流水线设计

一、 实验要求（请写出完成实验的具体要求）

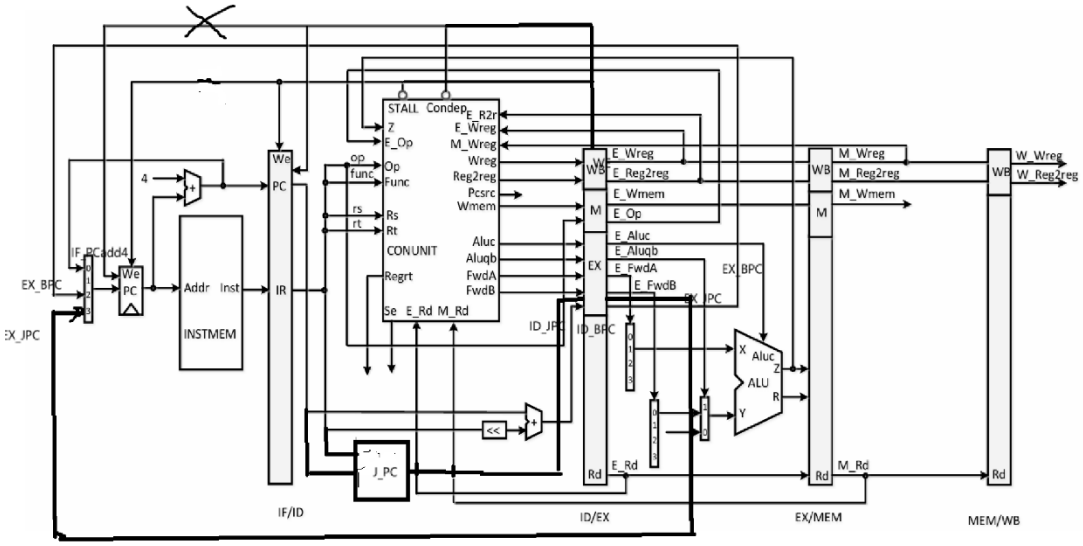
1、至少支持 add、sub、and、or、addi、andi、ori、lw、sw、beq、bne 和 j 十二条指令。

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。

二、 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。



a） 原理图

三、 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
```

2. 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
```

3. 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
```

4. 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
```

5. 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
```

6. 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
```

7. 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
```

8. 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
```

9. 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
```

10. 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
```

11.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;
```

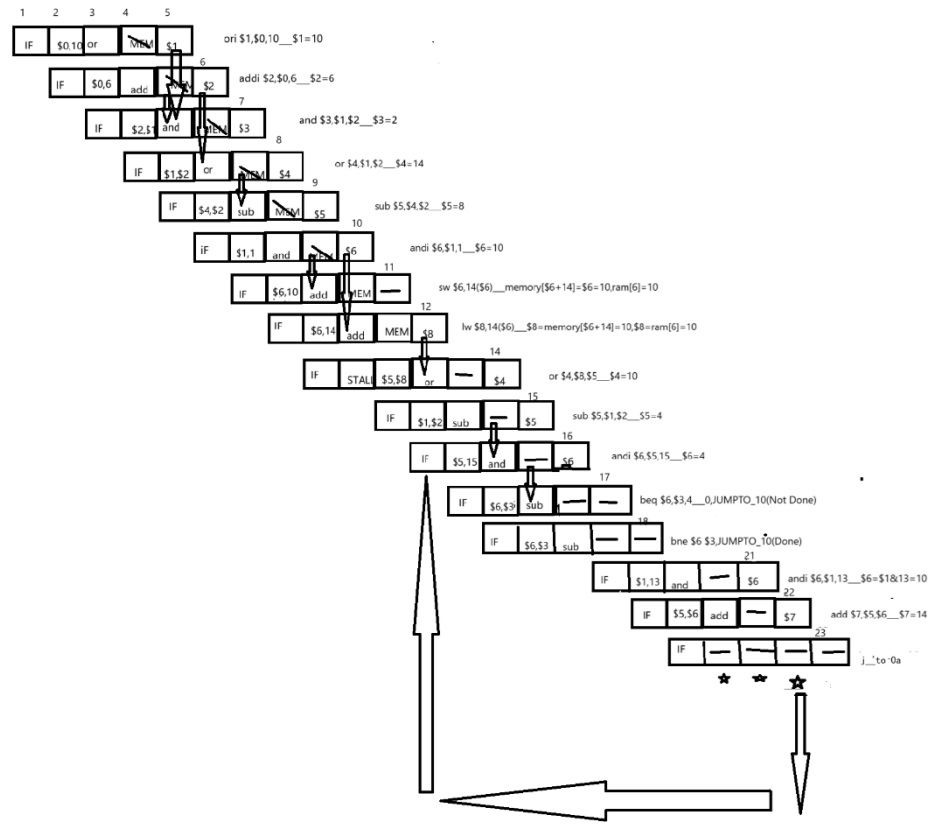
12.endmodule 测试文件:

