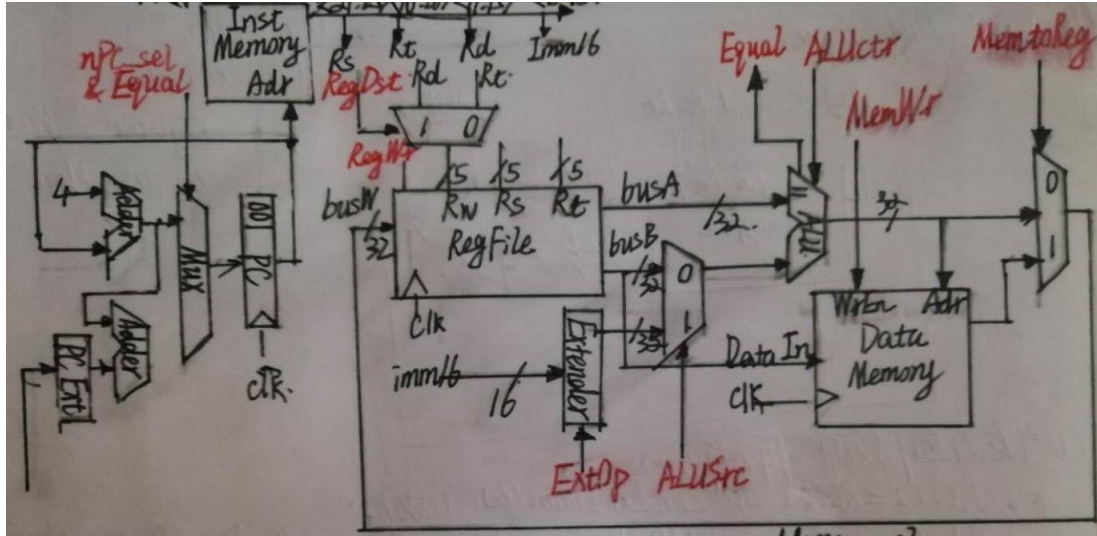


实验三 单周期 CPU

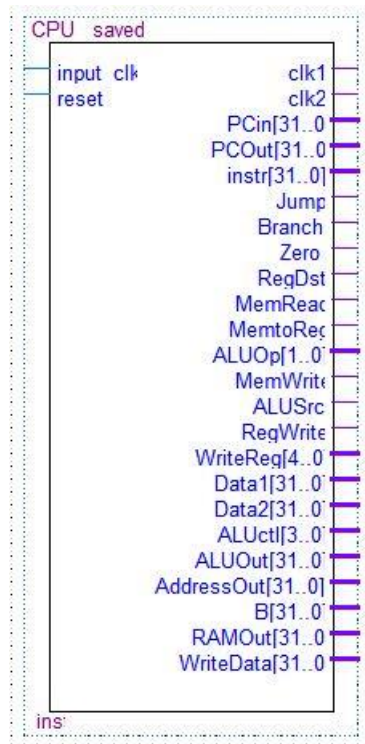
自 83 杜奇修 2018011505

一、通路绘制



二、模块分析

1. CPU 模块

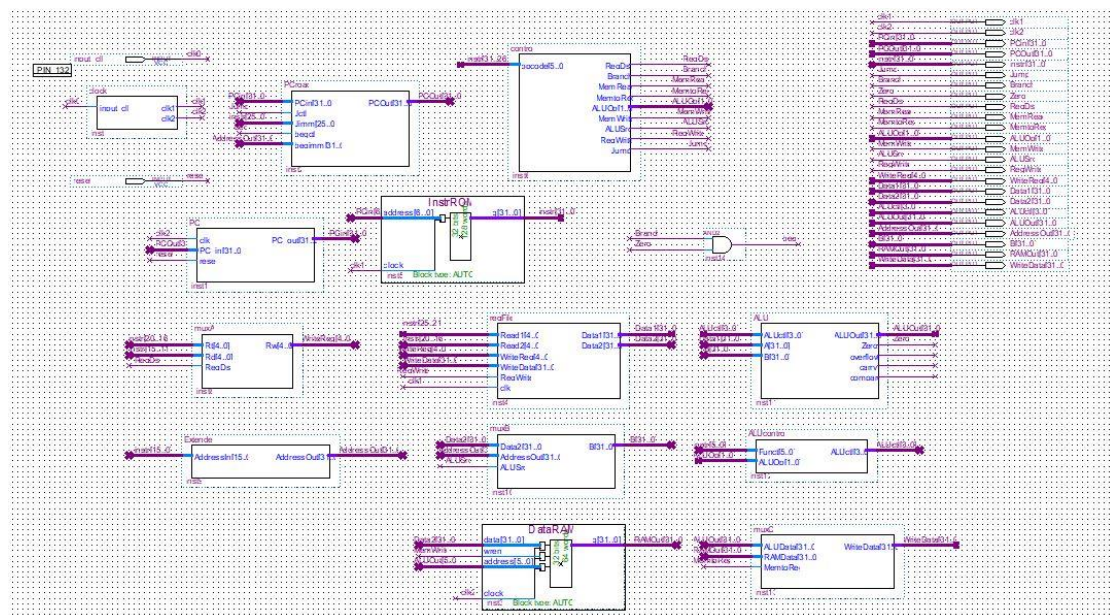


顶层模块有一个输入端`input_clk`和一个置0端`reset`，只需要在最开始将`reset`置零，就能保证PC初始值为0x00000000，输出端用于监视和仿真。

输出端口	功能
<code>clk1, clk2</code>	<code>clk1</code> 比 <code>clk2</code> 领先半个时钟周期， <code>clk1</code> 的上升沿时指令存储器读取地址，并且寄存器堆完成写回； <code>clk2</code> 的上升沿时PC完成更新并且DataRAM开始写入。

<i>PCIn</i> [31:0]	<i>PC</i> 寄存器的输出，即 <i>PCread</i> 的输入。
<i>PCOut</i> [31:0]	<i>PC</i> 寄存器的输入，即 <i>PCread</i> 的输出。
<i>instr</i> [31:0]	取出的指令 2 进制表示
<i>Jump</i>	<i>Jump</i> = 1表示执行 <i>j</i> 语句，否则 <i>Jump</i> = 0
<i>Branch</i>	表示是否分支，用于监测 <i>beq</i>
<i>Zero</i>	来自 <i>ALU</i> 输出，表示运算结果是否为 0
<i>RegDst</i>	<i>RegDst</i> = 1表示写寄存器的目标寄存器号来自 <i>rd</i> 字段， <i>RegDst</i> = 0表示写寄存器的目标寄存器号来自 <i>rt</i> 字段。
