lab03 report

郑航 520021911347

1 实验目的

- 理解主控制部件或单元、 ALU 控制器单元、 ALU 单元的原理
- 熟悉所需的 Mips 指令集
- 主控制器 (Ctr) 的实现
- ALU控制器 (ALUCtr) 的实现
- 算术逻辑运算单元 (ALU) 的实现
- 使用 Vivado 进行功能模块的行为仿真

2 实验原理

2.1 Ctr原理分析

2.1.1 模块描述

- 功能：对指令的高6位OpCode 进行分析，对不同的指令和指令类型发出对应的一系列控制信号
- 输入：6位的OPCode（2.1.2详细说明）
- 输出：一系列控制信号（2.1.3详细说明）

2.1.2 输入的OpCode

不同的OpCode对应不同的指令类型和指令，对应关系如下表：

OpCode	指令
000000	R型
000010	j
000100	beq
100011	lw
101011	sw

2.1.3 输出的控制信号

Ctr是控制中心，需要根据输入指令，输出一系列控制信号指导各部件的功能执行，具体的信号和信号对应含义如下表：

输出信号	含义
ALUSrc	算术逻辑运算单元 (ALU) 的第二个操作数的来源 (0: 使用 rt； 1: 使用立即数)
ALUOp	ALU控制信号，发送给运算单元控制器 (ALUCtr) 用来进一步解析运算类型
Branch	条件跳转信号，高电平说明当前指令是条件跳转指令 (branch)
Jump	无条件跳转信号，高电平说明当前指令是无条件跳转指令 (jump)
memRead	内存读的enable信号，高电平说明当前指令需要进行内存读取 (load)
memToReg	写寄存器的数据来源 (0: 使用 ALU 运算结果； 1: 使用内存读取数据)
memWrite	内存写的enable信号，高电平说明当前指令需要进行内存写入 (store)
regDst	目标寄存器选择信号 (0: 写入 rt 代表的寄存器； 1: 写入 rd 代表的寄存器)
regWrite	寄存器写的enable信号，高电平说明当前指令需要进行寄存器写入

2.1.4 OpCode和输出信号的对应关系

输出信号	000000	000010	000100	100011	101011
ALUSrc	0	0	0	1	1
ALUOp	1x	00	01	00	00
Branch	0	0	1	0	0
Jump	0	1	0	0	0
memRead	0	0	0	1	0
memToReg	0	0	0	1	0
memWrite	0	0	0	0	1
regDst	1	0	0	0	0
regWrite	1	0	0	1	0

2.2 ALUCtr原理分析

2.2.1 模块描述

- 结合Ctr传入的ALUOp和指令的后6位Funct，综合分析后通过ALUCtrOut指导ALU执行具体的功能
- 输入：2位的ALUOp和6位的Funct
- 输出：4位的ALUCtrOut

2.2.2 输入的ALUOp和Funct和输出的ALUCtrOut的对应关系

ALUOp	Funct	ALUCtrOut	对应指令	ALU操作
1x	100000	0010	add	加法
1x	100010	0110	sub	减法
1x	100100	0000	and	逻辑与
1x	100101	0001	or	逻辑或
1x	101010	0111	slt	小于时置位
00	xxxxxx	0010	lw	加法
00	xxxxxx	0010	sw	加法
01	xxxxxx	0110	beq	减法
00	xxxxxx	0010	j	加法

2.3 ALU原理分析

2.3.1 模块描述

- 对两个输入的数据，根据ALUCtrOut指定的运算方式对其进行运算，并输出运算结果，并根据结果是否为零将zero进行置位
- 输入：32位的inputA和inputB，4位的ALUCtrOut
- 输出：32位的aluRes，即计算结果；1位的zero，本质上是一个置位行为，若aluRes为0则将其置为1，否则为0

