实验报告六

PB14000556 陈晓彤

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

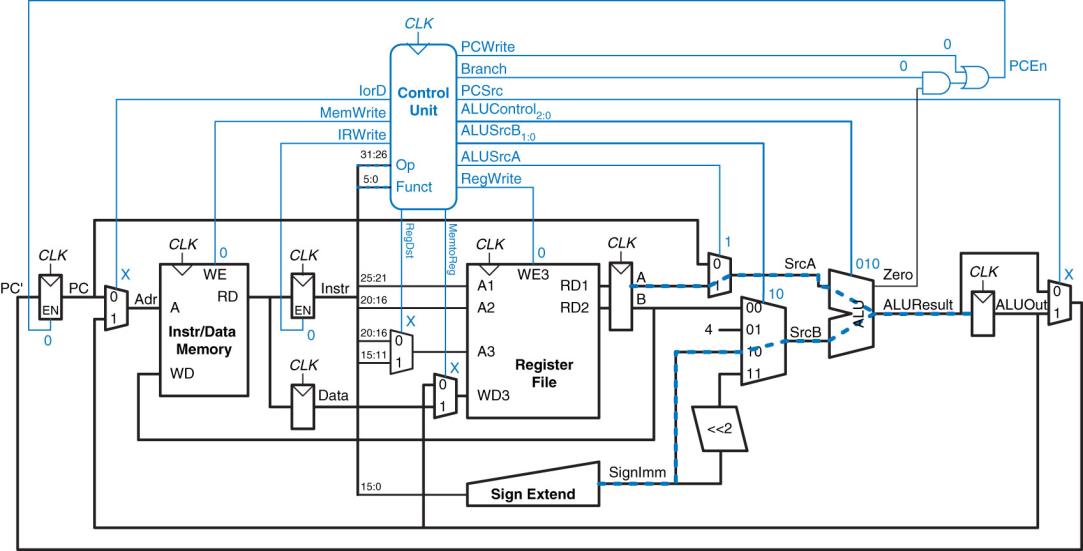
实验要求：

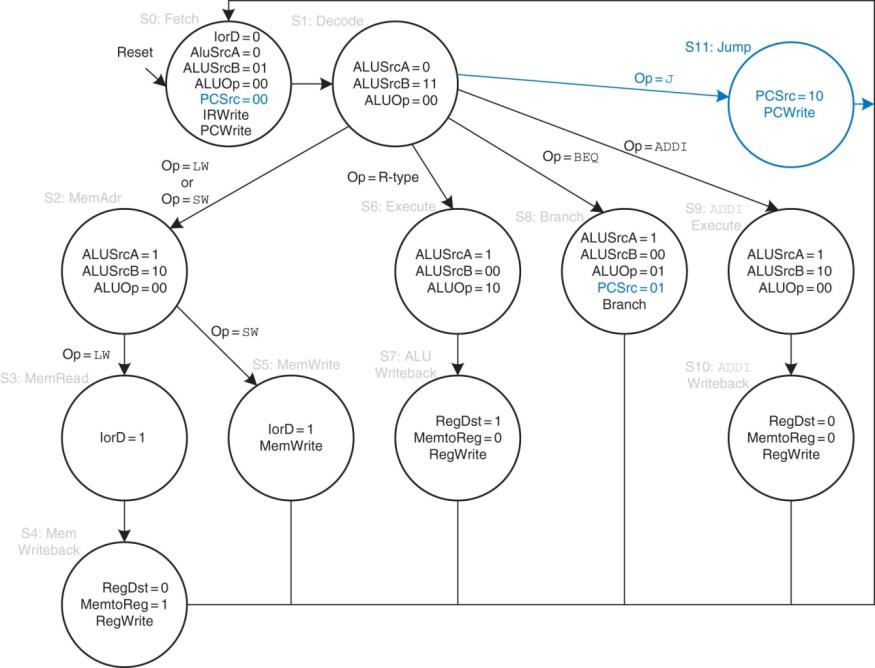
1. CPU类型为MIPS体系结构
2. 能运行add addi lw sw bgtz j六条指令
3. 核心部件为Memory RegisterFile ALU ControlUnit
4. 设计合理的状态机产生分时控制信号，保证系统在多周期执行完指令

