计算机系统结构试验 Lab03

518030910022 杨涵章

目录

1 实验概述：

1.1 实验名称：

简单的类 MIPS 单周期处理器部件实现– 主控制器，ALU，ALU 控制器

1.2 实验目的：

1) 理解 CPU 控制器、ALU 的原理
2) 主控制器 Ctr 的实现
3) 运算单元控制器 ALUCtr 的实现
4）ALU 的实现
5）使用功能仿真

1.3 实验步骤：

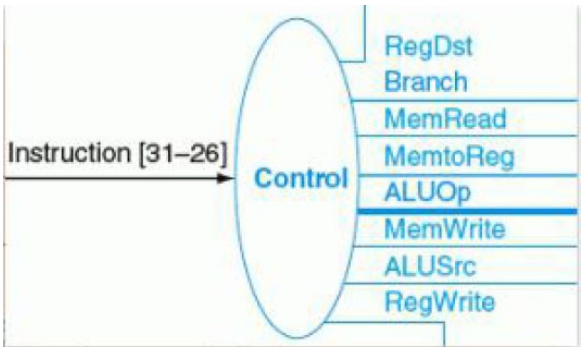
1） 按照实验手册指示分别创建 Ctr、ALUCtr、ALU 模块
2） 分模块对三个部分进行代码实现
3） 按照手册上的波形图进行波形测试，以得到和手册相同的波形图

2 实验描述：

2.1 主控制单元模块 Ctr

2.1.1 模块描述

如图所示，主控制单元接受一个指令的输入，经过主控制单元的处理，根据指令的 Opcode 设置各种输出信号，完成对于处理器其余模块的控制。



2.1.2 模块代码

主控制单元为一个译码器，主要作用为接受 MIPS 指令的[31:26]位字段作为输入，将控制信号输出，给 ALUCtr、data memory、registers 和 ALUSrc 等处理器原件输出控制信号，使得各部分能够正确操作、协调运行。代码主体部分为按照实验手册所给出的框架和控制信号真值表进行补充。将 opcode 的各种可能作为若干的 case，在每个 case 内进行控制信号的赋值。

下图为输入输出真值表，根据真值表设置信号值即可。

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

```
module Ctr(
    input [5:0] opCode,
    output reg regDst,
    output reg aluSrc,
    output reg memToReg,
    output reg regWrite,
    output reg memRead,
    output reg memWrite,
    output reg brance,
    output reg [1:0] aluOp,
    output reg jump
    );
    always @(opCode)
    begin
        case(opCode)
            6'b000000://R type
            begin
                regDst=1;
                aluSrc=0;
                memToReg=0;
                regWrite=1;
                memRead=0;
                memWrite=0;
                brance=0;
                aluOp=2'b10;
                jump=0;
            end
            6'b100011://lw
            begin
                regDst=0;
                aluSrc=1;
                memToReg=1;
                regWrite=1;
                memRead=1;
                memWrite=0;
                brance=0;
                aluOp=2'b00;
                jump=0;
            end
             6'b101011://sw
            begin
                regDst=0;
                aluSrc=1;
                memToReg=0;
                regWrite=0;
                memRead=0;
                memWrite=1;
                brance=0;
                aluOp=2'b00;
                jump=0;
            end
            6'b000100://beq
            begin
                regDst=0;
                aluSrc=0;
                memToReg=0;
                regWrite=0;
                memRead=0;
                memWrite=0;
                brance=1;
                aluOp=2'b01;
                jump=0;
            end
```

```
            6'b000010://jump
            begin
                regDst=0;
                aluSrc=0;
                memToReg=0;
                regWrite=0;
                memRead=0;
                memWrite=0;
                brance=0;
                aluOp=2'b00;
                jump=1;
            end
            default:
            begin
                regDst=0;
                aluSrc=0;
                memToReg=0;
                regWrite=0;
                memRead=0;
                memWrite=0;
                brance=0;
                aluOp=2'b00;
                jump=0;
            end
            endcase
    end
endmodule
```

