实验五 数据通路与控制单元设计

一、实验目的

1 掌握 Verilog 语言和 Vivado、Logisim 开发平台的使用；

2 掌握数据通路与控制单元的设计和测试方法。

二、实验内容

1 数据通路的设计；

2 控制单元的设计。

三、实验要求

1 掌握 Vivado 或 Logisim 开发工具的使用，掌握以上电路的设计和测试方法；

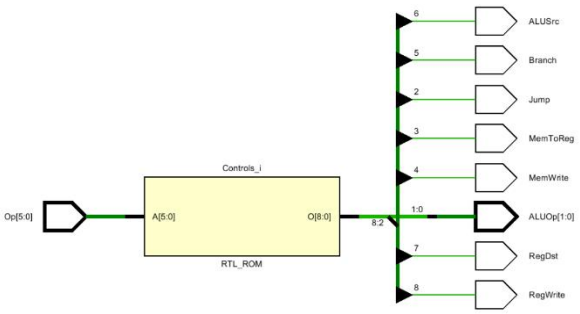
2 记录设计和调试过程（Verilog 代码/电路图/表达式/真值表，Vivado 仿真结果，Logisim 验证结果等）；

3 分析 Vivado 仿真波形/Logism 验证结果，注重输入输出之间的对应关系。

四、实验过程及分析

主译码器：

```
module MainDec(
    input [5:0] Op,
    output MemToReg, MemWrite,
    output Branch, ALUSrc,
    output RegDst, RegWrite,
    output Jump,
    output [1:0] ALUOp);
    reg [8:0] Controls;
    assign {RegWrite, RegDst, ALUSrc, Branch, MemWrite, MemToReg, Jump, ALUOp} = Controls;
    always@(*)
        case(Op)
            6'b000000: Controls <= 9'b110000010;
            6'b100011: Controls <= 9'b101001000;
            6'b101011: Controls <= 9'b001010000;
            6'b000100: Controls <= 9'b000100001;
            6'b001000: Controls <= 9'b101000000;
            6'b000010: Controls <= 9'b000000100;
            default:  Controls <= 9'bxxxxxxxxx;
        endcase
endmodule
```



该模块接收一个 6 位的操作码（Op），并产生一系列控制信号，用于控制计算机系统中的各种功能单元。

module MainDec(: 定义了一个 Verilog 模块，命名为 MainDec。

input [5:0] Op,: 声明一个 6 位的输入端口 Op，这是操作码的输入。

output MemToReg, MemWrite, Branch, ALUSrc, RegDst, RegWrite, Jump,: 声明多个输出端口，分别为控制信号 MemToReg, MemWrite, Branch, ALUSrc, RegDst, RegWrite, Jump。

output [1:0] ALUOp);: 声明一个 2 位的输出端口 ALUOp，用于控制 ALU 的操作。

reg [8:0] Controls;: 声明一个 9 位的寄存器 Controls，用于存储生成的控制信号。

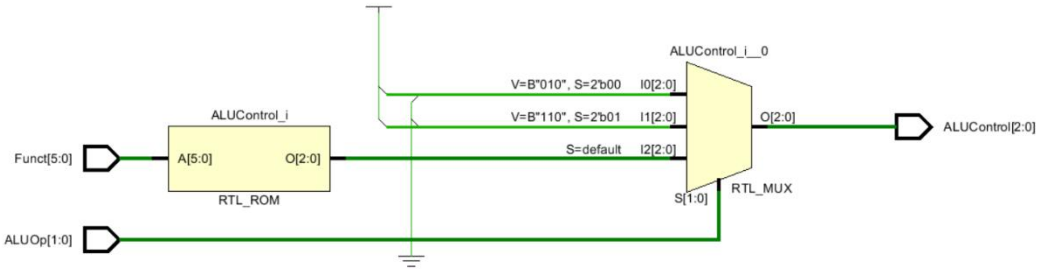
assign {RegWrite, RegDst, ALUSrc, Branch, MemWrite, MemToReg, Jump, ALUOp} = Controls;: 使用 assign 语句将 Controls 寄存器中的值分配给多个输出端口，通过花括号 {} 实现。这里假设 Controls 的最低位对应于 ALUOp 的最低位，依次类推。
always @(*): 使用 always 块，该块会在任何输入发生变化时执行。
case(Op): 使用 case 语句，根据输入的操作码 Op 进行分支选择。
每个 case 分支对应于一个特定的操作码，将对应的控制信号加载到 Controls 寄存器中。
default: Controls <= 9'bxxxxxxxxx;: 如果操作码不匹配任何 case 分支，使用 default 分支将 Controls 寄存器的值设置为 9'bxxxxxxxxx，表示未知状态。

