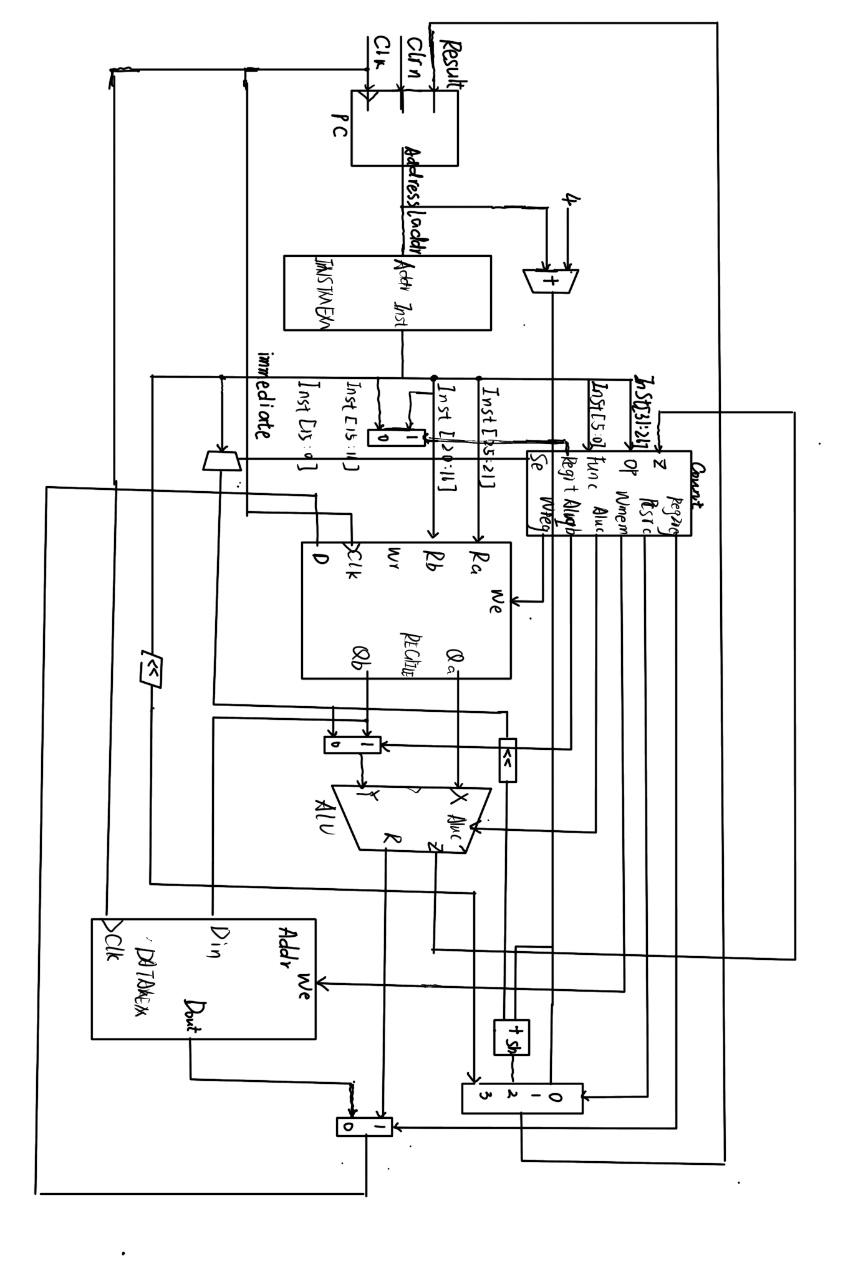
单周期CPU设计

## 实验要求（请写出完成实验的具体要求）

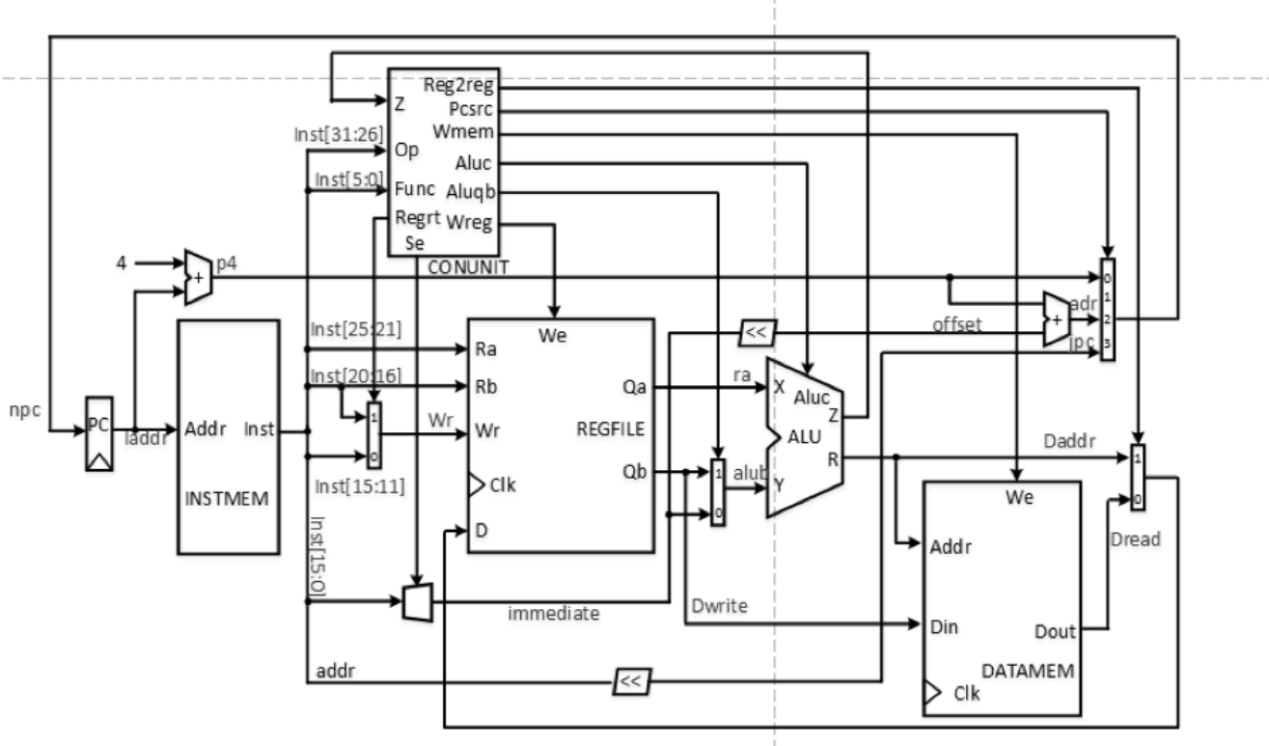
1. 完成单周期处理器所有模块的拼接、代码调试和完成仿真测试，支持指令存储器里所有指令的执行，能通过观察仿真信号判断指令是否正常执行。
2. 至少支持add、sub、and、or、addi、andi、ori、lw、sw、beq、bne和j十二条指令。

