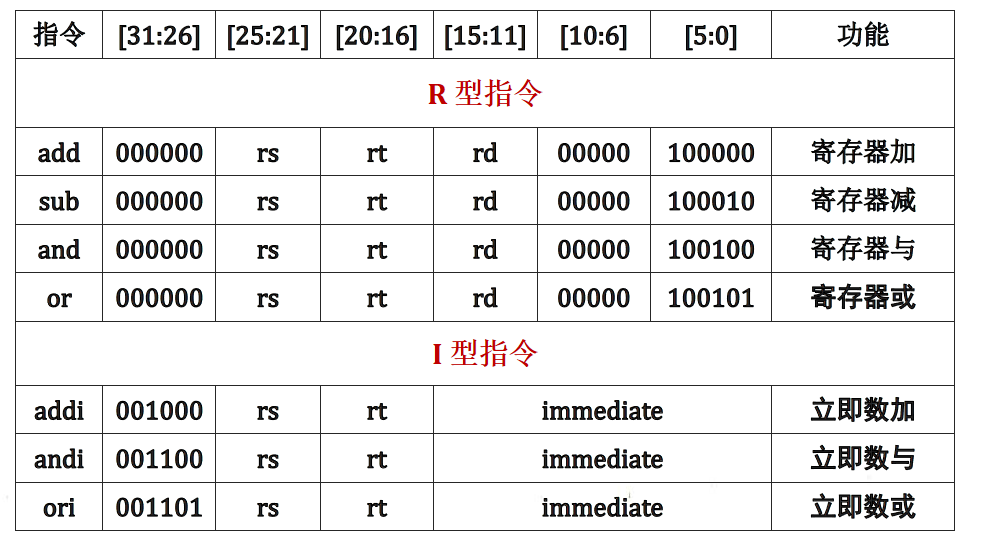
# 单周期CPU

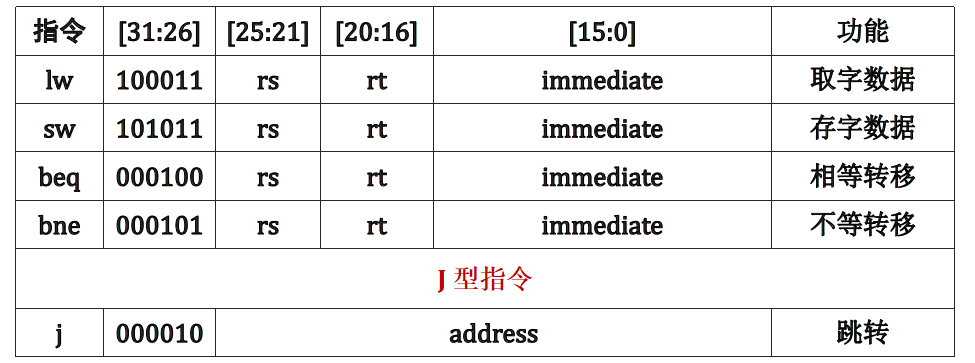
余超逸 刘常乐

## 一、原理

本次单周期CPU根据MIPS指令完成，语言为VERILOG，要求支持add、sub、and、or、addi、andi、ori、lw、sw、beq、bne、j、xor、sll、srl、sra、lui、jal、xori二十条指令。

