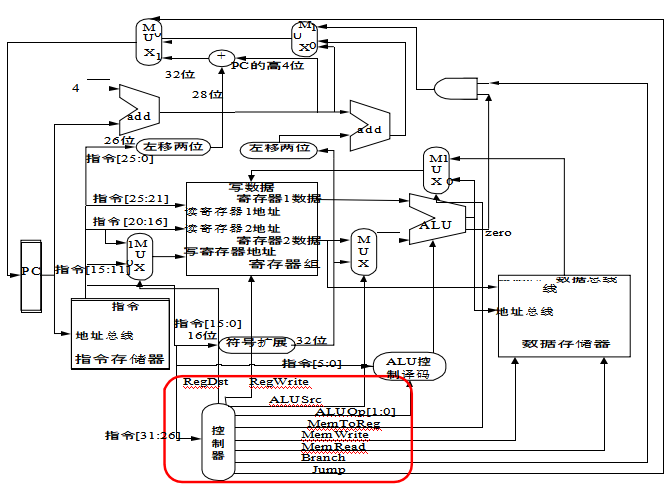
微处理器设计

电磁1702 桂正泰

U201713557

实验任务&目标

利用Verilog HDL语言，基于Xilinx FPGA nexys4实验平台，设计一个能够执行以下MIPS指令集的单周期类MIPS处理器，要求完成所有支持指令的功能仿真，验证指令执行的正确性，要求编写汇编程序将本人学号的ASCII码存入RAM的连续内存区域



微处理器各个模块硬件设计原理、Verilog代码

顶层模块

ROM块采用异步输出，RAM同步输出，异步写入。该方案有三个部分用到了时钟信号：指令指针赋值，寄存器文件写入数据，数据存储器输出数据。指令指针赋值与始终上升沿同步，而寄存器文件数据写入以及数据存储器数据输出、输入与时钟下降沿同步

Verilog代码

module top(

input clk,

input reset

);

reg [31:0] PC;

wire J,B,Bzero,zero,M2R,memwr,Imm,regwr,RtDst;

wire [1:0] ALUop;

wire [3:0] ALUctr;

wire [31:0] NewPC,JMPPC,BranPC,Imm32L2,Imm32,NotJMPPC,TempPC;

wire [31:0] Instr,WriteData,DataOut,Res,RsData,RtData,in2;

wire [4:0] WriteAddr;

assign Imm32={{16{Instr[15]}},Instr[15:0]};

assign WriteAddr=RtDst?Instr[15:11]:Instr[20:16];

assign in2=Imm?Imm32:RtData;

assign WriteData=M2R?DataOut:Res;

assign NotJMPPC=Bzero?BranPC:NewPC;

assign TempPC=J?JMPPC:NotJMPPC;

assign Bzero=B&zero;

assign Imm32L2=Imm32<<2;

assign JMPPC={NewPC[31:28],Instr[25:0],2'b00};

assign BranPC=NewPC+Imm32L2;

assign NewPC=PC+4;

always@(posedge clk)

if(!reset)

PC=TempPC;

else

PC=32'b0;

mainctr mainctr1(Instr[31:26],ALUop,RtDst,regwr,Imm,memwr,B,J,M2R);

ALU alu(RsData,in2,ALUctr,Res,zero);

aluctr\_gzt aluctr1(ALUop,Instr[5:0],ALUctr);

DataRAM dram(Res[7:2],RtData,!clk,memwr,DataOut);

InstrROM irom(PC[8:2],Instr);

regFile regfile(clk,reset,regwr,Instr[25:21],Instr[20:16],WriteAddr,WriteData,RsData,RtData);

endmodule

主控制器

module mainctr(

input [5:0] opCode,

output [1:0] ALUop,

output RtDst,

output regwr,

output Imm,

output memwr,

output B,

output J,

output M2R

);

reg [8:0] outputtemp;

assign RtDst=outputtemp[8];

assign Imm=outputtemp[7];

assign M2R=outputtemp[6];

assign regwr=outputtemp[5];

assign memwr=outputtemp[4];

assign B=outputtemp[3];

assign J=outputtemp[2];

assign ALUop=outputtemp[1:0];

always@(opCode)

case(opCode)

