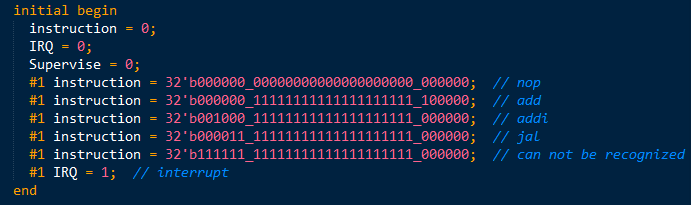
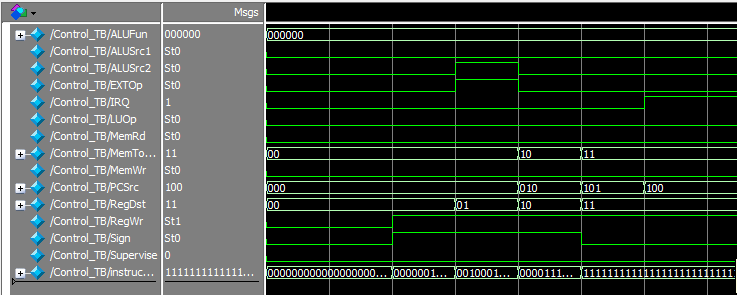
1.Control功能仿真

由于不存在时序逻辑电路，所以仿真的时候不需要时钟，只需要在一定的时间间隔内取一条不同的instruction就可以，我们选取的几条测试指令如下：



仿真的波形如下：



从波形中可以看出如下命令的控制信号：

（1）instruction = 32'b000000\_00000000000000000000\_000000; // nop

PCSrc = 000; RegDst = 00; RegWr = 0; ALUSrc1 = 0; ALUSrc2 = 0;

ALUFun = 000000; Sign = 0; MemWr = 0; MemRd = 0; MemToReg = 00;

EXTOp = 0; LUOp = 0;

（2）instruction = 32'b000000\_11111111111111111111\_000000; // add

PCSrc = 000; RegDst = 00; RegWr = 1; ALUSrc1 = 0; ALUSrc2 = 0;

ALUFun = 000000; Sign = 1; MemWr = 0; MemRd = 0; MemToReg = 00;

EXTOp = 0; LUOp = 0;

（3）instruction = 32'b001000\_11111111111111111111\_000000; // addi

PCSrc = 000; RegDst = 01; RegWr = 1; ALUSrc1 = 0; ALUSrc2 = 1;

ALUFun = 000000; Sign = 1; MemWr = 0; MemRd = 0; MemToReg = 00;

EXTOp = 1; LUOp = 0;

（4）instruction = 32'b000011\_11111111111111111111\_000000; // jal

PCSrc = 010; RegDst = 10; RegWr = 1; ALUSrc1 = 0; ALUSrc2 = 0;

ALUFun = 000000; Sign = 1; MemWr = 0; MemRd = 0; MemToReg = 10;

EXTOp = 0; LUOp = 0;

（5）instruction = 32'b111111\_11111111111111111111\_000000; // can not be recognized

PCSrc = 101; RegDst = 11; RegWr = 1; ALUSrc1 = 0; ALUSrc2 = 0;

ALUFun = 000000; Sign = 0; MemWr = 0; MemRd = 0; MemToReg = 11;

EXTOp = 0; LUOp = 0;

（6）IRQ = 1; // interrupt

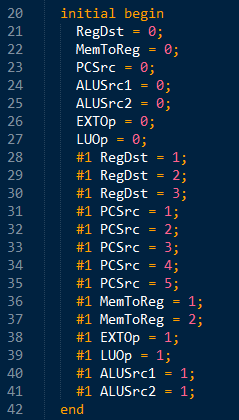
PCSrc = 100; RegDst = 11; RegWr = 1; ALUSrc1 = 0; ALUSrc2 = 0;

ALUFun = 000000; Sign = 0; MemWr = 0; MemRd = 0; MemToReg = 11;