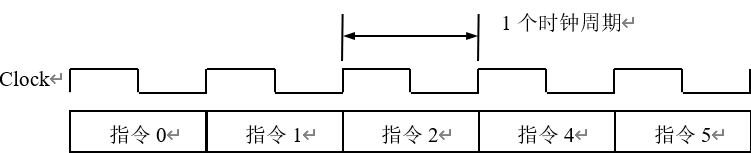
如下图为MIPS指令





本次单周期CPU由数据通路和控制部件两大部分组成。数据通路由组合逻辑部件ALU和状态控制部件、状态储存部件寄存器堆和存储器构成。

单周期（Single Cycle）CPU是指CPU从取出1条指令到执行完该指令只需1个时钟周期。



一条指令的执行过程包括：取指令→分析指令→执行指令→保存结果。对于单周期CPU来说，这些执行步骤均在一个时钟周期内完成

下一条指令的地址有4种情况：

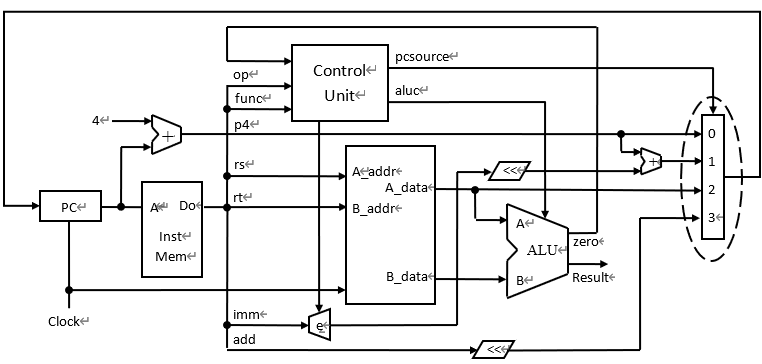
1. 程序不转移时下一条指令的地址为PC+4；

2. 执行beq和bne指令发生转移时，下一条指令的地址是PC加4，再加上符号扩展的偏移量左移2位的和；

3. 执行jr指令时转移的目标地址就是rs寄存器中的内容；

4. 执行j和jal指令时转移的目标地址是指令中的低26位地址左移2位，再与PC+4的高4位拼接在一起。

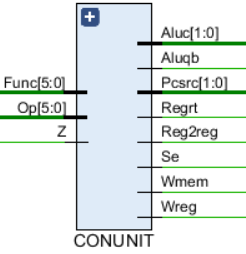
下一条指令地址的产生和选择电路：



在上图中控制器（Control Unit）根据op、func和zero（对于beq和bne指令）信号产生相应的转移控制选择信号pcsource。

## 二、代码

### 1.CONUNIT



**模块功能**

CONUNIT是作为CPU控制信号产生的器件，通过通过解析指令功能段得到该指令的各种控制信号，使其他器件有效或无效或选择信号。

**引脚**

Func：输入信号。

Op：输入信号。

Z：零标志信号，对Pcsrc有影响，输入信号。

Regrt：控制输入寄存器的Wr端口，输出信号。

Se：控制扩展模块，输出信号。

Wreg：控制寄存器端的写使能信号，输出信号。

Aluqb：控制ALU的Y端口的输入值，输出信号。

Aluc：控制ALU的计算种类，输出信号。

Wmem：控制数据存储器的写使能信号，输出信号。

Pcsrc：控制目标指令地址，输出信号。

Reg2reg：控制REHFILE更新值的来源。

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,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\_or = 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\_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\_sw|I\_lw|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;

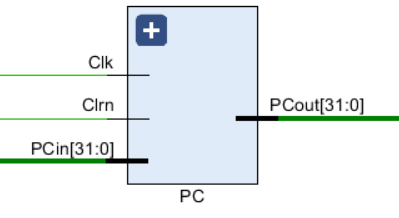
assign Pcsrc[1]=I\_beq&Z|I\_bne&~Z|I\_J;

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

### 2.PC寄存器



**模块功能**

用于给出指令在指令储存器中的地址。

**引脚及控制信号**

Clk：时钟周期，输入信号

Clrn：控制信号，输入信号

PCin：目标地址，可能是跳转地址或者是下一条指令的地址，输入信号

PCout：指令地址，输出信号

module PC(Clk,Clrn,newAddress,currentAddress);

input Clk;

input Clrn;

input [31:0] newAddress;

output reg [31:0] currentAddress;

always@(posedge Clk or negedge Clrn )

if(Clrn==0) begin currentAddress <= 0;

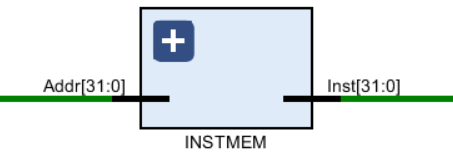
end else begin

currentAddress<= newAddress;

end

endmodule

### 3.INSTMEM



**模块功能**

依据当前PC，读取指令寄存器中相对应地址PCout[6:2]的指令。

**引脚及控制信号**

Addr：指令地址，输入信号

Inst：指令编码，输出信号

module INSTMEM(

input [31:0] Addr,

output [31:0] Inst

);

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

assign Rom[5'h00]=32'h20010008;

assign Rom[5'h01]=32'h3402000c;

assign Rom[5'h02]=32'h00411822;

assign Rom[5'h03]=32'h2004000a;

assign Rom[5'h04]=32'h00442825;

assign Rom[5'h05]=32'h00443024;

assign Rom[5'h06]=32'h00223820;

assign Rom[5'h07]=32'h08000009;

assign Rom[5'h08]=32'h14220001;

assign Rom[5'h09]=32'h1026fffe;

assign Rom[5'h0a]=32'hac03000c;

assign Rom[5'h0b]=32'h8c04000c;

assign Rom[5'h0c]=32'h3048000a;

