计算机系统结构实验报告　实验 3

简单的类 MIPS 单周期处理器部件实现：控制器与 ALU

2022 年 3 月 22 日

摘要

本实验实现了简单的类 MIPS 处理器的几个重要部件：主控制单元 (Ctr)、ALU 控制单元 (ALUCtr) 以及算术逻辑单元 (ALU)，其作用分别是产生处理器所需要的各种控制信号、产生算术逻辑单元 (ALU) 所需要的控制信号以及根据 ALU 控制单元 (ALUCtr) 发出的信号执行相应的算术逻辑运算输出结果。本实验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

本次实验有如下几个实验目的：

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 HD 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

2.1 主控制单元 (Ctr) 原理分析

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

本实验中，主控制器模块可以识别 R 型指令 (add, sub, and, or, slt)、I 型指令中的 load 指令 (lw, sw, beq)、J 型指令 (j) 并输出对应的控制信号。控制器产生的控制信号如表1所示。

信号	具体说明
ALUOp	2 位信号，发送给 ALU 控制单元 (ALUCtr) 用来进一步解析运算类型的控制信号
ALUSrc	ALU 第二个操作数来源选择信号；低电平：选择 rt 寄存器值，高电平：选择立即数
Branch	条件跳转信号，高电平说明当前指令是条件跳转指令
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
MemToReg	写寄存器的数据来源选择信号；低电平：选择 ALU 运算结果，高电平：选择内存读取数据
MemRead	内存读使能信号，高电平有效
MemWrite	内存写使能信号，高电平有效
RegDst	目标寄存器的选择信号；低电平：写入 rt 寄存器；高电平：写入 rd 寄存器
RegWrite	寄存器写使能信号，高电平说明当前指令需要进行寄存器写入

表 1: 主控制单元 (Ctr) 产生的控制信号

对于 ALUOp 信号需要进行一些额外的说明，所代表的具体含义以及解析方式如表 2 所示。

ALUOp 的信号内容	指令	具体说明
00	lw, sw	ALU 执行加法运算
01	beq	ALU 执行减法运算
1x	R-format	ALUCtr 结合指令 Funct 段决定最终操作

表 2: ALUOp 信号的具体含义以及解析方式

从表 2 中我们可以发现，ALUOp 实际上相当于在主控制单元 (Ctr) 中预先得出的部分非 R 型指令（如 beq 指令、j 指令、lw 指令以及 sw 指令等）的 ALU 控制信号；当然该控制信号并不能直接

送入 ALU，还需要经过 ALU 控制单元 (ALUCtr) 进行进一步的处理，得到最终的运算单元控制信号 ALUCtrOut，之后才能送入 ALU 进行对其的控制。

主控制单元 (Ctr) 产生的各种控制信号与指令 OpCode 段的对应方式如表3所示。

OpCode 指令	000000 R-format	100011 lw	101011 sw	000100 beq	000010 j
ALUSrc	0	1	1	0	0
ALUOp	10	00	00	01	00
Branch	0	0	0	1	0
Jump	0	0	0	0	1
MemRead	0	1	0	0	0
MemToReg	0	1	x	x	0
MemWrite	0	0	1	0	0
RegDst	1	0	x	x	0
RegWrite	1	1	0	0	0

表 3: 各指令对应的主控制单元 (Ctr) 控制信号

2.2 ALU 控制单元 (ALUCtr) 原理分析

算数逻辑单元 ALU 的控制单元（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 位区分 R 型指令。综合这两种输入，以控制 ALU 做正确操作。ALU 控制单元 (ALUCtr) 的解析方式如表 4 所示。

指令	ALUOp	Funct	ALUCtrOut	ALU 操作类型
lw	00	xxxxxx	0010	加法运算
sw	00	xxxxxx	0010	加法运算
beq	01	xxxxxx	0110	减法运算
j	00	xxxxxx	0010	加法运算
add	1x	xx0000	0010	加法运算
sub	1x	xx0010	0110	减法运算
and	1x	xx0100	0000	逻辑与运算
or	1x	xx0101	0001	逻辑或运算
slt	1x	xx1010	0111	小于时置位运算

表 4: ALU 控制单元 (ALUCtr) 的解析方式

从表 4 中可以看出，运算单元控制信号 ALUCtrOut 与 ALU 执行的操作类型有一一对应的关系，我们将在第 2.3 节给出具体的对应方式。

2.3 算术逻辑单元 (ALU) 原理分析

算术逻辑单元 ALU 根据 ALUCtr 的控制信号将两个输入执行与之对应的操作。ALURes 为输出结果。若减法操作 ALURes 的结果为 0 时，则 Zero 输出置为 1。该信号用于与 branch 指令结合判断

