

Lab3\_Ram

金泽文 PB15111604

**实验目的：**

理解有限状态机。

**实验内容：**

# 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地址开始的地方取两个数，从ram 101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。



### **实验分析与设计：**

ALU 模块完全同于上次实验。

Regfile 完全同于上次实验。

Top 不同于上次，这次只调用模块，无其他控制功能。

Control 不同于上次。详细说一下 control：

首先，分为 4 个周期，用 `current_state`，`next_state` 表示现态，次态。

用 `is_on` 控制是否继续，通过判断读入数据是否为-1 以及现态是否为 S0 控制。

对于从 ram 读入数据的地址 `addr_base_number`，在 S1，S2 阶段分别加一，

对于从 ram 读入数据，分别在 S0，S1 阶段读入。

对于写入 reg，通过地址 `addr_to_reg`，在 S1 时为 0，在 S2 时为 1，

对于写寄存器使能 `we_reg`，根据 S1 到 S2 的上升沿时得到的 S1 的地址 0，S2 到 S3 的上升沿得到的地址 1，写入对应寄存器。

S3 时得到 alu 得到结果。

并且在 S3 下降沿 `we_ram` 赋值 1，写入 ram。

### **意见建议：**

无

附录：

ALU 部分：

```
1  module ALU(  
2  input signed [31:0] alu_a,  
3  input signed [31:0] alu_b,  
4  input [4:0] alu_op,  
5  output reg [31:0] alu_out  
6  );  
7  
8  parameter A_NOP = 5'h00;  
9  parameter A_ADD = 5'h01;  
10 parameter A_SUB = 5'h02;  
11 parameter A_AND = 5'h03;  
12 parameter A_OR = 5'h04;  
13 parameter A_XOR = 5'h05;  
14 parameter A_NOR = 5'h06;  
15  
16 always@(*)  
17 begin  
18     case(alu_op)  
19         A_NOP: alu_out = 0;  
20         A_ADD: alu_out = alu_a + alu_b;  
21         A_SUB: alu_out = alu_a - alu_b;  
22         A_AND: alu_out = alu_a & alu_b;  
23         A_OR: alu_out = alu_a | alu_b;  
24         A_XOR: alu_out = (~alu_a & alu_b) | (alu_a & ~alu_b);  
25         A_NOR: alu_out = ~(alu_a | alu_b);  
26     endcase  
27 end  
28  
29 endmodule
```



Top 部分 :

```
1 `timescale 1ns / 1ps
2
3 module top(
4   input clk,
5   input rst_n
6   );
7
8   wire [31:0] alu_out, r1_dout, r2_dout, ram_out;
9   wire [7:0]  addr_ram_out, addr_save, addr_to_reg;
10  wire we_ram, we_reg;
11  reg [4:0] r1 = 0, r2 = 1;
12
13  ALU ALU1(r1_dout, r2_dout, ram_out, alu_out);
14  REG_FILE REG_FILE1(clk, rst_n, r1, r2, addr_to_reg,
15  ram_out, we_reg, r1_dout, r2_dout);
16  ram ram1(clk, we_ram, addr_save, alu_out, clk,
17  addr_ram_out, ram_out);
18  control control1(clk, rst_n, ram_out, addr_ram_out,
19  addr_save, addr_to_reg, we_ram, we_reg);
20
21 endmodule
```

Regfile 部分 :

```
1 module REG_FILE(  
2 input clk,  
3 input rst,  
4 input [4:0] r1_addr,  
5 input [4:0] r2_addr,  
6 input [4:0] r3_addr,  
7 input [31:0] r3_din,  
8 input r3_wr,  
9 output [31:0] r1_dout,  
10 output [31:0] r2_dout  
11 );  
12  
13 reg [31:0] regs[31:0];  
14  
15 always @(posedge clk or posedge rst) begin  
16     if (rst) begin  
17         regs[0] <= 32'b1;  
18         regs[1] <= 32'b1;  
19         regs[2] <= 32'b1;  
20         regs[3] <= 32'b1;  
21         regs[4] <= 32'b1;  
22         regs[5] <= 32'b1;  
23         regs[6] <= 32'b1;  
24         regs[7] <= 32'b1;  
25         regs[8] <= 32'b1;  
26         regs[9] <= 32'b1;  
27         regs[10] <= 32'b1;  
28         regs[11] <= 32'b1;  
29         regs[12] <= 32'b1;  
30         regs[13] <= 32'b1;  
31         regs[14] <= 32'b1;  
32         regs[15] <= 32'b1;  
33         regs[16] <= 32'b1;  
34         regs[17] <= 32'b1;  
35         regs[18] <= 32'b1;  
36         regs[19] <= 32'b1;  
37         regs[20] <= 32'b1;  
38         regs[21] <= 32'b1;  
39         regs[22] <= 32'b1;  
40         regs[23] <= 32'b1;  
41         regs[24] <= 32'b1;  
42         regs[25] <= 32'b1;  
43         regs[26] <= 32'b1;  
44         regs[27] <= 32'b1;  
45         regs[28] <= 32'b1;  
46         regs[29] <= 32'b1;  
47         regs[30] <= 32'b1;  
48         regs[31] <= 32'b1;  
49     end  
50     else if (r3_wr)begin  
51         regs[r3_addr] <= r3_din;  
52     end  
53 end  
54  
55 assign r1_dout = regs[r1_addr];  
56 assign r2_dout = regs[r2_addr];  
57 endmodule  
58
```