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

endmodule

module INSTMEM\_TEST;

reg [31:0]Addr;

wire [31:0]Inst;

INSTMEM uut(.Addr(Addr),.Inst(Inst));

initial begin

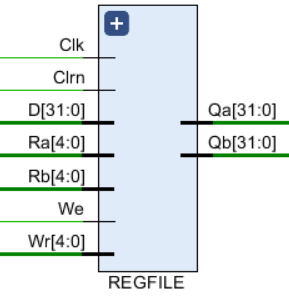
Addr = 32'b00000000000000000000000000000000;

end

always #100Addr=Addr+4;

endmodule

### 4.REGFILE



**模块功能**

给出要读取的两个寄存器编号和要写入的寄存器编号，然后由Qa和Qb端口更新Ra和Rb端口的输入编号分别输入其值。

**引脚及控制信号**

Ra：读取寄存器编号1，输入信号。

Rb：读取寄存器编号2或立即数，输入信号。

D：寄存器更新值，输入信号。

Wr：写入寄存器编号3，输入信号。

We：写使能信号，为0的时候不能写入，D值不更新，为1的时候能写入，D值更新，输入信号。

Clk：时钟周期，输入信号。

Clrn：清零信号，输入信号。

Qa：输出寄存器1的值，输入信号。

Qb：输出寄存器2的值，输入信号。

module INSTMEM(

input [31:0] Addr,

output [31:0] Inst

);

//写入只读指令

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

assign Rom[5'h00]=32'h20010008; //addi r1,r0,8 [r1=8];

assign Rom[5'h01]=32'h3402000c; //ori r2,r0,12 [r2=12];

assign Rom[5'h02]=32'h00411822; //sub r3,r2,r1 [r3=4];

assign Rom[5'h03]=32'h2004000a; //addi r4,r0,10 [r4=10];

assign Rom[5'h04]=32'h00442825; //or r5,r2,r4 [r5=14];

assign Rom[5'h05]=32'h00443024; //and r6,r2,r4 [r6=8];

assign Rom[5'h06]=32'h00223820; //add r7,r1,r2 [r7=20];

assign Rom[5'h07]=32'h08000009; //j 9;

assign Rom[5'h08]=32'h14220001; //bne r1 r2 不等跳转到 10;

assign Rom[5'h09]=32'h1026fffe; //beq r1 r6 相等跳转到 8;

assign Rom[5'h0a]=32'hac03000c; //sw r3,12(r0) [0000000c=4]; dout=4

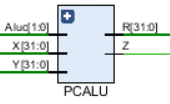
assign Rom[5'h0b]=32'h8c04000c; //lw r4,12(r0) [r4=4];

assign Rom[5'h0c]=32'h3048000a; //andi r8,r2,10 [r8=8];

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

endmodule

### 5.ALU



**模块功能**

算数逻辑部件，需要实现加，减，按位与，按位或。

**引脚及控制信号**

Qa：寄存器1的值。

Y：寄存器2的值或立即数。

Aluc：控制信号。

R：输入寄存器端口D的计算结果，输出信号。

Z：当值为1时代表两个输入信号值相等，当值为0时代表两个输入信号不等，输出信号。

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\_or,d\_and,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

module ALU\_TEST;

// Inputs

reg [31:0]X,Y;

reg [3:0]Aluc ;

// Outputs

wire [31:0]R;

wire Z;

ALU uut (.X(X),.Y(Y),.Aluc(Aluc),.R(R),.Z(Z));

initial begin

X = 0;

Y = 0;

Aluc = 0;

end;

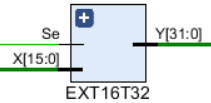
always #5 X=X+1;

always #10 Y=Y+1;

always #20 Aluc=Aluc+1;

endmodule

### 6.EXT16T32



**模块功能**

I指令的addi需要对立即数进行符号拓展，andi和ori需要对立即数进行零扩展，所以需要一个扩展模块。

**引脚及控制信号**

X：I型指令的立即数字段，输入信号。

Se：选择零扩展或是符号扩展的控制模块，输入信号。

Y：扩展后的立即数，输出信号。

module EXT16T32\_(X,Se,Y);

input[15:0] X;

input Se;

output [31:0]Y;

wire [31:0] E0,E1;