## 设计结构（请画出设计的主要处理器结构原理图）

其中分为自己画的（图a）和参考图（图b），自己画的把PC寄存器完整了一下。



1. 自己画的结构图



1. 参考图

设计的思路是按照书上的路径，最最主要的模块是ALU、指令存储器、寄存器堆、数据存储器、PC寄存器以及控制部件。

## 主要实现代码（请给出主要的/核心的实现代码）

* 1. 四位加法器（CLA\_4）：

module CLA\_4 (A, B, Cin, S,Cout );

input [3:0] A;

input [3:0] B;

input Cin;

output [3:0] S;

output Cout;

wire P0, P1, P2, P3;

wire G0, G1, G2, G3;

wire C1, C2, C3;

assign P0 = A[0] ^ B[0];

assign P1 = A[1] ^ B[1];

assign P2 = A[2] ^ B[2];

assign P3 = A[3] ^ B[3];

assign G0 = A[0] & B[0];

assign G1 = A[1] & B[1];

assign G2 = A[2] & B[2];

assign G3 = A[3] & B[3];

assign C1 = G0 | (P0 & Cin);

assign C2 = G1 | (P1 & G0) | (P1 & Cin);

assign C3 = G2 | (P2 & G1) | (P2 & P1 & Cin);

assign Cout = G3 | (P3 & G2) | (P3 & P1 & Cin) | (P3 & P0 & G0);

assign S[0] = P0 ^ Cin;

assign S[1] = P1 ^ C1;

assign S[2] = P2 ^ C2;

assign S[3] = P3 ^ C3;

endmodule

* 1. 32位加减法器（ADDSUB\_32）：

module ADDSUB\_32(X, Y, Sub, S);

input [31:0] X, Y;

input Sub;

output [31:0] S;

wire Cout0, Cout1, Cout2, Cout3, Cout4, Cout5, Cout6, Cout7;

CLA\_4 adder0(X[3:0], Y[3:0] ^ {4{Sub}}, Sub, S[3:0], Cout0);

CLA\_4 adder1(X[7:4], Y[7:4] ^ {4{Sub}}, Cout0, S[7:4], Cout1);

CLA\_4 adder2(X[11:8], Y[11:8] ^ {4{Sub}}, Cout1, S[11:8], Cout2);

CLA\_4 adder3(X[15:12], Y[15:12] ^ {4{Sub}}, Cout2, S[15:12], Cout3);

CLA\_4 adder4(X[19:16], Y[19:16] ^ {4{Sub}}, Cout3, S[19:16], Cout4);

CLA\_4 adder5(X[23:20], Y[23:20] ^ {4{Sub}}, Cout4, S[23:20], Cout5);

CLA\_4 adder6(X[27:24], Y[27:24] ^ {4{Sub}}, Cout5, S[27:24], Cout6);

CLA\_4 adder7(X[31:28], Y[31:28] ^ {4{Sub}}, Cout6, S[31:28], Cout7);

endmodule

* 1. 移位器：

module SHIFTER (in, Sa, Arith, Right, out);

input [31:0] in;

input [4:0] Sa;

input Arith;

input Right;

output [31:0] out;

wire [31:0] T4, S4, T3, S3, T2, S2, T1, S1, T0;

wire [31:0] L1u, L1d, L2u, L2d, L3u, L3d, L4u, L4d, L5u, L5d;

wire a = in[31] & Arith;

wire [15:0] e = {16{a}};

assign L1u = {in[15:0], 16'b0};

assign L1d = {e, in[31:16]};

MUX2X32 M11(L1u, L1d, Right, T4);

MUX2X32 M1r(in, T4, Sa[4], S4);

assign L2u = {S4[23:0], 8'b0};

assign L2d = {e[7:0], S4[31:8]};

MUX2X32 M21(L2u, L2d, Right, T3);

MUX2X32 M2r(S4, T3, Sa[3], S3);

assign L3u = {S3[27:0], 4'b0};

assign L3d = {e[3:0], S3[31:4]};

MUX2X32 M31(L3u, L3d, Right, T2);

MUX2X32 M3r(S3, T2, Sa[2], S2);

assign L4u = {S2[29:0], 2'b0};

assign L4d = {e[1:0], S2[31:2]};

MUX2X32 M41(L4u, L4d, Right, T1);

MUX2X32 M4r(S2, T1, Sa[1], S1);

assign L5u = {S1[30:0], 1'b0};

assign L5d = {e[0], S1[31:1]};

MUX2X32 M51(L5u, L5d, Right, T0);

MUX2X32 M5r(S1, T0, Sa[0], out);

endmodule

* 1. 32位左移两位移位器：

module SHIFTER32(in, out);

input [31:0]in;

output [31:0]out;

SHIFTER(in, 5'b00010, 1, 0, out);

endmodule

* 1. 32位拓展器：

module EXT16T32(X, Se, Y);

input[15:0] X;

input Se;

output[32:0]Y;

wire [31:0] E0, E1;

wire [15:0] e = {16{X[15]}};

parameter z = 16'b0;

assign E0 = {z, X};

assign E1 = {e, X};