是否满足转移条件。ALUCtrOut 与 ALU 操作的对应关系如表 5 所示。

ALUCtrOut	ALU 操作
0000	逻辑与 and
0001	逻辑或 or
0010	加法 add
0110	减法 sub
0111	小于时置位 slt
1100	逻辑或非 nor

表 5: ALUCtrOut 与 ALU 操作的对应关系

3 功能实现

3.1 主控制单元 (Ctr) 功能实现

在第 2.1 节中我们详细介绍了主控制单元 (Ctr) 的设计，我们只需要按照第 2.1 节中的表 3 进行相应信号的实现即可。注意，这里的 `default` 选项表示出现不支持指令时，我们将所有信号置零，当作一条空指令 (nop) 进行处理，对数据不产生影响。

完整代码详见附录 A.1。

```
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;
```

```
23              MemWrite = 0;
24              Branch = 0;
25              ALUOp = 2'b00;
26              Jump = 0;
27          end
28          6'b101011:    // sw
29          begin
30              RegDst = 0;
31              ALUSrc = 1;
32              MemToReg = 0;
33              RegWrite = 0;
34              MemRead = 0;
35              MemWrite = 1;
36              Branch = 0;
37              ALUOp = 2'b00;
38              Jump = 0;
39          end
40          6'b000100:    // beq
41          begin
42              RegDst = 0;
43              ALUSrc = 0;
44              MemToReg = 0;
45              RegWrite = 0;
46              MemRead = 0;
47              MemWrite = 0;
48              Branch = 1;
49              ALUOp = 2'b01;
50              Jump = 0;
51          end
52          6'b000010:    // jump
53          begin
54              RegDst = 0;
55              ALUSrc = 0;
56              MemToReg = 0;
57              RegWrite = 0;
58              MemRead = 0;
59              MemWrite = 0;
60              Branch = 0;
61              ALUOp = 2'b00;
62              Jump = 1;
63          end
```

```
64            default:      // default
65            begin
66                RegDst = 0;
67                ALUSrc = 0;
68                MemToReg = 0;
69                RegWrite = 0;
70                MemRead = 0;
71                MemWrite = 0;
72                Branch = 0;
73                ALUOp = 2'b00;
74                Jump = 0;
75            end
76        endcase
77    end
```

3.2 ALU 控制单元 (ALUCtr) 功能实现

在第 2.2 节中我们详细介绍了 ALU 控制单元 (ALUCtr) 的设计，我们只需要按照第 2.2 节中的表 4 进行相应信号的实现即可。

完整代码详见附录 A.2。

```
1  always @ (aluOp or funct)
2     begin
3         casex ({aluOp, funct})
4             8'b00xxxxxx:  //lw, sw
5                 ALUCtrOut = 4'b0010;
6             8'b01xxxxxx:  //beq
7                 ALUCtrOut = 4'b0110;
8             8'b1xxx0000:  //add
9                 ALUCtrOut = 4'b0010;
10            8'b1xxx0010:  //sub
11                ALUCtrOut = 4'b0110;
12            8'b1xxx0100:  //and
13                ALUCtrOut = 4'b0000;
14            8'b1xxx0101:  //or
15                ALUCtrOut = 4'b0001;
16            8'b1xxx1010:  //slt
17                ALUCtrOut = 4'b0111;
18        endcase
19    end
```

3.3 算术逻辑单元 (ALU) 功能实现

在第 2.3 节中我们详细介绍了 ALU 控制单元 (ALUCtr) 的设计，我们只需要按照第 2.3 节中的表 5 进行相应信号的实现即可。

完整代码详见附录 A.3。

```
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:  // slt
                ALURes = ($signed(inputA) < $signed(inputB));
            4'b1100:  // nor
                ALURes = ~(inputA | inputB);
            default:
                ALURes = 0;
        endcase
        if (ALURes == 0)
            Zero = 1;
        else
            Zero = 0;
    end
```

4 结果验证

4.1 主控制单元 (Ctr) 结果验证

我们使用 Verilog 编写激励文件，采用软件仿真的形式对主控制单元 (Ctr) 模块进行测试（代码实现参见附录 A.1）。我们在激励文件中对于 R 型指令、load 指令 (lw)、store 指令 (sw)、branch 指令 (beq)、jump 指令 (j) 以及其他不支持的指令都进行了测试，测试结果如图 1 所示。波形图符合理论预期。

