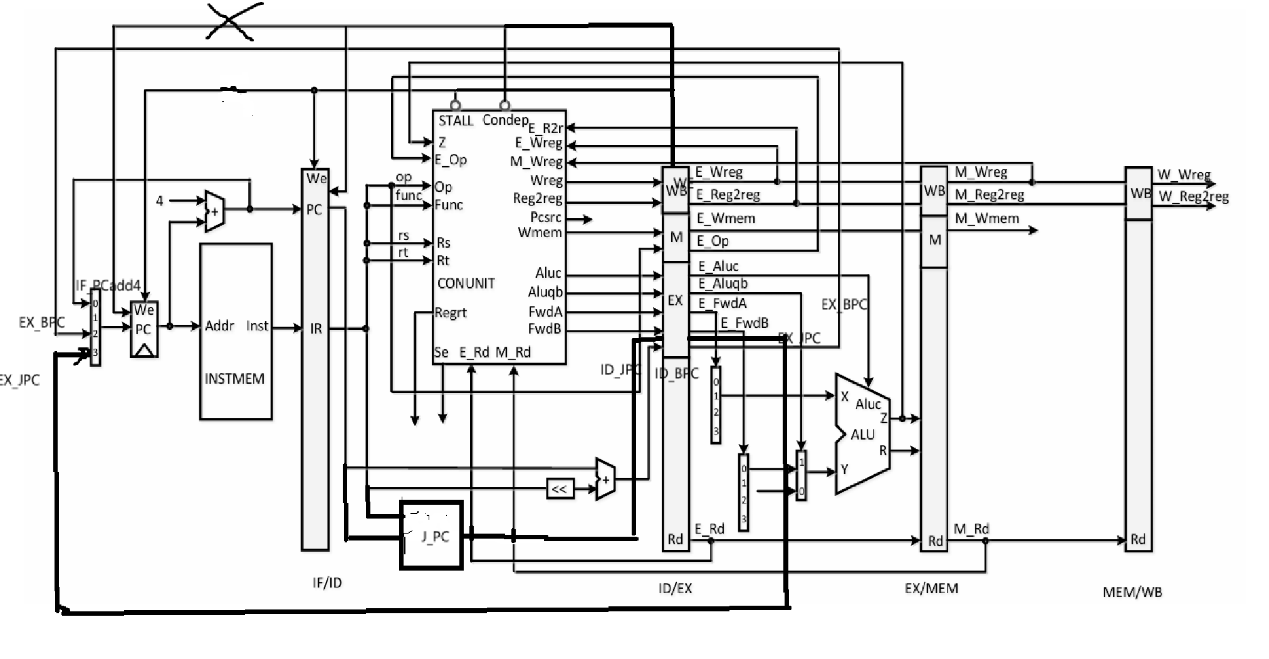
流水线设计

## 实验要求（请写出完成实验的具体要求）

1. 至少支持add、sub、and、or、addi、andi、ori、lw、sw、beq、bne和j十二条指令。
2. 对于“流水线处理器的设计和实现”，必须具有数据冒险、控制冒险的检测和处理机制。
3. 增加流水钱寄存器，讲单周期处理器改成流水线式执行，完成相关模块的重新拼接、代码调试和完成仿真测试，支持指令存储器里所有指令的执行，能通过观察仿真信号判断指令是否正常执行。

## 设计结构（请画出设计的主要处理器结构原理图）

相比于单周期CPU，流水线主要增加了4个流水线寄存器，我分别命名为REG\_IF\_ID，REG\_ID\_EX，REG\_EX\_MEM，REG\_MEM\_WB，但这只是最最基础的，我还必须想办法规避数据冒险和控制冒险，这其中数据冒险的解决办法主要为三种，其一是将寄存器读出的时间往后延半个周期，其二为内部前推，其三为停顿。这三种方法分别解决第一条指令和第四条指令的冲突，第一条指令和第二、三条指令的冲突，lw指令和后面一条指令的冲突。控制冒险的解决方法书上只给出了条件分支指令的解决办法，也就是清除两个周期的指令，那么J指令呢？很显然，我也可以这么解决，原理图如下。相比于书上的，增加了J指令的跳转，以及修改了Condep的输入位置（不应该输入到PC，应该输入到ID/EX寄存器）和增加了STALL对ID/EX的输入。



1. 原理图

## 主要实现代码（请给出主要的/核心的实现代码）

大部分代码在单周期CPU中给出，这里只写出改变的，也就是指令存储器（为了测试数据冒险和控制冒险）、控制单元和PC寄存器，还有增加的，也就是2、5、6位的寄存器和四个流水线寄存器，以下是实现的代码：

1. PC寄存器：

module PC(IF\_Result, Clk, En, Clrn, IF\_Addr, stall);

input [31:0] IF\_Result;

input Clk, En, Clrn, stall;

output [31:0] IF\_Addr;

wire En\_S = En & ~stall;

D\_FFEC32 pc(IF\_Result, Clk, En\_S, Clrn, IF\_Addr);

endmodule

1. 指令存储器：

module INSTMEM(Addr,Inst);

input [31:0]Addr;

output [31:0]Inst;

wire [31:0]Rom[31:0];

assign Rom[5'h00]=32'b001101\_00000\_00001\_00000\_00000\_001010; //ori $1,$0,10\_\_\_$1=10 Rom[5'h00]=32'h3401000a

assign Rom[5'h01]=32'b001000\_00000\_00010\_00000\_00000\_000110; //addi $2,$0,6\_\_\_$2=6 Rom[5'h01]=32'h20020006

assign Rom[5'h02]=32'b000000\_00001\_00010\_00011\_00000\_100100; //and $3,$1,$2\_\_\_$3=2

assign Rom[5'h03]=32'b000000\_00001\_00010\_00100\_00000\_100101; //or $4,$1,$2\_\_\_$4=14

assign Rom[5'h04]=32'b000000\_00100\_00010\_00101\_00000\_100010; //sub $5,$4,$2\_\_\_$5=8

assign Rom[5'h05]=32'b001100\_00001\_00110\_00000\_00000\_001011; //andi $6,$1,1\_\_\_$6=10