MUX2X32 i(E0,E1, Se, Y);

endmodule

* 1. 32位二选一选择器（MUX2X32）:

module MUX2X32 (

input [31:0] a0,

input [31:0] a1,

input sel,

output [31:0] c

);

function [31:0] select;

input [31:0] a0, a1;

input S;

begin

case(S)

1'b0: select = a0;

1'b1: select = a1;

default: select = 32'h00000000;

endcase

end

endfunction

assign c = select(a0, a1, sel);

endmodule

* 1. 五位二选一选择器：

module MUX2X5(A0,A1,S,Y);

input [4:0] A0,A1;

input S;

output [4:0] Y;

function [4:0] select;

input [4:0] A0,A1;

input S;

case(S)

0:select=A0;

1:select=A1;

endcase

endfunction

assign Y=select(A0,A1,S);

endmodule

* 1. 32位4选1选择器：

module MUX4X32 (A0, A1, A2, A3, S, Y);

input [31:0] A0, A1, A2, A3;

input [1:0] S;

output [31:0] Y;

function [31:0] select;

input [31:0] A0, A1, A2, A3;

input [1:0] S;

case(S)

2'b00: select = A0;

2'b01: select = A1;

2'b10: select = A2;

2'b11: select = A3;

endcase

endfunction

assign Y = select (A0, A1, A2, A3, S);

endmodule

* 1. 32位译码器：

module DEC5T32E(

input [4:0] I,

input En,

output reg [31:0] Y

);

always @(\*) begin

if (En) begin

Y = 32'b0;

Y[I] = 1'b1;

end else begin

Y = 32'b0;

end

end

endmodule

* 1. PC寄存器：

module PC(Clk,Reset,R,Address);

input Clk,Reset;

input [31:0]R;

output reg[31:0]Address;

initial begin

Address <= 0;

end

always @(posedge Clk or negedge Reset)

begin

if (!Reset)

begin

Address <= 0;

end

else

begin

Address = R;

end

end

endmodule

* 1. PC+4运算器：

module PCadd4(PC\_o,PCadd4);

input [31:0] PC\_o;

output [31:0] PCadd4;

ADDSUB\_32 addsub32(PC\_o,4,0, PCadd4);

Endmodule

* 1. ALU：

module ALU(X,Y,Aluc,R,Z);

input [31:0]X,Y;

input [1:0]Aluc;

output [31:0]R;

output Z;

wire [31:0] d\_as, d\_and, d\_or, d\_and\_or;

ADDSUB\_32 as32(X, Y, Aluc[0], d\_as);

assign d\_and = X & Y;

assign d\_or = X | Y;

MUX2X32 select1(d\_and, d\_or, Aluc[0], d\_and\_or);

MUX2X32 select2(d\_as, d\_and\_or, Aluc[1], R);

assign Z = ~|R;

endmodule

* 1. 数据存储器：

module DATAMEM(Addr,Din,Clk,We,Dout);

input [31:0]Addr,Din;

input Clk,We;

output [31:0]Dout;

reg [31:0] Ram[31:0];

assign Dout = Ram[Addr[6:2]];

always @(posedge Clk) begin

if(We) Ram[Addr[6:2]] <= Din;

end

integer i;

initial begin

for(i = 0; i < 32; i = i + 1)

Ram[i] = 0;

end

endmodule

* 1. D锁存器：

module D\_Latch(D, En, Q, Qn);

input D, En;

output Q, Qn;

wire Sn, Rn, Dn;

not i0(Dn, D);

nand i1(Sn, D, En);

nand i2(Rn, En, Dn);

nand i3(Q, Sn, Qn);

nand i4(Qn, Q, Rn);

endmodule

* 1. D触发器：

module D\_FF(D, Clk, Q, Qn);

input D, Clk;

output Q, Qn;

wire Clkn, Q0, Qn0;

not i0(Clkn, Clk);

D\_Latch d0(D, Clkn, Q0, Qn0);

D\_Latch d1(Q0, Clk, Q, Qn);

endmodule

* 1. D寄存器：

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

input D, Clk, En, Clrn;

output Q, Qn;

wire Y0, Y\_C;

MUX2X1 m0(.a0(Q), .a1(D), .sel(En), .c(Y0));

and i0(Y\_C, Y0, Clrn);

D\_FF d0(.D(Y\_C), .Clk(Clk), .Q(Q), .Qn(Qn));

endmodule

* 1. 32位寄存器：

module D\_FFEC32(D, Clk, En, Clrn, Q, Qn);

input [31:0] D;

input Clk, En, Clrn;

output [31:0] Q;

output [31:0] 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]);

D\_FFEC d6(D[6], Clk, En, Clrn, Q[6], Qn[6]);

D\_FFEC d7(D[7], Clk, En, Clrn, Q[7], Qn[7]);

D\_FFEC d8(D[8], Clk, En, Clrn, Q[8], Qn[8]);

D\_FFEC d9(D[9], Clk, En, Clrn, Q[9], Qn[9]);

D\_FFEC d10(D[10], Clk, En, Clrn, Q[10], Qn[10]);

D\_FFEC d11(D[11], Clk, En, Clrn, Q[11], Qn[11]);

D\_FFEC d12(D[12], Clk, En, Clrn, Q[12], Qn[12]);

D\_FFEC d13(D[13], Clk, En, Clrn, Q[13], Qn[13]);

D\_FFEC d14(D[14], Clk, En, Clrn, Q[14], Qn[14]);

D\_FFEC d15(D[15], Clk, En, Clrn, Q[15], Qn[15]);

D\_FFEC d16(D[16], Clk, En, Clrn, Q[16], Qn[16]);

D\_FFEC d17(D[17], Clk, En, Clrn, Q[17], Qn[17]);

D\_FFEC d18(D[18], Clk, En, Clrn, Q[18], Qn[18]);

D\_FFEC d19(D[19], Clk, En, Clrn, Q[19], Qn[19]);

D\_FFEC d20(D[20], Clk, En, Clrn, Q[20], Qn[20]);

D\_FFEC d21(D[21], Clk, En, Clrn, Q[21], Qn[21]);

D\_FFEC d22(D[22], Clk, En, Clrn, Q[22], Qn[22]);

D\_FFEC d23(D[23], Clk, En, Clrn, Q[23], Qn[23]);

