计算机系统结构实验 Lab3

David Wang

2020 年 5 月

1 概述

1.1 实验名称

简单的类 MIPS 处理器部件实现：控制器、ALU

1.2 实验目的

1. 理解 CPU 控制器、ALU 控制器、ALU 的原理
2. 实现主控制器 Ctr
3. 实现 ALU 控制器 ALUCtr
4. 实现 ALU
5. 功能仿真

2 主控制器单元模块

2.1 模块描述

主控制器单元模块以指令的高六位 opcode 作为输入，解码后给 ALUCtr、数据存储器、寄存器、多路选择器等输出正确的控制信号。

MIPS 指令集主要可以分为 R-指令, I-指令, J-指令三种。其中, R-指令是运算指令, 运算数来源和写入的目的地均为寄存器。最低的六位——funct 域控制运算种类，供 ALUCtr 使用 opcode 均为 0。而 I-指令与 J-指令均只需要 opcode 作为控制信号。所以，主控制器只需要输入 opcode 进行信号解析。

表 1: 不同类型指令的 opcode

指令	**opcode**
R-指令	000000
lw	100011
sw	101011
beq	000100
j	000010

表 2: 译码功能

指令类型	regDest	ALUSrc	MemtoReg	regWrite	memRead	memWrite	Branch	Jump	Aluop
R-指令	1	0	0	1	0	0	0	0	10
lw	0	1	1	1	1	0	0	0	00
sw	X	1	X	0	0	1	0	0	00
beq	X	0	X	0	0	0	1	0	01
j	0	0	0	0	0	0	0	1	01

2.2 模块实现

```
module Ctr(
    input [5:0] opCode,
    output reg regDest,
    output reg aluSrc,
    output reg memToReg,
    output reg regWrite,
    output reg memRead,
    output reg memWrite,
    output reg Branch,
    output reg [1:0] ALUop,
    output reg Jump
    );
    always @(opCode)
    begin
        case(opCode)
            6'b000000://R-type
            begin
                regDest=1;
                aluSrc=0;
                memToReg=0;
                regWrite=1;
                memRead=0;
                memWrite=0;
                Branch=0;
                Jump=0;
                ALUop=2'b10;
```

```
            end
            6'b000010://jump
            begin
                regDest=0;
                aluSrc=0;
                memToReg=0;
                regWrite=0;
                memRead=0;
                memWrite=0;
                Branch=0;
                Jump=1;
                ALUop=2'b00;
            end
            6'b100011://lw
            begin
                regDest=0;
                aluSrc=1;
                memToReg=1;
                regWrite=1;
                memRead=1;
                memWrite=0;
                Branch=0;
                Jump=0;
                ALUop=2'b00;
            end
            6'b101011://sw
            begin
                regDest=0
                aluSrc=1;
                memToReg=0;
                regWrite=0;
                memRead=0;
                memWrite=1;
                Branch=0;
                Jump=0;
                ALUop=2'b00;
            end
            6'b000100://beq
            begin
                regDest=0;
                aluSrc=0;
                memToReg=0;
                regWrite=0;
                memRead=0;
                memWrite=0;
                Branch=1;
                Jump=0;
                ALUop[0]=2'b01;
            end
        endcase
```

```
    end
endmodule
```

2.3 仿真程序

```
    module Ctr_tb(

    );
        reg [5:0] opCode;
        wire regDest;
        wire aluSrc;
        wire memToReg;
        wire regWrite;
        wire memRead;
        wire memWrite;
        wire Branch;
        wire [1:0] ALUop;
        wire Jump;

        Ctr uut (
                .opCode(opCode),
                .regDest(regDest),
                .aluSrc(aluSrc),
                .memToReg(memToReg),
                .regWrite(regWrite),
                .memRead(memRead),
                .memWrite(memWrite),
                .Branch(Branch),
                .ALUop(ALUop),
                .Jump(Jump)
        );

        initial begin
                opCode = 0;
                #200;
                #100 opCode = 6'b100011;
                #100 opCode = 6'b101011;
                #100 opCode = 6'b000100;
                #100 opCode = 6'b000010;
                #100 opCode = 6'b010101;
        end
endmodule
```