2.3.2 ALUCtrOut与ALU运算方式的对应关系

ALUCtrOut	ALU运算方式
0000	逻辑与
0001	逻辑或
0010	加法
0110	减法
0111	小于时置位

3 实验过程

3.1 Ctr

考虑该模块功能是对不同的输入进行对应的输出，因此主体设计为一个case块，对不同的输入情况给予不同的反应，对各输出信号进行对应赋值

注意到，为了防止没有case与输入对应的情况，应该设置default的处理方法，在此设计为将每个信号都设为0，可以防止进行误操作

完整代码如下：

```
module Ctr(
    input [5:0] opCode,
    output regDst,
    output aluSrc,
    output memToReg,
    output regWrite,
    output memRead,
    output memWrite,
    output branch,
    output [1:0] aluOp,
    output jump
    );

    reg RegDst;
    reg ALUSrc;
    reg MemToReg;
    reg RegWrite;
    reg MemRead;
    reg MemWrite;
    reg Branch;
```

```
    reg [1:0] ALUOp;
    reg Jump;

    always @(opCode)
    begin
        case(opCode)
        6'b000000: //R type
        begin
            RegDst = 1;
            ALUSrc = 0;
            MemToReg = 0;
            RegWrite = 1;
            MemRead = 0;
            MemWrite = 0;
            Branch = 0;
            ALUOp = 2'b10;
            Jump = 0;
        end
        6'b100011: //lw
        begin
            RegDst = 0;
            ALUSrc = 1;
            MemToReg = 1;
            RegWrite = 1;
            MemRead = 1;
            MemWrite = 0;
            Branch = 0;
            ALUOp = 2'b00;
            Jump = 0;
        end
        6'b101011: //sw
        begin
            RegDst = 0;
            ALUSrc = 1;
            MemToReg = 0;
            RegWrite = 0;
            MemRead = 0;
            MemWrite = 1;
            Branch = 0;
            ALUOp = 2'b00;
            Jump = 0;
        end
        6'b000100: //beq
        begin
            RegDst = 0;
            ALUSrc = 0;
            MemToReg = 0;
            RegWrite = 0;
            MemRead = 0;
            MemWrite = 0;
            Branch = 1;
            ALUOp = 2'b01;
            Jump = 0;
        end
        6'b000010: //j
        begin
            RegDst = 0;
            ALUSrc = 0;
```

```
            MemToReg = 0;
            RegWrite = 0;
            MemRead = 0;
            MemWrite = 0;
            Branch = 0;
            ALUOp = 2'b00;
            Jump = 1;
        end
        default:
        begin
            RegDst = 0;
            ALUSrc = 0;
            MemToReg = 0;
            RegWrite = 0;
            MemRead = 0;
            MemWrite = 0;
            Branch = 0;
            ALUOp = 2'b00;
            Jump = 0;
        end
        endcase
    end

    assign regDst = RegDst;
    assign aluSrc = ALUSrc;
    assign memToReg = MemToReg;
    assign regWrite = RegWrite;
    assign memRead = MemRead;
    assign memWrite = MemWrite;
    assign branch = Branch;
    assign aluOp = ALUOp;
    assign jump = Jump;
endmodule
```

3.2 ALUCtr

与3.1Ctr功能类似，也是将对应输入转化为对应输出信号，因此程序主体是一个casex块（即带通配符x的case块）

注意到case中，利用了实验指导书中介绍的位拼接符号{，}对两个输入信号拼接后进行分析，有效提高了编程效率，让程序更简洁

完整代码如下：

```
module ALUCtr(
    input [1 : 0] aluOp,
    input [5 : 0] funct,
    output [3 : 0] aluCtrOut
    );

    reg [3 : 0] ALUCtrOut;

    always @ (aluOp or funct)
    begin
        casex ({aluOp, funct})
            8'b00xxxxxx:                //lw sw j
                ALUCtrOut = 4'b0010;
```

```
            8'b01xxxxx:                 //beq
                ALUCtrOut = 4'b0110;
            8'b10100000:                //add
                ALUCtrOut = 4'b0010;
            8'b10100010:                //sub
                ALUCtrOut = 4'b0110;
            8'b10100100:                //and
                ALUCtrOut = 4'b0000;
            8'b10100101:                //or
                ALUCtrOut = 4'b0001;
            8'b10101010:                //slt
                ALUCtrOut = 4'b0111;
        endcase
    end

    assign aluCtrOut = ALUCtrOut;
endmodule
```