<i>MemRead</i>	<i>MemRead</i> = 1表示数据存储器读使能有效，此处不使用这个控制信号。
<i>MemtoReg</i>	<i>MemtoReg</i> = 1表示写回的数据来自数据存储器， <i>MemtoReg</i> = 0表示写回的数据来自 <i>ALU</i>
<i>ALUOp</i> [1:0]	通过输入信号的 <i>opcode</i> 给出，和 <i>funct</i> 字段结合通过 <i>ALUcontrol</i> 模块得到 <i>ALU</i> 的控制信号。
<i>MemWrite</i>	<i>MemWrite</i> = 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。
<i>ALUSrc</i>	<i>ALUSrc</i> = 1表示第二个 <i>ALU</i> 操作数为指令低 16 位的符号扩展， <i>ALUSrc</i> = 0表示第二个 <i>ALU</i> 操作数为来自寄存器堆的第二个输出。
<i>RegWrite</i>	<i>RegWrite</i> = 1表示寄存器堆写使能有效。
<i>WriteReg</i> [4:0]	写寄存器堆的寄存器编号，从0~31变化。
<i>Data1</i> [31:0]	寄存器堆的第一个输出。
<i>Data2</i> [31:0]	寄存器堆的第二个输出。
<i>ALUctl</i> [3:0]	由上面的 <i>ALUOp</i> 指定的 <i>ALU</i> 直接控制信号。
<i>ALUOut</i> [31:0]	<i>ALU</i> 的输出。
<i>AddressOut</i> [31:0]	符号拓展器的 32 位输出。
<i>B</i> [31:0]	<i>muxB</i> 选择的 <i>ALU</i> 的第二个操作数。
<i>RAMOut</i> [31:0]	<i>DataRAM</i> 的输出。
<i>WriteData</i> [31:0]	写入 <i>DataRAM</i> 的数据。