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

parameter z= 16'b0;

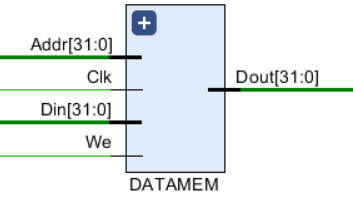
assign E0={z,X};

assign E1={e,X};

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

endmodule

### 7.DATAMEM



**模块功能**

数据存储器，通过控制信号，对数据寄存器进行读或者写操作。

**引脚及控制信号**



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

input [31:0]Addr,Din;

input Clk,We;

output [31:0]Dout;

reg [31:0]ram[0:31];

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

module DATAMEM\_TEST();

reg [31:0]Addr,Din;

reg Clk,We;

wire [31:0]Dout;

DATAMEM uut(.Addr(Addr),.Din(Din),.Clk(Clk),.We(We),.Dout(Dout));

initial begin

Addr=32'b00000000000000000000000000000100;

Din=32'b00000000000000000000000000000111;

Clk=0;

We=1;

end

always #100 Addr=Addr+1;

always #50 Din = Din+1;

always #20 Clk = 1-Clk;

always #10 We=1-We;

endmodule

### 8.CLA\_32

module CLA\_32(X,Y,Cin,S,Cout);

input [31:0] X,Y;

input Cin;

output [31:0]S;

output Cout;

wire cout0,cout1,cout2,cout3,cout4,cout5,cout6;

CLA\_4 add0(X[3:0],Y[3:0],Cin,S[3:0],cout0);

CLA\_4 add1(X[7:4],Y[7:4],cout0,S[7:4],cout1);

CLA\_4 add2(X[11:8],Y[11:8],cout1,S[11:8],cout2);

CLA\_4 add3(X[15:12],Y[15:12],cout2,S[15:12],cout3);

CLA\_4 add4(X[19:16],Y[19:16],cout3,S[19:16],cout4);

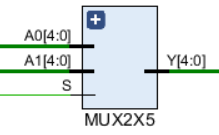
CLA\_4 add5(X[23:20],Y[23:20],cout4,S[23:20],cout5);

CLA\_4 add6(X[27:24],Y[27:24],cout5,S[27:24],cout6);

CLA\_4 add7(X[31:28],Y[31:28],cout6,S[31:28],Cout);

endmodule

### 9.MUX2X5



**模块功能**

根据控制信号从四个五位输入信号中选择输出信号

**引脚及控制信号**

A0、A1：两个需要被选择的信号，输入信号

S：对地址进行选择的控制信号，输入信号

Y：目标信号，输出信号

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)

1'b0:select=A0;

1'b1:select=A1;

endcase

endfunction

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

endmodule

module TEST;

reg [4:0] A0,A1;

reg S;

wire [4:0]Y;

MUX2X5\_describe uut (.A0(A0),.A1(A1),.S(S),.Y(Y));

initial begin

A0 = 00000;

A1 = 00000;

S = 0;

end

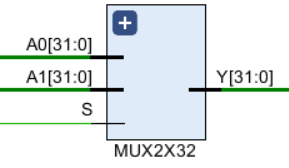
always #20 A0=A0+1;

always #10 A1=A1+1;

always #5 S=1-S;

endmodule

### 10.MUX2X32



**模块功能**

根据控制信号从两个32位输入信号中选择输出信号

**引脚及控制信号**

A0、A1：两个需要被选择的信号，输入信号

S：对地址进行选择的控制信号，输入信号

Y：目标信号，输出信号

module MUX2X32(A0,A1,S,Y);

input[31:0] A0,A1;

input S;

output [31:0]Y;

function [31:0] select;

input [31:0]A0,A1;

input S;

case(S)

1'b0:select=A0;

1'b1:select=A1;

endcase

endfunction

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

endmodule

module TEST;

reg [31:0] A0,A1;

reg S;

wire [31:0]Y;

MUX2X32\_describe uut (.A0(A0),.A1(A1),.S(S),.Y(Y));

initial begin

A0 = 0;

A1 = 0;