6'b000010:outputtemp =9'bxxx0\_001\_xx; //imp

6'b000000:outputtemp =9'b1001\_000\_10; //R

6'b100011:outputtemp =9'b0111\_000\_00; //lw

6'b101011:outputtemp =9'bx1x0\_100\_00; //sw

6'b000100:outputtemp =9'bx0x0\_010\_01; //beq

6'b001000:outputtemp =9'b0101\_000\_11;

default:outputtemp =9'b000000000;

endcase

endmodule

ALU运算模块

module ALU(

input [31:0] in1,

input [31:0] in2,

input [3:0] ALUctr,

output reg [31:0] Res,

output reg zero

);

always @(in1 or in2 or ALUctr)

begin

case(ALUctr)

4'b0000: //and

Res = in1&in2;

4'b0001: //or

Res = in1 | in2;

4'b0010: //add

Res = in1 + in2;

4'b0110: //sub

begin

Res = in1 - in2;

zero = (Res == 0)?1:0;

end

4'b0111:

Res=(in1<in2)?1:0;

default: //others

Res = 0;

endcase

end

endmodule

ALU控制器

module aluctr\_gzt(

input [1:0] ALUop,

input [5:0] func,

output reg [3:0] ALUctr

);

always @(ALUop or func)

casex({ALUop,func})

8'b00xxxxxx: ALUctr=4'b0010; //lw,sw

8'b01xxxxxx: ALUctr=4'b0110; //beq

8'b10xx0000: ALUctr=4'b0010; //add

8'b11xxxxxx: ALUctr=4'b0010;

8'b10xx0010: ALUctr=4'b0110; //sub

8'b10xx0100: ALUctr=4'b0000; //and

8'b10xx0101: ALUctr=4'b0001; //or

8'b10xx1010: ALUctr=4'b0111; //slt

default : ALUctr=4'b0000;

endcase

endmodule

寄存器文件

module regFile(

input clk,

input reset,

input regwr,

input [4:0] RsAddr,

input [4:0] RtAddr,

input [4:0] WriteAddr,

input [31:0] WriteData,

output [31:0] RsData,

output [31:0] RtData

);

reg [31:0] regs [0:31];

assign RsData = (RsAddr == 5'h0)?32'h0:regs[RsAddr];

assign RtData = (RsAddr == 5'h0)?32'h0:regs[RtAddr];

integer i;

always@(negedge clk or posedge reset)

if (reset)

for(i=0;i<32;i=i+1)

regs[i]<=0;

else if(regwr)

regs[WriteAddr] <= WriteData;

endmodule

ROM汇编程序设计代码

main:

add $14,$12,$13

addi $11,$0,200

addi $12,$0,85 #U

sw $12,36($11)

addi $13,$0,50 #2

sw $13,32($11)

addi $14,$0,48 #0

sw $14,28($11)

addi $15,$0,49 #1

sw $15,24($11)

addi $16,$0,55 #7

sw $16,20($11)

addi $17,$0,49 #1

sw $17,16($11)

addi $18,$0,51 #3

sw $18,12($11)

addi $19,$0,53 #5

sw $19,8($11)

addi $14,$0,53 #5

sw $14,4($11)

addi $18,$0,55 #7

sw $18,0($11)

add $4,$2,$3

lw $4,4($2)

sw $2,8($2)

sub $2,$4,$3

or $2,$4,$3

and $2,$4,$3

slt $2,$4,$3

beq $4,$3,exit

j main

exit: lw,$2,0($3)

j main

Regfile模块仿真激励代码

module regSim(

);

reg clk ;

reg reset;

reg regwr;

reg [4:0] RsAddr;

reg [4:0] RtAddr;

reg [4:0] WriteAddr;

reg [31:0] WriteData;

wire [31:0] RsData;

wire [31:0] RtData;

regFile uut(clk,reset,regwr,RsAddr,RtAddr,WriteAddr,WriteData,RsData,RtData);

parameter PERIOD = 10; //clk period 10ns

always begin

clk = 1'b0;

#(PERIOD/2) clk = 1'b1; //rising edge

#(PERIOD/2); //falling edge

end

initial begin

reset=1; //reset 14 ns

RsAddr=3;

RtAddr=0;

WriteAddr=5;

WriteData=8;