assign Rom[5'h06]=32'b101011\_00110\_00110\_00000\_00000\_001110; //sw $6,14($6)\_\_\_memory[$6+14]=$6=10,ram[6]=10

assign Rom[5'h07]=32'b100011\_00110\_01000\_00000\_00000\_001110; //lw $8,14($6)\_\_\_$8=memory[$6+14]=10,$8=ram[6]=10

assign Rom[5'h08]=32'b000000\_01000\_00101\_00100\_00000\_100101; //or $4,$8,$5\_\_\_$4=10

assign Rom[5'h09]=32'b000000\_00001\_00010\_00101\_00000\_100010; //sub $5,$1,$2\_\_\_$5=4

assign Rom[5'h0A]=32'b001100\_00101\_00110\_00000\_00000\_001111; //andi $6,$5,15\_\_\_$6=4

assign Rom[5'h0B]=32'b000100\_00110\_00011\_00000\_00000\_000100; //beq $6,$3,4\_\_\_0,JUMPTO\_10(Not Done)

assign Rom[5'h0C]=32'b000101\_00110\_00011\_00000\_00000\_000100; // bne $6 $3,8\_\_\_2,JUMPTO\_12(Done)

assign Rom[5'h0D]=32'b000000\_00100\_00101\_00011\_00000\_100100; //and $3,$4,$5\_\_\_

assign Rom[5'h0E]=32'b000000\_00001\_00101\_00100\_00000\_100101; //or $4,$1,$5\_\_\_

assign Rom[5'h0F]=32'b000000\_00001\_00010\_00101\_00000\_100010; //sub $5,$1,$2\_\_\_

assign Rom[5'h10]=32'b001100\_00001\_00110\_00000\_00000\_001011; //andi $6,$1,13\_\_\_$6=$1&13=10

assign Rom[5'h11]=32'b000000\_00101\_00110\_00111\_00000\_100000; //add $7,$5,$6\_\_\_$7=14

assign Rom[5'h12]=32'b000010\_00000\_00000\_00000\_00000\_001010; //j 01100\_\_\_\_0a Rom[5'h07]=32'h0800000a

assign Rom[5'h13]=32'hXXXXXXXX;

assign Rom[5'h14]=32'hXXXXXXXX;

assign Rom[5'h15]=32'hXXXXXXXX;

assign Rom[5'h16]=32'hXXXXXXXX;

assign Rom[5'h17]=32'hXXXXXXXX;

assign Rom[5'h18]=32'hXXXXXXXX;

assign Rom[5'h19]=32'hXXXXXXXX;

assign Rom[5'h1A]=32'hXXXXXXXX;

assign Rom[5'h1B]=32'hXXXXXXXX;

assign Rom[5'h1C]=32'hXXXXXXXX;

assign Rom[5'h1D]=32'hXXXXXXXX;

assign Rom[5'h1E]=32'hXXXXXXXX;

assign Rom[5'h1F]=32'hXXXXXXXX;

assign Inst=Rom[Addr[6:2]];

endmodule

1. 控制单元：

module CONUNIT(E\_Op,Op,Func,Z,Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Rs,Rt,E\_Rd,M\_Rd,E\_Wreg,M\_Wreg,FwdA,FwdB,E\_Reg2reg,stall,condep);

input [5:0]Op,Func,E\_Op;

input Z;

input E\_Wreg,M\_Wreg,E\_Reg2reg;

input [4:0]E\_Rd,M\_Rd,Rs,Rt;

output Regrt,Se,Wreg,Aluqb,Wmem,Reg2reg,stall,condep;

output [1:0]Pcsrc,Aluc;

output reg [1:0]FwdA,FwdB;

wire R\_type=~|Op;

wire I\_add=R\_type&Func[5]&~Func[4]&~Func[3]&~Func[2]&~Func[1]&~Func[0];

wire I\_sub=R\_type&Func[5]&~Func[4]&~Func[3]&~Func[2]&Func[1]&~Func[0];

wire I\_and=R\_type&Func[5]&~Func[4]&~Func[3]&Func[2]&~Func[1]&~Func[0];

wire I\_or=R\_type&Func[5]&~Func[4]&~Func[3]&Func[2]&~Func[1]&Func[0];

wire I\_addi=~Op[5]&~Op[4]&Op[3]&~Op[2]&~Op[1]&~Op[0];

wire I\_andi=~Op[5]&~Op[4]&Op[3]&Op[2]&~Op[1]&~Op[0];

wire I\_ori=~Op[5]&~Op[4]&Op[3]&Op[2]&~Op[1]&Op[0];

wire I\_lw=Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0];

wire I\_sw=Op[5]&~Op[4]&Op[3]&~Op[2]&Op[1]&Op[0];

wire I\_beq=~Op[5]&~Op[4]&~Op[3]&Op[2]&~Op[1]&~Op[0];

wire I\_bne=~Op[5]&~Op[4]&~Op[3]&Op[2]&~Op[1]&Op[0];

wire E\_beq=~E\_Op[5]&~E\_Op[4]&~E\_Op[3]&E\_Op[2]&~E\_Op[1]&~E\_Op[0];

wire E\_bne=~E\_Op[5]&~E\_Op[4]&~E\_Op[3]&E\_Op[2]&~E\_Op[1]&E\_Op[0];

wire E\_J = ~E\_Op[5]&~E\_Op[4]&~E\_Op[3]&~E\_Op[2]&E\_Op[1]&~E\_Op[0];

wire I\_J=~Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&~Op[0];

wire E\_Inst = I\_add|I\_sub|I\_and|I\_or|I\_sw|I\_beq|I\_bne;

assign Regrt = I\_addi|I\_andi|I\_ori|I\_lw|I\_sw|I\_beq|I\_bne|I\_J;

assign Se = I\_addi|I\_lw|I\_sw|I\_beq|I\_bne;

assign Wreg = I\_add|I\_sub|I\_and|I\_or|I\_addi|I\_andi|I\_ori|I\_lw;

assign Aluqb = I\_add|I\_sub|I\_and|I\_or|I\_beq|I\_bne|I\_J;

assign Aluc[1] = I\_and|I\_or|I\_andi|I\_ori;

assign Aluc[0] = I\_sub|I\_or|I\_ori|I\_beq|I\_bne;