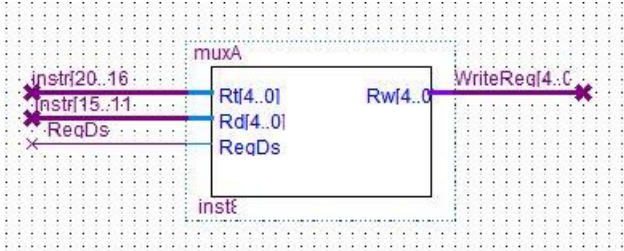
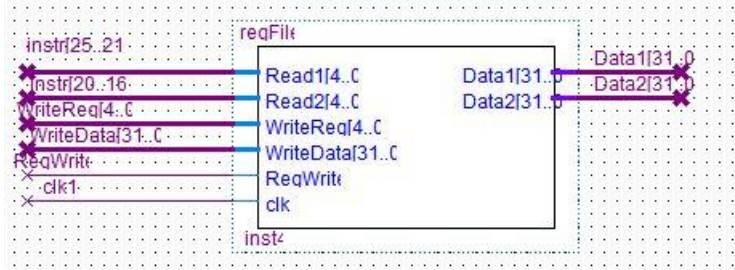
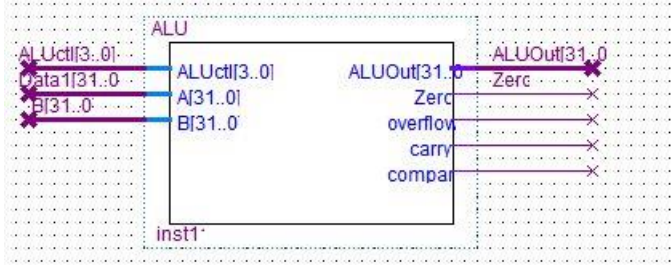

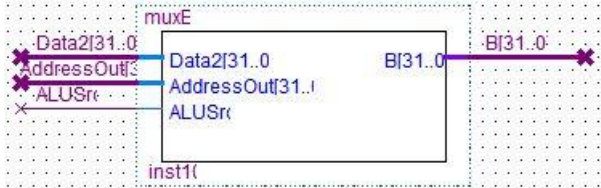
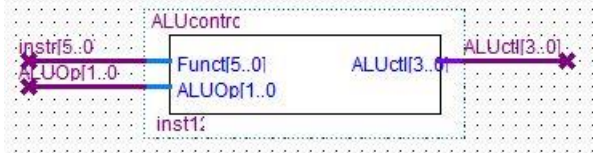
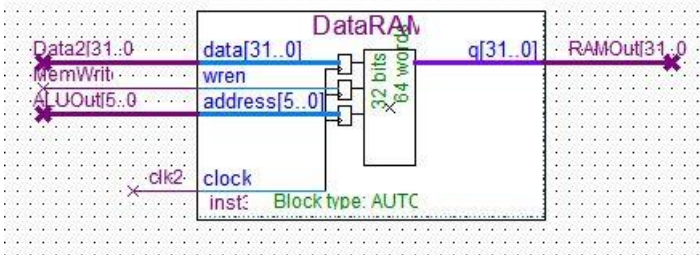
CPU 模块展开图如下：