D\_FFEC d24(D[24], Clk, En, Clrn, Q[24], Qn[24]);

D\_FFEC d25(D[25], Clk, En, Clrn, Q[25], Qn[25]);

D\_FFEC d26(D[26], Clk, En, Clrn, Q[26], Qn[26]);

D\_FFEC d27(D[27], Clk, En, Clrn, Q[27], Qn[27]);

D\_FFEC d28(D[28], Clk, En, Clrn, Q[28], Qn[28]);

D\_FFEC d29(D[29], Clk, En, Clrn, Q[29], Qn[29]);

D\_FFEC d30(D[30], Clk, En, Clrn, Q[30], Qn[30]);

D\_FFEC d31(D[31], Clk, En, Clrn, Q[31], Qn[31]);

endmodule

* 1. 基本寄存器堆模块：

module REG32(D, En, Clk, Clrn, Q31, Q30, Q29, Q28, Q27, Q26, Q25, Q24, Q23, Q22, Q21, Q20, Q19, Q18, Q17, Q16, Q15, Q14, Q13, Q12, Q11, Q10, Q9, Q8, Q7, Q6, Q5, Q4, Q3, Q2, Q1, Q0);

input [31:0] D, En;

input Clk, Clrn;

output [31:0]Q31, Q30, Q29, Q28, Q27, Q26, Q25, Q24, Q23, Q22, Q21, Q20, Q19, Q18, Q17, Q16, Q15, Q14, Q13, Q12, Q11, Q10, Q9, Q8, Q7, Q6, Q5, Q4, Q3, Q2, Q1, Q0;

D\_FFEC32 q31(D, Clk, En[31], Clrn, Q31);

D\_FFEC32 q30(D, Clk, En[30], Clrn, Q30);

D\_FFEC32 q29(D, Clk, En[29], Clrn, Q29);

D\_FFEC32 q28(D, Clk, En[28], Clrn, Q28);

D\_FFEC32 q27(D, Clk, En[27], Clrn, Q27);

D\_FFEC32 q26(D, Clk, En[26], Clrn, Q26);

D\_FFEC32 q25(D, Clk, En[25], Clrn, Q25);

D\_FFEC32 q24(D, Clk, En[24], Clrn, Q24);

D\_FFEC32 q23(D, Clk, En[23], Clrn, Q23);

D\_FFEC32 q22(D, Clk, En[22], Clrn, Q22);

D\_FFEC32 q21(D, Clk, En[21], Clrn, Q21);

D\_FFEC32 q20(D, Clk, En[20], Clrn, Q20);

D\_FFEC32 q19(D, Clk, En[19], Clrn, Q19);

D\_FFEC32 q18(D, Clk, En[18], Clrn, Q18);

D\_FFEC32 q17(D, Clk, En[17], Clrn, Q17);

D\_FFEC32 q16(D, Clk, En[16], Clrn, Q16);

D\_FFEC32 q15(D, Clk, En[15], Clrn, Q15);

D\_FFEC32 q14(D, Clk, En[14], Clrn, Q14);

D\_FFEC32 q13(D, Clk, En[13], Clrn, Q13);

D\_FFEC32 q12(D, Clk, En[12], Clrn, Q12);

D\_FFEC32 q11(D, Clk, En[11], Clrn, Q11);

D\_FFEC32 q10(D, Clk, En[10], Clrn, Q10);

D\_FFEC32 q9(D, Clk, En[9], Clrn, Q9);

D\_FFEC32 q8(D, Clk, En[8], Clrn, Q8);

D\_FFEC32 q7(D, Clk, En[7], Clrn, Q7);

D\_FFEC32 q6(D, Clk, En[6], Clrn, Q6);

D\_FFEC32 q5(D, Clk, En[5], Clrn, Q5);

D\_FFEC32 q4(D, Clk, En[4], Clrn, Q4);

D\_FFEC32 q3(D, Clk, En[3], Clrn, Q3);

D\_FFEC32 q2(D, Clk, En[2], Clrn, Q2);

D\_FFEC32 q1(D, Clk, En[1], Clrn, Q1);

D\_FFEC32 q0(D, Clk, En[0], Clrn, Q0);

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

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

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'b000010\_00000\_00000\_00000\_00000\_001100;//j 01100\_\_\_\_0C

assign Rom[5'h07]=32'hXXXXXXXX;

assign Rom[5'h08]=32'hXXXXXXXX;

assign Rom[5'h09]=32'hXXXXXXXX;

assign Rom[5'h0A]=32'hXXXXXXXX;

assign Rom[5'h0B]=32'hXXXXXXXX;

assign Rom[5'h0C]=32'b000100\_00001\_00010\_00000\_00000\_000100;//beq $1,$2,4\_\_\_0D

assign Rom[5'h0D]=32'b000101\_00001\_00011\_00000\_00000\_000100;// bne $1 $3,4\_\_\_12

assign Rom[5'h0E]=32'hXXXXXXXX;

assign Rom[5'h0F]=32'hXXXXXXXX;

assign Rom[5'h10]=32'hXXXXXXXX;

assign Rom[5'h11]=32'hXXXXXXXX;

assign Rom[5'h12]=32'b000000\_00101\_00110\_00111\_00000\_100000;//add $7,$5,$6\_\_\_$7=18

assign Rom[5'h13]=32'b101011\_00111\_00110\_00000\_00000\_001010;//sw $6,10($7)\_\_\_memory[$7+10]=$6=10

assign Rom[5'h14]=32'b100011\_00111\_01000\_00000\_00000\_001010;//lw $8,10($7)\_\_\_$8=memory[$7+10]=10

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. 控制部件：

module CONUNIT(Op,Func,Z,Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg);

input [5:0]Op,Func;

input Z;

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

output [1:0]Pcsrc; output [1:0]Aluc;

