计算机结构实验 lab03

张露伊

2022 年 3 月 26 日

摘要

本实验实现了简单的类 MIPS 处理器的几个重要部件：主控制器 (Ctr)、运算单元控制器 (ALUCtr) 以及算术逻辑运算单元 (ALU)。但只能实现 lw、sw、beq、add、sub、and、or、slt 这几条基本指令。之后通过软件仿真的形式进行实验结果的验证。

目录

1 实验目的

1.1 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）

1.2 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 VerilogHD 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

2 原理分析

Mips 基本格式指令大致如下：

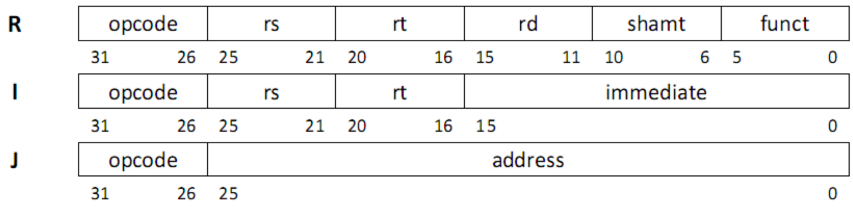

图 1: Mips 基本指令格式

2.1 主控制器 Ctr 的设计

主控制单元(Ctr)的输入为指令的 opCode 字段，操作码经过 Ctr 的译码，给 ALUCtr、Data Memory、Registers、Muxs 等功能单元输出正确的控制信号。

其输出具体包括：

表 1: 主控制器控制信号

信号名称	具体含义
ALUSrc	ALU 的第二个操作数来源。0：使用 Rt 值，1：使用立即数
RegDst	目标寄存器选择。0：写入 Rt，1：写入 Rd
MemToReg	是否写回内存读取内容。
RegWrite	寄存器写使能信号。高电平有效
MemRead	内存读使能。高电平有效
MemWrite	内存写使能。高电平有效
Branch	条件跳转使能信号。高电平有效
AlUOp	ALU 需要执行的操作类型
Jump	无条件跳转使能信号。高电平有效

其中 ALU 是一个包含两位的信号，其取值与含义对应关系如下：

表 2: ALUop 信号含义

ALUOp[1:0]	含义
10	进行 R 型运算，具体运算取决于 funct
00	ALU 执行加法运算
01	ALU 执行减法运算

主控制模块对于 opCode 进行译码操作，其对应真值表如下：

表 3: 主控制模块真值表

opCode[5:0]	000000	100011	101011	000100	000010
指令类型	R:add,sub,and,or,slt	I:lw	I:sw	I:beq	J:j
RegDst	1	0	x	x	0
ALUSrc	0	1	1	0	0
MemtoReg	0	1	x	x	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
Jump	0	0	0	0	1
ALUOp	10	00	00	01	00

根据表 3可以实现 Ctr 的功能，当出现非法指令时，所有控制器的信号清零。

2.2 算术逻辑单元的控制单元（ALUCtr）的设计

ALUCtr 是根据主控制器的 ALUOp 控制信号来判断指令类型，并依据指令的后 6 位（funct）区分 R 型指令。故而它的输入为 funct 和 ALUOp[1:0]，输出为 ALUCtrOut。ALUCtrOut 可以决定 ALU 模块进行的计算操作。

ALUCtr 真值表以及 ALUCtrOut 信号对应运算如下：

表 4: ALUCtr 真值表

指令	ALUOp	funct	ALUCtrOut
lw	00	xxxxxx	0010
sw	00	xxxxxx	0010
beq	01	xxxxxx	0110
add	10	100000	0010
sub	10	100010	0110
and	10	100100	0000
or	10	100101	0001
slt	10	101010	0111

表 5: ALUCtrOut 与 ALU 运算的关系

ALUCtrOut	Function
0000	and
0001	or
0010	add
0110	sub
0111	slt
1100	nor

2.3 算术逻辑单元 ALU 的设计

ALU 根据 ALUCtr 的控制信号（ALUCtrOut）将两个输入的数据执行与之对应的计算，并将结果保存到 ALURes。如果是减法操作，当 $ALURes = 0$ 时，Zero 的输出置为 1。