2.1.3 仿真代码

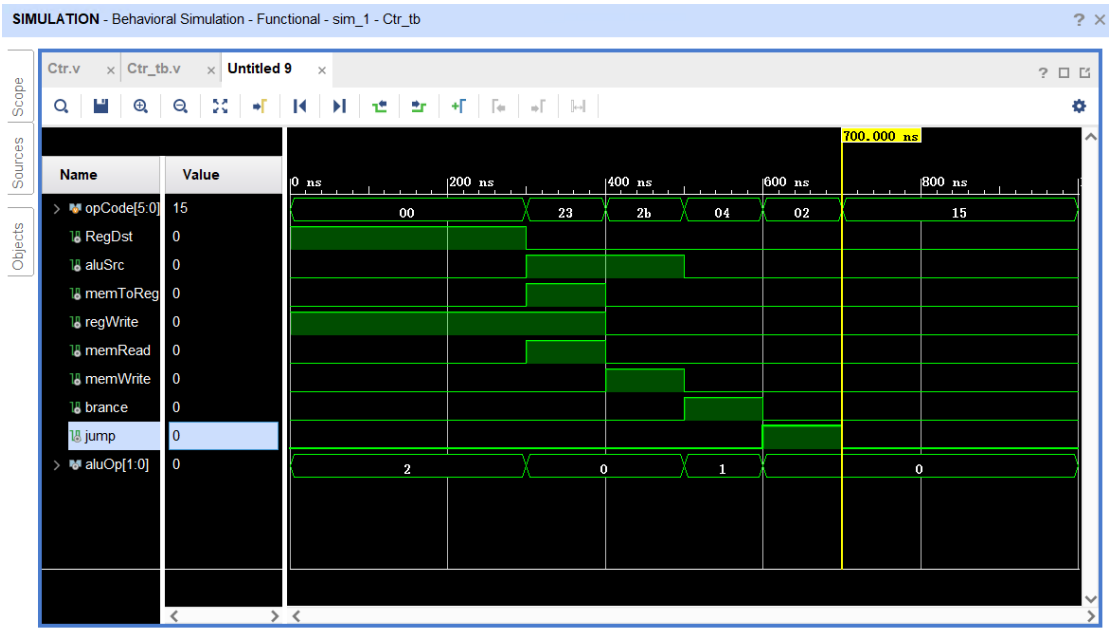
仿真代码的书写较为容易，实例化模块，以 100 为周期输入 opCode 值，观察仿真波形是否符合预期即可。与实验手册对照可知波形基本一致。

```
reg [5:0] opCode;
wire RegDst;
wire aluSrc;
wire memToReg;
wire regWrite;
wire memRead;
wire memWrite;
wire brance;
wire jump;
wire [1:0] aluOp;
Ctr u0(
        .opCode(opCode),
        .regDst(RegDst),
        .aluSrc(aluSrc),
        .memToReg(memToReg),
        .regWrite(regWrite),
        .memRead(memRead),
        .memWrite(memWrite),
        .brance(brance),
        .jump(jump),
         .aluOp(aluOp)
);
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;
   end
```

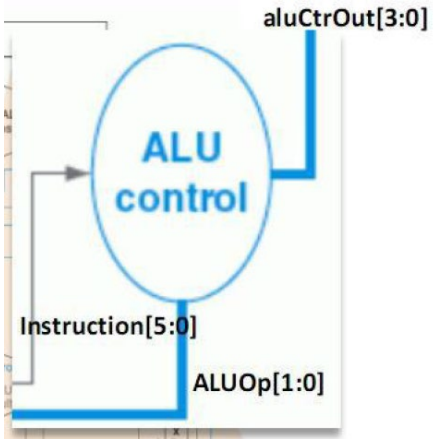
2.1.4 仿真结果



2.2 算术逻辑单元（ALU）控制器模块

2.2.1 模块描述

该模块用于控制算术逻辑单元 ALU 的执行。ALUCtr 根据主控制器输出信号中 ALUOp 判断指令类型，从而向 ALU 输出正确的运算控制信号。其中 R-type 仅按照 ALUOp 无法输出完整信号，需要结合指令[5:0]后 6 位(funct)部分进行判断。译码结果将用于 ALU 的操作。