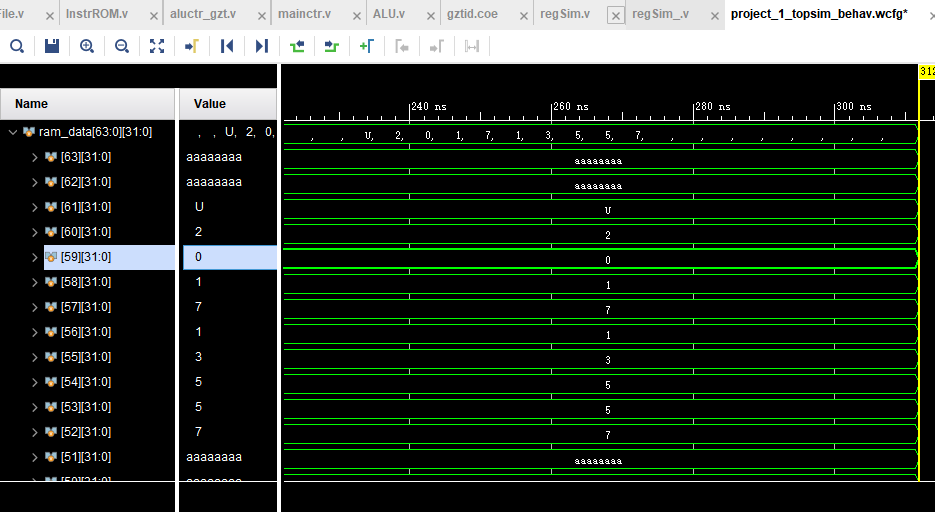
regwr=1;

#14 reset=0; //20ns writedata stored in writeaddr $5=8

#20 RsAddr=5; //34ns RsData=8;

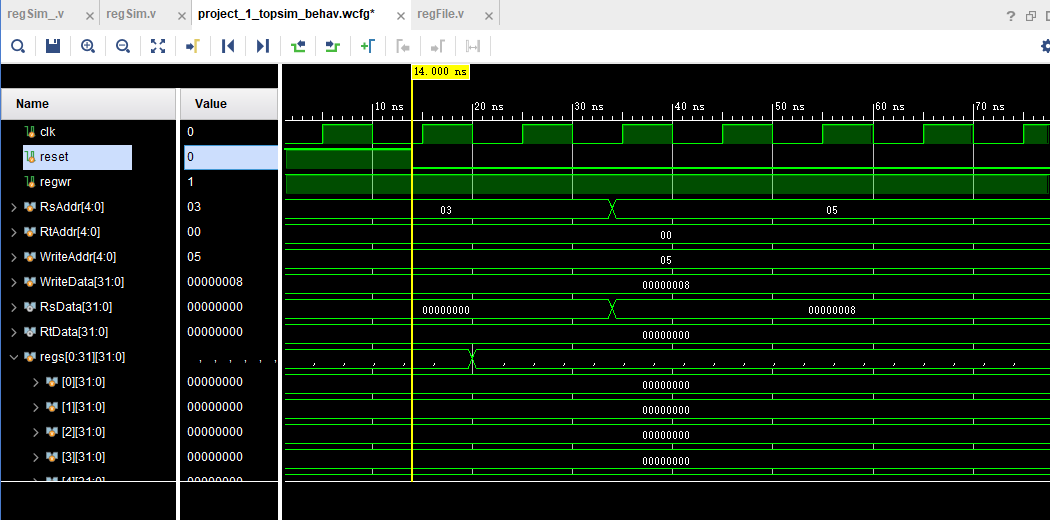
end

endmodule

仿真截图

验证Regfile模块正确性

首先将光标移到14ns处，reset一直为有效电平，故regs值一直为0，符合复位功能要求。后reset信号无效，regwr信号有效，此时WriteData数据在时钟下降沿写入WriteAddr编号寄存器，即在20ns时数据8写入5号寄存器，如图所示，符合同步写入功能要求。寄存器数据为异步输出，与时钟无关，因此寄存器输入地址一旦发生变化，相应数据直接输出。34ns之前RsAddr的值为3不变，因此输出3号寄存器的值为0；34ns之后RsAddr的值为5，且5号寄存器的值在此之前被修改为8，因此RsData直接输出8，而RtData一直为0.如图所示。



心得体会

在写代码的过程当中，当写完源代码以及仿真激励代码之后，发现运行是不成功的，回过头来对着每段代码进行观察，最后发现是书本上的ALU控制器模块中没有出现addi这个运算单元，导致在后面的仿真当中不能执行。在处理完这个问题之后，也可以开始仿真，然而在仿真过程当中，会出现无法观测仿真波形，在同学的指导下，才知道要先restart一下，再run，再restart，再run就会没有问题了。实际上，在导出regs，ram\_data的波形之后，还是未对寄存器的执行方式有着太清楚的认识，直到当导出spo即指令码所对应的波形之后，才能通过指令码对应的数字以及寄存器数值的变化较清楚地认识处理器是如何通过外界的指令完成所要达到的目的。