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

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]=I\_beq&Z|I\_bne&~Z|I\_J;

assign Pcsrc[0]=I\_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;

endmodule

* 1. REGFILE:

module REGFILE(Ra,Rb,D,Wr,We,Clk,Clrn,Qa,Qb);

input [4:0]Ra,Rb,Wr;

input [31:0]D;

input We,Clk,Clrn;

output [31:0]Qa,Qb;

wire [31:0]Y\_mux,Q31\_reg32,Q30\_reg32,Q29\_reg32,Q28\_reg32,Q27\_reg32,Q26\_reg32,Q25\_reg32,Q24\_reg32,Q23\_reg32,Q22\_reg32,Q21\_reg32,Q20\_reg32,Q19\_reg32,Q18\_reg32,Q17\_reg32,Q16\_reg32,Q15\_reg32,Q14\_reg32,Q13\_reg32,Q12\_reg32,Q11\_reg32,Q10\_reg32,Q9\_reg32,Q8\_reg32,Q7\_reg32,Q6\_reg32,Q5\_reg32,Q4\_reg32,Q3\_reg32,Q2\_reg32,Q1\_reg32,Q0\_reg32;

DEC5T32E dec(Wr,We,Y\_mux);

REG32 A(D,Y\_mux,Clk,Clrn,Q31\_reg32,Q30\_reg32,Q29\_reg32,Q28\_reg32,Q27\_reg32,Q26\_reg32,Q25\_reg32,Q24\_reg32,Q23\_reg32,Q22\_reg32,Q21\_reg32,Q20\_reg32,Q19\_reg32,Q18\_reg32,Q17\_reg32,Q16\_reg32,Q15\_reg32,Q14\_reg32,Q13\_reg32,Q12\_reg32,Q11\_reg32,Q10\_reg32,Q9\_reg32,Q8\_reg32,Q7\_reg32,Q6\_reg32,Q5\_reg32,Q4\_reg32,Q3\_reg32,Q2\_reg32,Q1\_reg32,Q0\_reg32);

MUX32X32 select1(Q0\_reg32,Q1\_reg32,Q2\_reg32,Q3\_reg32,Q4\_reg32,Q5\_reg32,Q6\_reg32,Q7\_reg32,Q8\_reg32,Q9\_reg32,Q10\_reg32,Q11\_reg32,Q12\_reg32,Q13\_reg32,Q14\_reg32,Q15\_reg32,Q16\_reg32,Q17\_reg32,Q18\_reg32,Q19\_reg32,Q20\_reg32,Q21\_reg32,Q22\_reg32,Q23\_reg32,Q24\_reg32,Q25\_reg32,Q26\_reg32,Q27\_reg32,Q28\_reg32,Q29\_reg32,Q30\_reg32,Q31\_reg32,Ra,Qa);

MUX32X32 select2(Q0\_reg32,Q1\_reg32,Q2\_reg32,Q3\_reg32,Q4\_reg32,Q5\_reg32,Q6\_reg32,Q7\_reg32,Q8\_reg32,Q9\_reg32,Q10\_reg32,Q11\_reg32,Q12\_reg32,Q13\_reg32,Q14\_reg32,Q15\_reg32,Q16\_reg32,Q17\_reg32,Q18\_reg32,Q19\_reg32,Q20\_reg32,Q21\_reg32,Q22\_reg32,Q23\_reg32,Q24\_reg32,Q25\_reg32,Q26\_reg32,Q27\_reg32,Q28\_reg32,Q29\_reg32,Q30\_reg32,Q31\_reg32,Rb,Qb);

endmodule

* 1. J型指令拼装：

module SHIFTER\_COMBINATION(X,PCADD4,Sh);

input [25:0] X;

input [31:0] PCADD4;

output [31:0] Sh;

parameter z=2'b00;

assign Sh={PCADD4[31:28],X[25:0],z};

endmodule

* 1. CPU封装：

module CPU(Clk,Reset,Addr,Inst,Qa,Qb,ALU\_R,NEXTADDR, D);

input Clk,Reset;

output [31:0] Inst,NEXTADDR,ALU\_R,Qb,Qa,Addr,D;

wire [31:0]Result,PCadd4,EXTIMM,InstL2,EXTIMML2,Y,mux4x32\_2,R,Dout;

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

wire [1:0]Aluc,Pcsrc;

wire [4:0]Wr;

PC pc(Clk,Reset,Result,Addr);

PCadd4 pcadd4(Addr,PCadd4);

INSTMEM instmem(Addr,Inst);

CONUNIT conunit(Inst[31:26],Inst[5:0],Z,Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg);

MUX2X5 mux2x5(Inst[15:11],Inst[20:16],Regrt,Wr);

EXT16T32 ext16t32(Inst[15:0],Se,EXTIMM);

SHIFTER\_COMBINATION shifter1(Inst[26:0],PCadd4,InstL2);

SHIFTER32 shifter2(EXTIMM,EXTIMML2);

REGFILE regfile(Inst[25:21],Inst[20:16],D,Wr,Wreg,Clk,Reset,Qa,Qb);

MUX2X32 mux2x321(EXTIMM,Qb,Aluqb,Y);

ALU alu(Qa,Y,Aluc,R,Z);

DATAMEM datamem(R,Qb,Clk,Wmem,Dout);

ADDSUB\_32 addsub\_32(PCadd4,EXTIMML2,0,mux4x32\_2);

MUX2X32 mux2x322(Dout,R,Reg2reg,D);

MUX4X32 mux4x32(PCadd4,'b0,mux4x32\_2,InstL2,Pcsrc,Result);

assign NEXTADDR=Result;

assign ALU\_R=R;

endmodule

* 1. 仿真代码：

`timescale 10ns / 1ps

module CPU\_tb;

reg Clk;

reg Reset;

wire [31:0] Addr, Inst, Qa, Qb, ALU\_R, NEXTADDR, D;

CPU cpu(Clk,Reset,Addr,Inst,Qa,Qb,ALU\_R,NEXTADDR, D);

initial begin

Clk=0;

Reset=0;

#10;

Reset=1;