2.2.2 模块代码

模块实现同样参照实验手册所提供的代码框架，按照 ALUOp 和 funct 拼接之后的值使用 case 语句进行判断。

下图为 ALUOp 与 funct 组成的真值表，根据真值表设置代码即可。

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

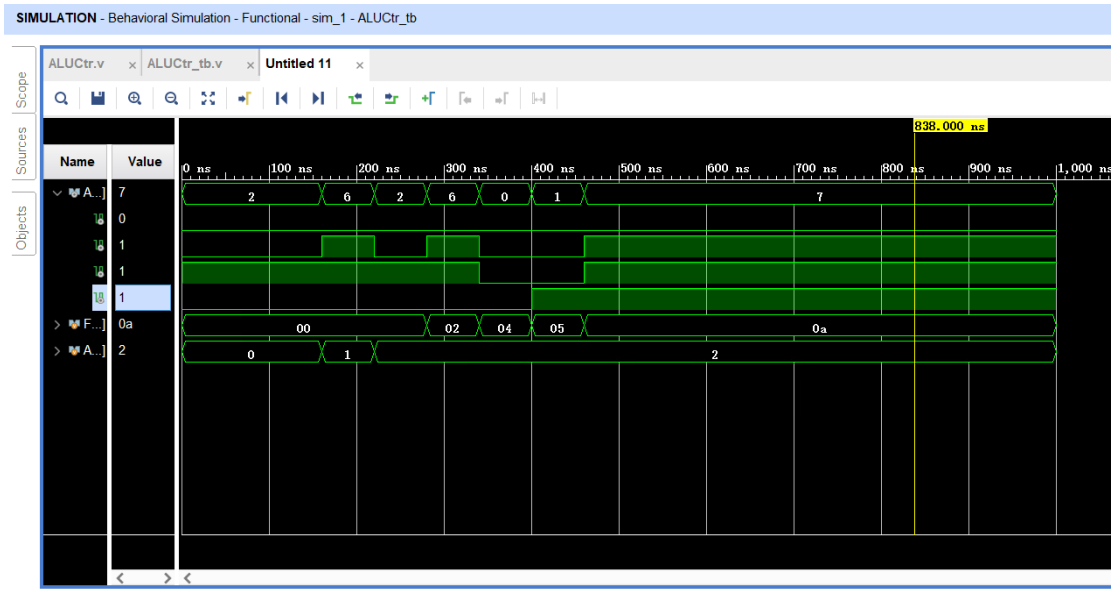
```
23  module ALUCtr(
24      input [1:0] aluop,
25      input [5:0] funct,
26      output reg [3:0] aluctrout
27      );
28      always @ (aluop or funct)
29      begin
30          casex({aluop,funct})
31          8'b00xxxxxx:
32          aluctrout=4'b0010;
33          8'bx1xxxxxx:
34          aluctrout=4'b0110;
35          8'b1xxx0000:
36          aluctrout=4'b0010;
37          8'b1xxx0010:
38          aluctrout=4'b0110;
39          8'b1xxx0100:
40          aluctrout=4'b0000;
41          8'b1xxx0101:
42          aluctrout=4'b0001;
43          8'b1xxx1010:
44          aluctrout=4'b0111;
45
46      endcase
47    end
48  endmodule
```

2.2.3 仿真代码

在实验手册中给出了两种波形，一种对于未定义位直接按照 x 输入，第二种是对于各个位都有具体的赋值，我这里采用第二种方式进行赋值，通过观察手册提供的波形，设置对应的 ALUOp 和 funct 的值。得到的波形与手册中的波形相同。

```
23  module ALUCtr_tb(
24
25      );
26
27
28  wire [3:0] ALUCtrOut;
29  reg [5:0] Funct;
30  reg [1:0] ALUOp;
31  ALUCtr u0(
32      .aluop(ALUOp),
33      .funct(Funct),
34      .aluctrout(ALUCtrOut));
35  initial begin
36      ALUOp=2'b00;
37      Funct=6'b000000;
38      #160
39      ALUOp=2'b01;
40      #60
41      ALUOp=2'b10;
42      #60
43      Funct=6'b000010;
44      #60
45      Funct=6'b00100;
46      #60
47      Funct=6'b000101;
48      #60
49      Funct=6'b001010;
50      end
```