assign Wmem = I\_sw;

assign Pcsrc[1] = (E\_beq&Z)|(E\_bne&~Z)|E\_J;

assign Pcsrc[0] = E\_J;

assign Reg2reg =I\_add|I\_sub|I\_and|I\_or|I\_addi|I\_andi|I\_ori|I\_sw|I\_beq|I\_bne|I\_J;

always@(E\_Rd,M\_Rd,E\_Wreg,M\_Wreg,Rs,Rt)begin

FwdA=2'b00;

if((Rs==E\_Rd)&(E\_Rd!=0)&(E\_Wreg==1))begin

FwdA=2'b10;

end else begin

if((Rs==M\_Rd)&(M\_Rd!=0)&(M\_Wreg==1))begin

FwdA=2'b01;

end

end

end

always@(E\_Rd,M\_Rd,E\_Wreg,M\_Wreg,Rs,Rt)begin

FwdB=2'b00;

if((Rt==E\_Rd)&(E\_Rd!=0)&(E\_Wreg==1))begin

FwdB=2'b10;

end else begin

if((Rt==M\_Rd)&(M\_Rd!=0)&(M\_Wreg==1))begin

FwdB=2'b01;

end

end

end

assign stall=((Rs==E\_Rd)|(Rt==E\_Rd))&(E\_Reg2reg==0)&(E\_Rd!=0)&(E\_Wreg==1);

assign condep=(E\_beq&Z)|(E\_bne&~Z)|E\_J;

endmodule

1. 2位寄存器：

module D\_FFEC2(D, Clk, En, Clrn, Q, Qn);

input [1:0] D;

input Clk, En, Clrn;

output [1:0] Q, Qn;

D\_FFEC d0(D[0], Clk, En, Clrn, Q[0], Qn[0]);

D\_FFEC d1(D[1], Clk, En, Clrn, Q[1], Qn[1]);

endmodule

1. 5位寄存器：

module D\_FFEC5(D, Clk, En, Clrn, Q, Qn);

input [4:0] D;

input Clk, En, Clrn;

output [4:0] Q, Qn;

D\_FFEC d0(D[0], Clk, En, Clrn, Q[0], Qn[0]);

D\_FFEC d1(D[1], Clk, En, Clrn, Q[1], Qn[1]);

D\_FFEC d2(D[2], Clk, En, Clrn, Q[2], Qn[2]);

D\_FFEC d3(D[3], Clk, En, Clrn, Q[3], Qn[3]);

D\_FFEC d4(D[4], Clk, En, Clrn, Q[4], Qn[4]);

endmodule

1. 6位寄存器：

module D\_FFEC6(D, Clk, En, Clrn, Q, Qn);

input [5:0] D;

input Clk, En, Clrn;

output [5:0] Q, Qn;

D\_FFEC d0(D[0], Clk, En, Clrn, Q[0], Qn[0]);

D\_FFEC d1(D[1], Clk, En, Clrn, Q[1], Qn[1]);

D\_FFEC d2(D[2], Clk, En, Clrn, Q[2], Qn[2]);

D\_FFEC d3(D[3], Clk, En, Clrn, Q[3], Qn[3]);

D\_FFEC d4(D[4], Clk, En, Clrn, Q[4], Qn[4]);

D\_FFEC d5(D[5], Clk, En, Clrn, Q[5], Qn[5]);

endmodule

1. IF\_ID寄存器：

module REG\_IF\_ID(D0, D1, En, Clk, Clrn, Q0, Q1,stall, condep);

input [31:0] D0, D1;

input En, Clk, Clrn, stall, condep;

output [31:0] Q0, Q1;

wire En\_S = En & ~stall;

wire Clrn\_C = Clrn & ~condep;

D\_FFEC32 q0(D0, Clk, En\_S, Clrn\_C, Q0);

D\_FFEC32 q1(D1, Clk, En\_S, Clrn\_C, Q1);

endmodule

1. ID\_EX寄存器：

module REG\_ID\_EX(D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11,D12,D13,En,Clk,Clrn,Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,Q11,Q12,Q13,stall,condep);

// Wreg, Reg2reg, Wmem, ID\_Inst[31:26], Aluc, Aluqb, //ID\_JPC,ID\_Qa, ID\_Qb, ID\_I, ID\_Wr, FwdA, FwdB,ID\_BPC, En, Clk, Clrn,

//E\_Wreg, E\_Reg2reg, E\_Wmem, E\_Op, E\_Aluc, E\_Aluqb, //EX\_JPC, E\_R1, E\_R2, E\_I, E\_Rd, E\_FwdA, E\_FwdB, EX\_BPC, stall, condep

input [31:0] D7,D8,D9,D13,D6;

input [5:0] D3;

input [4:0] D10;

input [1:0] D4, D11, D12;

input D0, D1, D2, D5;

input En, Clk, Clrn, stall, condep;

output [31:0] Q7,Q8,Q9,Q13,Q6;

output [5:0] Q3;

output [4:0] Q10;

output [1:0] Q4, Q11, Q12;

output Q0, Q1, Q2, Q5;

wire Clrn\_SC = Clrn & ~condep;

D\_FFEC q0(D0,Clk,En,Clrn\_SC,Q0);

D\_FFEC q1(D1,Clk,En,Clrn\_SC,Q1);

D\_FFEC q2(D2,Clk,En,Clrn\_SC,Q2);

D\_FFEC6 q3(D3,Clk,En,Clrn\_SC,Q3);

D\_FFEC2 q4(D4,Clk,En,Clrn\_SC,Q4);

D\_FFEC q5(D5,Clk,En,Clrn\_SC,Q5);

D\_FFEC32 q6(D6,Clk,En,Clrn\_SC,Q6);

D\_FFEC32 q7(D7,Clk,En,Clrn\_SC,Q7);

D\_FFEC32 q8(D8,Clk,En,Clrn\_SC,Q8);

D\_FFEC32 q9(D9,Clk,En,Clrn\_SC,Q9);

D\_FFEC5 q10(D10,Clk,En,Clrn\_SC,Q10);

D\_FFEC2 q11(D11,Clk,En,Clrn\_SC,Q11);