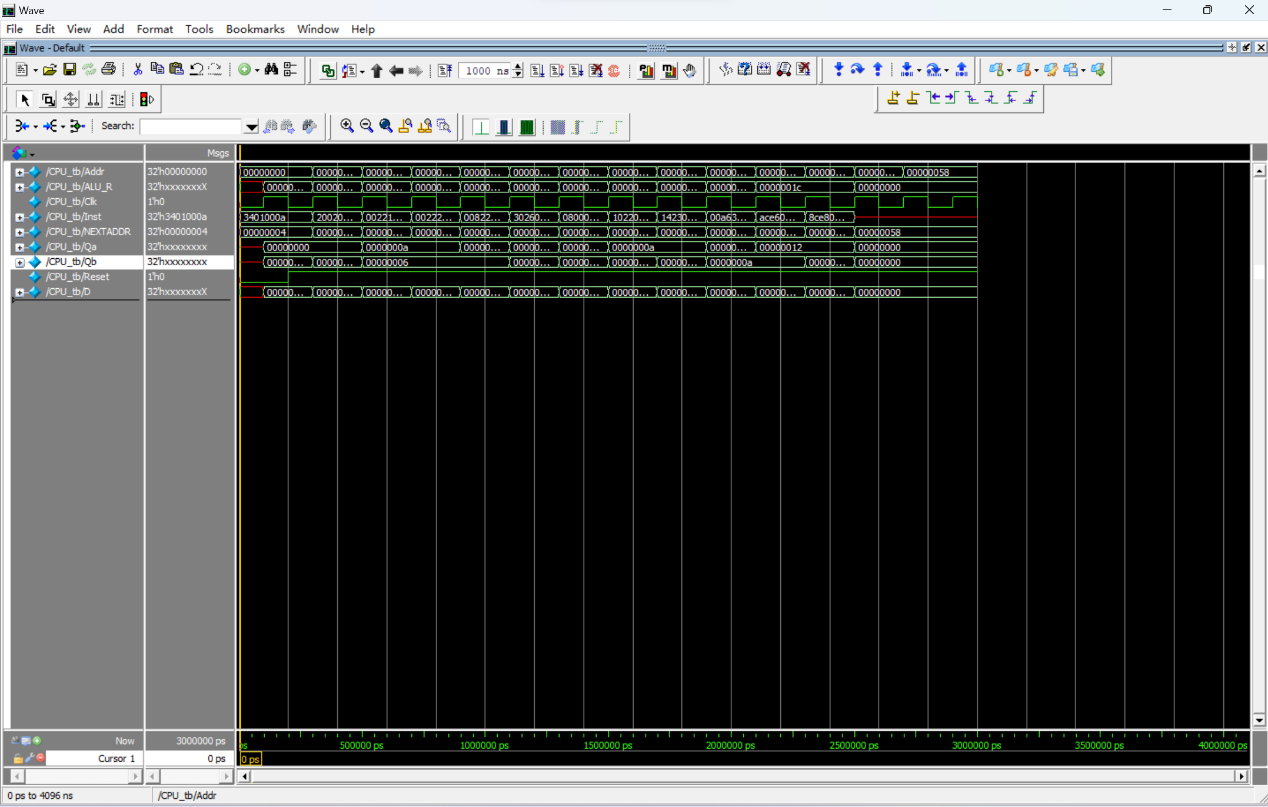
end

always #10 Clk=~Clk;

endmodule

## 仿真结果（请给出仿真波形截图和对应的结果分析阐述）

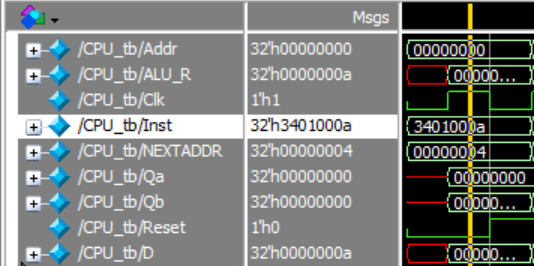
最终波形如下：



1. 总体波形

下面进行每条指令的分析：

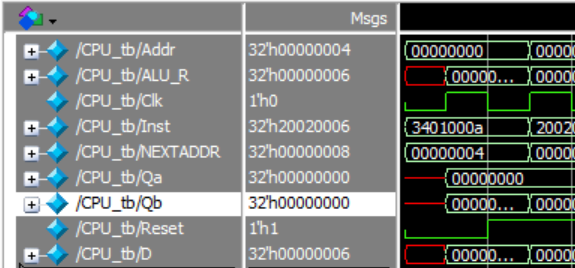
第一条指令：32'b001101\_00000\_00001\_00000\_00000\_001010;//ori $1,$0,10\_\_\_$1=10



1. Ori指令

我们应该关注的是ALU\_R和D，可以看到ALU计算出来的结果为10（a转换为十进制后），写回寄存器堆的结果（也就是D）也为10，与预期结果相符。

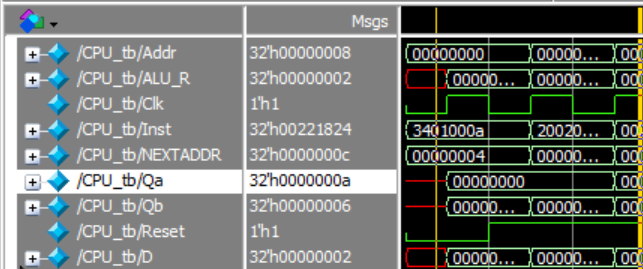
第二条指令：32'b001000\_00000\_00010\_00000\_00000\_000110;//addi $2,$0,6\_\_\_$2=6



1. Addi指令

我们应该关注的是ALU\_R和D，可以看到计算出来的的结果和写回寄存器堆的结果为6，与预期相符。

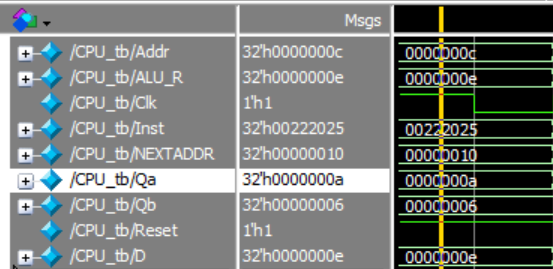
第三条指令：32'b000000\_00001\_00010\_00011\_00000\_100100;//and $3,$1,$2\_\_\_$3=2