S = 0;

end

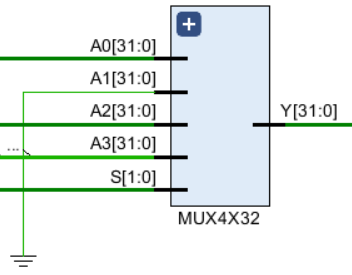
always #20 A0=A0+1;

always #10 A1=A1+1;

always #5 S=1-S;

endmodule

### 11.MUX4X32



**模块功能**

根据控制信号从四个输入信号中选择输出信号

**引脚及控制信号**

A0、A1、A2、A3：四个需要被选择的信号，输入信号

S：对地址进行选择的控制信号，输入信号

Y：目标信号，输出信号

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

module TEST;

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

reg [1:0]S;

wire [31:0]Y;

MUX2X32\_describe uut (.A0(A0),.A1(A1),.A2(A2),.A3(A3),.S(S),.Y(Y));

initial begin

A0 = 0;

A1 = 0;

A2 = 0;

A3 = 0;

S = 2'b01;

end

always #10 A0=A0+1;

always #10 A1=A1+1;

always #10 A2=A2+1;

always #10 A3=A3+1;

always #10 S=S+01;

endmodule

### 12.封装

module CPU(Clk,Clrn,Inst,currentaddress,R,Dout,D,rd,rt,rs,sa,func,op,immediate,address,newaddress,Qb);

input Clk,Clrn;

output [31:0]Inst,currentaddress,R,Dout,D,newaddress;

output [4:0]rd,rt,rs,sa;

output [5:0]func,op;

output [15:0]immediate;

output [25:0]address;

output [31:0]Qb;

wire[31:0] newaddress,currentaddress\_4,address1,currentaddress\_immediate;

//newaddress 是pc寄存器的输入 currentaddresss\_4是pc加4之后的值 address1是j型指令的跳转的目标地址

//currentaddress\_immediate是pc加4 与 立即数扩展、移位 后相加的结果

wire [31:0]kong;

//kong就是一个空的东西 里面没有内容 存在的意义是为32位4选1 提供一个输入的窗口 因为这个四选一需要5个输入 而实际上只有4个

wire [4:0]rd,rt,rs,sa;

wire [5:0]func,op;

wire [15:0]immediate;

wire [25:0]address;

//rd rs 这些就是r i j型指令的相应的段

wire [31:0] kuozhan;//16位立即数扩展成32位的结果

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

wire [1:0] Pcsrc,Aluc;

//控制器的相关输出

wire [31:0]Y;//ALU的一个输入

wire [31:0]Qa,Qb;//寄存器堆的输出

wire [4:0]Wr;//寄存器堆的要写的地址的端口

assign rd=Inst[15:11];

assign rt=Inst[20:16];

assign rs=Inst[25:21];

assign sa=Inst[10:6];

assign func=Inst[5:0];

assign op=Inst[31:26];

assign immediate=Inst[15:0];

assign address=Inst[25:0];

//下面的就是一些相关的部件的调用

CONUNIT conunit(op,func,Z,Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg);

PC pc(Clk,Clrn,newaddress,currentaddress);

INSTMEM instmem (currentaddress,Inst);

REGFILE regfile (rs,rt,D,Wr,Wreg,Clk,Clrn,Qa,Qb);

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

EXT16T32\_ ext16t32 (immediate,Se,kuozhan);

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

CLA\_32 pcplus4(currentaddress,32'h4,1'b0,currentaddress\_4);

CLA\_32 br\_adr (currentaddress\_4,(kuozhan << 2),1'b0,currentaddress\_immediate);

assign address1={currentaddress\_4[31:28],address,2'b00};

MUX2X5 mux2x5(rd,rt,Regrt,Wr);

MUX2X32 mux2x322 (kuozhan,Qb,Aluqb,Y);

MUX2X32 mux2x323 (Dout,R,Reg2reg,D);

MUX4X32 mux4x32 (currentaddress\_4,kong,currentaddress\_immediate,address1,Pcsrc,newaddress);

endmodule

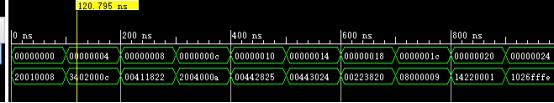
## 三、测试结果

### 1.各个部件单独测试

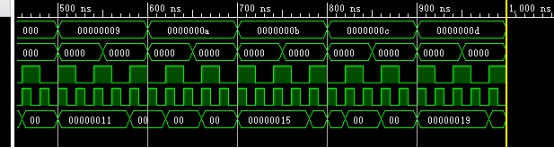
ALU:



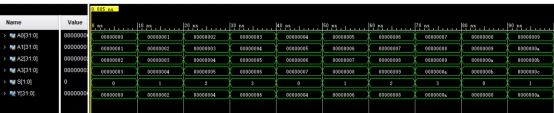
INSTMEM:



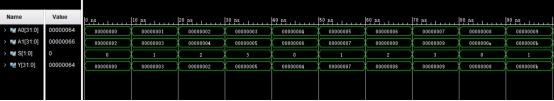
DATAMEM:



MUX4X32



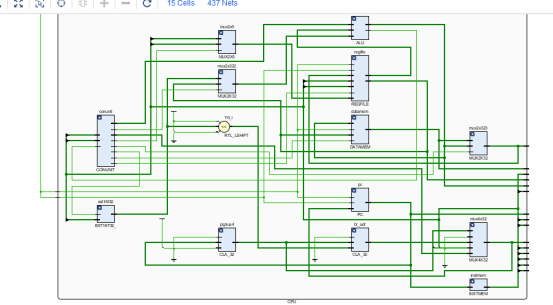
MUX2X32



MUX2X5

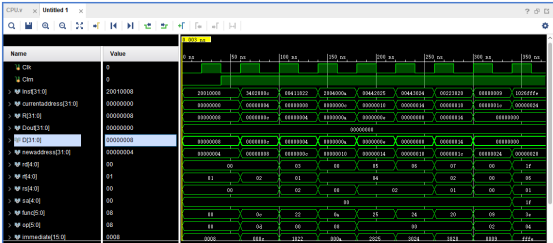


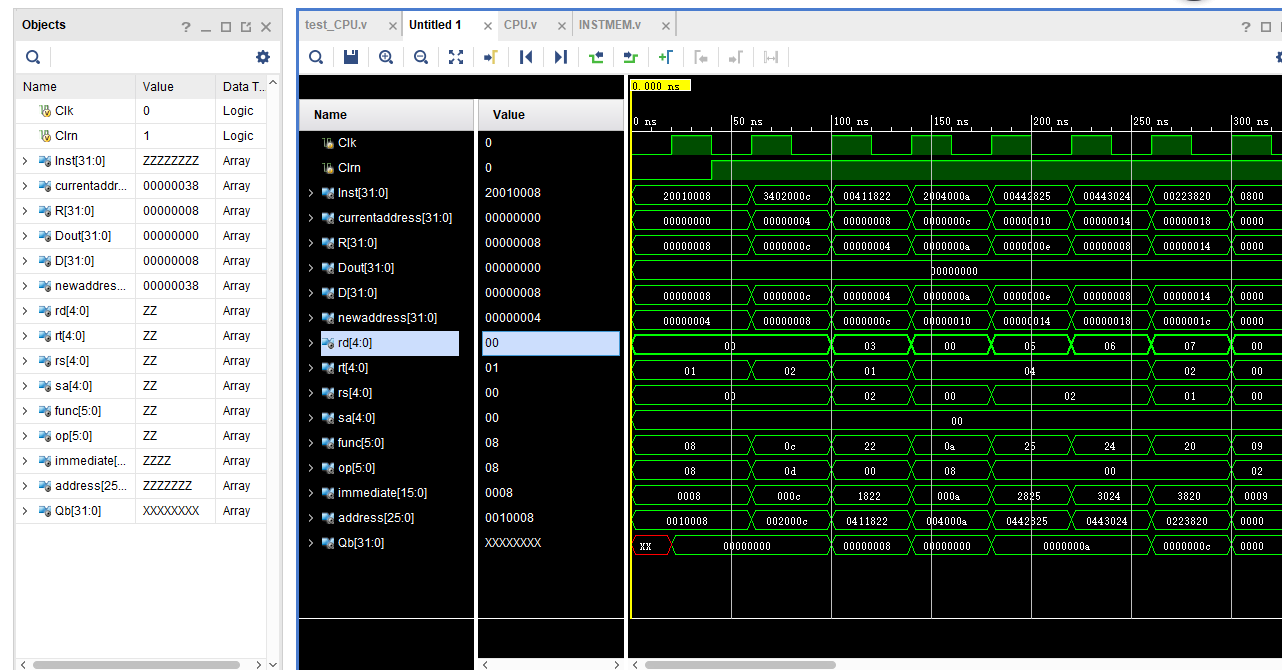
### 2.RTL图



和书上的图主要器件部分大致相同

### 3.总体运行结果





整体CPU功能实现成功