**实验四 运算控制 时序与状态机**

**姓名：姜庆彩**

**学号：PB1505187**

* 实验功能要求：

综合利用三次实验的结果，完成以下功能：

* + 通过例化，向ram中0地址到13地址存入14个数，比如10-23；向ram中100地址到106地址存入7个数，比如0~6，分别代表运算符，向ram 107地址写入-1
  + 运算控制：
    - 从ram 0地址开始的地方取两个数，从ram 100地址开始的地方取一个运算符，计算之后，把结果存入ram地址200
    - 从ram 2地址开始的地©取一个运算符，计算之后，把结果存入ram地址201
    - ……
    - 如果取出操作符为-1，则结束。
* 实验要求：

使用状态机或分频clk，或联合使用这两种技术控制运算过程（数据读取，计算，数据写入），每部加法运算所用时钟数不允许超过五个。

仿真激励文件模块只允许出现clk和rst信号输入。

本次实验依据运算所用周期数进行评分，周期越多分越低。

* 实验设计要求：

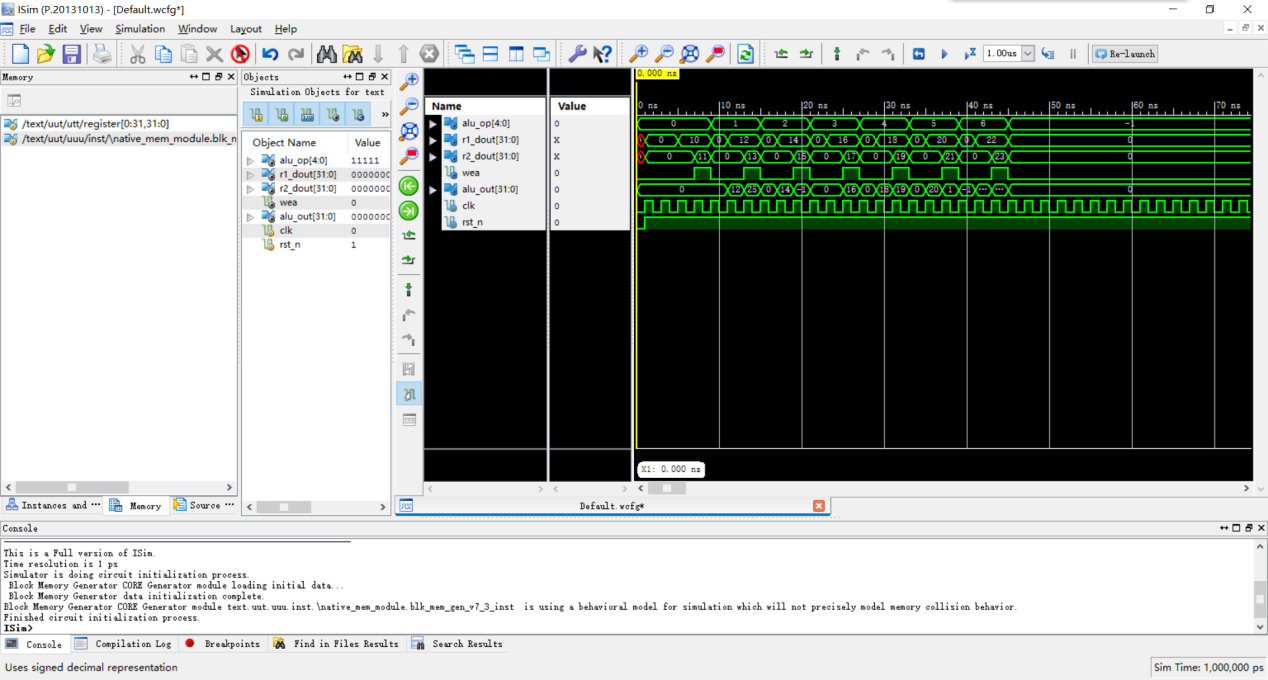
实现一个control模块，完成整个运算的控制。

实现一个顶层模块Top

* + 调用Ram模块
  + 调用RegFile
  + 调用ALU完成加法运算
  + 调用control模块，完成运算控制

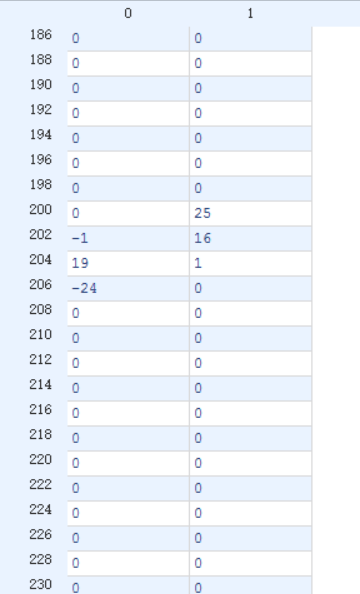
实验结果：

模拟波形图：



由图中可以看出，用22.5个周期实现了7次运算以及停止的操作，其中21的中期用于运算，也就是说每次运算3个周期，完成了实验的要求。

ram内容：



我的实现简介：

这次的实验实现较为复杂，我使用了4个状态的状态机，其中第一、二个状态用于读第一、二个操作数，第三个状态用于取操作数以及运算（由于运算结果与ram直接相连，写入ram相当于不需要浪费周期），第四个状态用于停止。由于这次的ram读写的位置跨度很大，为了增加偏移的稳定性，我使用了addr1与addr2来记录ram的位置。这次实验的时间主要花在了调试上，因为原来不知道ram的延迟时间，所以花了很久的时间用于地址的赋值上。