3 功能实现

3.1 主控制器 Ctr 的实现

主控制器的输出是由 opCode 决定的，我们可以使用 case 语句，根据不同的 opCode 的值对应不同的输出。部分核心代码如下：

```
46    always@(opCode)
47    begin
48        case(opCode)
49        6'b000000:   //R type
50        begin
51            RegDst = 1;
52            ALUSrc = 0;
53            MemToReg = 0;
54            RegWrite = 1;
55            MemRead = 0;
56            MemWrite = 0;
57            Branch = 0;
58            ALUOp = 2'b10;
59            Jump = 0;
60        end
61        6'b100011:   //I lw
62        begin
```

图 2: Ctr.v

```
123       assign regDst = RegDst;
124       assign aluSrc = ALUSrc;
125        assign memToReg = MemToReg;
126        assign regWrite = RegWrite;
127        assign memRead = MemRead;
128        assign memWrite = MemWrite;
129        assign branch = Branch;
130        assign aluOp = ALUOp;
131        assign jump = Jump;
```

图 3: Ctr.v

其中不同的 opCode 值对应不同的输出，RegDst、MemRead 等为控制信号寄存器，与相应的输出信号线相连。

3.2 算术逻辑单元的控制单元 ALUCtr 的实现

ALUCtr 根据 ALUOp 和 func 的值来决定 ALU 应该执行什么操作。并且在 ALUOp 和 funct 中有一些位是我们不关心的位（x），因此我们使用 casex 来代替 case。在 casex 中，我们可以使用 x 来表示我们不关心的位。$\{a, b\}$ 为 Verilog 位拼接运算符。部分核心代码如下：

```
always @ (aluOp or funct)
begin
    casex({aluOp,funct})
    8'b00xxxxxx: ALUCtrOut = 4'b0010;   //add
    8'b01xxxxxx: ALUCtrOut = 4'b0110;   //sub
    8'b10xx0000: ALUCtrOut = 4'b0010;   //add
    8'b10xx0010: ALUCtrOut = 4'b0110;   //sub
    8'b10xx0100: ALUCtrOut = 4'b0000;   //and
    8'b10xx0101: ALUCtrOut = 4'b0001;   //or
    8'b10xx1010: ALUCtrOut = 4'b0111;   //slt
    endcase
end
assign aluCtrOut = ALUCtrOut;
```

图 4: ALUCtr.v

3.3 算术逻辑单元 ALU 的实现

ALU 根据 ALUCtr 的控制信号将两个输入执行与之对应的操作。将结果存入 ALURes 中。如果减法操作后 $ALURes = 0$，则 $Zero = 1$。部分核心代码如下：

```
35      always @ (input1 or input2 or aluCtr)
36      begin
37          case(aluCtr)
38          4'b0000:    //AND
39              ALURes = input1 & input2;
40          4'b0001:    //OR
41              ALURes = input1 | input2;
42          4'b0010:    //ADD
43              ALURes = input1 + input2;
44          4'b0110:    //SUB
45              ALURes = input1 - input2;
46          4'b0111:    //SLT
47              ALURes = ($signed(input1) < $signed(input2));
48          4'b1100:    //nor
49              ALURes = ~(input1 | input2);
50          endcase
51          if(ALURes==0)
52              Zero = 1;
53          else
54              Zero = 0;
55      end
```

图 5: ALU.v

4 结果验证

4.1 主控制器 Ctr 的测试

先后向 Ctr 模块输入 R 型、lw、sw、beq、jump 和未定义指令的 OpCode，具体数据如下表所示：

表 6: Ctr 仿真测试数据

OpCode	指令类型
000000	R 型
100011	lw
101011	sw
000100	beq
000010	jump
010101	default

仿真结果截图如下：

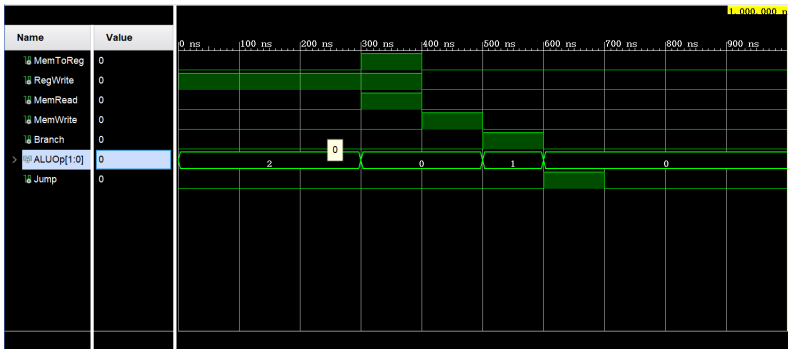

图 6: Ctr 仿真波形图

通过观察波形图可以判断结果符合预测，模块实现正确。

4.2 算术逻辑单元的控制单元 ALUCtr 的测试

通过输入不同的 opCode 和 funct 来对 ALUCtr 模块进行测试，具体数据如下表所示：

表 7: ALUCtr 仿真测试数据

ALUOp	funct	指令类型
00	000000	R 型（0010）
01	000000	sub（0110）
10	000000	add（0010）
10	000010	sub（0110）
10	000100	and（0000）
10	000101	or（0001）
10	001010	slt(0111)