D\_FFEC2 q12(D12,Clk,En,Clrn\_SC,Q12);

D\_FFEC32 q13(D13,Clk,En,Clrn\_SC,Q13);

endmodule

1. EX\_MEM寄存器：

module REG\_EX\_MEM(D0, D1, D2, D3, D4, D5, En, Clk, Clrn, Q0, Q1, Q2, Q3, Q4, Q5);

input D0, D1, D2;

input [31:0] D3,D4;

input [4:0] D5;

output En, Clk, Clrn;

output Q0, Q1, Q2;

output [31:0] Q3, Q4;

output [4:0] Q5;

D\_FFEC q0(D0, Clk, En, Clrn, Q0);

D\_FFEC q1(D1, Clk, En, Clrn, Q1);

D\_FFEC q2(D2, Clk, En, Clrn, Q2);

D\_FFEC32 q3(D3, Clk, En, Clrn, Q3);

D\_FFEC32 q4(D4, Clk, En, Clrn, Q4);

D\_FFEC5 q5(D5, Clk, En, Clrn, Q5);

endmodule

1. MEM\_WB寄存器：

module REG\_MEM\_WB(D0, D1, D2, D3, D4, En, Clk, Clrn, Q0, Q1, Q2, Q3, Q4);

input [31:0] D2, D3;

input [4:0] D4;

input D0, D1, En, Clk, Clrn;

output Q0, Q1;

output [31:0] Q2, Q3;

output [4:0] Q4;

D\_FFEC q0(D0, Clk, En, Clrn, Q0);

D\_FFEC q1(D1, Clk, En, Clrn, Q1);

D\_FFEC32 q2(D2, Clk, En, Clrn, Q2);

D\_FFEC32 q3(D3, Clk, En, Clrn, Q3);

D\_FFEC5 q4(D4, Clk, En, Clrn, Q4);

endmodule

1. 流水线CPU：

module PIPELINE(Clk, En, Clrn,EX\_X, EX\_Y, EX\_R, E\_Rd, M\_Rd, IF\_Inst, ID\_Inst,Pcsrc);

input Clk, En, Clrn;

output [31:0] EX\_X, EX\_Y, EX\_R, IF\_Inst, ID\_Inst;

output [4:0] E\_Rd, M\_Rd;

output [1:0] Pcsrc;

wire [31:0] IF\_Result, IF\_Addr, IF\_PCadd4, D, ID\_Qa, ID\_Qb, ID\_PCadd4, ID\_Inst;

wire [31:0] E\_R1, E\_R2, E\_I, E\_I\_L2, Y, E\_R, M\_R, M\_S, Dout, W\_R, W\_Dout, ID\_I, Alu\_X, E\_num, ID\_I\_L2;

wire [31:0] ID\_JPC, ID\_BPC, EX\_JPC, EX\_BPC;

wire [5:0] E\_Op;

wire [4:0] ID\_Rd, W\_Rd;

wire [1:0] Aluc, E\_Aluc, FwdA, FwdB, E\_FwdA, E\_FwdB;

wire Regrt, Se, Wreg, Aluqb, Reg2reg, Wmem, Z;

wire E\_Wreg, E\_Reg2reg, E\_Wmem, E\_Aluqb, Cout, M\_Wreg, M\_Reg2reg, M\_Wmem, W\_Wreg, W\_Reg2reg, stall, condep;

// IF

MUX4X32 select\_pc(IF\_PCadd4, 0, EX\_BPC, EX\_JPC, Pcsrc, IF\_Result);

PC program\_counter(IF\_Result, Clk, En, Clrn, IF\_Addr, stall);

assign IF\_PCadd4 = IF\_Addr + 4;

INSTMEM fetch\_instruction(IF\_Addr, IF\_Inst);

REG\_IF\_ID reg\_IF\_ID(IF\_PCadd4, IF\_Inst, En, Clk, Clrn, ID\_PCadd4, ID\_Inst,stall, condep);

// ID

CONUNIT control\_unit(E\_Op, ID\_Inst[31:26], ID\_Inst[5:0], Z, Regrt, Se, Wreg, Aluqb, Aluc, Wmem, Pcsrc, Reg2reg, ID\_Inst[25:21], ID\_Inst[20:16], E\_Rd, M\_Rd, E\_Wreg, M\_Wreg, FwdA, FwdB, E\_Reg2reg, stall, condep);

MUX2X5 select\_rt\_rd(ID\_Inst[15:11], ID\_Inst[20:16], Regrt, ID\_Rd);

EXT16T32 extend\_immediate(ID\_Inst[15:0], Se, ID\_I);

REGFILE register\_file(ID\_Inst[25:21], ID\_Inst[20:16], D, W\_Rd, W\_Wreg, ~Clk, Clrn, ID\_Qa, ID\_Qb);

SHIFTER32 shift\_immediate\_l2(ID\_I, ID\_I\_L2);

CLA\_32 get\_B\_addr(ID\_PCadd4, ID\_I\_L2, 0, ID\_BPC, Cout);

SHIFTER\_COMBINATION shifter\_combination(ID\_Inst[25:0], ID\_PCadd4, ID\_JPC);

REG\_ID\_EX reg\_ID\_EX(Wreg, Reg2reg, Wmem, ID\_Inst[31:26], Aluc, Aluqb,ID\_JPC,ID\_Qa, ID\_Qb, ID\_I, ID\_Rd, FwdA, FwdB,ID\_BPC, En, Clk, Clrn, E\_Wreg, E\_Reg2reg, E\_Wmem, E\_Op, E\_Aluc, E\_Aluqb,EX\_JPC,E\_R1, E\_R2, E\_I, E\_Rd, E\_FwdA, E\_FwdB, EX\_BPC, stall, condep);

//EX

MUX4X32 select\_alu\_x(E\_R1, D, M\_R, 0, E\_FwdA, Alu\_X);

MUX4X32 select\_alu\_y\_first(E\_R2, D, M\_R, 0, E\_FwdB, E\_num);

MUX2X32 select\_alu\_y\_second(E\_I, E\_num, E\_Aluqb, Y);

ALU alu(Alu\_X,Y,E\_Aluc,E\_R,Z);