1. And指令

与前面相同，ALU\_R和D的值为2，与与其计算结果相符。

第四条指令：32'b000000\_00001\_00010\_00100\_00000\_100101;//or $4,$1,$2\_\_\_$4=14



1. Or指令

与前面相同，ALU\_R和D的结果为14，与预期结果相同。

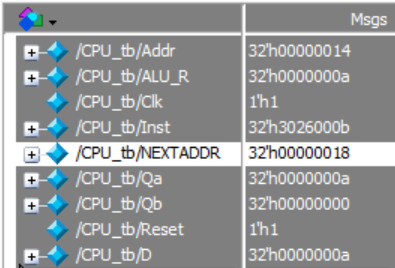
第五条指令：32'b000000\_00100\_00010\_00101\_00000\_100010;//sub $5,$4,$2\_\_\_$5=8



1. Sub指令

与前面相同，ALU\_R和D的结果为8，这与预期计算结果相同。

第六条指令：32'b001100\_00001\_00110\_00000\_00000\_001011;//andi $6,$1,1\_\_\_$6=10



1. Addi指令

与前面相同，ALU\_R和D的结果为10，这与预期结果相符合。

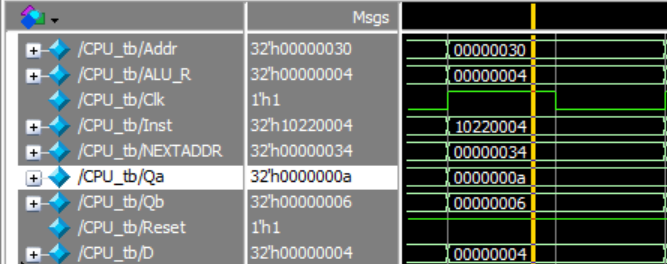
第七条指令：32'b000010\_00000\_00000\_00000\_00000\_001100;//j 01100\_\_\_\_0C



1. J指令

这条指令我们应该关注的是NEXTADDR，也就是PC的下一个地址，我们得到的结果为00000030（16进制），而预期结果的计算过程如下。此时PC地址为00000018（16进制），加4之后为0000001c（16进制），最高四位（2进制）为0000，而J指令的address字段左移两位与0000拼接后得到的值为\_\_（省略26个0)110000，转换为16进制得到为00000030，与实际结果相符合。

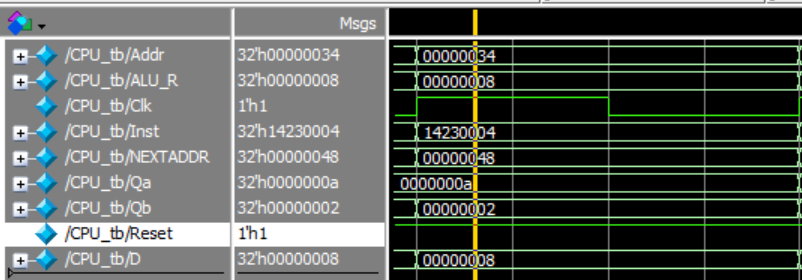
第八条指令：32'b000100\_00001\_00010\_00000\_00000\_000100;//beq $1,$2,4\_\_\_0D



1. Beq指令

这是相等跳转指令，我们从寄存器堆中取值为10（Qa）和6（Qb），，可以明显知道不相等，因此不转移，PC下一次接受的地址为PC+4，和NEXTADDR的值00000034相符。

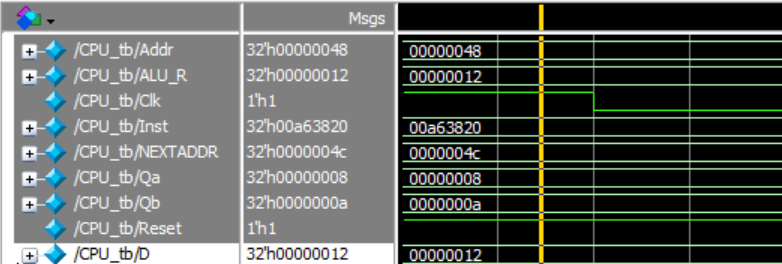
第九条指令：32'b000101\_00001\_00011\_00000\_00000\_000100;// bne $1 $3,4\_\_\_12



1. Bne指令

这是不等跳转指令，可以看到从寄存器中取出的值（Qa和Qb）分别为10和2，不相等，进行跳转，预期的结果计算如下。拓展offset再左移两位得到00000010（16进制）再与PC+4相加得到00000048（16进制），而这与得到的NEXTADDR结果相符合。

