上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

计算机系统结构实验报告lab3

姓　　名：	张卫鸣
学　　号：	520021911141
专　　业：	计算机科学与技术

2022 年 3 月 22 日

计算机系统结构实验lab3

目录

一、实验概述

（一）实验名称

简单的类MIPS单周期处理器功能部件的设计与实现（一）

（二）实验目的

1. 理解主控制部件或单元、ALU控制器单元、ALU单元的原理
2. 熟悉所需的MIPS指令集
3. 使用Verilog设计与实现主控制器部件（Ctr）
4. 使用Verilog设计与实现ALU控制器部件（ALUCtr）
5. ALU功能部件的实现
6. 使用Vivado进行功能模块的行为仿真

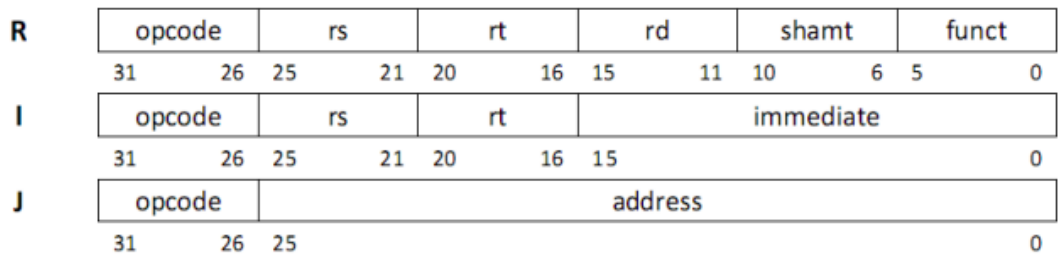
二、实验描述

（一）主控制单元模块Ctr

1. Ctr模块描述

主控制单元（Ctr）的输入为指令的opCode字段，操作码经过Ctr, ALUCtr, Data Memory, Registers, Muxs等部件输出正确的控制信号。

图1是MIPS指令的三种基本格式，我们的输入取指令的最高六位。



本实验中，主控制单元模块可以识别R型指令、lw、sw、beq、jump指令并输出对应的控制信号。主控制单元产生的各种控制信号和opCode段的对应方式如表1所示。

Opcode指令	000000(R)	100011(lw)	101011(sw)	000100(beq)	000010(j)
regDst	1	0	0	0	0
aluSrc	0	1	1	0	0
memToReg	0	1	0	0	0
regWrite	1	1	0	0	0
memRead	0	1	0	0	0
memWrite	0	0	1	0	0
branch	0	0	0	1	0
aluOp	10	00	00	01	00
jump	0	0	0	0	1

表1

2. Ctr模块代码

利用case语句将opCode分成R-type、lw、sw、beq、jump五类，根据真值表解码并输出相关控制信号。以下列出实现代码：

```
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型指令
            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:         //jump
            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
```

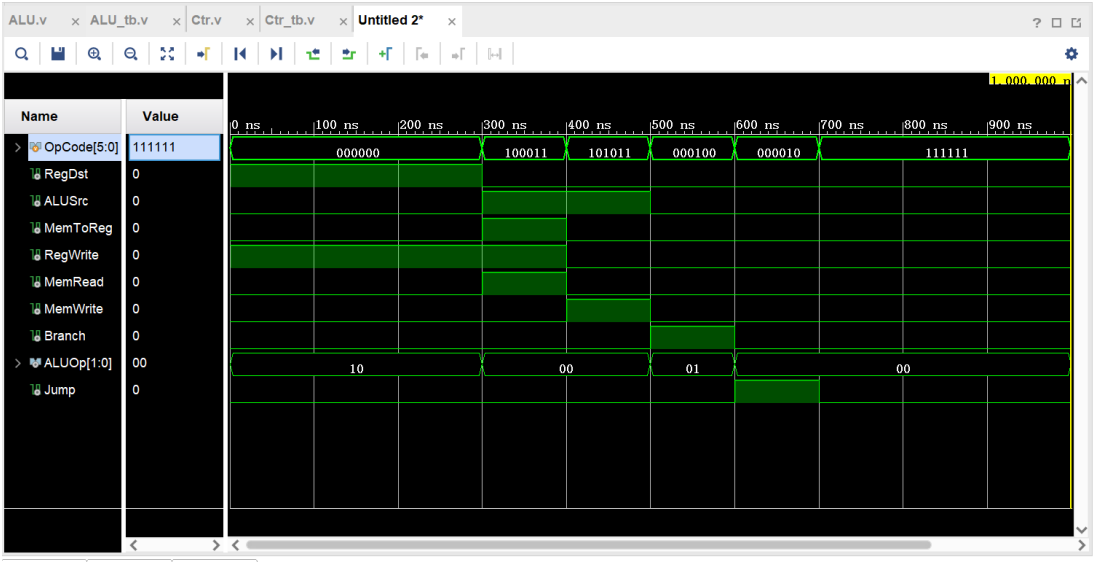
3. 仿真测试代码

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

```
    wire Jump;
    Ctr u(.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'b111111;
    end
endmodule
```