仿真结果截图如下：

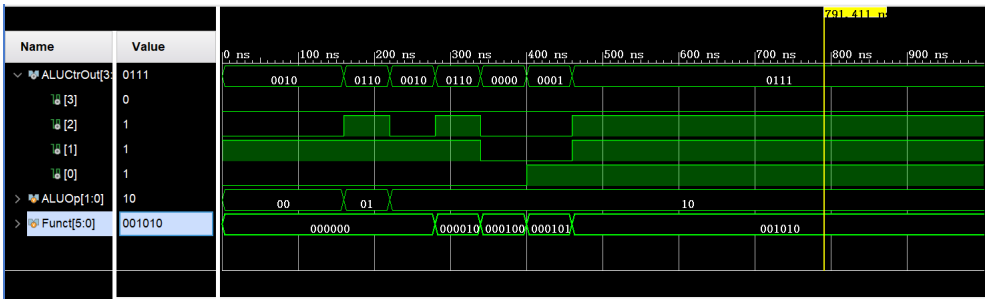

图 7: ALUCtr 仿真波形图

通过观察波形图可以判断结果符合预测，模块实现正确。在实验中，还有可能得到另一种波形图，即结果含有 x，表示数据的不确定性，我们不关心 x 的数字具体是什么，因为是 1 或者 0 并不影响结果。

4.3 算术逻辑单元 ALU 的测试

通过输入不同的 ALUCtr 和两个操作数来测试 ALU 模块，具体数据及仿真结果截图如下：

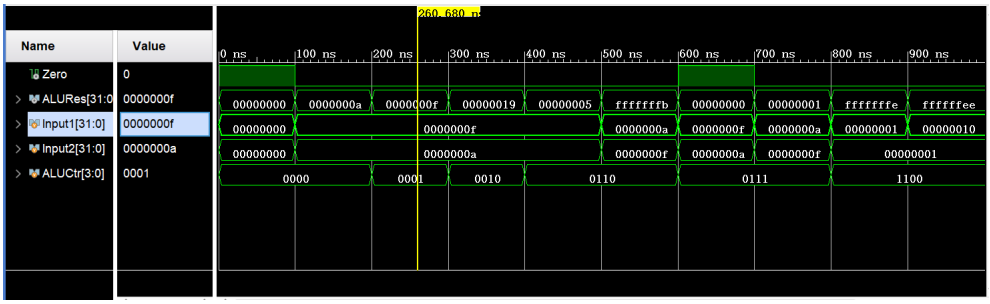

图 8: ALU 仿真波形图

通过观察波形图可以判断结果符合预测，模块实现正确。

表 8: ALU 仿真测试数据

ALUCtr	操作数 1	操作数 2	结果
0000	15	10	15 and 10
0001	15	10	15 or 10
0010	15	10	15 + 10
0110	15	10	15 - 10
0110	10	15	10 - 15
0111	15	10	15 <10
0111	10	15	10 <15
1100	1	1	1 nor 1
1100	16	1	16 nor 1

5 心得体会

这次的三个实验在充分理解了输入输出以及它们之间的相互关系的基础上并不难进行实现，在编写实现代码时只要仿照给出的例子就可以举一反三地完成整个部件的设计。

在写激励文件时我遇到了比较大的困难，因为这次的实验并没有给出激励文件的例子，只能根据前两个实验来仿照书写，在编写时我以为三个模块之间相互联系，不知道如何将几个模块统一在一起，使得一个模块的输出成为另外一个模块的输出。在理解了设为顶层模块的含义之后发现只需要单独定义每个模块的输入即可，使得仿真激励文件的编写难度大大降低。

这次实验的几个部件实现的功能较少，比较单一，无法满足后续实验要求，但是非常适合模块设计的入门，为之后功能的设计拓展打下了一定的基础。