实验报告五

PB14000556 陈晓彤

实验题目：设计一个单周期CPU 并成功运行一个计算Fibonacci数列的程序

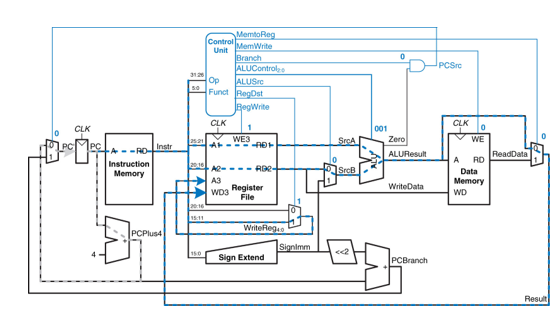
实验要求：

1. CPU类型为MIPS体系结构
2. 能运行add addi lw sw bgtz j六条指令
3. 核心部件为InstructionMemory RegisterFile ALU DataMemory ControlUnit
4. 设计合理的控制信号，保证系统在单周期执行完指令

设计思路：

1. DataMemory 由ipcore—distributed single port RAM实现，异步读取，同步写入，初始化文件命名为RAM\_initial.coe 其内容为Mars编译出的汇编指令数据段，但一定要注意修改格式：各个单元数据之间要加逗号，开头两行要加上MEMORY\_INITIALIZATION\_RADIX=10; MEMORY\_INITIALIZATION\_VECTOR=
2. Instruction Memory由ipcore—distributed ROM实现，异步读取，初始化文件命名为RAM\_initial.coe 其内容为Mars编译出的汇编指令代码段，同样注意修改格式
3. RegisterFile 由32个32位寄存器实现，采用异步读取，同步写入的方式，初始化使0号寄存器内容为0
4. ControlUnit发出信号：MemtoReg MemWrite Bgtz RegDst RegWrite ALUSrc j
5. MemtoReg—选择存入寄存器的数据来源ALU/Memory
6. MemWrite RegWrite—数据内存和寄存器写使能
7. Bgtz j—跳转指令控制信号
8. RegDst—寄存器写地址来源 rt/rd
9. ALUSrc—alu\_b来源 reg\_read\_2/imm16
10. ALU只进行加法计算，控制信号固定为3’b010,alu\_a来源为寄存器，alu\_b来源由ALUSrc决定
11. PC同步更新，在PC+4 PC+4+bgtz offset j\_address 三者间选择PC\_next
12. I-type指令需要signextended部件保证数据对齐

具体原理图：



源代码：

综合系统：

module cpu(

input clk,

output [31:0] ALU\_result

);

parameter Instruction\_Memory\_size=10,Data\_Memory\_size=7;

reg [31:0] PC;

wire [4:0] Reg\_write\_address;

reg sign\_gtz;

wire [31:0] Reg\_write\_data;

reg [31:0] PC\_next;

wire [31:0] alu\_b;

wire [31:0] Instruction;

wire [31:0] signextended;

wire [31:0] j\_address;

wire [31:0] Mem\_read\_data;

wire [31:0] Reg\_read\_data1;

wire [31:0] Reg\_read\_data2;

wire RegDst,j,bgtz,MemtoReg,ALUSrc,RegWrite,MemWrite;

ROM Instruction\_Memory(

.a (PC[Instruction\_Memory\_size+1:2]),

.spo (Instruction[31:0])

);

ControlUnit ControlUnit1(

.opcode (Instruction[31:26]),

.RegDst (RegDst),

.j (j),

.bgtz (bgtz),

.MemtoReg (MemtoReg),

.MemWrite (MemWrite),

.ALUSrc (ALUSrc),

.RegWrite (RegWrite)

);

RAM Data\_Memory(

.clk (clk),

.a (ALU\_result[Data\_Memory\_size+1:2]),

.d (Reg\_read\_data2),

.we (MemWrite),

.spo (Mem\_read\_data)

);

Registers Registers1(

.clk (clk),

.read\_address1 (Instruction[25:21]),

.read\_address2 (Instruction[20:16]),

.write\_address (Reg\_write\_address),

.write\_data (Reg\_write\_data),

.write\_enable (RegWrite),

.read\_data1 (Reg\_read\_data1),

.read\_data2 (Reg\_read\_data2)