4. 仿真波形

仿真出来的波形如图2所示：



从仿真波形图中可以看出，主控制器正确产生了所有控制信号，对于其他指令采取了置零的操作。

（二）ALU控制单元模块ALUCtr

1. ALUCtr模块描述

ALU的控制单元模块（ALUCtr）根据主控制器（Ctr）输出的ALUOp来判断指令类型,结合ALUOp和Funct输出对应的ALU Control信号。

ALU Control（即aluCtrOut[3:0]）的值与ALU操作的对应关系如下表所示：

ALU Control	Function
0000	AND
0001	OR
0010	ADD
0110	SUB
0111	set on less than
1100	NOR

表2

进一步，我们把Funct，ALUOp与ALU Control的编码关系转化为输入输出真值表，如表3所示：

ALUOp	Funct	Operation
00	XXXXXX	0010
X1	XXXXXX	0110
1X	XX0000	0010
1X	XX0010	0110
1X	XX0100	0000
1X	XX0101	0001
1X	XX1010	0111

表3

2. ALUCtr模块代码

利用casex语句实现输入输出功能，并按照真值表赋值，具体代码如下：

```
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:     //add
                ALUCtrOut = 4'b0010;
            8'b01xxxxxx:    //sub
                ALUCtrOut = 4'b0110;
            8'b1xxx0000:    //add
                ALUCtrOut = 4'b0010;
            8'b1xxx0010:    //sub
                ALUCtrOut = 4'b0110;
            8'b1xxx0100:    //and
                ALUCtrOut = 4'b0000;
            8'b1xxx0101:    //or
                ALUCtrOut = 4'b0001;
            8'b1xxx1010:    //set on less than
                ALUCtrOut = 4'b0111;
        endcase
    end
    assign aluCtrOut=ALUCtrOut;
endmodule
```

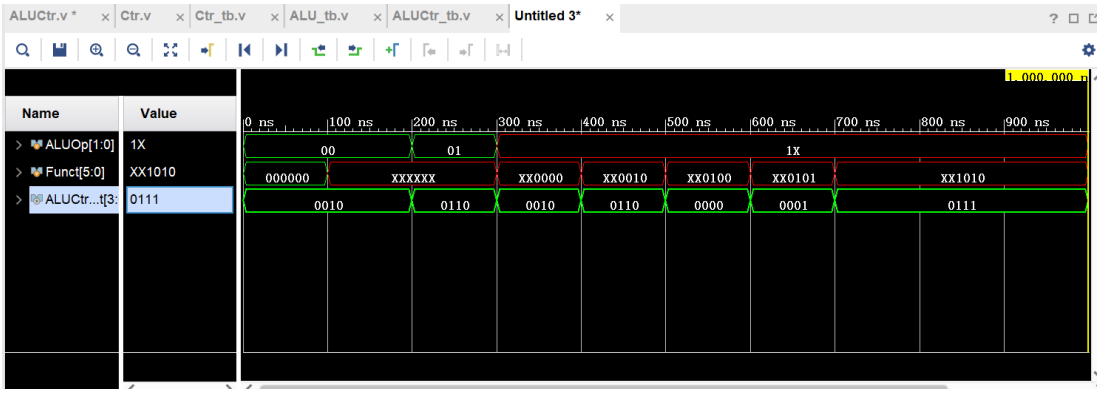
3. ALUCtr仿真测试代码

```
module ALUCtr_tb(

    );
    reg [1:0] ALUOp;
    reg [5:0] Funct;
    wire [3:0] ALUCtrOut;
    ALUCtr u0(.aluOp(ALUOp),
              .funct(Funct),
              .aluCtrOut(ALUCtrOut)
    );
    initial begin
        ALUOp=0;
        Funct=0;
        #100;
        ALUOp=2'b00;
        Funct=6'bxxxxxx;
        #100;
        ALUOp=2'b01;
        Funct=6'bxxxxxx;
        #100;
        ALUOp=2'b1x;
        Funct=6'bxx0000;
        #100;
        ALUOp=2'b1x;
```

```
25          Funct=6'bxx0010;
26          #100;
27          ALUOp=2'b1x;
28          Funct=6'bxx0100;
29          #100;
30          ALUOp=2'b1x;
31          Funct=6'bxx0101;
32          #100;
33          ALUOp=2'b1x;
34          Funct=6'bxx1010;
35          #100;
36      end
37  endmodule
```

4. 仿真波形



在仿真激励文件中，我们对加法指令、减法指令、逻辑与指令、逻辑或指令、小于时置位指令、lw指令、beq指令等都进行了验证。如图所示，能够产生正确的ALUCtrOut控制信号。