REG\_EX\_MEM reg\_EX\_MEM(E\_Wreg, E\_Reg2reg, E\_Wmem, E\_R, E\_num, E\_Rd, En, Clk, Clrn, M\_Wreg, M\_Reg2reg, M\_Wmem, M\_R, M\_S, M\_Rd);

//MEM

DATAMEM data\_mem(M\_R, M\_S, Clk, M\_Wmem, Dout);

REG\_MEM\_WB reg\_MEM\_WB(M\_Wreg, M\_Reg2reg, M\_R, Dout, M\_Rd, En, Clk, Clrn, W\_Wreg, W\_Reg2reg, W\_R, W\_Dout, W\_Rd);

//WB

MUX2X32 select\_R\_Dout(W\_Dout,W\_R,W\_Reg2reg,D);

assign EX\_R = E\_R;

assign EX\_X = Alu\_X;

assign EX\_Y = Y;

1. endmodule测试文件：

module PIPELINE\_tb;

reg Clk, En, Clrn;

wire [31:0] EX\_R;

wire [31:0] EX\_X;

wire [31:0] EX\_Y, IF\_Inst, ID\_Inst;

wire [4:0] E\_Rd, M\_Rd;

wire [1:0] Pcsrc;

PIPELINE cpu(.Clk(Clk), .En(En), .Clrn(Clrn),.EX\_R(EX\_R), .EX\_X(EX\_X), .EX\_Y(EX\_Y), .E\_Rd(E\_Rd), .M\_Rd(M\_Rd),.IF\_Inst(IF\_Inst), .ID\_Inst(ID\_Inst),.Pcsrc(Pcsrc));

initial begin

Clk=0;Clrn=0;En=1;

#20;

Clk=1;

#20;

Clrn=1;

Clk=0;

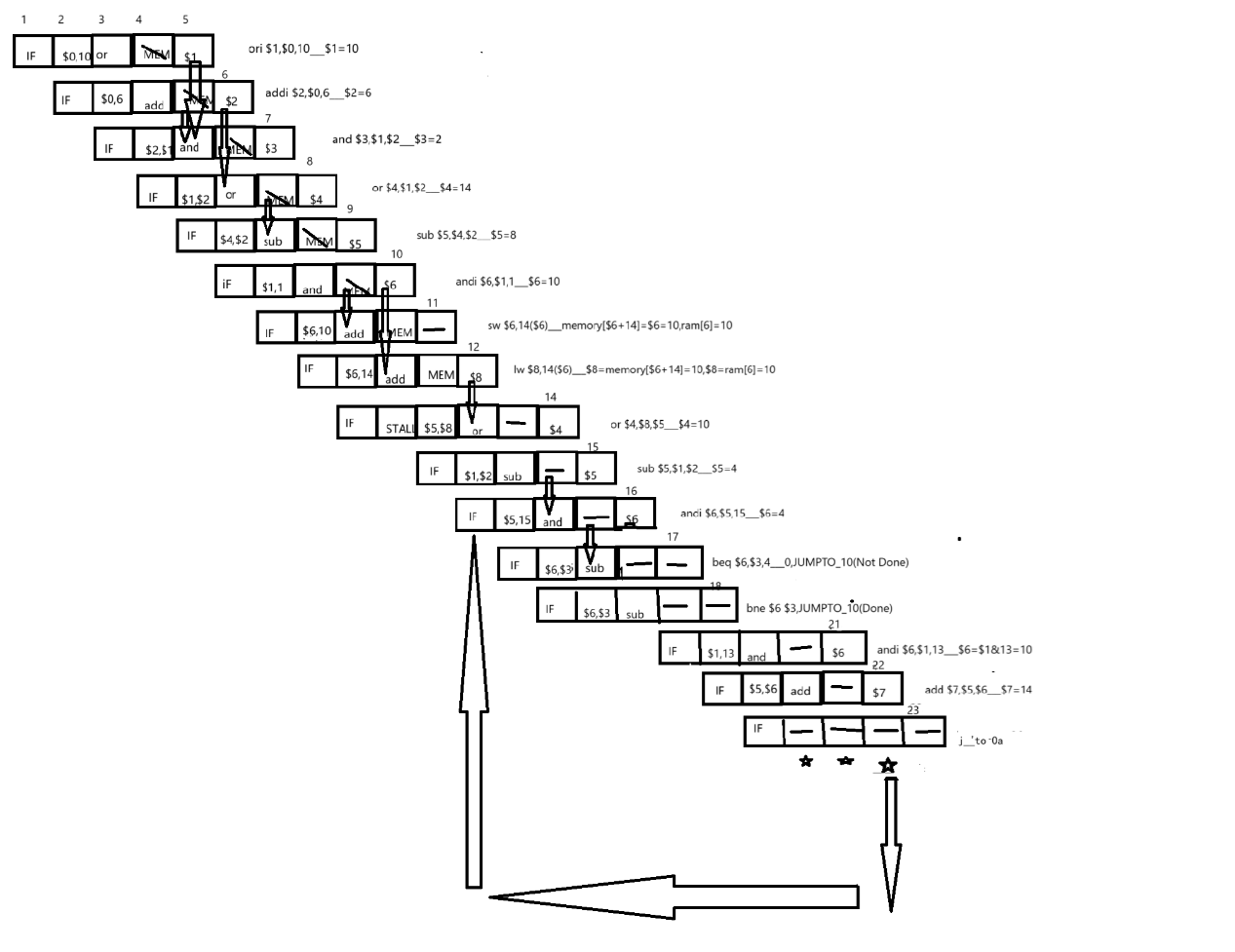
forever #20 Clk=~Clk;

