计算机体系结构实验

实验3报告书

作者：李珉超

学号：515030910361

2017.4.2

1 实验介绍

1.1 实验名称

简单的类MIPS单周期处理器实现- 控制器，ALU

1.2 实验目的

1.理解CPU控制器，ALU的原理

1.3 实验范围

1.ISE 13.4 的使用

2.Spartan 3E实验板的使用

3.使用VerilogHDL进行逻辑设计

4.CPU控制器的实现

5.ALU的实现

2 实验过程

2.1 Ctr

2.1.1 实验原理

Ctr 是CPU中负责decode opcode的硬件模块。

Ctr 接受一个带宽为6的信号作为输入，它是指令的最高6位。另外接受一个带宽为1的reset信号，作为初始化用。

Ctr 输出的信号中，除了aluOp的带宽为2，其他信号的带宽均为1。aluOp是aluControl的输入之一。

regDst用来说明写回register的地址是rt还是rd

jump用来说明当前指令是否为跳转指令

branch用来说明当前指令是否是分支指令

MemRead用来说明是否需要从data memory读取数据

MemWrite用来说明是否需要写数据到data memory

MemtoReg用来说明写回register的数据来自于data memory 还是ALU

ALUSrc用来说明ALU的一个操作数来自register还是sign extend

RegWrite用来说明register是否需要被更新

在本实验中，实现了jump, R-type, lw, sw, beq五种情况。

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

主控制模块真值表（OpCode 与控制输出的编码关系）

注：Jump 指令编码是 000010，Jump 信号输出 1，其余输出 0

2.1.2 实验模块

```
module Ctr(
    input [5:0] opCode,
    input reset,
    output reg regDst,
    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 @ (negedge reset)
    begin
        regDst = 0;
        aluSrc = 0;
        memToReg = 0;
        regWrite = 0;
        memRead = 0;
        memWrite = 0;
        branch = 0;
        aluOp = 2'b00;
        jump = 0;
    end

    always @ (opCode)
    begin
    case(opCode)
        6'b000010: // jump
        begin
```

```
        regDst = 0;
        aluSrc = 0;
        memToReg = 0;
        regWrite = 0;
        memRead = 0;
        memWrite = 0;
        branch = 0;
        aluOp = 2'b00;
        jump = 1;
    end

    6'b000000: // R
    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 = x;
        aluSrc = 1;
        memToReg = x;
        regWrite = 0;
        memRead = 0;
        memWrite = 1;
        branch = 0;
        aluOp = 2'b00;
        jump = 0;
    end

    6'b000100: // beq
    begin
        regDst = x;
        aluSrc = 0;
        memToReg = x;
        regWrite = 0;
        memRead = 0;
        memWrite = 0;
        branch = 1;
        aluOp = 2'b01;
        jump = 0;
    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
endmodule
```