3.3 ALU

模块主要功能是根据ALUCtrOut，对两个操作数进行对应的运算，故也是采用一个case块进行编写，并在得到结果后判断结果是否为0，对zero进行置位

完整代码如下：

```
module ALU(
    input [31 : 0] inputA,
    input [31 : 0] inputB,
    input [3 : 0] aluCtrOut,
    output zero,
    output [31 : 0] aluRes
    );

    reg Zero;
    reg [31 : 0] ALURes;

    always @ (inputA or inputB or aluCtrOut)
    begin
        case (aluCtrOut)
            4'b0000:    // and
                ALURes = inputA & inputB;
            4'b0001:    // or
                ALURes = inputA | inputB;
            4'b0010:    // add
                ALURes = inputA + inputB;
            4'b0110:    // sub
                ALURes = inputA - inputB;
            4'b0111:    // set on less than
                ALURes = ($signed(inputA) < $signed(inputB));
            4'b1100:    // nor
                ALURes = ~(inputA | inputB);
            default:
                ALURes = 0;
        endcase
        if (ALURes == 0)
            Zero = 1;
        else
```

```
            Zero = 0;
    end

    assign zero = Zero;
    assign aluRes = ALURes;
endmodule
```

4 实验结果

4.1 Ctr仿真结果

使用 Verilog 编写激励文件，采用软件仿真的形式对主控制器 (Ctr) 模块进行测试

4.1.1 激励文件编写

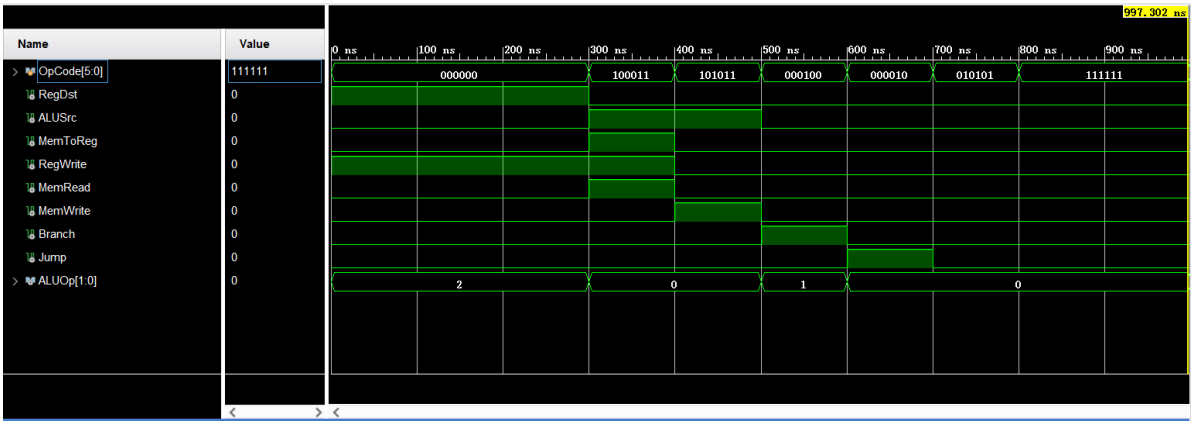
在激励文件中，我们每隔100ns改变一下OpCode的值对Ctr进行测试，其中最后一项是对错误OpCode的输入测试（应按default处理）

