上海交通大學

《计算机组成与系统结构》实验 3 报告

学生姓名　　：梁展诚 Zhin Sheng Leong

学生学院（系）　：船舶海洋与建筑工程学院（土木工程）

学生学号　　：517010990022

实验名称　　：简单的类 MIPS 单周期处理器部件实现——控制器，ALU

目录

简单的类 *MIPS* 单周期处理器部件实现 ——控制器 ，*ALU*

1. 实验目的

- 理解 *CPU* 控制器， *ALU* 的原理
- 实现主控制器 *Ctr*
- 实现运算单元控制器 *ALUCtr*
- 实现 *ALU*
- 使用 *Verilog HDL* 进行逻辑设计
- 使用功能仿真

2. 实验原理

MIPS 指令的长度为 32 位，所有指令均可分为 *R* 型、*I* 型、*J* 型三种类型，其格式如下图所示：

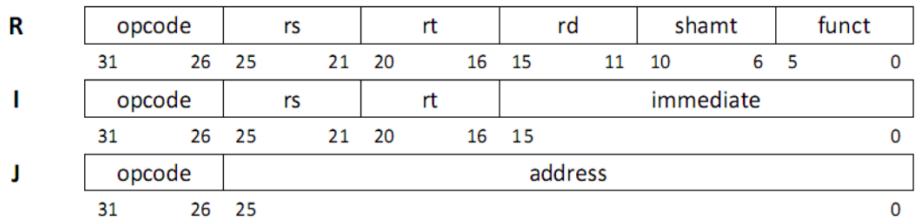

图1：*MIPS* 基本指令格式

2.1 主控制单元 Ctr 的设计

一个简单的 *MIPS* 处理器主要包括主控制单元 *Ctr* 和控制单元模块 *ALUCtr* 的实现。主控制单元的输入指令为指令的 *OpCode* 字段，操作码经过 *Ctr* 的译码后，将对 *ALUCtr*，*Data Memory*，*Registers* 等部件输出正确的控制信号。本次实验需要实验的控制信号如表 1 所示：

信号	具体含义
ALUSrc	*ALU* 第 2 操作数的选择
RegWrite	寄存器的写有效信号
RegDst	目标寄存器的选择

$MemRead$	内存的读有效信号
$MemWrite$	内存的写有效信号
$MemToReg$	写回寄存器的有效信号
$Branch$	条件跳转的有效信号
$Jump$	无条件跳转的有效信号
$ALUOp$	ALU 需要执行的操作类型

表 1：主控制器的控制信号

$ALUOp$ [1:0]	具体含义
00	ALU 执行加法运算
01	ALU 执行减法运算
10	R 型指令，具体运算取决于 $Funct\ field$

表 2：$ALUOp$ 的信号类型及其含义

$OpCode$ [5:0]	000000	100011	101011	000100	000010
指令类型	R 型	I 型	I 型	I 型	J 型
$ALUSrc$	0	1	1	0	0
$RegWrite$	1	1	0	0	0
$RegDst$	1	0	X	X	0
$MemRead$	0	1	0	0	0
$MemWrite$	0	0	1	0	0
$MemToReg$	0	1	X	X	0
$Branch$	0	0	0	1	0
$Jump$	0	0	0	0	1
$ALUOp$ [1:0]	10	00	00	01	00

表 3：主控制模块真值表

根据表 3 实现主控制器的功能。任何不在表 3 的指令可认为是非法指令，在这里不考虑异常处理，因此在遇到非法指令时将所有控制信号置为 0，以确保寄存器和内存中的数据不受影响。

2.2 运算控制单元 $ALUCtr$ 的设计

指令	$ALUop[1:0]$	$Funct\ field\ [5:0]$	$ALUCtr[3:0]$
lw	00	$XXXXXX$	0010
sw	00	$XXXXXX$	0010
beq	01	$XXXXXX$	0110
ADD	10	100000	0010
$SUBTRACT$	10	100010	0110
AND	10	100100	0000
OR	10	100101	0001
$SET\ ON\ LESS\ THAN$	10	101010	0111

表 4：运算控制单元的真值表

根据主控制器的 $ALUOp$ 来判断指令的类型及其相应的 $ALUCtr$ 控制信号。对于 R 型指令，还需要根据指令的最低 6 位 $[5:0]\ Funct\ field$，才可以进一步决定具体的运送种类，并判断对应的 $ALUCtr$ 控制信号。