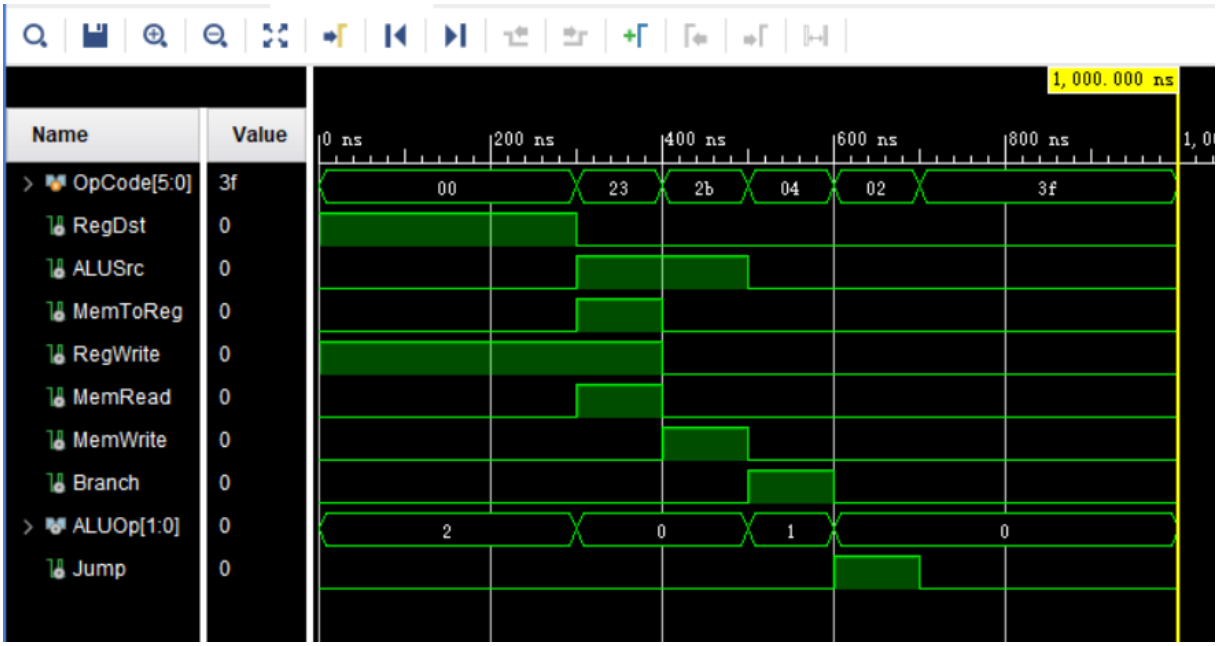

图 1: 对于主控制单元 (Ctr) 的测试结果

4.2 ALU 控制单元 (ALUCtr) 结果验证

我们使用 Verilog 编写激励文件，采用软件仿真的形式对 ALU 控制单元 (ALUCtr) 模块进行测试（代码实现参见附录 A.2）。我们在激励文件中对于加法指令 (add)、减法指令 (sub)、逻辑与指令 (and)、逻辑或指令 (or)、小于时置位指令 (slt) 以及 ALUOp 为 00 或 01 的指令（如 load 指令 (lw)、branch 指令 (beq) 等）都进行了测试，测试结果如图 2 所示。波形图符合理论预期。