2.1.3 仿真

初始化opCode为0

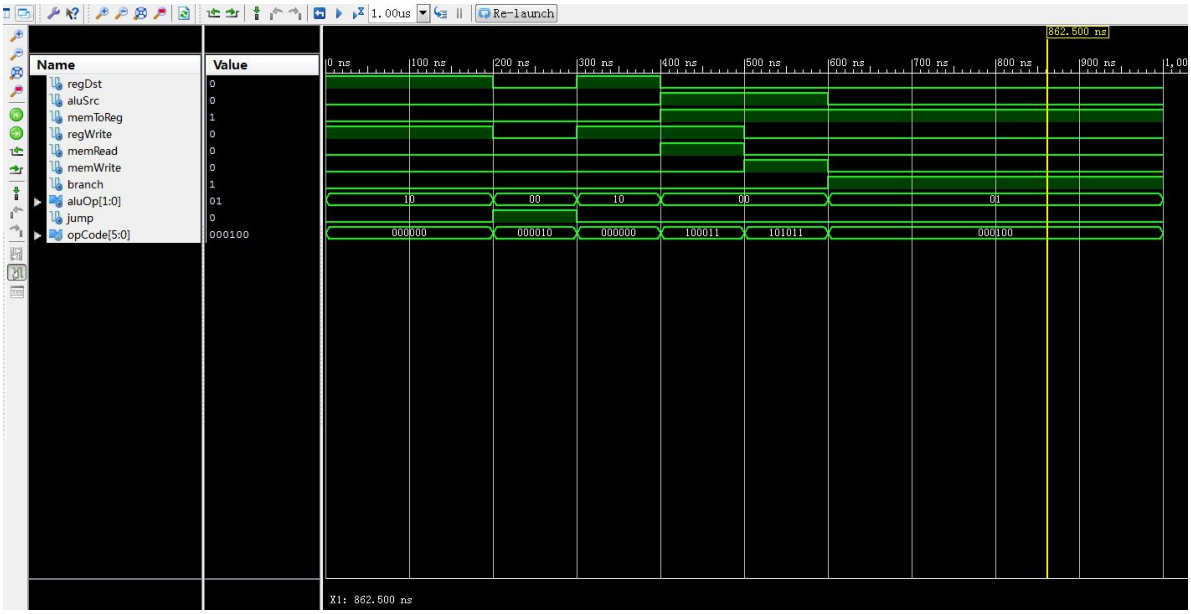
并对实现了的5种opCode进行100ns的测试，观察全部的输出信号是否正确。

```
module test_for_Ctr;
    // Inputs
    reg [5:0] opCode;

    // Outputs
    wire regDst;
    wire aluSrc;
    wire memToReg;
    wire memRead;
    wire memWrite;
    wire regWrite;
```

```
wire branch;
wire jump;
wire [1:0] aluOp;

// Instantiate the Unit Under Test (UUT)
Ctr uut (
.opCode(opCode),
.regDst(regDst),
.aluSrc(aluSrc),
.memToReg(memToReg),
.regWrite(regWrite),
.memRead(memRead),
.memWrite(memWrite),
.branch(branch),
.aluOp(aluOp),
.jump(jump)
);

initial begin
    // Initialize Inputs
    opCode = 0;

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

```
        // Add stimulus here
        #100 opCode = 6'b000010;// jump
        #100 opCode = 6'b000000;// R
        #100 opCode = 6'b100011;// lw
        #100 opCode = 6'b101011;// sw
        #100 opCode = 6'b000100;// beq
    end
endmodule
```

2.1.4 仿真图



仿真显示，信号输出均正确

2.2 AluCtr

2.2.1 实验原理

AluCtr是CPU中负责输出aluCtr，以告诉ALU具体操作的硬件模块。

AluCtr接受一个带宽为2的信号量aluOp，它来自Control unit。

另外，AluCtr还接受一个带宽为6的信号量，它来自于指令的低六位，充当opCode的补充，以满足更丰富的指令要求。

ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

输入输出真值表

2.2.2 实验模块

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

    reg [3:0] aluCtr;
    always @ (aluOp or funct)
    begin
        if(aluOp == 2'b00) aluCtr = 4'b0010;
        else if(aluOp[0] == 1) aluCtr = 4'b0110;
        else
        begin
```

```
            if(funct[3:0] == 4'b0000) aluCtr = 4'b0010;
            if(funct[3:0] == 4'b0010) aluCtr = 4'b0110;
            if(funct[3:0] == 4'b0100) aluCtr = 4'b0000;
            if(funct[3:0] == 4'b0101) aluCtr = 4'b0001;
            if(funct[3:0] == 4'b1010) aluCtr = 4'b0111;
            if(funct[3:0] == 4'b0111) aluCtr = 4'b1100;
        end
    end
endmodule
```