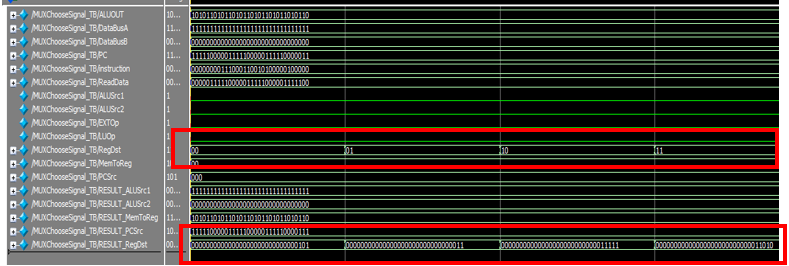
EXTOp = 0; LUOp = 0;

2.MUXChooseSignal功能仿真

由于该模块的功能是输入选择信号，输出选择器的选择结果。采用的方式是每次改变一个信号，观察相应的选择结果的改变。信号改变的顺序如下：



（1）RegDst改变



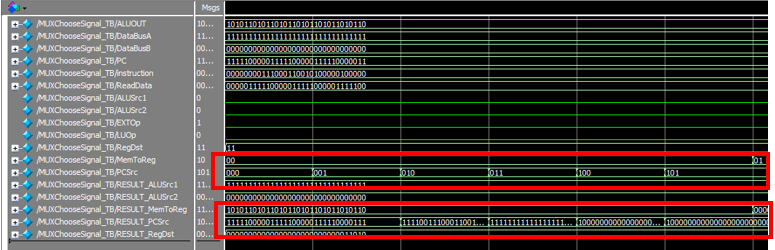
上图中是改变RegDst的结果，在输出波形中可以看出RESUKT\_RegDsr信号发生了三次改变，对应的结果分别为：RegDst == 0 ? instruction[15 : 11]: // rd

RegDst == 1 ? instruction[20 : 16]: // rt

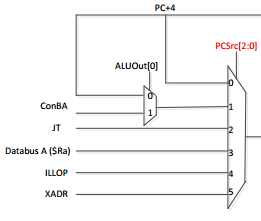
RegDst == 2 ? 5'b11111: // $ra (31)

RegDst == 3 ? 5'b11010;

（2）PCSrc改变



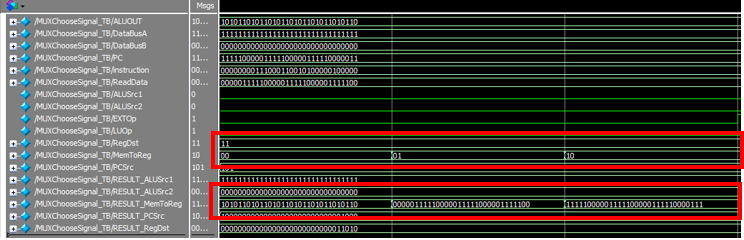
上图是改变PCSrc信号的效果，在波形中可以看到PCSrc产生了5次变化，但是RESULT\_PCSrc只产生了4次变化，第一次PCSrc变化时输出结果并没有变化，这是因为当PCSrc = 1时的输入信号来源有两个，如下图：



由ALUOut[0]进行控制，但是我们规定的ALUOut最低位为0，所以选择的结果是PC+4，这个结果与PCSrc = 0的时候是一致的，所以在波形上看并没有改变输出结果。

通过观察PC = 11111000001111100000111110000011，而PC+4的结果为11111000001111100000111110000111，这正好是PCSrc输出的结果，证明工作正常。