设计思路：

1. Memory由ipcore—blocked RAM实现，同步读取，初始化文件Initial.coe 其内容为两部分，指令段为Mars编译出的汇编指令代码段，注意到指令有18行，所以为了凑整从第32个地址开始存数据段，所以中间要加上14个0，数据段前20个地址同样要填充0以存放后来算出的数列，所以总共要留出34个空位放0
2. RegisterFile 由32个32位寄存器实现，采用异步读取，同步写入的方式，初始化时0号寄存器内容为0
3. ControlUnit发出两类信号：一类是部件使能信号，包括PCWrite IRWrite MemWrite RegWrite Branch；另一类为数据选择信号，包括RegDst ALUSrcA ALUSrcB MemtoReg IorD PCSrc，作用时间见状态图，注意使能信号如下一状态不使用需要改为关闭状态以免发生数据错误，数据选择信号如下一周期不更改可不予赋值
4. ALU只进行加法计算,数据来源由ALUSrcA ALUSrcB决定
5. PC同步更新，在PC+4 PC+4+bgtz offset j\_address 三者间选择PC\_next
6. I-type指令需要signextended部件保证数据对齐

具体原理图：





注意：此状态图为异步时钟情况，改成同步时钟需要在S0和S1之间加一个状态，因为读Memory数据还需要延时一个时钟周期

源代码：

综合系统：

module cpu(

input clk,

input reset

);

reg [31:0] PC\_Reg;

reg [31:0] Instr\_Reg;

reg [31:0] ALU\_Reg;

reg [31:0] Data;

wire [31:0] Mem\_Addr;

wire [31:0] Mem\_Out;

reg [31:0] Instruction;

wire [31:0] Reg\_Addr3;

wire [31:0] Reg\_In;

reg [31:0] Reg\_ALU;

wire [31:0] Signextended;

wire [31:0] Shifted;

wire [31:0] A;

wire [31:0] B;

reg [31:0] Reg\_RD1;

reg [31:0] Reg\_RD2;

wire [31:0] ALU\_Result;

wire [31:0] ALU\_A;

wire [31:0] ALU\_B;

wire ALU\_Zero;

reg [31:0] ALU\_Out;

wire [31:0] PC\_Next;

wire [31:0] PC\_Jump;

wire PCen;

wire MemRead;

wire MemWrite;

wire RegDst;

wire IRWrite;

wire PCWrite;

wire bgtz;

wire MemtoReg;

wire ALUSrcA;

wire [1:0] ALUSrcB;

wire [1:0] PCSrc;

wire IorD;

wire RegWrite;

Memory Memory1(

.clka (clk),

.wea (MemWrite),

.addra (Mem\_Addr[11:2]),

.dina (Reg\_RD2),

.douta (Mem\_Out)

);

ControlUnit ControlUnit1(

.reset (reset),

.opcode (Instruction[31:26]),

.clk (clk),

.RegDst (RegDst),

.IRWrite (IRWrite),

.PCWrite (PCWrite),

.bgtz (bgtz),

.MemtoReg (MemtoReg),

.MemWrite (MemWrite),

.ALUSrcA (ALUSrcA),

.ALUSrcB (ALUSrcB),

.PCSrc (PCSrc),

.RegWrite (RegWrite),

.IorD (IorD)

);

Registers Registers1(

.clk (clk),

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

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

.write\_address (Reg\_Addr3),

.write\_data (Reg\_In),

.write\_enable (RegWrite),

.read\_data1 (A),

.read\_data2 (B)

);

ALU ALU1(

.A (ALU\_A),

.B (ALU\_B),

.ALU\_Zero (ALU\_Zero),

.ALU\_result (ALU\_Result)