## Control :

```
1 `timescale 1ns / 1ps
2
3 module control(
4 input clk,
5 input rst_n,
6 input [31:0] ram_out,
7 output reg [7:0] addr_ram_out,
8 output reg [7:0] address2,
9 output reg [7:0] address3,
10 output reg we_ram,
11 output reg we_reg
12 );
13
14 reg [1:0] current_state = 0, next_state = 1;
15 reg [7:0] addr_base_number = 0, addr_op = 8'd100, addr_save = 8'd200,
16   addr_to_reg = 0;
17 parameter s0 = 2'd0, s1 = 2'd1, s2 = 2'd2, s3 = 2'd3;
18 reg is_on = 1;
19
20 always@(posedge clk, negedge rst_n) begin
21     if(~rst_n)
22         is_on <= 1;
23     else if(current_state == s3 && ram_out == 32'hfffffff)
24         is_on <= 0;
25 end
26
27 always@(*) begin
28     case(current_state)
29         s0: next_state = s1;
30         s1: next_state = s2;
31         s2: next_state = s3;
32         s3: next_state = s0;
33         default: next_state = s0;
34     endcase
35 end
36
37 always@(posedge clk, negedge rst_n) begin
38     if(~rst_n)
39         current_state <= 0;
40     else if(is_on)
41         current_state <= next_state;
42 end
43
44 current_state <= next_state;
45 end
46
47 always@(posedge clk, negedge rst_n) begin
48     if(~rst_n)
49         addr_base_number <= 0;
50     else if(current_state == s1 || current_state == s0)
51         addr_base_number <= addr_base_number + 1;
52 end
53
54 always@(posedge clk, negedge rst_n) begin
55     if(~rst_n)
56         addr_to_reg <= 0;
57     else if(current_state == s1 || current_state == s0)
58         addr_to_reg <= 0;
59     else if(current_state == s1)
60         addr_to_reg <= 1;
61 end
62
63 always@(*) begin
64     address3 = addr_to_reg;
65 end
66
67 always@(posedge clk, negedge rst_n) begin
68     if(~rst_n)
69         addr_op <= 8'd100;
70     else if(current_state == s2)
71         addr_op <= addr_op + 1;
72 end
73
74 always@(posedge clk, negedge rst_n) begin
75     if(~rst_n)
76         addr_save <= 8'd200;
77     else if(current_state == s3)
78         addr_save <= addr_save + 1;
79 end
80
81 always@(posedge clk, negedge rst_n) begin
82     address2 = addr_save;
83 end
84
85 always@(*) begin
```



```

80
81 always@(*) begin
82     if(~rst_n)
83         addr_ram_out = 0;
84     else
85     begin
86         case(current_state)
87             s0: addr_ram_out = addr_base_number;
88             s1: addr_ram_out = addr_base_number;
89             s2: addr_ram_out = addr_op;
90             s3: addr_ram_out = addr_base_number;
91             default: addr_ram_out = 0;
92         endcase
93     end
94 end
95
96 always@(negedge clk,negedge rst_n) begin
97     if(~rst_n)
98         we_ram = 0;
99     else begin
100        case(current_state)
101            s0: we_ram = 0;
102            s1: we_ram = 0;
103            s2: we_ram = 0;
104            s3: we_ram = 1;
105            default: we_ram = 0;
106        endcase
107    end
108 end
109
110 always@(posedge clk,negedge rst_n) begin
111     if(~rst_n)
112         we_reg = 0;
113     else begin
114         case(current_state)
115             s0: we_reg = 0;
116             s1: we_reg = 1;
117             s2: we_reg = 1;
118             s3: we_reg = 0;
119             default: we_reg = 0;
120         endcase
121     end
122 end
123
124 endmodule

```

Testfile :

```
1 `timescale 1ns / 1ps
2
3 module test;
4
5     // Inputs
6     reg clk;
7     reg rst_n;
8
9     // Instantiate the Unit Under Test (UUT)
10 top uut (
11     .clk(clk),
12     .rst_n(rst_n)
13 );
14
15 initial begin
16     // Initialize Inputs
17     clk = 0;
18     rst_n = 0;
19
20     // Wait 100 ns for global reset to finish
21     #10;
22     rst_n = 1;
23     forever #1 clk = ~clk;
24
25
26     // Add stimulus here
27
28 end
29
30 endmodule
31
32
```