测试代码如下：

```
module Ctr_tb(
    );
    reg [5:0] OpCode;
    wire RegDst;
    wire ALUSrc;
    wire MemToReg;
    wire RegWrite;
    wire MemRead;
    wire MemWrite;
    wire Branch;
    wire Jump;
    wire [1:0] ALUOp;

    Ctr ctr(
        .opCode(OpCode),
        .regDst(RegDst),
        .aluSrc(ALUSrc),
        .memToReg(MemToReg),
        .regWrite(RegWrite),
        .memRead(MemRead),
        .memWrite(MemWrite),
        .branch(Branch),
        .aluOp(ALUOp),
        .jump(Jump)
    );

    initial begin
        OpCode = 0;

        #100;

        #100 OpCode = 6'b000000;
        #100 OpCode = 6'b100011;
        #100 OpCode = 6'b101011;
```

```
        #100 OpCode = 6'b000100;
        #100 OpCode = 6'b000010;
        #100 OpCode = 6'b010101;
        #100 OpCode = 6'b111111;

    end
endmodule
```

4.1.2 仿真图像



由图像可知，各输出控制信号都符合我们预期，实验成功

4.2 ALUCtr仿真结果

使用 Verilog 编写激励文件，采用软件仿真的形式对ALU控制器（ALUCtr) 模块进行测试

4.2.1 激励文件编写

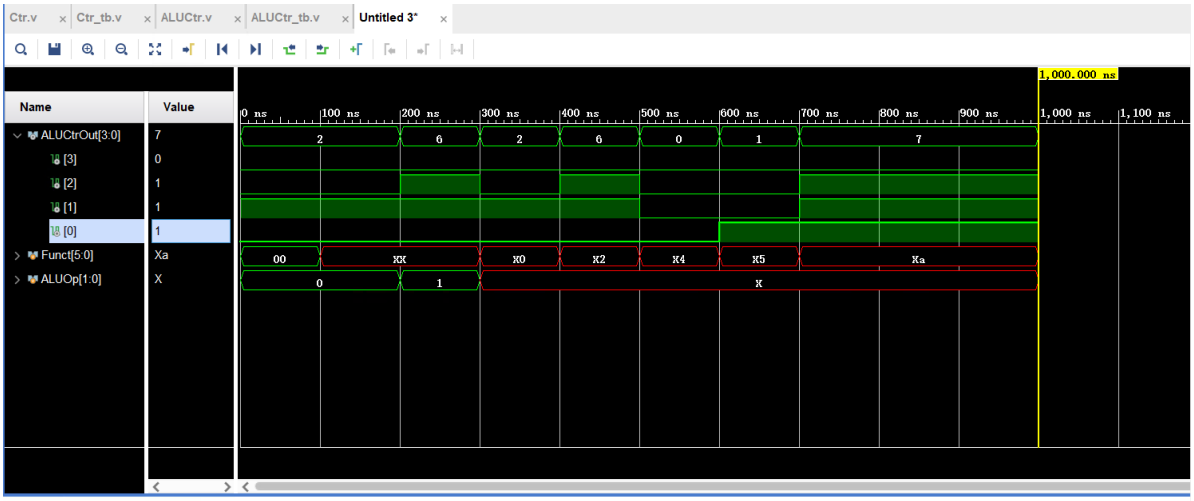
在激励文件中，我们每隔100ns改变一下ALUOp和Funct的值对ALUCtr进行测试