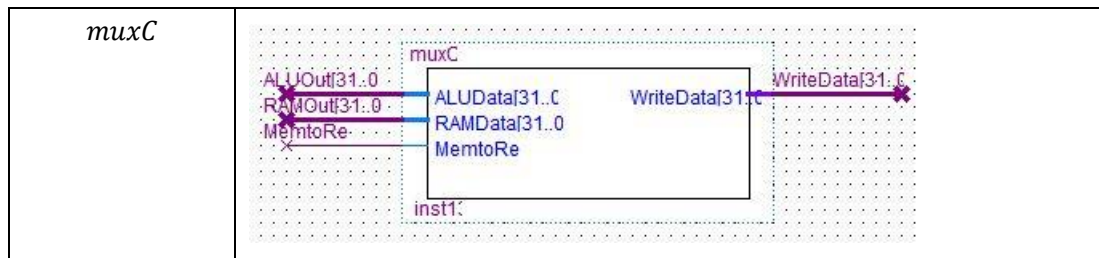


2.其他模块

A.图示

模块名	图示
<i>clock</i>	
<i>PCroad</i>	
<i>control</i>	
<i>PC</i>	
<i>InstrROM</i>	

<i>muxA</i>	 <p>The muxA block has three inputs: <code>instr[20..16]</code>, <code>instr[15..11]</code>, and <code>ReqDs</code>. It has two internal signals: <code>Rt[4..0]</code> and <code>Rd[4..0]</code>. The output is <code>WriteReg[4..0]</code>. The block is labeled <code>inst0</code>.</p>
<i>regFile</i>	 <p>The regFile block has multiple inputs: <code>instr[25..21]</code>, <code>instr[20..16]</code>, <code>WriteReg[4..0]</code>, <code>WriteData[31..0]</code>, <code>ReqWrite</code>, and <code>clk1</code>. It has two internal signals: <code>Read1[4..0]</code> and <code>Read2[4..0]</code>. The outputs are <code>Data1[31..0]</code> and <code>Data2[31..0]</code>. The block is labeled <code>inst4</code>.</p>
<i>ALU</i>	 <p>The ALU block has three inputs: <code>ALUctl[3..0]</code>, <code>Data1[31..0]</code>, and <code>B[31..0]</code>. It has two internal signals: <code>A[31..0]</code> and <code>BI[31..0]</code>. The output is <code>ALUOut[31..0]</code>. The block is labeled <code>inst1</code>.</p>
<i>Extender</i>	 <p>The Extender block has one input: <code>instr[15..0]</code>. It has one internal signal: <code>AddressIn[15..0]</code>. The output is <code>AddressOut[31..0]</code>. The block is labeled <code>inst5</code>.</p>
<i>muxB</i>	 <p>The muxE block has three inputs: <code>Data2[31..0]</code>, <code>AddressOut[31..0]</code>, and <code>ALUSrc</code>. It has two internal signals: <code>Data2[31..0]</code> and <code>AddressOut[31..0]</code>. The output is <code>BI[31..0]</code>. The block is labeled <code>inst10</code>.</p>
<i>ALUcontrol</i>	 <p>The ALUcontr block has two inputs: <code>instr[5..0]</code> and <code>ALUOp[1..0]</code>. It has two internal signals: <code>Func[5..0]</code> and <code>ALUOp[1..0]</code>. The output is <code>ALUctl[3..0]</code>. The block is labeled <code>inst12</code>.</p>
<i>DataRAM</i>	 <p>The DataRAM block has three inputs: <code>Data2[31..0]</code>, <code>MemWrit</code>, and <code>ALUOut[5..0]</code>. It has two internal signals: <code>data[31..0]</code> and <code>address[5..0]</code>. The output is <code>q[31..0]</code>. The block is labeled <code>inst3</code> and has a block type of <code>AUTC</code>.</p>



B.功能简介:

<i>clock</i>	分频器，产生 <i>clk1,clk2</i>
<i>PCroad</i>	和 <i>PC</i> 相关的数据通路，能处理 <i>PC</i> 自增和 <i>beq,j</i> 指令带来的跳转
<i>control</i>	根据 <i>opcode</i> 生成主要控制指令
<i>PC</i>	指令计数器
<i>InstrROM</i>	指令存储器
<i>muxA</i>	用于选择寄存器堆的写寄存器号
<i>regFile</i>	寄存器堆
<i>ALU</i>	算术逻辑单元
<i>Extender</i>	符号扩展器，进行无符号扩展
<i>muxB</i>	用于选择 <i>ALU</i> 的第二个操作数
<i>ALUcontrol</i>	用于直接控制 <i>ALU</i> 的运算方式
<i>DataRAM</i>	数据存储器，用于模拟内存
<i>muxC</i>	用于选择写回寄存器堆数据来源

三、MIPS处理器仿真结果

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00000000	0x00432020	add \$4,\$2,\$3	2: add \$4,\$2,\$3
<input type="checkbox"/>	0x00000004	0x8c440004	lw \$4,0x00000004(\$2)	3: lw \$4,4(\$2)
<input type="checkbox"/>	0x00000008	0xac420008	sw \$2,0x00000008(\$2)	4: sw \$2,8(\$2)
<input type="checkbox"/>	0x0000000c	0x0831022	sub \$2,\$4,\$3	5: sub \$2,\$4,\$3
<input type="checkbox"/>	0x00000010	0x0831025	or \$2,\$4,\$3	6: or \$2,\$4,\$3
<input type="checkbox"/>	0x00000014	0x0831024	and \$2,\$4,\$3	7: and \$2,\$4,\$3
<input type="checkbox"/>	0x00000018	0x083102a	slt \$2,\$4,\$3	8: slt \$2,\$4,\$3
<input type="checkbox"/>	0x0000001c	0x10830001	beq \$4,\$3,0x00000001	9: beq \$4,\$3,exit
<input type="checkbox"/>	0x00000020	0x08000000	j 0x00000000	10: j main
<input type="checkbox"/>	0x00000024	0x8c620000	lw \$2,0x00000000(\$3)	12: lw \$2,0(\$3)
<input type="checkbox"/>	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、signal tap II仿真结果