2.3 算术逻辑单元 ALU 的设计

算数逻辑单元作为一个组合逻辑模块，根据 $ALUCtr$，对 2 个源操作数进行相应的操作运算，输出运算结果。该模块还需要 1 个 $Zero$ 信号，当运算结果为 0 时输出高电位，若运算结果为 1 时则输出低电位，具体关系参见表 5：

$ALUCtr[3:0]$	具体运算或操作
0000	AND 与运算
0001	OR 或运算
0010	ADD加法运算
0110	$SUBTRACT$ 减法运算
0111	$SET\ ON\ LESS\ THAN$ 小于时置位
1100	NOR 与或运算

表 5：$ALUCtr$ 的信号类型及其含义

3. 实验设计

3.1 主控制单元 *Ctr* 的实现

主控制器根据输入信号 *OpCode* 决定输出信号的值，可通过 *Verilog* 中的 *case* 语句实现其功能，部分代码如下：

```
module ctr(
    input [5:0] OpCode,
    output RegDst,
    output ALUSrc,
    output RegWrite,
    output MemToReg,
    output MemRead,
    output MemWrite,
    output Branch,
    output [1:0] ALUOp,
    output Jump);

    reg regdst;
    reg alusrc;
    reg memtoreg;
    reg regwrite;
    reg memwrite;
    reg memread;
    reg branch;
    reg [1:0] aluop;
    reg jump;

    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

```

后续代码与上述相似，在此省略；对于非法指令，通过 *default* 模块定义，将所有控制信号的输出值置为 0

3.2 运算控制单元 *ALUCtr* 的实现

运算控制单元与主控制单元的设计相似，且其输入信号中 *Funct field* 存在可不关心的位，因此通过 *Verilog* 中的 *casex* 语句实现其功能，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: aluctrout = 4'b0010;
            8'b01xxxxxx: aluctrout = 4'b0110;
            8'b1xxx0000: aluctrout = 4'b0010;
            8'b1xxx0010: aluctrout = 4'b0110;
            8'b1xxx0100: aluctrout = 4'b0000;
            8'b1xxx0101: aluctrout = 4'b0001;
            8'b1xxx1010: aluctrout = 4'b0111;
            default:     aluctrout = 4'b1111;
        endcase
    end

    assign ALUCtrOut = aluctrout;

endmodule
```

3.3 算术逻辑单元 *ALU* 的实现

算数逻辑单元根据运算控制单元 *ALUCtr* 的输出信号，对 2 个源操作数进行相应的操作运算，产生 *ALURes* 以及 *Zero* 标志信号的输出信号。通过 *Verilog* 中的 *case* 语句实现其功能，代码如下：

```
module ALURes( input [31:0] input1, input [31:0] input2, input [3:0] ALUCtr, output
 Zero, output [31:0] ALURes);

    reg [31:0] alures;
    reg zero;

    always @(input1 or input2 or ALUCtr)
    begin
        case (ALUCtr)
        4'b0010: alures = input1 + input2;
        4'b0110: alures = input1 - input2;
        4'b0000: alures = input1 & input2;
        4'b0001: alures = input1 | input2;
        4'b0111:
        begin
            if (input1 < input2)
                alures = 32'b00000000000000000000000000000001;
            else
                alures = 32'b00000000000000000000000000000000;
        end
        endcase
    end
    always @ (*) zero = ( ! alures )? 1:0;
    assign ALURes = alures;
    assign Zero = zero;