源代码:

module control(

input clk,

input rst\_n,

input [31:0] doutb,

output reg [4:0] alu\_op,

output reg [4:0]r1\_addr,

output reg [4:0]r2\_addr,

output reg [4:0]r3\_addr,

output reg r3\_wr,

output reg wea,

output reg [7:0] addra,

output reg [7:0] addrb

);

parameter firstread=2'b00,secondread=2'b01,get\_op\_and\_wr=2'b10,finish=2'b11;

reg [1:0] curstate;

reg [1:0] nextstate;

reg [7:0] addr1;

reg [7:0] addr2;

always@(posedge clk or negedge rst\_n)

begin

if(~rst\_n) curstate<=firstread;

else curstate<=nextstate;

end

always@(\*)

begin

case(curstate)

firstread:

nextstate<=secondread;

secondread:

if(doutb==32'b1111\_1111\_1111\_1111\_1111\_1111\_1111\_1111)

nextstate<=finish;

else nextstate<=get\_op\_and\_wr;

get\_op\_and\_wr:

nextstate<=firstread;

finish: nextstate<=finish;

endcase

end

always@(posedge clk or negedge rst\_n)

begin

if(~rst\_n)

begin

r3\_addr=5'b0;

addr1=8'd0;

addr2=8'd100;

addra=8'd199;

addrb=8'b0;

r3\_wr=0;

wea=0;

r1\_addr=5'b0;

r2\_addr=5'b1;

alu\_op=5'b0;

end

else

begin

case(curstate)

firstread:

begin

r3\_wr=0;

if(r3\_addr==0) r3\_addr=r3\_addr;

else

begin

r3\_addr=r3\_addr+1;

r1\_addr=r1\_addr+2;

r2\_addr=r2\_addr+2;

wea=1;

end

if(addr1==0) addr1=8'b0;

else addr1=addr1+1;

addrb=addr1;

end

secondread:

begin

r3\_wr=1;

wea=0;

addr1=addr1+1;

addrb=addr1;

if(r3\_addr!=0)

alu\_op=doutb[4:0];

end

get\_op\_and\_wr:

begin

r3\_addr=r3\_addr+1;

r3\_wr=1;

wea=0;

addr2=addr2+1;

addrb=addr2;

addra=addra+1;

end

finish:

begin

r3\_wr=0;

wea=0;

end

endcase

end

end

endmodule

module REG\_FILE(

input clk,input rst\_n,input [4:0] r1\_addr,input [4:0] r2\_addr,input [4:0] r3\_addr,input [31:0] r3\_din,input r3\_wr,output reg [31:0] r1\_dout,output reg [31:0] r2\_dout );

reg [31:0] register[0:31];

integer k,i;

always@(posedge clk or negedge rst\_n)

begin

if(~rst\_n)

begin

register[0]=32'b0;

register[1]=32'b0;

for(k=2;k<32;k=k+1)

register[k]=32'b0;

end

else if(r3\_wr)

begin

i=r3\_addr;

register[i]=r3\_din;

i=r1\_addr;

r1\_dout=register[i];

i=r2\_addr;

r2\_dout=register[i];

end

else

begin

i=r1\_addr;

r1\_dout=register[i];

i=r2\_addr;

r2\_dout=register[i];

end

end

endmodule

module alu(

input signed [31:0] alu\_a,

input signed [31:0] alu\_b,

input [4:0] alu\_op,

output reg [31:0] alu\_out

);

always@(\*)

begin

case(alu\_op)

1: alu\_out <= alu\_a + alu\_b;

2: alu\_out <= alu\_a - alu\_b;

3: alu\_out <= alu\_a & alu\_b;

4: alu\_out <= alu\_a | alu\_b;

5: alu\_out <= alu\_a ^ alu\_b;

6: alu\_out <= ~(alu\_a | alu\_b);

default:alu\_out <= 0;

endcase

end

endmodule

module top(

input clk,

input rst\_n,

output [4:0] alu\_op,

output [31:0] r1\_dout,

output [31:0] r2\_dout,

output wea,

output [31:0] alu\_out

);

wire clk\_;

assign clk\_=~clk;

wire [4:0] r1\_addr;

wire [4:0] r2\_addr;

wire [4:0] r3\_addr;

wire [4:0] r3\_din;

wire r3\_wr;

wire [31:0] number;

wire [7:0] addra;

wire [7:0] addrb;

wire [31:0] dina;

wire [31:0] doutb;

alu uut(

.alu\_a(r1\_dout),

.alu\_b(r2\_dout),

.alu\_op(alu\_op),

.alu\_out(alu\_out)

);

REG\_FILE utt(

.clk(clk),

.rst\_n(rst\_n),

.r1\_addr(r1\_addr),

.r2\_addr(r2\_addr),

.r3\_addr(r3\_addr),

.r3\_din(doutb),

.r3\_wr(r3\_wr),

.r1\_dout(r1\_dout),

.r2\_dout(r2\_dout)

);

control ttt(

.clk(clk),

.rst\_n(rst\_n),

.doutb(doutb),

.r1\_addr(r1\_addr),

.r2\_addr(r2\_addr),

.r3\_addr(r3\_addr),

.r3\_wr(r3\_wr),

.alu\_op(alu\_op),

.wea(wea),

.addra(addra),

.addrb(addrb)

);

myram uuu(

.clka(clk\_),

.wea(wea),

.addra(addra),

.dina(alu\_out),

.clkb(clk),

.addrb(addrb),

.doutb(doutb)

);

endmodule