将采样深度设置为64，并且设定采样参数

Signal Configuration:

Clock: input_clk

Data

Sample depth: 64 RAM type: Auto

☒ Segmented: 64 1 sample segments

Storage qualifier:

Type: Continuous

Input port: auto_stp_external_storage_qualifier

☒ Record data discontinuities

☐ Disable storage qualifier

Trigger

Trigger flow control: Sequential

Trigger position: Pre trigger position

Trigger conditions: 1

☐ Trigger in

输出端口设置为

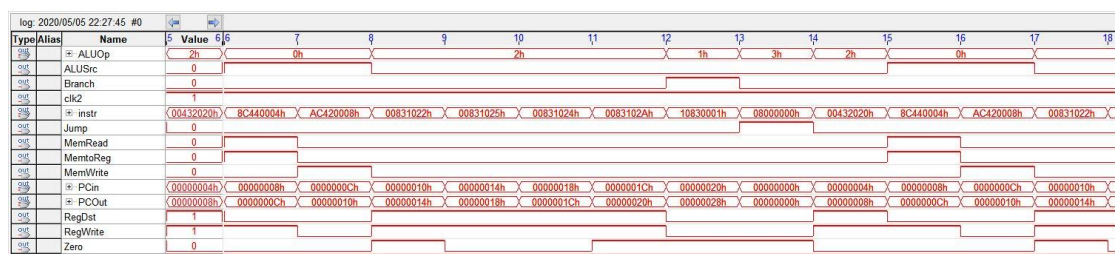
Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	109	109	Seq Basic AND
out		ALUOp	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Xh
out		ALUSrc	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
out		Branch	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
out		clk2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
out		instr	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
out		Jump	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
out		MemRead	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
out		MemtoReg	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
out		MemWrite	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
out		PCin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
out		PCOut	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
out		RegDst	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
out		RegWrite	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
out		Zero	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
out		clk1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

参数说明:

<i>clk1, clk2</i>	<i>clk1</i> 比 <i>clk2</i> 领先半个时钟周期， <i>clk1</i> 的上升沿时指令存储器读取地址，并且寄存器堆完成写回； <i>clk2</i> 的上升沿时 <i>PC</i> 完成更新并且 <i>DataRAM</i> 开始写入。
<i>PCin</i> [31:0]	<i>PC</i> 寄存器的输出，即 <i>PCread</i> 的输入。
<i>PCOut</i> [31:0]	<i>PC</i> 寄存器的输入，即 <i>PCread</i> 的输出。
<i>instr</i> [31:0]	取出的指令 2 进制表示
<i>Jump</i>	<i>Jump</i> = 1表示执行 <i>j</i> 语句，否则 <i>Jump</i> = 0

<i>Branch</i>	表示是否分支，用于监测 <i>beq</i>
<i>Zero</i>	来自 <i>ALU</i> 输出，表示运算结果是否为 0
<i>RegDst</i>	<i>RegDst</i> = 1表示写寄存器的目标寄存器号来自 <i>rd</i> 字段， <i>RegDst</i> = 0表示写寄存器的目标寄存器号来自 <i>rt</i> 字段。
<i>MemRead</i>	<i>MemRead</i> = 1表示数据存储器读使能有效，此处不使用这个控制信号。
<i>MemtoReg</i>	<i>MemtoReg</i> = 1表示写回的数据来自数据存储器， <i>MemtoReg</i> = 0表示写回的数据来自 <i>ALU</i>
<i>ALUOp</i> [1:0]	通过输入信号的 <i>opcode</i> 给出，和 <i>funct</i> 字段结合通过 <i>ALUcontrol</i> 模块得到 <i>ALU</i> 的控制信号。
<i>MemWrite</i>	<i>MemWrite</i> = 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。
<i>ALUSrc</i>	<i>ALUSrc</i> = 1表示第二个 <i>ALU</i> 操作数为指令低 16 位的符号扩展， <i>ALUSrc</i> = 0表示第二个 <i>ALU</i> 操作数为来自寄存器堆的第二个输出。