2.4 仿真波形图

如下图所示，波形图符合理论预期。

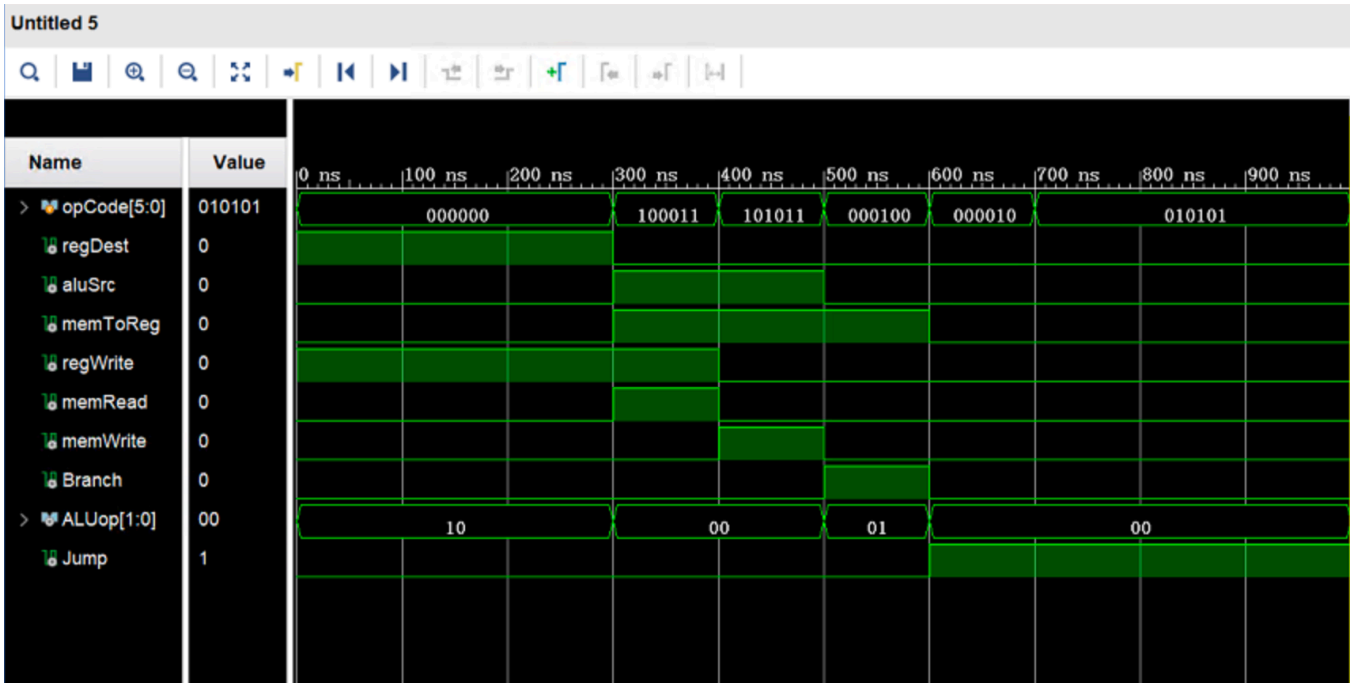

图 1:

3 ALU 控制器模块

3.1 模块描述

ALU 控制器模块以主控制器输出的 ALUop 和指令的低六位 funct 域作为输入，向 ALU 输出四位的操作指令。

ALU control lines	Function
0000	and
0001	or
0010	add
0110	subtract
0110	set on less than
1100	nor

表 3: ALUCtr 输出与 ALU 操作的对应关系

Instruction opcode	ALUop	funct field	ALU action	ALU control input
lw	00	XXXXXX	add	0010
sw	00	XXXXXX	add	0010
beq	01	XXXXXX	subtract	0110
add	10	100000	add	0010
sub	10	100010	subtract	0110
and	10	100100	and	0000
or	10	100101	or	0001
slt	10	101010	set on less than	0111

表 4: Funct,ALUop 与 ALU 控制信号的编码关系

ALUop	Funct field	operation
00	XXXXXX	0010
X1	XXXXXX	0110
1X	XX0000	0010
1X	XX0010	0110
1X	XX0100	0000
1X	XX0101	0001
1X	XX1010	0111