end

endmodule

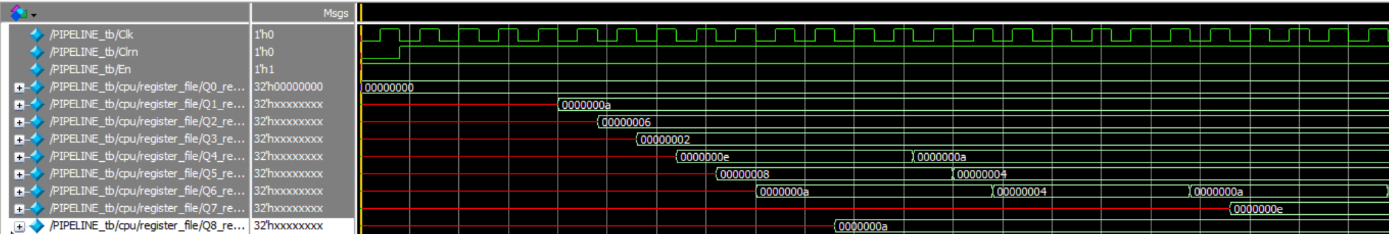
## 仿真结果（请给出仿真波形截图和对应的结果分析阐述）

在分析结果之前，我先把流水线模型机给出，此模型机标出了时间（以数字的形式）、指令、前推等：

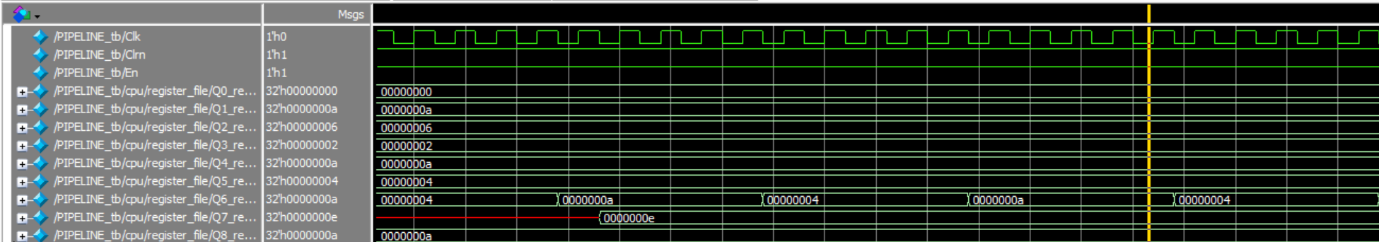


1. 流水线模型机

好的，我们可以回顾我们的指令代码，再结合这个图进行分析，一共有十六条指令，我们在这个实验中更关心的是数据冒险和控制冒险，因此判断指令的正确性我们先直接按照寄存器的结果来判断（当然包括变化的时机与预期是否符合），如下图：



1. 寄存器值图一



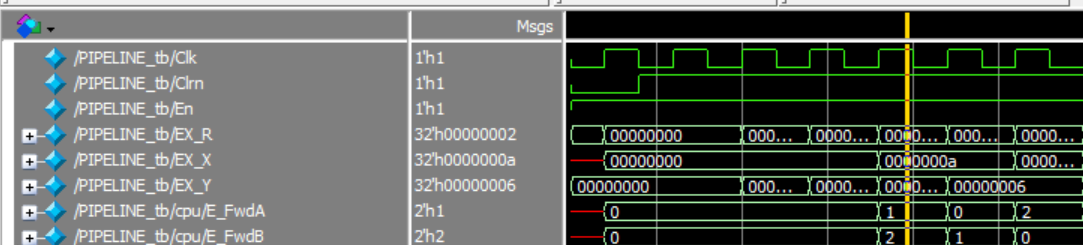
1. 寄存器图二

先解释一下时钟周期的问题，我们是从20ns开始第一个时钟周期，但我们可以发现Clrn信号此时为0，这是我们进行了置零操作，也就是PC的输入要为0，也就是说可以当作我们这时候开始了第一个时钟周期！注意在这里=为第一次赋值，->表示发生变化，首先看寄存器图一，我们可以清楚地看到$0=0(一直保持)，$1=10（在第五个始终周期下降沿），$2=6（在第六个时钟周期下降沿），$3=2（在第七个时钟周期下降沿），$4=14（在第八个时钟周期下降沿），$5=8（在第九个时钟周期下降沿），$6=10（在第十个时钟周期下降沿），$8=10（在第十二个时钟周期下降沿），$4->10（在第十四个时钟周期下降沿），$5->4（在第十五个时钟周期下降沿），$6->4（在第十六个时钟周期下降沿），$6->10（在第二十一个时钟周期下沿），$7->14（在第二十二个时钟周期下降沿）。可以看到J指令之前的指令都是正确的。那么如何验证J指令正确呢？由于我们设置的指令是一个循环，如果J指令没错的话，那么我们将会在第二十六个周期看到$6->4，也就是会进到不断地循环当中，我们看寄存器图二，符合我们地预期，也就是说J指令也正确。从而我们的程序运行是没有问题的，并且数据冒险和控制冒险都得到了解决。

我们接下来具体分析数据冒险和控制冒险：

1. 数据冒险
2. 一和四的数据冒险：这种冒险我们直接由上述寄存器值写入的时机为下降沿可知已得到解决。
3. 一和二的数据冒险：这里我由模型机可以看到有许多指令存在这种冲突，我们只以第二条和第三条指令的冲突作为例子。
4. 一和三的数据冒险：这里我有模型机可以看到有许多指令存在这种冲突，我只以第一条和第三条指令的冲突作为例子。

