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

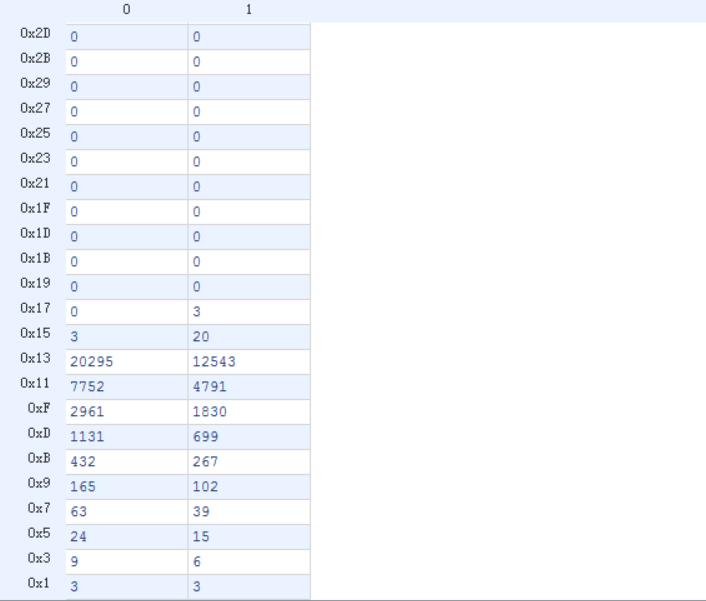
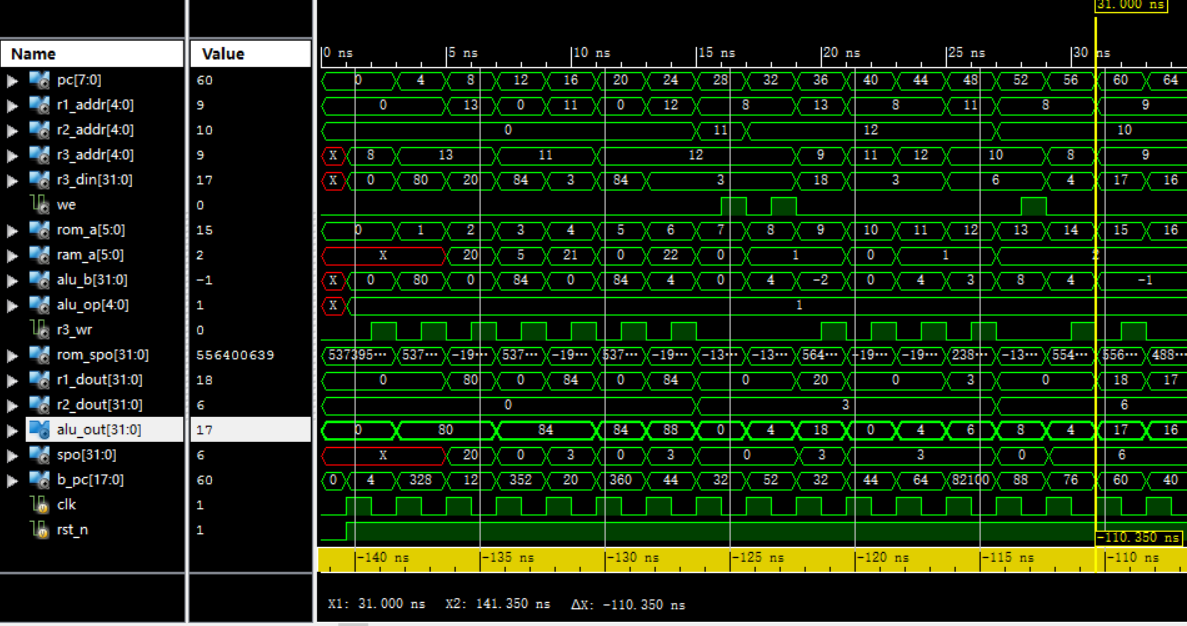
addi $t1, $t1, -1 # decrement loop counter

bgtz $t1, loop # repeat if not finished yet.

out:

j out

* 实验结果：



* 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,instruction[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,instruction[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)

);