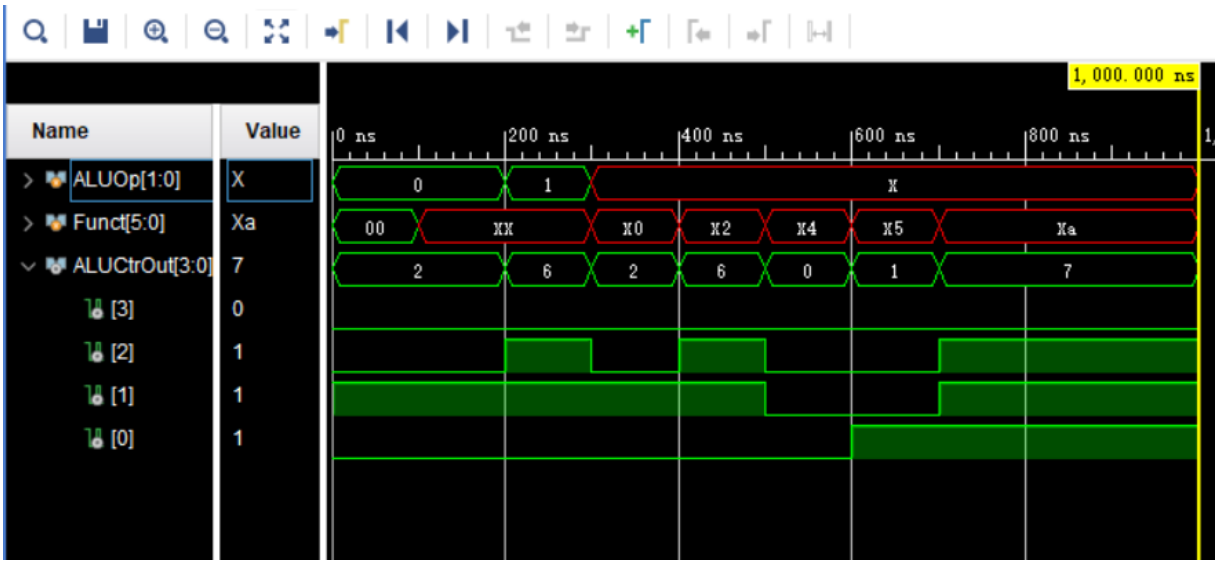

图 2: 对于 ALU 控制单元 (ALUCtr) 的测试结果

4.3 算术逻辑单元 (ALU) 结果验证

我们使用 Verilog 编写激励文件，采用软件仿真的形式对算术逻辑单元 (ALU) 模块进行测试（代码实现参见附录 A.3）。我们在激励文件中对于加法、减法、逻辑与、逻辑或、小于时置位、逻辑或非等算数逻辑运算都进行了测试，测试结果如图 3 所示。波形图符合理论预期。

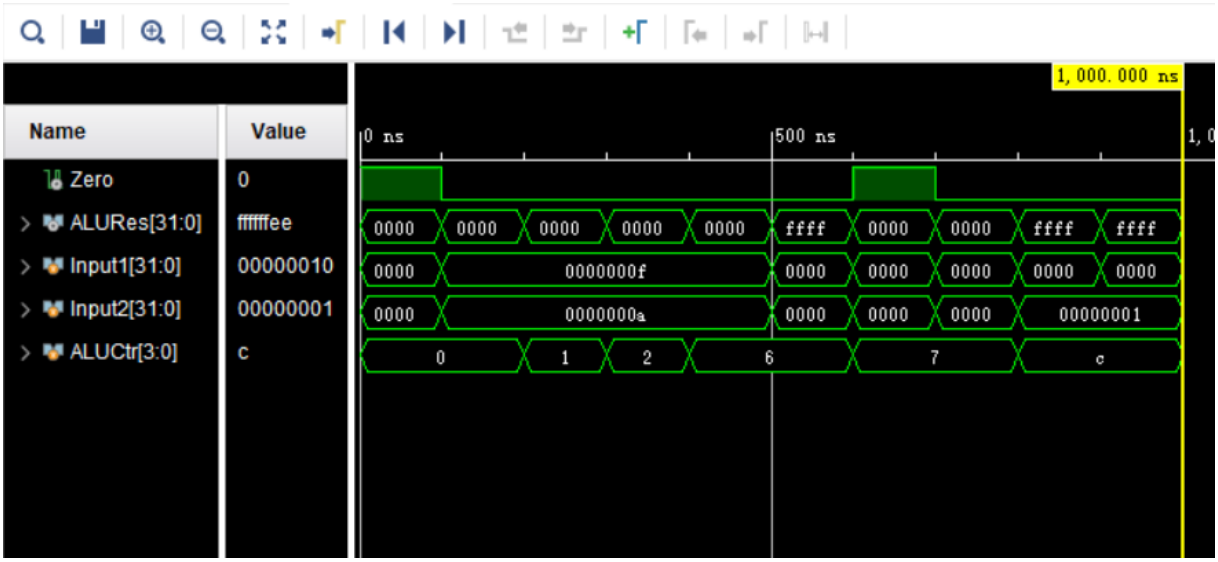

图 3: 对于算术逻辑单元 (ALU) 的测试结果

5 总结与反思

本实验设计并实现了类 MIPS 处理器的三个重要组成部件：主控制单元 (Ctr)、ALU 控制单元 (ALUCtr) 以及算术逻辑单元 (ALU)，并且通过软件仿真模拟的方法验证了它们的正确性。

同时，本次实验的实验指导比 1、2 更加简略，这需要我们进行独立思考，在真值表复现的过程中需要用到 case 和 casex 语句，需要处理的条件分支比较多，需要我们十足的耐心。在本次实验中，我的仿真波形与理论相符得很好，也激发了我对 MIPS 处理器更多的兴趣。

附录 A　设计文件完整代码实现

A.1　主控制单元 (Ctr) 的代码实现

参见代码文件 `Ctr.v` 及 `Ctr_tb.v`。

A.2　ALU 控制单元 (ALUCtr) 的代码实现

参见代码文件 `ALUCtr.v` 及 `ALUCtr_tb.v`。

A.3　算术逻辑单元 (ALU) 的代码实现

参见代码文件 `ALU.v` 及 `ALU_tb.v`。