endmodule
```

4. 结果仿真

对于上述的 3 个单元，采用软件仿真的方式进行校核测试。通过 Verilog 编写相应的激励文件，进行仿真，观察其结果，与预期的结果进行对比。

4.1 主控制单元 Ctr 的仿真

分别对 *Ctr* 模块的 *OpCode* 输入 *R* 型指令、*lw* 指令、*sw* 指令、*beq* 指令以及 *Jump* 指令，得出如下的仿真波形图：

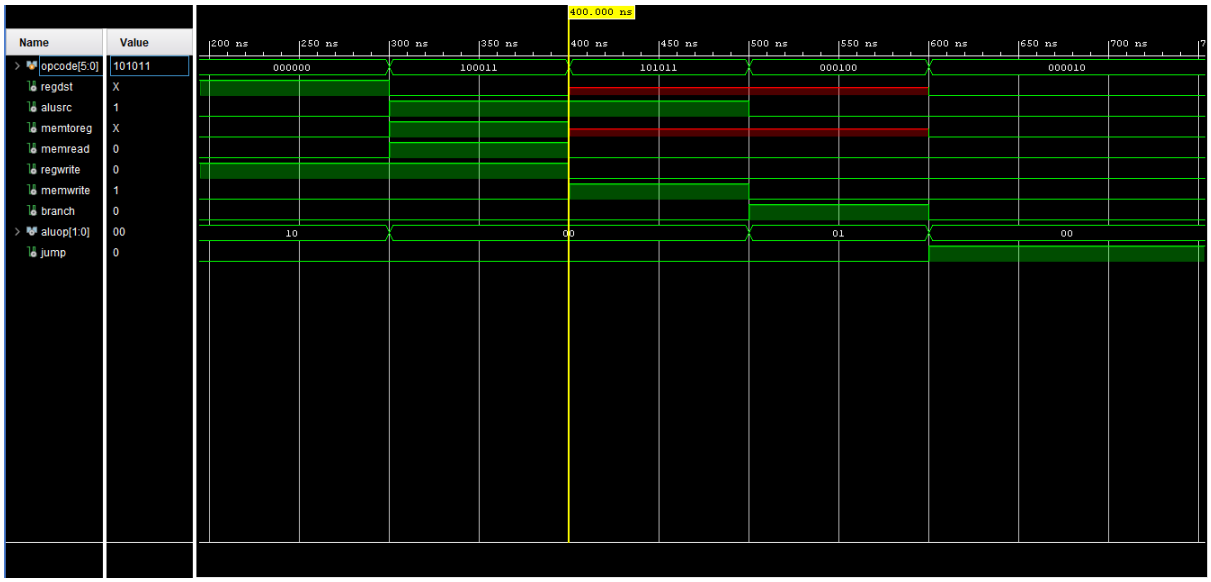

图 2：主控制单元 *Ctr* 的仿真结果

红色的输出为“*x*”值，表示输出信号为不确定。由图 2 可看出，所设计的模块对所有指令所输出的结果均与预期的一致，表示该模块的实现正确。

4.2 运算控制单元 *ALUCtr* 的仿真

分别对 *ALUCtr* 模块的 *ALUOp* 以及 *Funct* 输入 *lw* 指令（与*sw* 指令一致）、*beq* 指令、*R* 型 *ADD* 指令、*R* 型 *SUBTRACT* 指令、*R* 型 *AND* 指令、*R* 型 *OR* 指令以及 *R* 型 *SET ON LESS THAN* 指令，得出如下的仿真波形图：

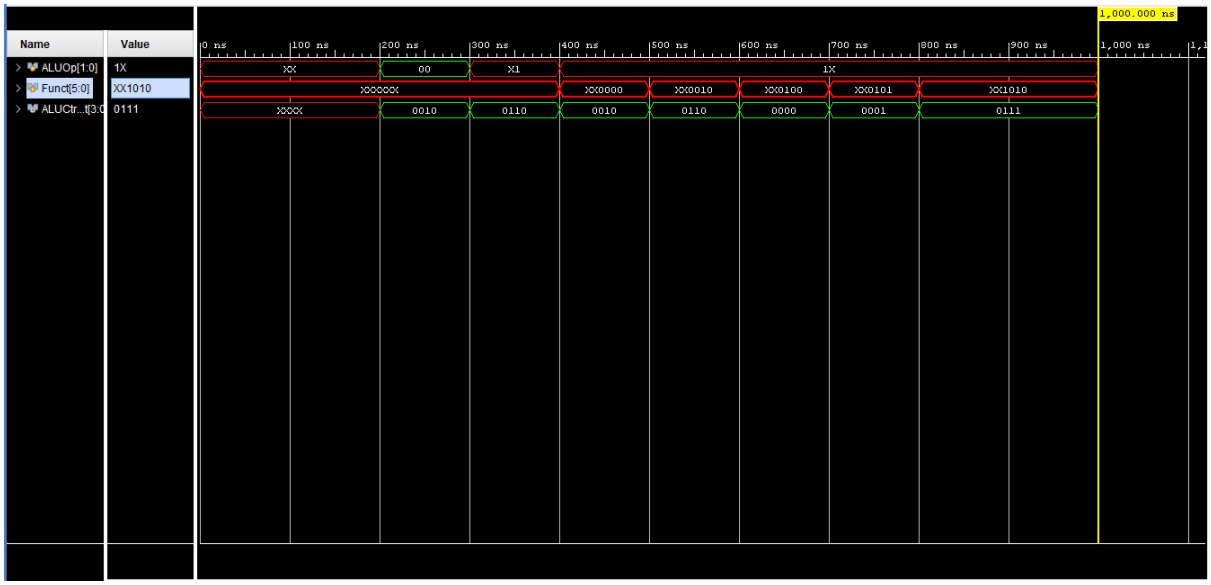

图 3：运算制单元 *ALUCtr* 的仿真结果