第十条指令：000000\_00101\_00110\_00111\_00000\_100000;//add $7,$5,$6\_\_\_$7=18

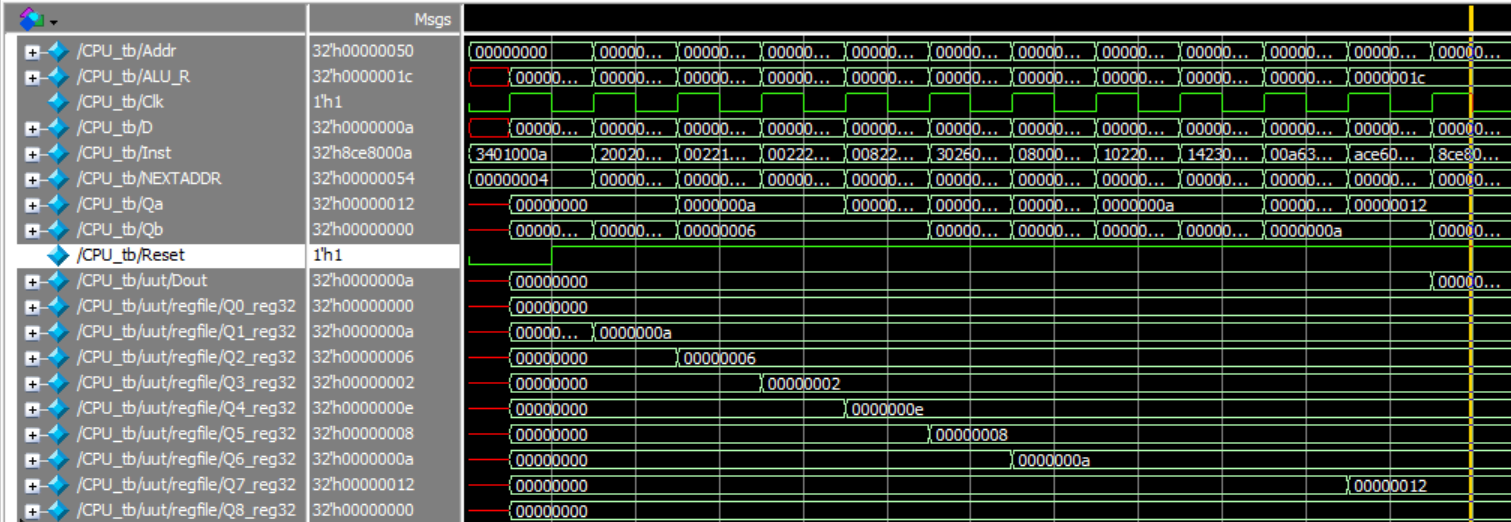


1. And指令

我们关注的ALU\_R和D都为00000012（16进制），转换为10进制之后为18，与预期结果相符合。

第十一条指令：

32'b101011\_00111\_00110\_00000\_00000\_001010;//sw $6,10($7)\_\_\_memory[$7+10]=$6=10

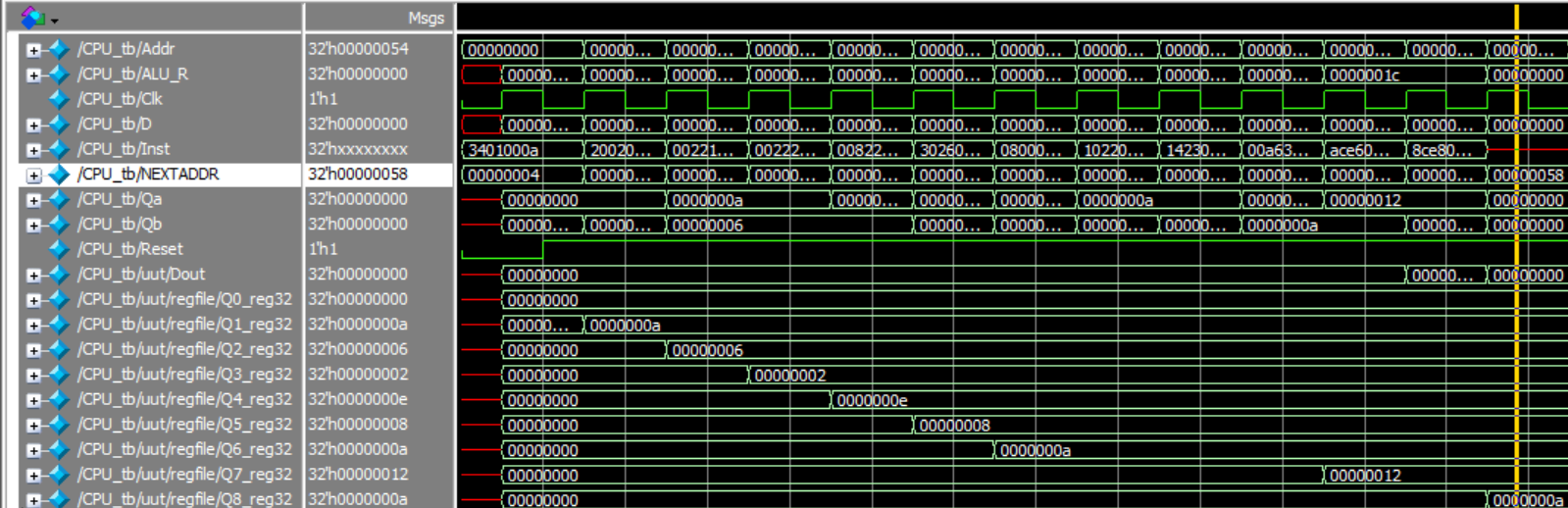


1. Sw指令

这是写存操作，为便于验证，直接看第十二条指令取出来的值，可以发现D为0000000a（16进制），转换为10进制就是10，这与我们预期相符合。在看一下计算出来的存储器地址ALU\_R为000000001c（16进制）为28（10进制），这与我们的预期$7（=18）+10=28（10进制）相符合。

第十二条指令：

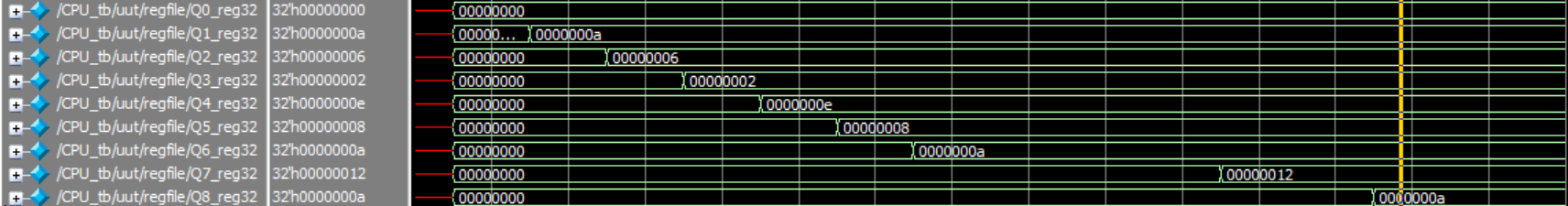
32'b100011\_00111\_01000\_00000\_00000\_001010;//lw $8,10($7)\_\_\_$8=memory[$7+10]=10



1. lw指令

为便于观察，直接查看下一个时钟周期，可以看到$8的值（Q8）为0000000a（16进制），也就是10，可以发现写入的地方和值都是符合预期的。

最后我们看一下各个寄存器的值：



1. 寄存器值

$0（Q0） = 0（初始值）, $1（Q1） = 10， $2（Q2） = 6, $3（Q3） = 2, $4（Q4）= 14， $5（Q5） = 8, $6（Q6） = 10, $7（Q7） = 18， $8（Q8） = 10。这都是符合我们所设定的指令的。