2.2.3 仿真

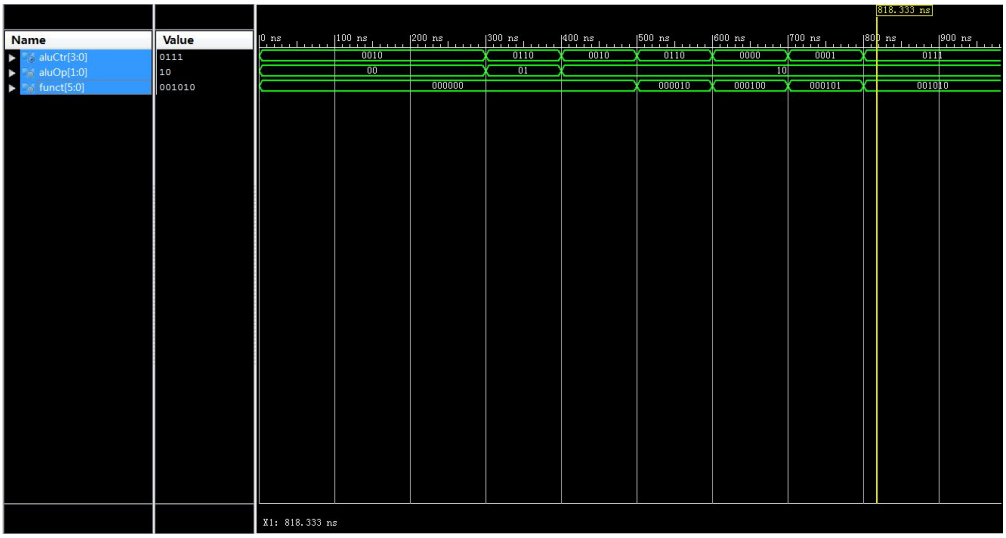
初始化aluOp, funct为0。

等待100ns后，测试每一个在模块中实现了的aluOp, funct 的情况，测试时间为100ns

```
module test_for_aluCtr;
    // Inputs
    reg [1:0] aluOp;
    reg [5:0] funct;

    // Outputs
    wire [3:0] aluCtr;

    // Instantiate the Unit Under Test (UUT)
    AluCtr uut (
    .aluOp(aluOp),
    .funct(funct),
    .aluCtr(aluCtr),
    );

    initial begin
        // Initialize Inputs
        aluOp = 0;
```

```
        funct = 0;

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

        // Add stimulus here
        #100 aluOp, funct = 8'b00000000;
        #100 aluOp, funct = 8'b01000000;
        #100 aluOp, funct = 8'b10000000;
        #100 aluOp, funct = 8'b10000010;
        #100 aluOp, funct = 8'b10000100;
        #100 aluOp, funct = 8'b10000101;
        #100 aluOp, funct = 8'b10001010;
    end
endmodule
```

2.2.4 仿真图



仿真表明，输出的aluCtr正确

2.3 Alu

2.3.1 实验原理

ALU 是CPU中负责计算的硬件单元。有两个带宽为32的输入，他们分别是ALU的两个32位的操作数。然后有一个带宽为32的输出，是ALU的计算结果。

此外ALU还接受一个带宽为4的输入，叫做aluCtr, 是指导ALU进行具体操作的信号。该信号由ALUControl给出。在模块中，使用if语句实现分支。

在本实验中，ALU还输出一个带宽为1的信号量，叫做zero（真实的情况下应该有更多的信号量）。zero 输出1，如果ALU 的计算结果是0；否则，输出0。

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

注：beq 实际是个减法操作

2.3.2 实验模块

```
module Alu(input1,input2,aluCtr,zero,aluRes);
    input [31:0] input1;
    input [31:0] input2;
    input [3:0] aluCtr;
    output zero;
    output [31:0] aluRes;

    reg zero;
    reg [31:0] aluRes;
    integer index;

    always @ (input1 or input2 or aluCtr)
```

```
begin
    if (aluCtr == 4'b0000)// AND
    begin
        for(index = 0; index ¡ 32; index = index + 1)
            aluRes[index] = input1[index] & input2[index];
        if (aluRes == 0) zero = 1;
        else zero = 0;
    end
    else if (aluCtr == 4'b0001)// OR
    begin
        for(index = 0; index ¡ 32; index = index + 1)
            aluRes[index] = input1[index] | input2[index];
        if (aluRes == 0) zero = 1;
        else zero = 0;
    end
    else if (aluCtr == 4'b0010)// add
    begin
        aluRes = input1 + input2;
        if (aluRes == 0) zero = 1;
        else zero = 0;
    end
    else if (aluCtr == 4'b0110)// subtract
    begin
        aluRes = input1 - input2;
        if (aluRes == 0) zero = 1;
        else zero = 0;
    end
    else if (aluCtr == 4'b0111)// slt
    begin
        aluRes = input1 < input2;
        if (aluRes == 0) zero = 1;
        else zero = 0;
    end
```

```
        else if (aluCtr == 4'b1100)// NOR
        begin
            for(index = 0; index ¡ 32; index = index + 1)
            aluRes[index] = ~ (input1[index] | input2[index]);
            if (aluRes == 0) zero = 1;
            else zero = 0;
        end
        else ;
    end
endmodule
```

