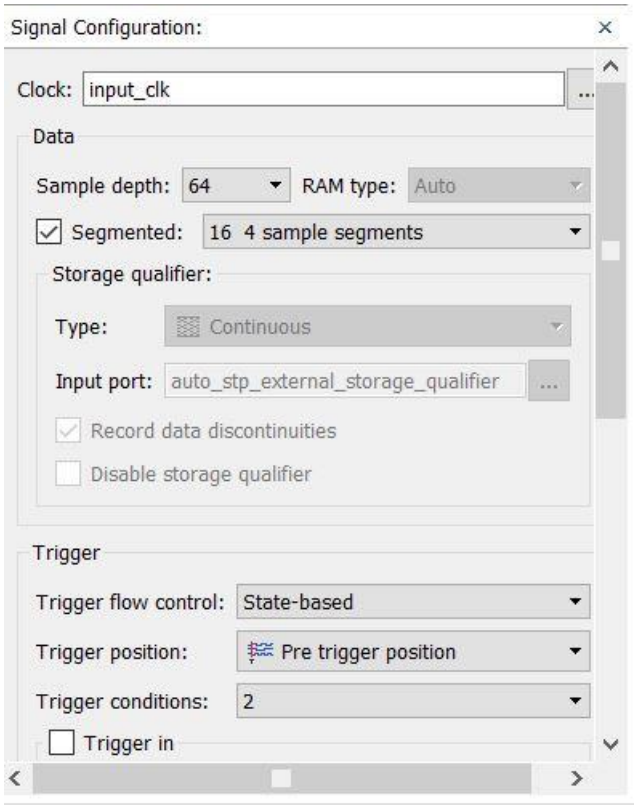
Text Segment

Bkpt	Address	Code	Basic		Source
	0x00000000	0x00432020	add $4, $2, $3	2:	add $4, $2, $3
	0x00000004	0x8c440004	lw $4, 0x00000004($2)	3:	lw $4, 4($2)
	0x00000008	0xac420008	sw $2, 0x00000008($2)	4:	sw $2, 8($2)
	0x0000000c	0x00831022	sub $2, $4, $3	5:	sub $2, $4, $3
	0x00000010	0x00831025	or $2, $4, $3	6:	or $2, $4, $3
	0x00000014	0x00831024	and $2, $4, $3	7:	and $2, $4, $3
	0x00000018	0x0083102a	slt $2, $4, $3	8:	slt $2, $4, $3
	0x0000001c	0x10830001	beq $4, $3, 0x00000001	9:	beq $4, $3, exit
	0x00000020	0x08000000	j 0x00000000	10:	j main
	0x00000024	0x8c620000	lw $2, 0x00000000($3)	12:	lw $2, 0($3)
	0x00000028	0x08000000	j 0x00000000	13:	j main