表 5: 输入输出真值表

3.2 模块实现

实现 ALUCtr 只需要将上述真值表表达出来。

```
    module ALUCtr(
    input [1:0] ALUop,
    input [5:0] functField,
    output reg [3:0] operation
    );
    always @ (ALUop or functField)
    begin
        casex({ALUop,functField})
            8'b00xxxxxx:operation=4'b0010;
            8'bx1xxxxxx:operation=4'b0110;
            8'b1xxx0000:operation=4'b0010;
            8'b1xxx0010:operation=4'b0110;
            8'b1xxx0100:operation=4'b0000;
            8'b1xxx0101:operation=4'b0001;
            8'b1xxx1010:operation=4'b0111;
        endcase
    end
endmodule
```

3.3 仿真程序

```
    module ALUCtr_tb();
    reg [1:0] ALUop;
    reg [5:0] functField;
```

```
    wire [3:0] operation;
    ALUCtr uu(
        .ALUop(ALUop),
        .functField(functField),
        .operation(operation)
    );
    initial begin
        ALUop=2'b00;
        functField=6'b000000;
        #100;
        ALUop=2'b00;
        functField=6'b000001;
        #100
        ALUop=2'b00;
        functField=6'b001101;
        #100
        ALUop=2'b01;
        functField=6'b100100;
        #100
        ALUop=2'b10;
        functField=6'b100000;
        #100
        functField=6'b010010;
        #100
        functField=6'b010100;
        #100
        functField=6'b100101;
        #100
        functField=6'b011010;
    end
endmodule
```