```
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_R
d(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

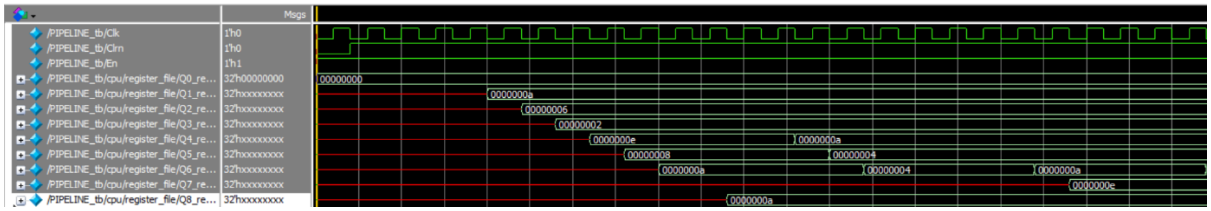
四、 仿真结果（请给出仿真波形截图和对应的结果分析阐述）

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：

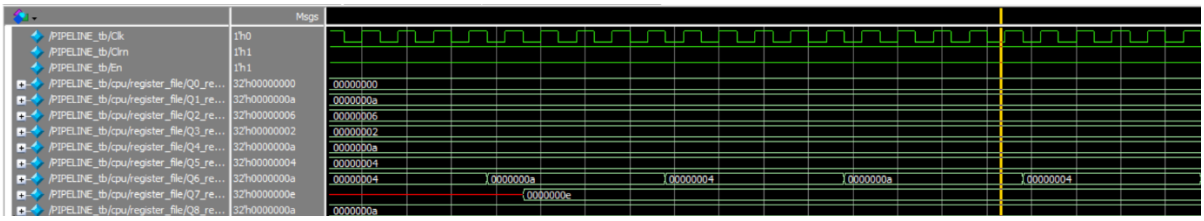


b） 流水线模型机

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：



c） 寄存器值图一



d) 寄存器图二

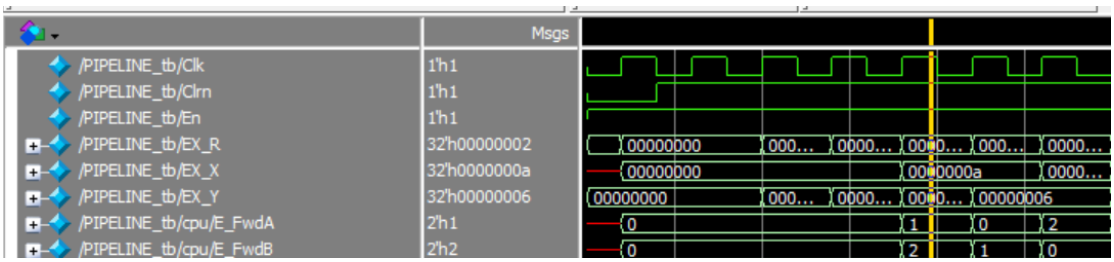
先解释一下时钟周期的问题，我们是从 20ns 开始第一个时钟周期，但我们可以发现 Clrn 信号此时为 0，这是我们进行了置零操作，也就是 PC 的输入要为 0，也就是说可以当作我们这时候开始了第一个时钟周期！注意在这里=为第一次赋值，->表示发生变化，首先看寄存器图一，我们可以清楚地看到$0=0(一直保持)，$1=10（在第五个始终周期下降沿），$2=6（在第六个时钟周期下降沿），$3=2（在第七个时钟周期下降沿），$4=14（在第八个时钟周期下降沿），$5=8（在第九个时钟周期下降沿），$6=10（在第十个时钟周期下降沿），$8=10（在第十二个时钟周期下降沿），$4->10（在第十四个时钟周期下降沿），$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：

(一) 数据冒险

1. 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。
2. 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。
3. 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。