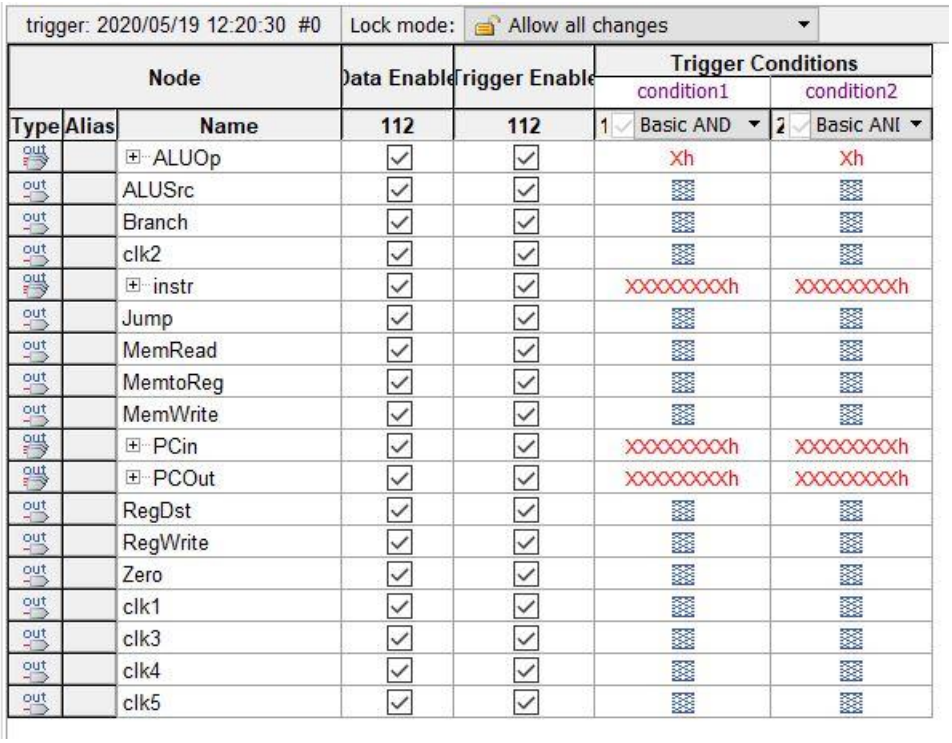
对照课本上的附录$A.10$可以得到这个结果是正确的。

四、$signaltapII$仿真结果

将采样深度设置为64，并且设定采样参数



输出端口设置为



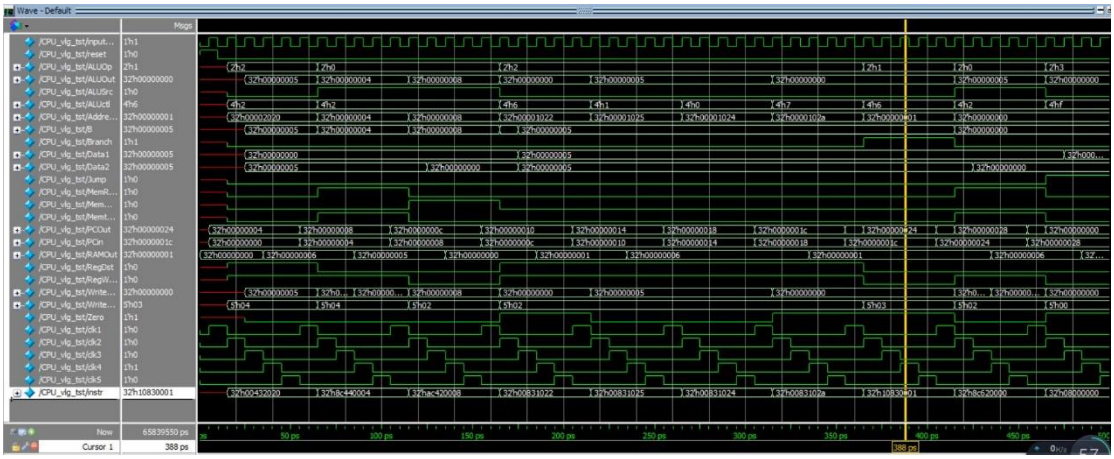
参数说明:

$clk1, clk2$	$clk1$到$clk5$依次对应多周期 CPU 的 5 个时钟信号，$clk1$上升沿控制PC更新，$clk2$上升沿控制指令寄存器的更新，$clk3$上升沿控制数据寄存器的更新，$clk4$上升沿控制$DataRAM$的读取，$clk5$上升沿控制写回时钟。
$PCin[31:0]$	PC寄存器的输出，即$PCroad$的输入。
$PCOut[31:0]$	PC寄存器的输入，即$PCroad$的输出。
$instr[31:0]$	取出的指令 2 进制表示
$Jump$	$Jump = 1$表示执行j语句，否则$Jump = 0$
$Branch$	表示是否分支，用于监测beq
$Zero$	来自ALU输出，表示运算结果是否为 0
$RegDst$	$RegDst = 1$表示写寄存器的目标寄存器号来自rd字段，$RegDst = 0$表示写寄存器的目标寄存器号来自rt字段。
$MemRead$	$MemRead = 1$表示数据存储器读使能有效，此处不使用这个控制信号。
$MemtoReg$	$MemtoReg = 1$表示写回的数据来自数据存储器，$MemtoReg = 0$表示写回的数据来自ALU
$ALUOp[1:0]$	通过输入信号的$opcode$给出，和$funct$字段结合通过$ALUcontrol$模块得到ALU的控制信号。
$MemWrite$	$MemWrite = 1$表示将写入数据输入端的数据写入到用地址指定的存储单元中去。
$ALUSrc$	$ALUSrc = 1$表示第二个ALU操作数为指令低 16 位的符号扩展，$ALUSrc = 0$表示第二个ALU操作数为来自寄存器堆的第二个输出。

最后的输出结果:

经检验，输出符合预期。

五、$modelsim$的仿真和$debug$



如图所示，实验结果符合预期，我预设了第一个周期会beq生效一次，在图中也能看到这个过程。

中途遇到了一个相当奇怪的问题，我一开始控制$InstrROM$的方式是使用$input_clk$直接控制，因为书上的多周期CPU的指令存储器模块是“只读”的，而宏生成的ROM似乎做不到这一点，因此需要一个很高的时钟来控制。但是这样的时钟产生了一个非常奇怪的结果，$PC = 32'h00000000$时，输出$instr$信号无变化，$PC = 32'h00000004$时，$instr = 32'h00432020$，也就是产生了一条指令的偏差。我通过分模块调试，发现问题出现在竞争冒险环节，由图中可以看到，在第二个时钟上升沿的时候，即指令寄存器被读取的时候，这个时候也恰好是$InstrROM$上升沿到达的时候，因此会有一次竞争冒险，符合预期。因此我增加了$clk6$，与$input_clk$反向，则恰好可以避免这种冲突。

六、总结

本次实验我学习了多周期CPU的编写，整体难度不大，增进了我对于$verilog$的使用熟练程度。