（3）MemToReg改变



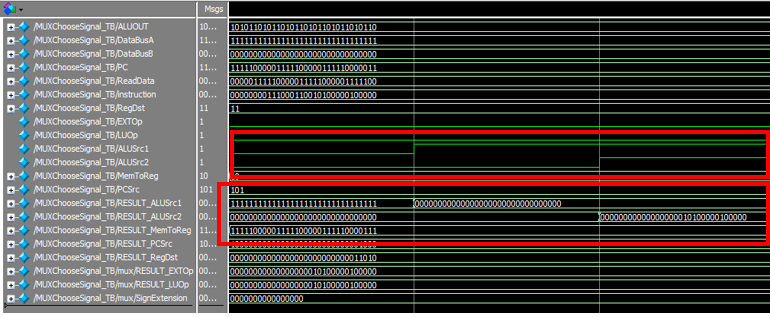
上图是MemToReg产生变化的结果，一共产生了两次变化，可以看到RESULT\_MemToReg也产生了两次变化，并且输出结果符合如下规则：

MemToReg == 0 ? ALUOUT:

MemToReg == 1 ? ReadData:

MemToReg == 2 ? (PC + 4);

（4）ALUSrc1和ALUSrc2的变化



上图中可以看出ALUSrc1和ALUSrc2的信号从0到1发生了变化。下面的红色框内的可以看出ALU的两个输入信号发生了变化，分析可知变化的过程是符合规则定义的：

RESULT\_ALUSrc1 = ALUSrc1 == 1 ? {27'b0, instruction[10 : 6]} :

ALUSrc1 == 0 ? DataBusA;

RESULT\_ALUSrc2 = ALUSrc2 == 1 ? RESULT\_LUOp :

ALUSrc2 == 0 ? DataBusB;

（5）EXTOp和LUOp的变化



这里所变化的信号是EXTOp和LUOp，可以看到当LUOp信号从低电平升到高电平时RESULT\_ALUOp信号发生了变化，同时RESULT\_ALUSrc2信号也发生了变化，这是合理的，因为ALU的第二个操作数是来自于拓展方式的选择的。

但是需要注意的一点是当EXTOp信号发生变化的时候并没有任何信号发生改变，经过分析发现是因为拓展方式如果选择符号拓展那么将根据instruction[15]来决定高位是1还是0，这里凑巧这一位是0，也就是说此时的符号拓展和无符号拓展没有区别。这是测试信号设计失误造成的，所以我们又选取了一组新的信号测试结果如下：



从上图可以看出拓展方式从高位补0变成了高位补1。

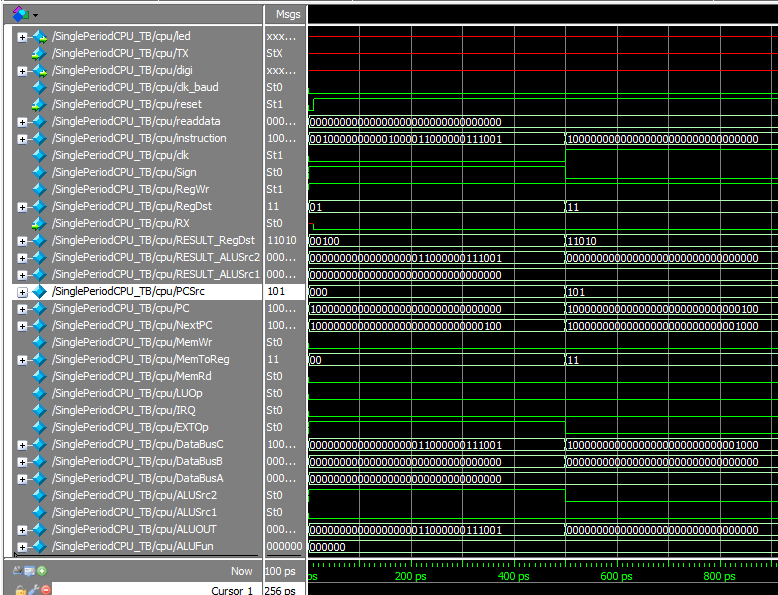
3.CPU整体仿真

仿真过程的第一步先逐一验证所有的指令，不考虑串口和中断等外设。

（1）addi

// addi $a0, $zero, 12345 #(0x3039)

data <= {6'h08, 5'd0 , 5'd4 , 16'h3039};

