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

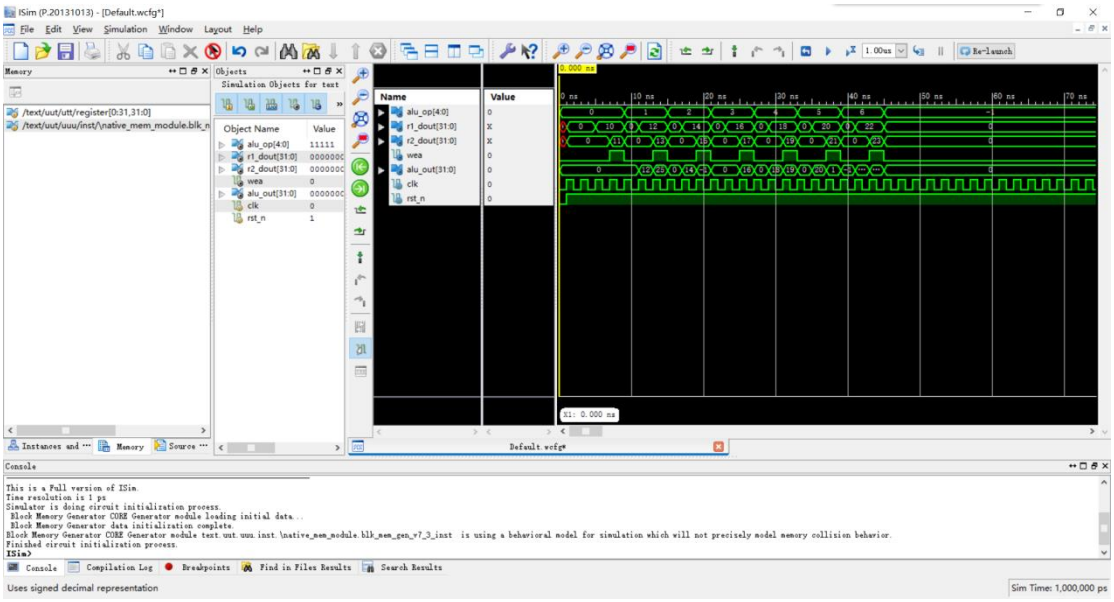
调用 RegFile

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：

	0	1
186	0	0
188	0	0
190	0	0
192	0	0
194	0	0
196	0	0
198	0	0
200	0	25
202	-1	16
204	19	1
206	-24	0
208	0	0
210	0	0
212	0	0
214	0	0
216	0	0
218	0	0
220	0	0
222	0	0
224	0	0
226	0	0
228	0	0
230	0	0

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
```