);

SignExtend SignExtend\_1(

.in (Instruction[15:0]),

.out (Signextended)

);

Get\_PC\_Jump Get\_PC\_Jump\_1(

.in (Instruction[25:0]),

.pc (PC\_Reg[31:28]),

.out (PC\_Jump)

);

assign Shifted[31:2]=Signextended[29:0];

assign Shifted[1:0]=2'b00;

assign Mem\_Addr=IorD?ALU\_Reg:PC\_Reg;

assign Reg\_Addr3=RegDst?Instruction[15:11]:Instruction[20:16];

assign Reg\_In=MemtoReg?Mem\_Out:ALU\_Reg;

assign ALU\_A=ALUSrcA?Reg\_RD1:PC\_Reg;

assign ALU\_B=(ALUSrcB==2'b00)?Reg\_RD2:((ALUSrcB==2'b01)?4:((ALUSrcB==2'b10)?Signextended:Shifted));

always@(posedge clk or negedge reset)

begin

if(~reset)

PC\_Reg<=32'b0;

else if(PCen)

PC\_Reg<=PC\_Next;

end

always@(posedge clk)

begin

Reg\_RD1<=A;

Reg\_RD2<=B;

end

always@(posedge clk)

begin

if(IRWrite)

Instruction<=Mem\_Out;

end

always@(posedge clk)

begin

ALU\_Reg<=ALU\_Result;

end

assign PC\_Next=(PCSrc==2'b00)?ALU\_Result:((PCSrc==2'b01)?ALU\_Reg:PC\_Jump);

assign PCen=PCWrite+(ALU\_Zero\*bgtz);

endmodule

RAM初始化文件：

MEMORY\_INITIALIZATION\_RADIX=16;

MEMORY\_INITIALIZATION\_VECTOR=

20080080,

200d00d0,

8dad0000,

200b00d4,

8d6b0000,

200c00d4,

8d8c0004,

ad0b0000,

ad0c0004,

21a9fffe,

8d0b0000,

8d0c0004,

016c5020,

ad0a0008,

21080004,

2129ffff,

1d20fff9,

08000011,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

00000000,

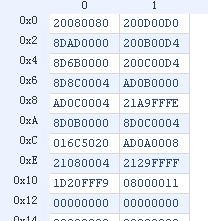
00000000,

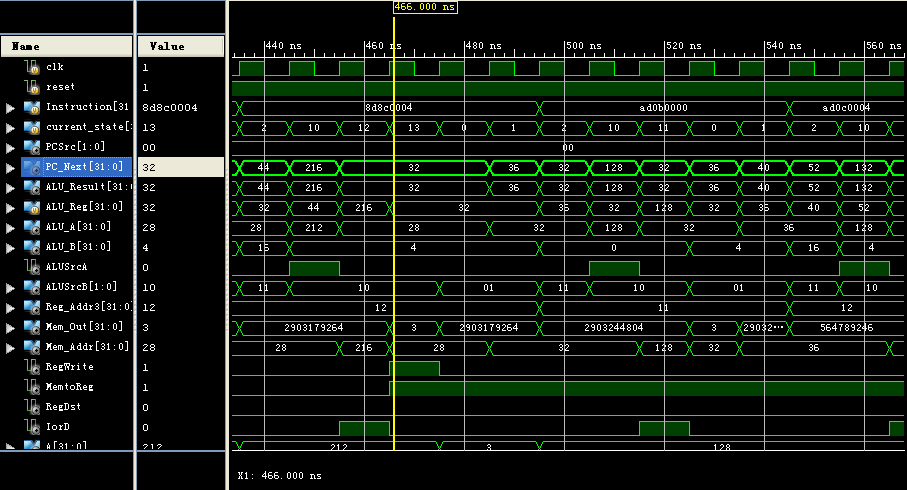
00000014,

00000003,

00000003,

00000000;

仿真结果

：