# 实验五　单周期 MIPS CPU 设计

姓名：姜庆彩

学号：PB15051087

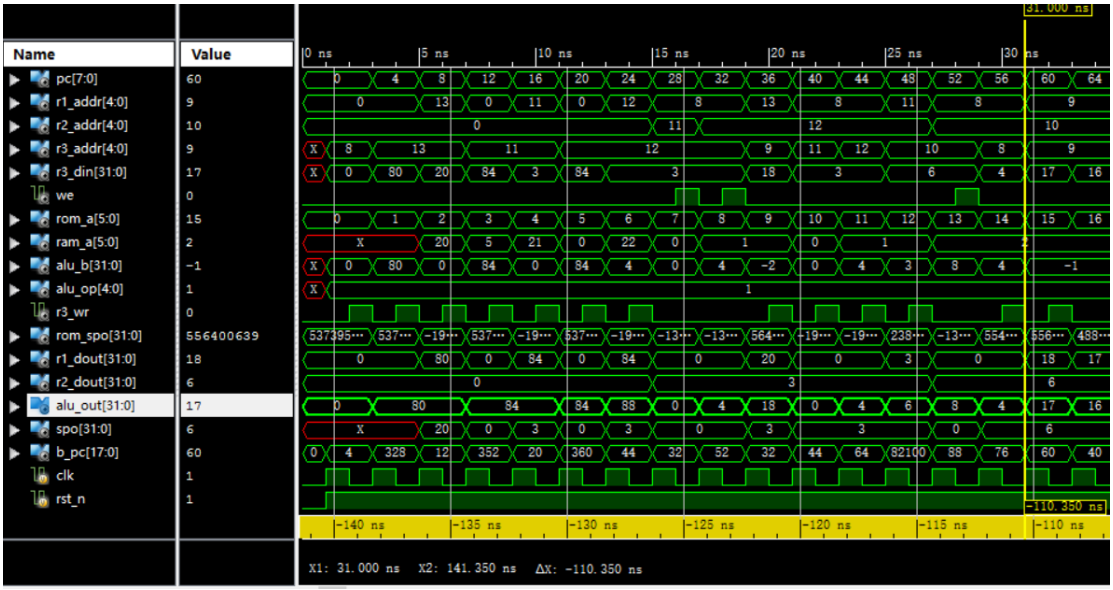
- 实验要求

  设计 CPU，完成以下程序代码的执行，其功能是起始数为 3 和 3 的斐波拉契数列的计算。只计算 20 个数。

```
    .data
    fibs: .word   0 : 20       # "array" of  20 words to
contain fib values
    size: .word  20           # size of "array"
    temp: .word 3 3
         .text
         la   $t0, fibs       # load address of array
         la   $t5, size       # load address of size
variable
         lw   $t5, 0($t5)     # load array size
         la   $t3, temp     # load
         lw   $t3, 0($t3)
         la   $t4, temp
         lw   $t4, 4($t4)

         sw   $t3, 0($t0)     # F[0] = $t3
         sw   $t4, 4($t0)     # F[1] = $t4
         addi $t1, $t5, -2    # Counter for loop, will
execute (size-2) times
    loop: lw   $t3, 0($t0)     # Get value from array F[n]
         lw   $t4, 4($t0)     # Get value from array F[n+1]
         add  $t2, $t3, $t4    # $t2 = F[n] + F[n+1]
         sw   $t2, 8($t0)     # Store F[n+2] = F[n] + F[n+1]
in array
         addi $t0, $t0, 4     # increment address of Fib.
number source
         addi $t1, $t1, -1    # decrement loop counter
         bgtz $t1, loop       # repeat if not finished yet.
    out:
    j out
```

- 实验结果：




| | 0 | 1 |
|---|---|---|
| 0x2D | 0 | 0 |
| 0x2B | 0 | 0 |
| 0x29 | 0 | 0 |
| 0x27 | 0 | 0 |
| 0x25 | 0 | 0 |
| 0x23 | 0 | 0 |
| 0x21 | 0 | 0 |
| 0x1F | 0 | 0 |
| 0x1D | 0 | 0 |
| 0x1B | 0 | 0 |
| 0x19 | 0 | 0 |
| 0x17 | 0 | 3 |
| 0x15 | 3 | 20 |
| 0x13 | 20295 | 12543 |
| 0x11 | 7752 | 4791 |
| 0xF | 2961 | 1830 |
| 0xD | 1131 | 699 |
| 0xB | 432 | 267 |
| 0x9 | 165 | 102 |
| 0x7 | 63 | 39 |
| 0x5 | 24 | 15 |
| 0x3 | 9 | 6 |
| 0x1 | 3 | 3 |

- 实验总结：

  这次实验要求的是异步读取，一开始不知道可以使用DRAM实现，而是用了与前几次一样的双端口 RAM，然后死活跑不出来，后来看了群里面的 dalao 发言才知道可以用 DRAM，这样就变得比较简单了；由于是异步读取，前面设计的 REGFILE 模块也需要改成非时钟沿触发而是直接赋值；反而数据通路不是很复杂，简单的逻辑判断就可了。