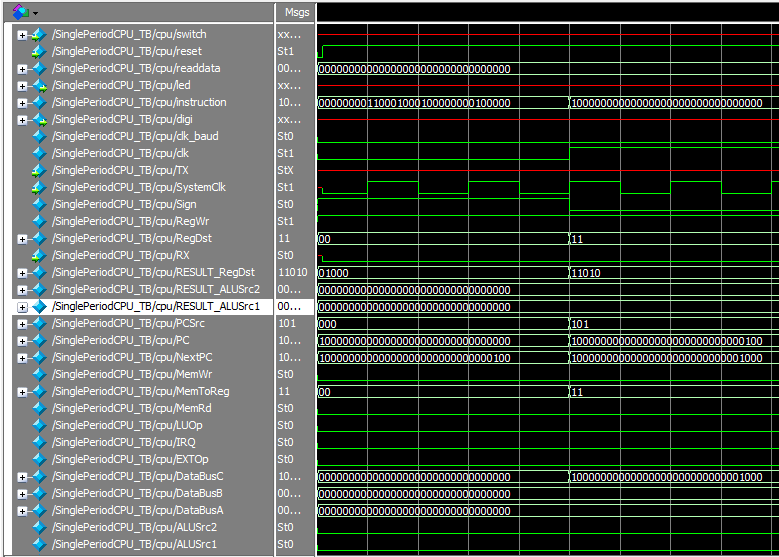


可以看到所有的控制信号都是正常的（以后再补）同时也验证了当执行完唯一的指令之后，程序的下一条指令读到了0x80000000，其控制信号也与预期的一致。

（2）add

// add $t0, $a2, $a0

data <= {6'h00, 5'd6 , 5'd4 , 5'd8 , 5'd0 , 6'h20};



接下来的指令验证都把所有的指令放在一个rom里面顺序执行。

4.发现的主要错误

（1）在写数据通路的选择信号时主要出现的问题是对变量声明的时候长度出现的问题，有的32位数据变量被不小心声明成了1位的控制信号。

（2）测试信号的变化顺序有问题，造成某些信号的控制效果不能观测到。

（3）在使用指令验证CPU功能时的跳转指令理解有问题，以为是相对指令地址的移动，造成跳转的结果不正确。

（4）某些指令的ALUFun出现问题，没有按照控制信号的要求赋值。后来发现是在control模块里面给控制信号赋值的时候使用的是ALUFun = 000000，显然这种方式默认的是十进制，对于大部分信号是错误的。

（5）一个困扰了一天的bug是由于实验指导书中的jr和jalr指令的跳转地址使用的都是PC+4，所以就直接给中断的返回地址$k0存入了PC+4，但是发现中断时当前指令并没有执行，所以应该减去4，也就是要把PC存入$k0。

（6）单周期里面的一个BUG是定时器的问题，定时器的触发周期时长不一致，一个短的时间触发后面跟着一个长的时间触发。后来发现是传递参数的时候将instruction[31]当成了PC[31]，结果一直持续不断的进入内核态，导致中断不能正常工作。这就造成了数码管不能正常显示。

（7）数码管的示数一直不正确，后来发现是汇编程序的问题。

（8）在给TL装载0xffffffff的时候，低四位的ffff使用了addi指令，这样造成的结果是使用了符号拓展，而ffff的符号位是1，所以在相加之后的结果是0xfffeffff，这样定时器就会进入一个不完整的计数过程。

（9）在将程序烧写进板子之后希望使用按键作为时钟信号调试，由于一开始装载了ffffffff到TL，所以按理说应该直接进入中断，但是返现并没有进中断，后来继续仿真发现是因为定时器初始化的过程放在了内核态里面。

5.流水线BUG

（1）一开始没有理解流水线的实质，所以对于最后一个回写的阶段的指令使用了之前的信号。