3.4 仿真波形图

如下图所示，波形图符合理论预期。

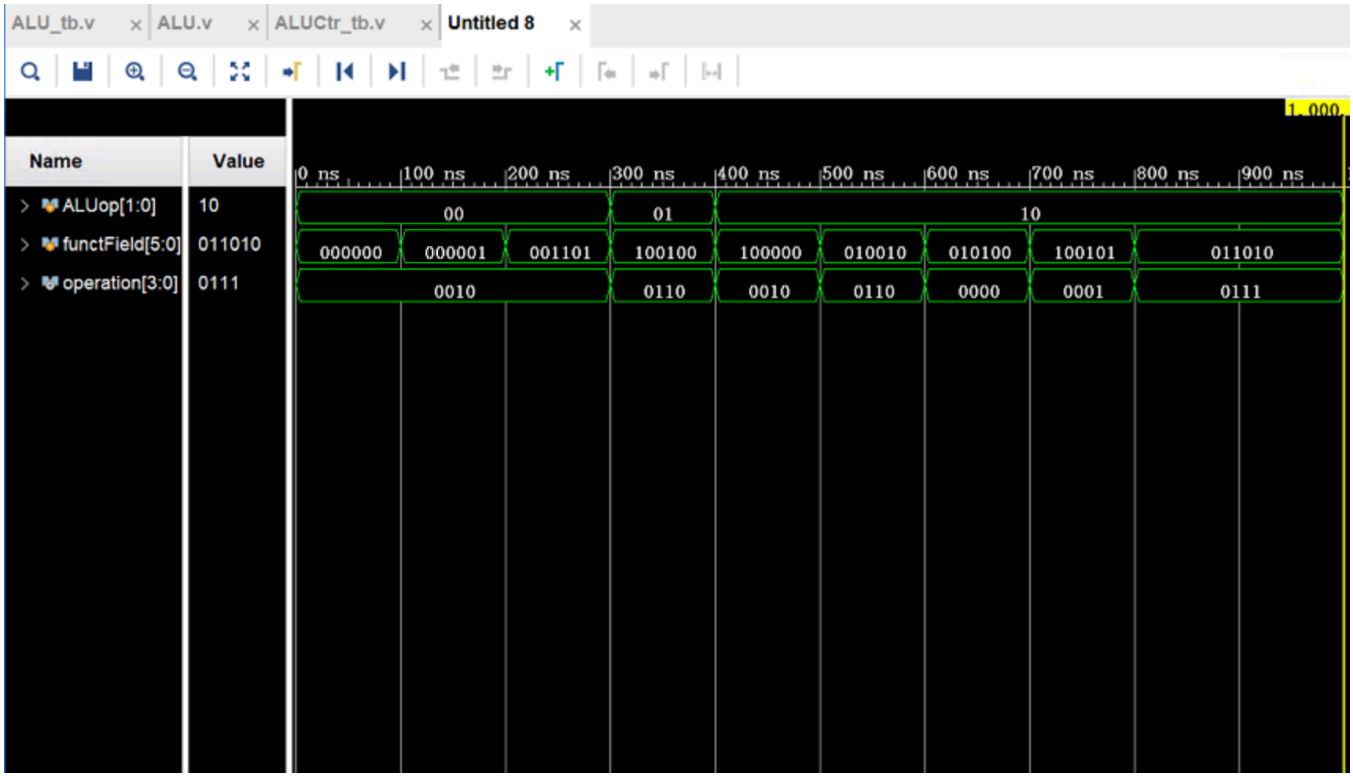

图 2:

4 ALU

4.1 模块描述

ALU 根据 ALUCtr 的信号将两个输入做算术操作，输出一个结果。如果运算结果为 0，则将 Zero 置为 1。

ALU 控制器输入 ALU 的信号与 ALU 的运算的关系如下表。

ALU control lines	Function
0000	and
0001	or
0010	add
0110	subtract
0111	set on less than
1100	nor

表 6: ALUCtr 和 ALU 操作对应关系

4.2 模块实现

```
1    module ALU(
2    input [31:0] input1,
3    input [31:0] input2,
4    input [3:0] aluCtr,
5    output reg zero,
6    output reg [31:0] aluRes
```

```
    );
    always @(input1 or input2 or aluCtr)
    begin
        case(aluCtr)
            4'b0010:
                begin
                    aluRes=input1+input2;
                    if(aluRes==0) zero=1;
                    else zero=0;
                end
            4'b0110:
                begin
                    aluRes=input1-input2;
                    if(aluRes==0) zero=1;
                    else zero=0;
                end
            4'b0000:
                begin
                    aluRes=input1 & input2;
                    if(aluRes==0) zero=1;
                    else zero=0;
                end
            4'b0001:
                begin
                    aluRes=input1 | input2;
                    if(aluRes==0) zero=1;
                    else zero=0;
                end
            4'b0111:
                begin
                    if(input1<input2) aluRes=1;
                    else aluRes=0;
                end
            4'b1100:
                begin
                    aluRes=~(input1|input2);
                    if(aluRes==0) zero=1;
                    else zero=0;
                end
        endcase
    end
endmodule
```

4.3 仿真程序

```
    module ALU_tb();
    reg [31:0] input1;
    reg [31:0] input2;
    reg [3:0] aluCtr;
```

```
    wire zero;
    wire [31:0] aluRes;

    ALU alu(
        .input1(input1),
        .input2(input2),
        .aluCtr(aluCtr),
        .zero(zero),
        .aluRes(aluRes)
    );
    initial begin
        input1=25;
        input2=13;
        aluCtr=0;
        #100
        aluCtr=4'b0000;
        #100
        aluCtr=4'b0001;
        #100
        aluCtr=4'b0010;
        #100
        aluCtr=4'b0110;
        #100
        aluCtr=4'b0111;
        #100
        aluCtr=4'b1100;
    end
endmodule
```

4.4 仿真波形图

如下图所示，波形符合预期。

图 3:

5 总结与反思

5.1 重点和难点

本次实验的指导比 1、2 更加简略。这需要我们进行独立思考。这三个模块主要涉及真值表的复现，需要用到 case 或 casex 语句，需要处理的条件分支比较多，这需要我们细心地写代码。

5.2 实验总结与感想

这次实验我实现了主控制器、ALU 控制器、ALU 三个模块。它们对于 MIPS 处理器的正确运作有至关重要的作用。这次实验的指导比 1 和 2 少了一些，锻炼了我独立思考与自学的能力。在 case 语句中处理多种情况也锻炼了我缜密思考细致考虑的能力。总的来说，当仿真波形与理论相符时我感到十分欣慰，这次实验激发了我对于 MIPS 处理器的兴趣。