测试代码如下：

```
module ALUCtr_tb(
    );

    reg [1 : 0] ALUOp;
    reg [5 : 0] Funct;
    wire [3 : 0] ALUCtrOut;

    ALUCtr aluctr(.aluOp(ALUOp), .funct(Funct), .aluCtrOut(ALUCtrOut));

    initial begin
        // Initialize Inputs
        ALUOp = 0;
        Funct = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // testing
        ALUOp = 2'b00;
        Funct = 6'bxxxxxx;
        #100;

        ALUOp = 2'b01;
```

```
        Funct = 6'bxxxxxx;
        #100;

        ALUOp = 2'b1x;
        Funct = 6'bxx0000;
        #100;

        ALUOp = 2'b1x;
        Funct = 6'bxx0010;
        #100;

        ALUOp = 2'b1x;
        Funct = 6'bxx0100;
        #100;

        ALUOp = 2'b1x;
        Funct = 6'bxx0101;
        #100;

        ALUOp = 2'b1x;
        Funct = 6'bxx1010;
        #100;
    end
endmodule
```

4.2.2 仿真图像



由图像可知，ALUCtrOut可以根据不同的输入作出正确输出，符合我们预期，实验成功

4.3 ALU仿真结果

使用 Verilog 编写激励文件，采用软件仿真的形式对算术逻辑运算单元 (ALU) 模块进行测试

4.3.1 激励文件编写

在激励文件中，我们每隔100ns改变一下inputA、inputB和ALUCtrOut的值对ALU进行测试，对加法、减法、逻辑与、逻辑或、小于时置位、逻辑或非等算数逻辑运算进行了测试，对其中一些运算我们进行了两次测试（inputA和inputB位置对调）