（三）ALU模块

1. ALU模块描述

根据ALUCtr的输出信号，ALU对两个输入执行对应的操作。ALURes为计算后输出的结果。若减法操作ALURes的结果0时，Zero信号的输出为1。

2. ALU模块代码

仍然使用case语句，根据aluCtr不同的值对两个操作数选择进行不同的操作。Zero的值用于进行branch条件的判断。

实现代码如下：

```
1   module ALU(
2       input [31:0] input1,
```

```
    input [31:0] input2,
    input [3:0] aluCtr,
    output zero,
    output [31:0] aluRes
    );
    reg Zero;
    reg [31:0] ALURes;
    always @ (input1 or input2 or aluCtr)
    begin
        case(aluCtr)
            4'b0000:
                ALURes=input1&input2;
            4'b0001:
                ALURes=input1|input2;
            4'b0010:
                ALURes=input1+input2;
            4'b0110:
                ALURes=input1-input2;
            4'b0111:
                ALURes=($signed(input1)<$signed(input2));
            4'b1100:
                ALURes=~(input1|input2);
            default:
                ALURes=0;
        endcase
        if(ALURes==0)
            Zero=1;
        else
            Zero=0;
    end
    assign zero=Zero;
    assign aluRes=ALURes;
endmodule
```

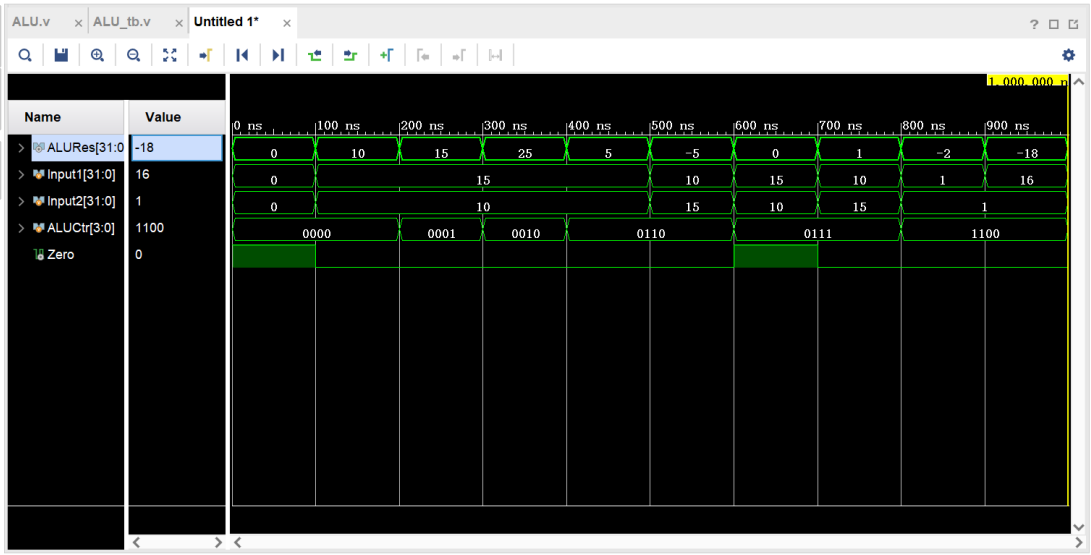
3. ALU仿真测试代码

```
module ALU_tb(

    );
    wire [31:0] ALURes;
    reg [31:0] Input1;
    reg [31:0] Input2;
    reg [3:0] ALUCtr;
    wire Zero;
    ALU uu(.input1(Input1),
          .input2(Input2),
          .aluCtr(ALUCtr),
          .zero(Zero),
```

```
13              .aluRes(ALURes)
14      );
15      initial begin
16          Input1=0;
17          Input2=0;
18          ALUCtr=0;
19          #100;
20          Input1=15;
21          Input2=10;
22          ALUCtr=4'b0000;
23          #100;
24          Input1=15;
25          Input2=10;
26          ALUCtr=4'b0001;
27          #100;
28          Input1=15;
29          Input2=10;
30          ALUCtr=4'b0010;
31          #100;
32          Input1=15;
33          Input2=10;
34          ALUCtr=4'b0110;
35          #100;
36          Input1=10;
37          Input2=15;
38          ALUCtr=4'b0110;
39          #100;
40          Input1=15;
41          Input2=10;
42          ALUCtr=4'b0111;
43          #100;
44          Input1=10;
45          Input2=15;
46          ALUCtr=4'b0111;
47          #100;
48          Input1=1;
49          Input2=1;
50          ALUCtr=4'b1100;
51          #100;
52          Input1=16;
53          Input2=1;
54          ALUCtr=4'b1100;
55      end
56  endmodule
```

4. 仿真波形



我们在激励文件中对上节提到的各种运算进行了测试。从图中可以发现，ALU的运算严格符合我们的编码，运算结果都是正确的。

三、实验心得

本次实验需要我们自己编写的内容较lab1和lab2多了许多，我对于一些基本的verilog语法有了更加具体的认识，在这里本质就是将一些线和自己设置的单元端口连接起来。

在本次实验中，多次用到case语句，这是本次实验的核心操作，实现不同控制的选择。在此过程中对Verilog的分支语句和运算操作都有了更深一步的了解。

我觉得这个实验的设计很巧妙，起了很好的衔接作用，且难度不是很大但能带给我比较多的启示。在这里编码的转换选择很灵活，其中的基本思想值得领会并运用在后期更复杂的指令集中。

四、参考资料

2022计算机系统结构实验指导书lab3