);

ALU ALU1(

.A (Reg\_read\_data1),

.B (alu\_b),

.op (3'b010),

.ALU\_result (ALU\_result)

);

SignExtend16 SignExtend16\_1(

.in (Instruction[15:0]),

.out (signextended)

);

SignExtend26 SignExtend26\_1(

.in (Instruction[25:0]),

.pc (PC[31:28]),

.out (j\_address)

);

always@(ALU\_result)

begin

sign\_gtz=(ALU\_result>0);

end

always@(\*)

begin

if(j)

PC\_next=j\_address;

else if(sign\_gtz && bgtz)

PC\_next=PC+32'b100+(signextended<<2);

else

PC\_next=PC+32'b100;

end

always@(posedge clk)

begin

if(PC\_next)

PC=PC\_next;

end

assign Reg\_write\_address=(RegDst)?(Instruction[15:11]):(Instruction[20:16]);

assign alu\_b=(ALUSrc)?(signextended):(Reg\_read\_data2);

assign Reg\_write\_data=(MemtoReg)?(Mem\_read\_data):(ALU\_result);

initial

begin

PC=32'h0;

end

endmodule

各个部件：

module ControlUnit(

input [5:0] opcode,

output bgtz,

output j,

output RegDst,

output ALUSrc,

output MemtoReg,

output MemWrite,

output RegWrite

);

wire add,lw,sw,addi;

assign bgtz=(opcode==6'd7);

assign j=(opcode==6'd2);

assign add=(opcode==6'b0);

assign lw=(opcode==6'd35);

assign sw=(opcode==6'd43);

assign addi=(opcode==6'd8);

assign RegDst=add;

assign ALUSrc=lw+sw+addi;

assign MemWrite=sw;

assign MemtoReg=lw;

assign RegWrite=add+lw+addi;

endmodule

module Registers(

input clk,

input [4:0] read\_address1,

input [4:0] read\_address2,

input [4:0] write\_address,

input [31:0] write\_data,

input write\_enable,

output [31:0] read\_data1,

output [31:0] read\_data2

);

reg [31:0] registers [31:0];

initial

begin

registers[0]=32'b0;

end

assign read\_data1=registers[read\_address1];

assign read\_data2=registers[read\_address2];

always@(posedge clk)

begin

if(write\_enable)

registers[write\_address]=write\_data;

end

endmodule

module ALU(

input [31:0] A,

input [31:0] B,

input [2:0] op,

output [31:0] ALU\_result

);

assign ALU\_result=(A+B)\*(op==3'b010)+(A-B)\*(op==3'b110)+(A|B)\*(op==3'b001);

endmodule

module SignExtend16(

input [15:0] in,

output [31:0] out

);

assign out[15:0]=in[15:0];

assign out[31]=in[15];assign out[30]=in[15];assign out[29]=in[15];assign out[28]=in[15];

assign out[27]=in[15];assign out[26]=in[15];assign out[25]=in[15];assign out[24]=in[15];

assign out[23]=in[15];assign out[22]=in[15];assign out[21]=in[15];assign out[20]=in[15];

assign out[19]=in[15];assign out[18]=in[15];assign out[17]=in[15];assign out[16]=in[15];

endmodule

module SignExtend26(

input [25:0] in,

input [3:0] pc,

output [31:0] out

);

assign out[31]=pc[3];assign out[30]=pc[2];assign out[29]=pc[1];assign out[28]=pc[0];

assign out[27:2]=in[25:0];

assign out[1]=0;assign out[0]=0;

endmodule

ROM与RAM初始化文件：

ROM（指令流）

MEMORY\_INITIALIZATION\_RADIX=16;

MEMORY\_INITIALIZATION\_VECTOR=

20080000,

200d0050,

8dad0000,

200b0054,

8d6b0000,

200c0054,

8d8c0004,

ad0b0000,

ad0c0004,

21a9fffe,

8d0b0000,

8d0c0004,

016c5020,

ad0a0008,

21080004,

2129ffff,

1d20fff9,

08000011,

0;

RAM（初始化数据）

MEMORY\_INITIALIZATION\_RADIX=16;

MEMORY\_INITIALIZATION\_VECTOR=

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000014,

00000003,

00000003,

00000000;

仿真结果：