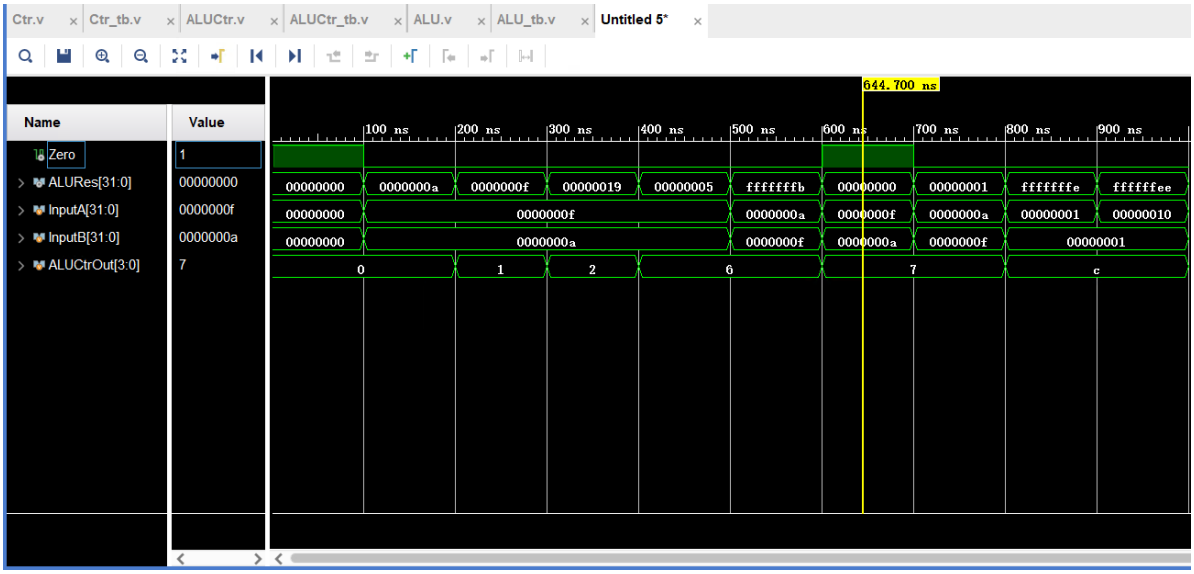
测试代码如下：

```
module ALU_tb(
    );
```

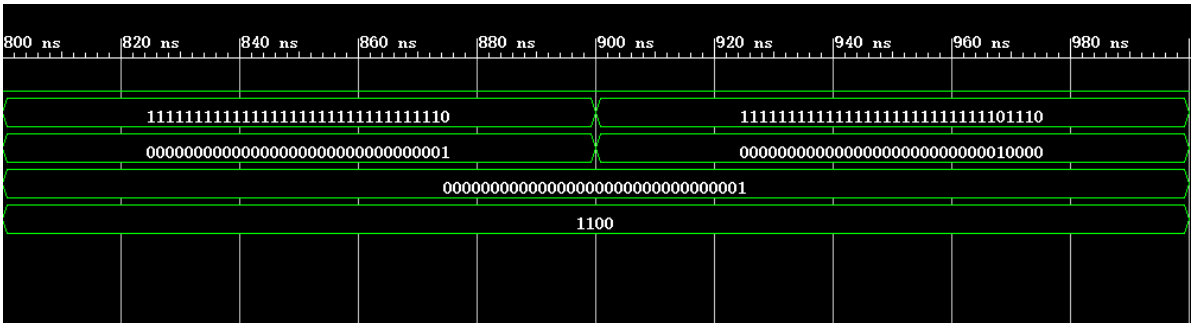
```
    wire [31 : 0] ALURes;
    reg [31 : 0] InputA;
    reg [31 : 0] InputB;
    reg [3 : 0] ALUCtrOut;
    wire Zero;

    ALU alu(.inputA(InputA), .inputB(InputB),
           .aluCtrOut(ALUCtrOut), .zero(Zero),
           .aluRes(ALURes));

    initial begin
        // Initialize Inputs
        InputA = 0;
        InputB = 0;
        ALUCtrOut = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // testing and
        InputA = 15;
        InputB = 10;
        ALUCtrOut = 4'b0000;
        #100;

        // testing or
        InputA = 15;
        InputB = 10;
        ALUCtrOut = 4'b0001;
        #100;

        // testing add
        InputA = 15;
        InputB = 10;
        ALUCtrOut = 4'b0010;
        #100;

        // testing sub 1
        InputA = 15;
        InputB = 10;
        ALUCtrOut = 4'b0110;
        #100;

        // testing sub 2
        InputA = 10;
        InputB = 15;
        ALUCtrOut = 4'b0110;
        #100;

        // testing set on less than 1
        InputA = 15;
        InputB = 10;
        ALUCtrOut = 4'b0111;
        #100;

        // testing set on less than 2
        InputA = 10;
```

```
61          InputB = 15;
62          ALUCtrOut = 4'b0111;
63          #100;
64
65          // testing nor 1
66          InputA = 1;
67          InputB = 1;
68          ALUCtrOut = 4'b1100;
69          #100;
70
71          // testing nor 2
72          InputA = 16;
73          InputB = 1;
74          ALUCtrOut = 4'b1100;
75          #100;
76      end
77  endmodule
```

4.3.2 仿真结果



其中特别展开并截取了逻辑或非nor的运算结果如下



检查可知，aluRes的输出正确，都符合我们的预期，在aluRes为0时zero也成功置位，可知实验成功

5 反思总结

本实验设计并实现了类 MIPS 处理器的三个重要组成部件：主控制器 (Ctr)、运算单元控制器 (ALUCtr) 以及算术逻辑运算单元 (ALU)，并且通过软件仿真模拟的方法验证了它们的正确性，为后面的单周期类 MIPS 处理器以及流水线处理器的实现奠定基础。

目前这三个实验模块只支持九条指令，功能较为简单，后续可以在此框架上进行扩展，通过增加一些case等来丰富模块功能，具体我们会在lab05中进行阐述

本实验的难度并没有想象中的大，主要就是对各种信号间的对应关系进行实现，所以用到了很多case和casex语句，也需要花费时间理一理各个信号的具体对应关系。实验开始时我由于惯性思维，并没有注意到如果含有通配符x，需要使用casex语句而不是case，后来通过错误信息发现了该错误。本实验难度适中，作为处理器系列实验的引入非常合适，让我对Verilog的语法有了更好的掌握，为后续完成较复杂的模块奠定了基础。