我选取了EX模块中的ALU的输入值（已经选择）EX\_X,EX\_Y和计算结果EX\_R，还有前推信号E\_FwdA和E\_FwdB。

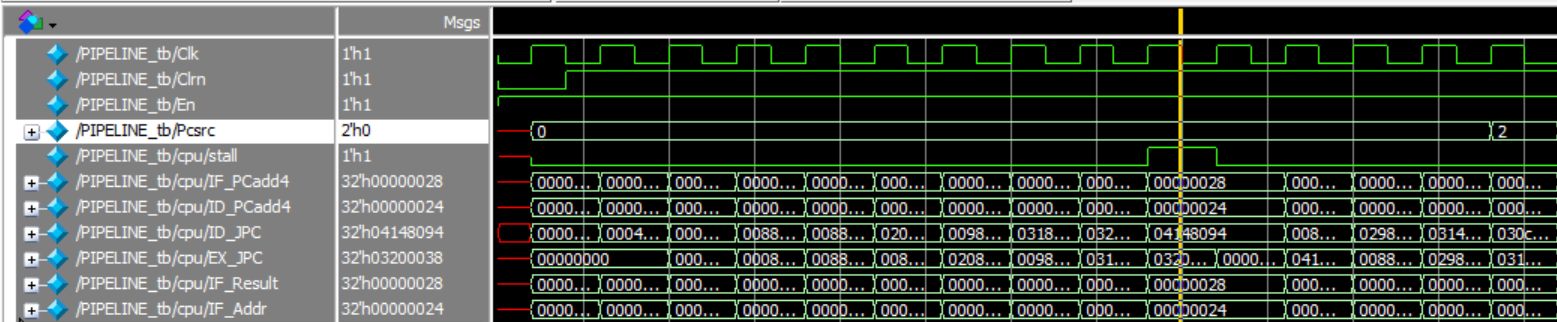


1. 数据冒险图一

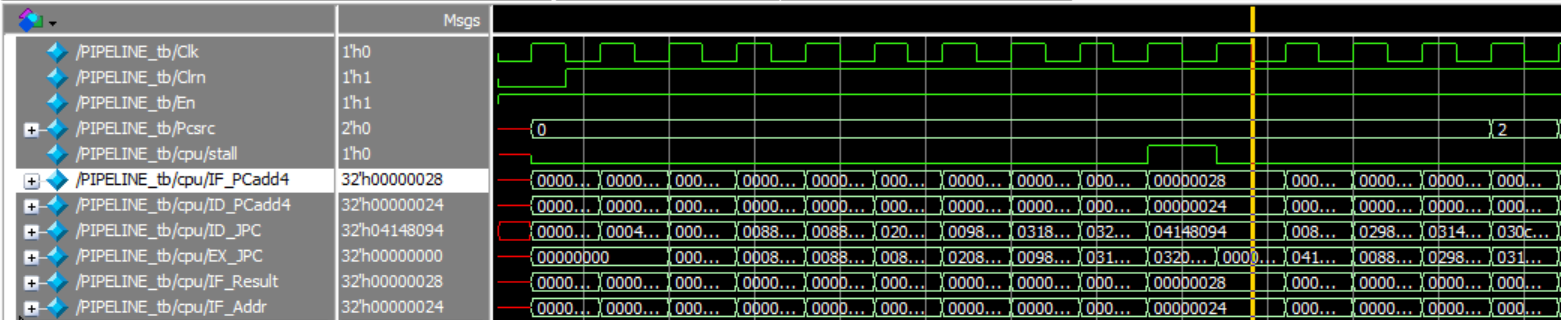
按照预期，在第五个时钟周期，也就是我们黄线的周期，第二条指令的$2要前推到EX\_X，第一条指令的$1要前推到EX\_Y，那么FwdA要为01（也就是1），FwdB要为10（也就是2），可以看到仿真结果符合预期，而EX\_X=10也就是$2的值，EX\_Y=6也就是$1的值，运算结果EX\_R=2，这些结果符合预期，证明我们的数据冒险全部得到解决。

1. lw的数据冒险:这里产生冒险的只有第八条和第九条指令。

按照我们的设计逻辑，在lw处于EX级的时候也就是第九个时钟周期结束时的上升沿时，stall=1，这样使得PC寄存器和IF\_ID寄存器暂停一个时钟周期（也就是第十个时钟周期），使得ID\_EX寄存器清零。所以我们选择IF\_Result和IF\_Addr作为判断PC寄存器是否暂停，选择IF\_PCadd4和ID\_Pcadd4作为判断IF\_ID寄存器是否暂停，ID\_JPC和EX\_JPC作为判断ID\_EX寄存器是否清零。



1. 第十个时钟周期

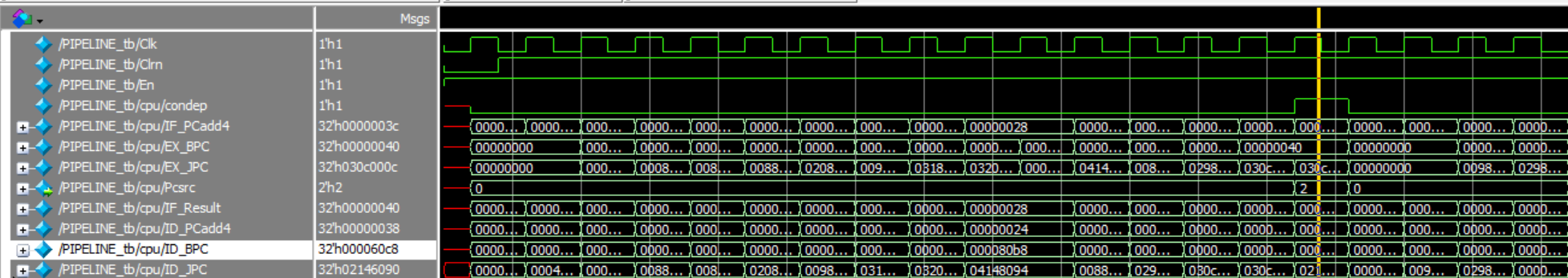


1. 第十一个时钟周期

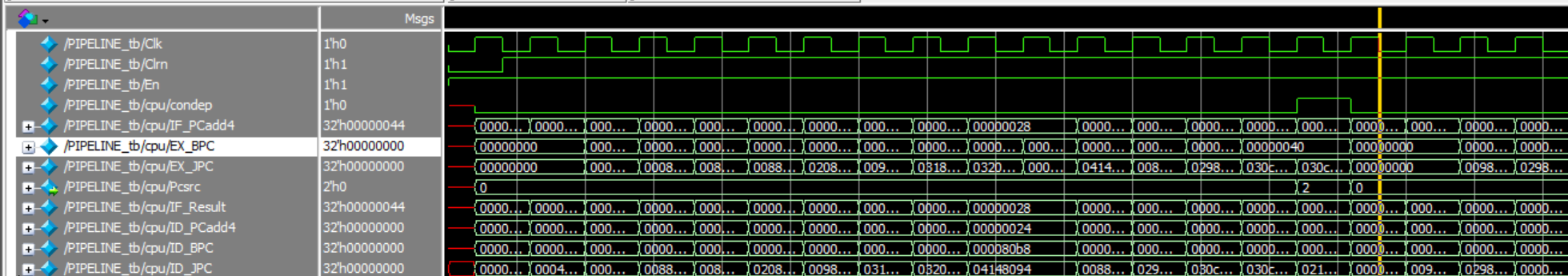
我们可以看到在第十个始时钟周期stall=1，符合我们的预期，IF\_PCadd4和ID\_Pcadd4在第十个和第十一个时钟周期值没有改变，证明IF\_ID寄存器在第十个时钟周期暂停了，IF\_Result和IF\_Addr在第十个和第十一个时钟周期值没有改变，证明PC寄存器在第十个时钟周期暂停了，ID\_JPC和EX\_JPC在第十一个周期为0，证明ID\_EX寄存器在第十个周期结束时的上升沿清零了。这都符合我们的预期，证明我们已经解决了lw指令的数据冒险。