- 源代码：

```
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 <= alu_out;
     endcase
     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 [31:0] r1_dout,
output [31:0] r2_dout
    );
reg [31:0] register[0:31];
assign r1_dout=register[r1_addr];
assign r2_dout=register[r2_addr];
integer k;
always@(posedge clk or negedge rst_n)
begin
 if(~rst_n)
 begin
   for(k=0;k<32;k=k+1)
    register[k]=32'b0;
 end
 else if(r3_wr)
     register[r3_addr]=r3_din;
end



endmodule
```

```
    module control(
    input clk,
    input rst_n,
    input [31:0] instruction,
    input [31:0] spo,
    input [31:0] r1_dout,
    input [31:0] r2_dout,
    input [31:0] alu_out,
    output reg signed [7:0] pc,
    output reg [4:0] r1_addr,
    output reg [4:0] r2_addr,
    output reg [4:0] r3_addr,
    output reg [31:0] r3_din,
    output reg we,
    output reg [5:0] rom_a,
    output reg [5:0] ram_a,
    output reg [31:0] alu_b,
    output reg [4:0] alu_op,
    output reg r3_wr,
    output reg signed [17:0] b_pc
    );
    reg flag;
    reg signed [31:0] judge;
    reg signed [17:0] offset;
    reg signed [17:0] next_pc;
    reg [27:0] j_pc;
    always@(*)
    begin
      if(~rst_n) next_pc=0;
      else if(flag==1) next_pc=pc+4;
      offset=instruction[15:0]<<2;
      b_pc=offset+next_pc;//branch
      j_pc=instruction[25:0]<<2;//jump
    end
    always@(posedge clk or negedge rst_n)
    begin
      if(~rst_n)
      begin
        pc=7'b0;
       flag=0;
      end
      else if(instruction[31:26]==6'b000010)
pc=j_pc[7:0];
      else
```

```
if(instruction[31:26]==6'b000111&&judge>0)
pc=b_pc[7:0];//bgzt
      else
      begin
        pc=next_pc[7:0];
       flag=1;
      end
    end
    always@(*)
    begin
      rom_a=pc>>2;
    end
    always@(*)
    begin
      if(~rst_n)
      begin
       r3_wr=0;
      we=0;
      r1_addr=0;
      r2_addr=0;
      end
      else if(instruction[31:26]==6'b001000)
//addi
      begin
       if(instruction[15]==1)

alu_b={16'b11111_1111_1111_1111,instruction
[15:0]};
     else
alu_b={16'b0000_0000_0000_0000,instruction[
15:0]};
      r1_addr=instruction[25:21];
      r3_addr=instruction[20:16];
      r3_din=alu_out;
      alu_op=5'h01;
      if(~clk) r3_wr=1;
      else r3_wr=0;
      end
      else if(instruction[31:26]==6'b000000)
//add
      begin
       r1_addr=instruction[25:21];
      r2_addr=instruction[20:16];
      r3_addr=instruction[15:11];
```

```
      alu_b=r2_dout;
      alu_op=5'h01;
      r3_din=alu_out;
      if(~clk) r3_wr=1;
      else r3_wr=0;
       end
       else if(instruction[31:26]==6'b100011)
//lw
       begin
        r1_addr=instruction[25:21];
      alu_b={16'b0000_0000_0000_0000,instruc
tion[15:0]};
      alu_op=5'h01;
      ram_a=alu_out[7:2];
      r3_addr=instruction[20:16];
      r3_din=spo;
      if(~clk) r3_wr=1;
      else r3_wr=0;
       end
       else if(instruction[31:26]==6'b101011)
//sw
       begin
        r1_addr=instruction[25:21];
      r2_addr=instruction[20:16];
      alu_b={16'b0000_0000_0000_0000,instruc
tion[15:0]};
      ram_a=alu_out[7:2];
      alu_op=5'h01;
      r3_wr=0;
      if(~clk) we=1;
      else we=0;
       end
       else if(instruction[31:26]==6'b000111)
//bgtz
       begin
        r1_addr=instruction[25:21];
      judge=r1_dout;
       end
    end
endmodule
```

```
module top(
input clk,
input rst_n,
output signed [7:0] pc,
output [4:0] r1_addr,
output [4:0] r2_addr,
output [4:0] r3_addr,
output [31:0] r3_din,
output we,
output [5:0] rom_a,
output [5:0] ram_a,
output [31:0] alu_b,
output [4:0] alu_op,
output  r3_wr,
output [31:0] rom_spo,
output [31:0] r1_dout,
output [31:0] r2_dout,
output [31:0] alu_out,
output [31:0] spo,
output [17:0] b_pc
    );
REG_FILE uuu(
.clk(clk),
.rst_n(rst_n),
.r1_addr(r1_addr),
.r2_addr(r2_addr),
.r3_addr(r3_addr),
.r3_din(r3_din),
.r3_wr(r3_wr),
.r1_dout(r1_dout),
.r2_dout(r2_dout)
);
alu ttt(
.alu_a(r1_dout),
.alu_b(alu_b),
.alu_op(alu_op),
.alu_out(alu_out)
);
control uut(
.clk(clk),
.rst_n(rst_n),
.instruction(rom_spo),
.spo(spo),
.r1_dout(r1_dout),
```

```
    .r2_dout(r2_dout),
    .alu_out(alu_out),
    .pc(pc),
    .r1_addr(r1_addr),
    .r2_addr(r2_addr),
    .r3_addr(r3_addr),
    .r3_din(r3_din),
    .we(we),
    .rom_a(rom_a),
    .ram_a(ram_a),
    .alu_b(alu_b),
    .alu_op(alu_op),
    .r3_wr(r3_wr),
    .b_pc(b_pc)
    );
    myrom utu(
    .a(rom_a),
    .spo(rom_spo)
    );
    myram err(
    .a(ram_a),
    .d(r2_dout),
    .clk(clk),
    .we(we),
    .spo(spo)
);
```