最后的输出结果：

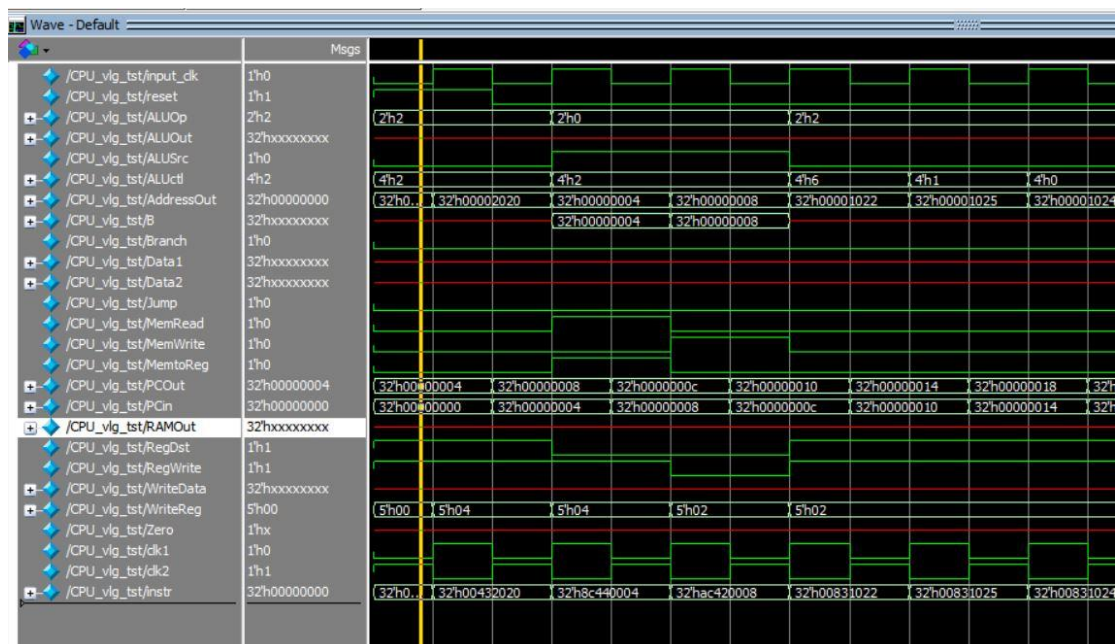


经检验，输出符合预期。

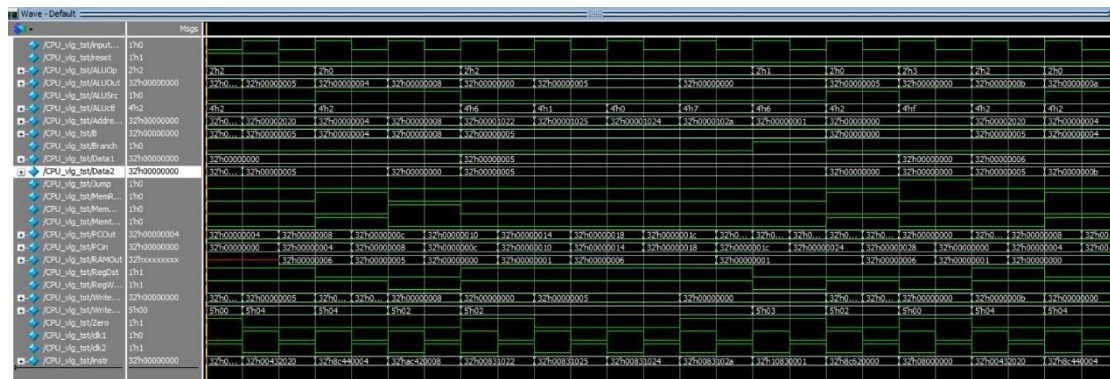
五、modelsim的仿真和debug

本次实验我的运气相对不错，在总线连接上没有遇到太大麻烦；对于所有的模块，我均进行了*modelsim*仿真，确定了所有模块均工作正常才开始整体仿真。

最开始我输出的仿真波形如下，可以看到，由于输入的数据没有确定，很多东西都无法确定。



然后我在网上寻找了对于寄存器堆的赋值语句，找到了`initial`语句，进行了仿真。我让\$3和\$4相等，可以看到，此处出现了`beq`语句的跳转。这也验证了正确性。



除此之外还有实验开始的一个小bug，一开始我在.v文件中用always语句无论如何也仿真不出信号，然后改成了always@(*)语句后就能看到了。这可能是verilog的一种语法。网上语法也基本是always@(*)，在此记下这个知识点。

六、总结

本次实验我学习了单周期CPU的编写，在这个任务下我真正理解了`signaltap`的各项参数，并且更深入地了解了`verilog`的编写和宏的使用，这很大程度上提高了我的动手能力。整体实验比较顺利，没有出现太多错误，感谢秦老师和同学全程的帮助。期待下一次实验。