1. 控制冒险
2. beq和bne的控制冒险：我们在第13条指令执行了bne的跳转，那么按照预期会延后两个周期，那么就会在第十七个周期结束时的上升沿才完成执行andi的取指。并且在第二十一个周期的下降沿将$6的值写入，我们可以看到寄存器的结果是符合预期的.那么我们可以具体分析一下，可知在第16个时钟周期，condep的值为1，那么在第16个时钟周期结束时的上升沿，IF\_ID，ID\_EX寄存器的值会清零。

我们选取ID\_BPC、ID\_JPC、EX\_BPC、EX\_JPC、IF\_PCadd4、IF\_Result、Pcsrc来判断是否发生跳转，同时ID\_BPC和EX\_JPC可以判断ID\_EX寄存器是否清零，选取IF\_PCadd4和ID\_PCadd4来判断IF\_ID寄存器是否清零。下图为仿真结果：



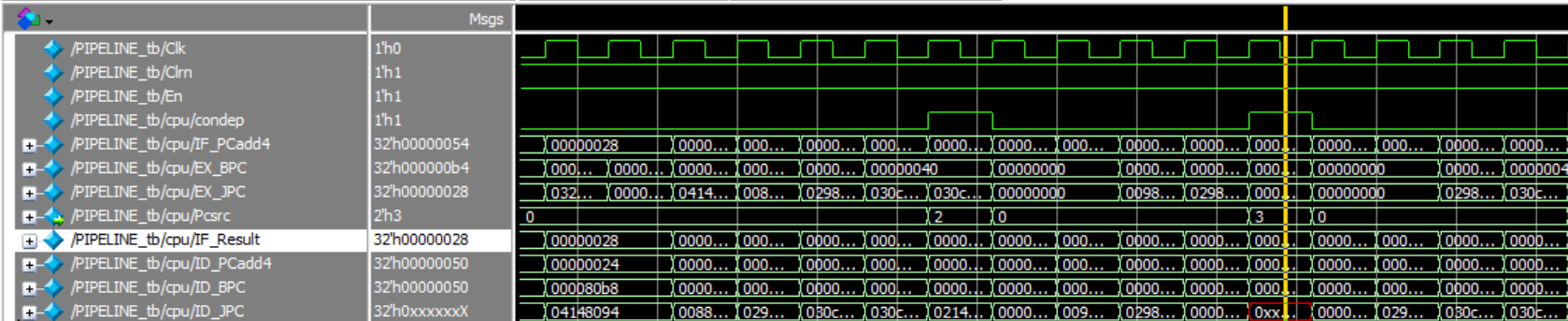
1. 第十六个时钟周期



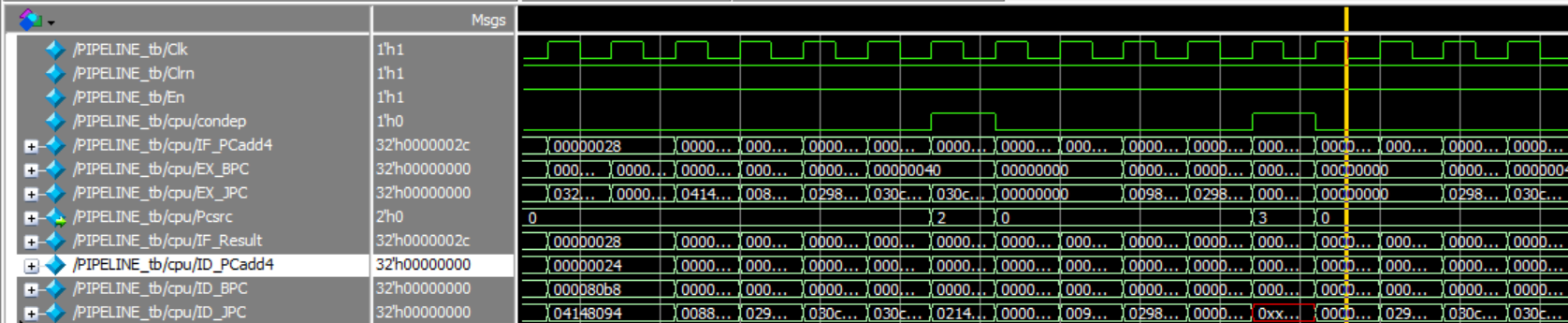
1. 第十七个时钟周期

我们可以看到第16个时钟周期时，condep=1，而在第十六个时钟周期IF\_Result的值和EX\_BPC的值相等，证明选择没有错误；而在16个时钟周期结束时的上升沿EX\_BPC=0，ID\_Pcadd4=0，证明IF\_ID寄存器和ID\_EX寄存器成功清零，而寄存器存入结果的时机又证明addi的发生时机是符合预期的，因此这类控制冒险正确。

1. J指令的控制冒险：我们在第16条指令产生了这个冒险，具体原理和分支控制冒险类似，并且我们同样由寄存器结果的存入时机可知andi延后了两个时钟周期执行。仿真结果如下：



1. 第二十二个时钟周期



1. 第二十三个时钟周期

我们可以看到第二十二个时钟周期时，condep=1，Pcsrc=3， IF\_Result的值和EX\_JPC的值相等，证明选择没有错误；而在16个时钟周期结束时的上升沿EX\_BPC=0，ID\_Pcadd4=0，证明IF\_ID寄存器和ID\_EX寄存器成功清零，而寄存器存入结果的时机又证明andi的发生时机是符合预期的，因此这类控制冒险顺利解决。