2.2.4 仿真结果



2.3 算术逻辑单元（ALU）

2.3.1 模块描述

该模块为算术逻辑单元，根据控制指令和操作数计算结果。下图为控制信号和 ALU 操作对照表。计算完成之后设置零位和结果即可。

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 模块代码

根据 ALUCtr 的控制信号，对两个操作数进行逻辑或算数运算，并将结果输出到 ALUres 中，同时设置 ZERO 标识符。

模块实现同样基本按照手册提供的代码框架进行填充，这里采用 if 语句进行判断，按照 ALUCtr 的值分别进行处理，在各个部分中记得添加 ZERO 表示符的判断。

```
23  module ALU(
24      input [31:0] input1,
25      input [31:0] input2,
26      input [3:0] aluctr,
27      output reg zero,
28      output reg [31:0] alures
29      );
30
31      always @ (input1 or input2 or aluctr)
32      begin
33          if(aluctr==4'b0010)//add
34              alures=input1+input2;
35          else if(aluctr==4'b0110)//sub
36          begin
37              alures=input1-input2;
38              if(alures==0)
39                  zero=1;
40              else
41                  zero=0;
42           end
43           else if(aluctr==4'b0000)//and
44              begin
45              alures=input1&input2;
46              if(alures==0)
47                  zero=1;
48              else
49                  zero=0;
50              end
51           else if(aluctr==4'b0001)//or
52              begin
53              alures=input1|input2;
54              if(alures==0)
55                  zero=1;
56              else
57                  zero=0;
58                  end
59           else if(aluctr==4'b0111)//set on less than
60           begin
61                if(input1<input2)
62                 alures=1;
63                else
64                alures=0;
65                if(alures==0)
66                 zero=1;
67              else
68                  zero=0;
69           end
70           else if(aluctr==4'b1100)//nor
71           begin
72              alures=~(input1|input2);
73              if(alures==0)
74                  zero=1;
75              else
76                  zero=0;
77                  end
78      end
79  endmodule
```

2.3.3 仿真代码

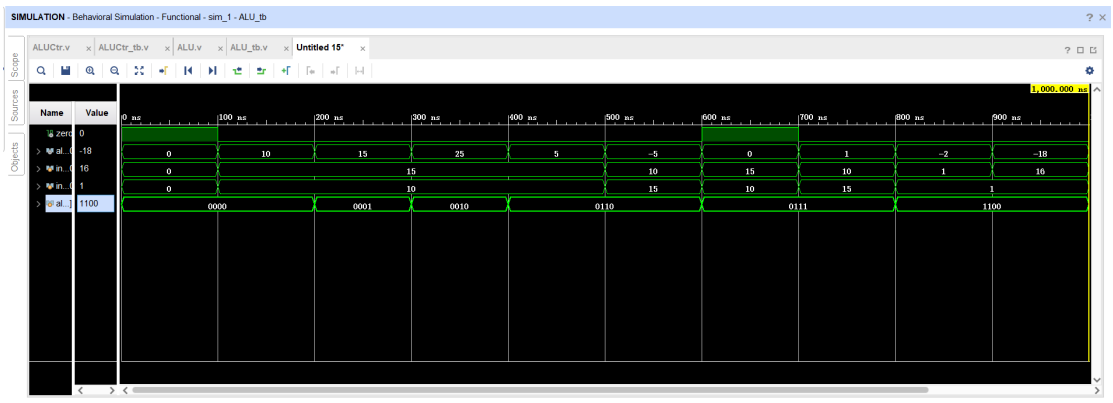
按照手册提供的波形设置输入值和操作符的值即可。最终得到和手册相同的波形。

```
module ALU_tb(

    );
 wire zero;
wire [31:0] alures;
reg [31:0] input1;
reg [31:0] input2;
reg [3:0] aluctr;

ALU u0(
    .input1(input1),
    .input2(input2),
    .aluctr(aluctr),
    .zero(zero),
    .alures(alures)
);
initial begin
    input1=0;
    input2=0;
    aluctr=0;
    #100
    input1=15;
    input2=10;
    #100
    aluctr=4'b0001;
    #100
    aluctr=4'b0010;
    #100
    aluctr=4'b0110;
    #100
    input1=10;
    input2=15;
    #100
    input1=15;
    input2=10;
    aluctr=4'b0111;
    #100
    input1=10;
    input2=15;
    #100
    input1=1;
    input2=1;
    aluctr=4'b1100;
    #100
    input1=16;
end
endmodule
```

2.3.4 仿真结果



3 总结与反思：

这一次实验指导书的指导较少，不像实验一二有着完整的实验流程阐述，所以想要完成这次的实验需要对 Verilog 语言有了一定的了解才能够完成。本次实验依旧给出了各个部分的代码框架，按照框架并对应真值表进行填充即可，总体流程较为清晰。

这次实验我花了较长的时间学习 Verilog 语言，与实验一二笼统的认知不同，这次实验很多细节的实现需要对 Verilog 语言有一定的了解，但好在之前有系统地学习过 C++，两者有一定的相似之处，学习的时间成本不是很高。

这次的实验是自己首次接触到模块化的具体设计和实现，在实验过后对于具体模块的创建已经有了较为深刻的认知，对于后续的实验也有了较好的帮助。