由图 3 可看出，所设计的模块对所有指令所输出的结果均与预期的一致，表示该模块的实现正确。

4.3 算术逻辑单元 *ALU* 的仿真

分别对 *ALU* 进行表 6 的测试，得出如下的仿真波形图：

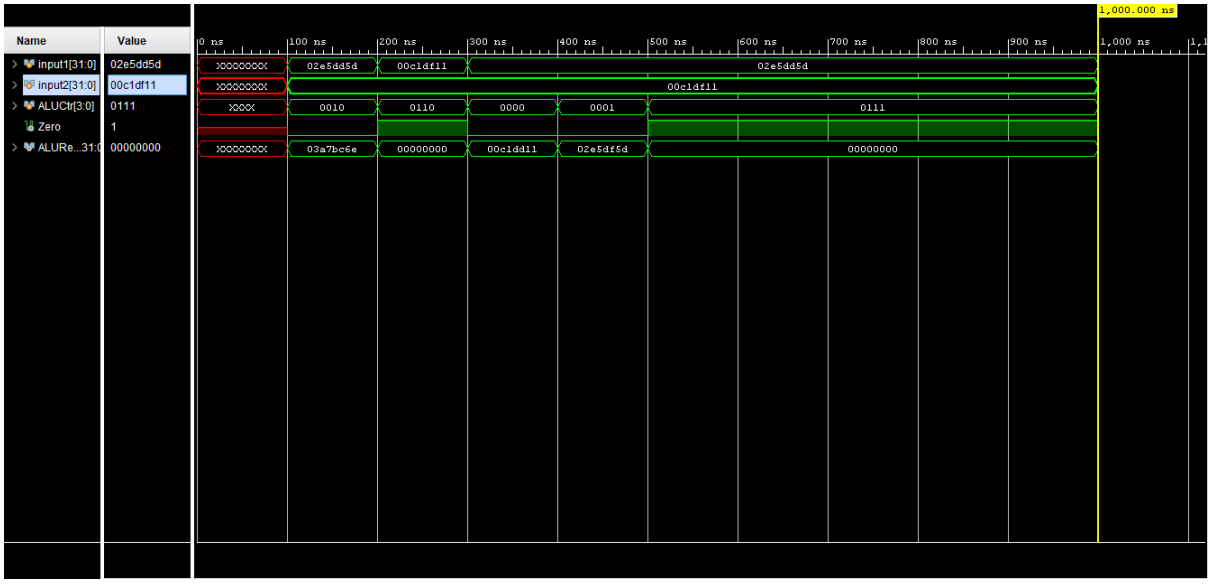

图 4：算术逻辑单元 *ALU* 的仿真结果

运算	操作数 1	操作数 2
ADD	$02e5dd5d$	$00c1df11$
$SUBTRACT$	$00c1df11$	$00c1df11$
AND	$02e5dd5d$	$00c1df11$
OR	$02e5dd5d$	$00c1df11$
$SET\ ON\ LESS\ THAN$	$02e5dd5d$	$00c1df11$

表 6：算术逻辑单元 ALU 的测试项目

由图 4 可看出，所设计的模块对所有指令所输出的结果均与预期的一致，表示该模块的实现正确。

5. 实验总结

本次实验主要着重于 $MIPS$ 处理器中的核心运算单元和控制单元。由于大部分模块只实现了部分的功能，且只需参照已提供的真值表完成编码即可实现，加上实验指导书提供了非常详细的指导，因此相对而言还是比较简单的。本次实验除了加深我对 $Verilog$ 语言的掌握、开发和仿真软件环境的操作，也为日后与 $MIPS$ 处理器相关的实验打下了良好的基础。