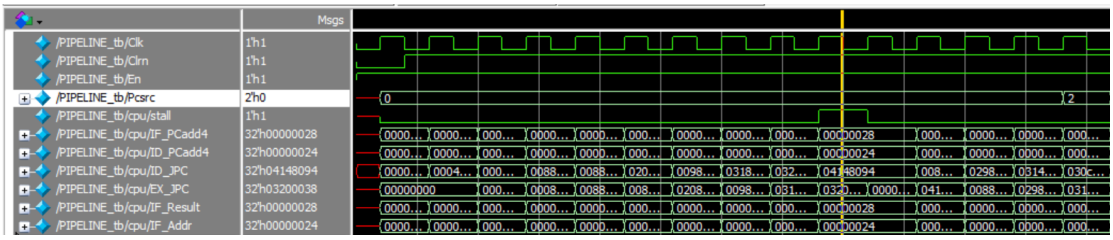


e） 数据冒险图一

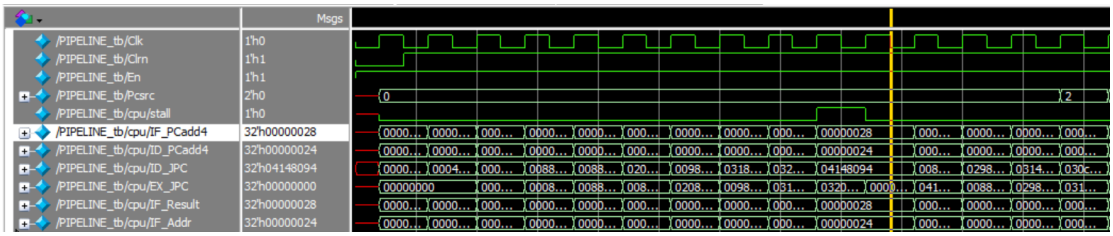
按照预期，在第五个时钟周期，也就是我们黄线的周期，第二条指令的$2 要前推到 EX_X，第一条指令的$1 要前推到 EX_Y，那么 FwdA 要为 01（也就是 1），FwdB 要为 10（也就是 2），可以看到仿真结果符合预期，而 EX_X=10 也就是$2 的值，EX_Y=6 也就是$1 的值，运算结果 EX_R=2，这些结果符合预期，证明我们的数据冒险全部得到解决。

4. 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。



f） 第十个时钟周期



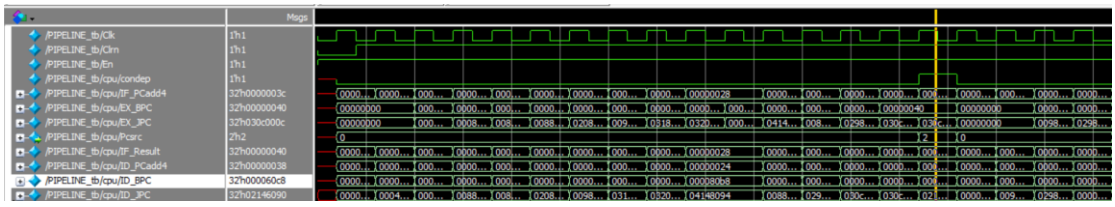
g） 第十一个时钟周期

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。

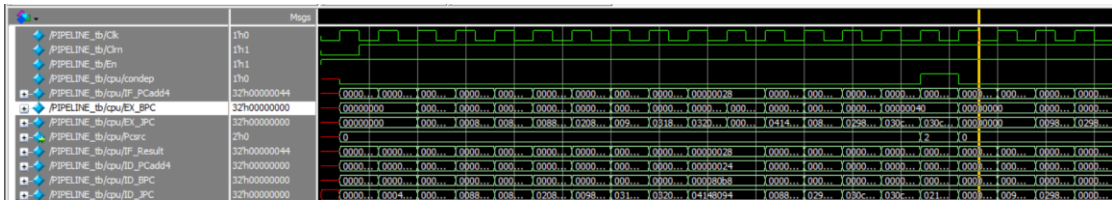
(二) 控制冒险

5. 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：



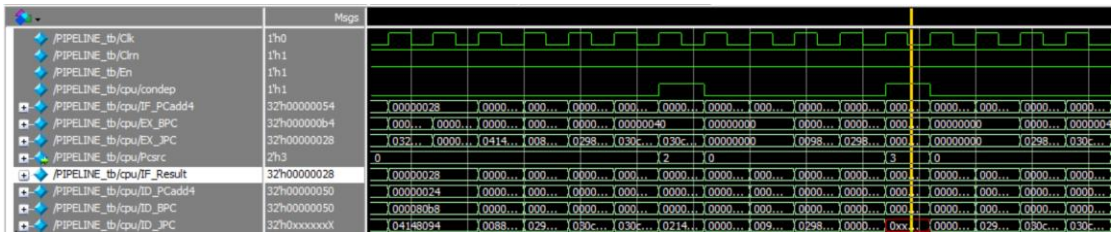
h） 第十六个时钟周期



i） 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。

6. 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：



j） 第二十二个时钟周期

k） 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。