2.3.3 仿真

初始化input1，input2，aluCtr为0。

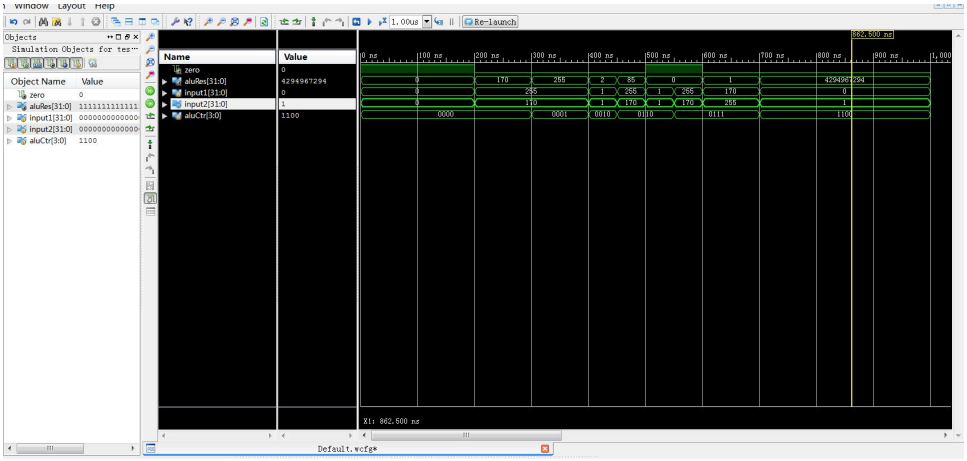
等待100ns后，测试每一个在模块中实现了的aluCtr的情况，测试时间为100ns

操作数是预设的。

```
module test_for_Alu;
    // Inputs
    reg [31:0] input1;
    reg [31:0] input2;
    reg [3:0] aluCtr;

    // Outputs
    wire zero;
    wire [31:0] aluRes;

    // Instantiate the Unit Under Test (UUT)
    Alu uut (
    .input1(input1),
    .input2(input2),
    .aluCtr(aluCtr),
    .zero(zero),
```

```
        .aluRes(aluRes)
        );

        initial begin
            // Initialize Inputs
            input1 = 0;
            input2 = 0;
            aluCtr = 0;

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

            // Add stimulus here
            #100 input1 = 255; input2 = 170; aluCtr = 4'b0000;
            #100 input1 = 255; input2 = 170; aluCtr = 4'b0001;
            #100 input1 = 1; input2 = 1; aluCtr = 4'b0010;
            #50 input1 = 255; input2 = 170; aluCtr = 4'b0110;
            #50 input1 = 1; input2 = 1; aluCtr = 4'b0110;
            #50 input1 = 255; input2 = 170; aluCtr = 4'b0111;
            #50 input1 = 170; input2 = 255; aluCtr = 4'b0111;
            #100 input1 = 0; input2 = 1; aluCtr = 4'b1100;
        end
endmodule
```

2.3.4 仿真图



注意到，仿真结果表明，加、减、按位和、按位与、小于、按位非或的计算结果均正确。

3 实验心得

了解了CPU中指令是如何解码的，以及信号是如何从Control unit中产生的，并学到了各个信号的作用。理解一个硬件电路是如何实现各种不同的功能的，这依赖于mux硬件模块和信号之间的配合，用来选择不同的输入来源。

了解了ALU是如何与AluCtr之间的通信，和ALU实现多种运算的方式和原理。