总体而言，这个 Verilog 模块的目的是根据输入的 6 位操作码生成与之对应的一系列控制信号，这些信号通常用于控制计算机系统中的各个功能单元。

ALU 译码器：

```
module ALUDec(
    input [5:0] Funct,
    input [1:0] ALUOp,
    output reg [2:0] ALUControl);

    always@(*)
        case(ALUOp)
          2'b00: ALUControl <= 3'b010;
          2'b01: ALUControl <= 3'b110;
           default: case(Funct)
               6'b100000: ALUControl <= 3'b010;
               6'b100010: ALUControl <= 3'b110;
               6'b100100: ALUControl <= 3'b000;
               6'b100101: ALUControl <= 3'b001;
               6'b101010: ALUControl <= 3'b111;
               default:ALUControl <= 3'bxxx;
           endcase
       endcase
endmodule
```



这是一个用于 ALU 控制的 Verilog 模块；
module ALUDec(: 定义了一个 Verilog 模块，命名为 ALUDec。
input [5:0] Funct,: 声明了一个 6 位的输入端口 Funct，用于接收功能码。
input [1:0] ALUOp,: 声明了一个 2 位的输入端口 ALUOp，用于接收 ALU 操作码。
output reg [2:0] ALUControl);: 声明了一个 3 位的寄存器输出端口 ALUControl，用于输出 ALU 的控制信号。
always @(*): 使用 always 块，该块会在任何输入发生变化时执行。
case(ALUOp): 使用 case 语句，根据输入的 ALU 操作码 ALUOp 进行分支选择。

2'b00: ALUControl <= 3'b010;: 如果 ALUOp 等于2'b00,则将 ALUControl 设置为3'b010。
2'b01: ALUControl <= 3'b110;: 如果 ALUOp 等于2'b01,则将 ALUControl 设置为3'b110。

default: case(Funct): 如果 ALUOp 不匹配任何 case 分支，使用 default 分支，进入嵌套的 case(Funct) 语句。
6'b100000: ALUControl <= 3'b010;: 如果 Funct 等于6'b100000，则将 ALUControl 设置为3'b010。
6'b100010: ALUControl <= 3'b110;: 如果 Funct 等于6'b100010，则将 ALUControl 设置为3'b110。
其余的 case 分支用于匹配不同的 Funct 值，并相应地设置 ALUControl 的值。
如果 Funct 不匹配任何 case 分支，使用 default 分支将 ALUControl 设置为3'bxxx，表示未知状态。
总体而言，这个Verilog模块的目的是根据输入的ALU操作码（ALUOp）和功能码（Funct）生成与之对应的ALU控制信号（ALUControl)。这些控制信号用于控制算术逻辑单元（ALU）的操作，例如加法、减法、逻辑运算等。

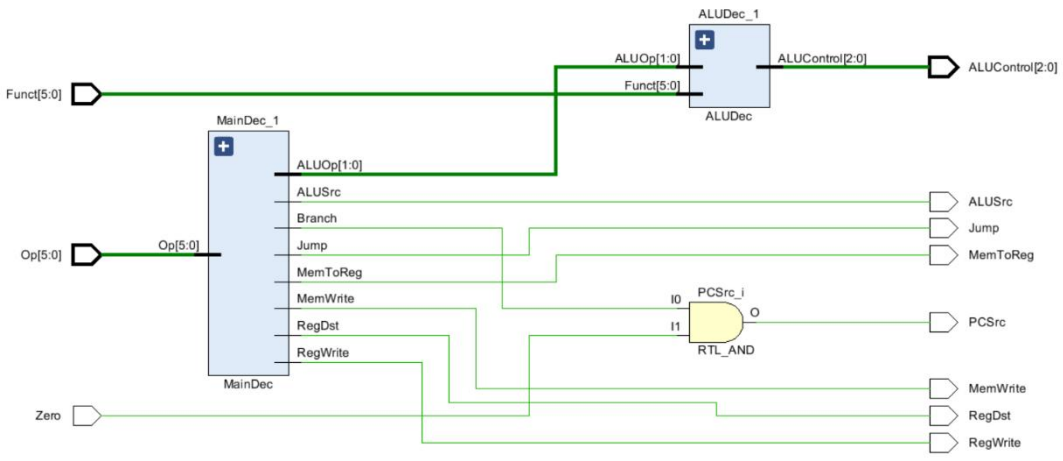
Controller：

```
module Controller(
    input [5:0] Op, Funct,
    input Zero,
    output MemToReg, MemWrite,
    output PCSrc, ALUSrc,
    output RegDst, RegWrite,
    output Jump,
    output [2:0] ALUControl);

    wire [1:0] ALUOp;
    wire Branch;

    MainDec MainDec_1(Op, MemToReg, MemWrite, Branch, ALUSrc, RegDst, RegWrite, Jump, ALUOp);
    ALUDec ALUDec_1(Funct, ALUOp, ALUControl);
    assign PCSrc = Branch & Zero;

endmodule
```



这是一个控制器（Controller）的 Verilog 模块；
module Controller(: 定义了一个 Verilog 模块，命名为 Controller。
input [5:0] Op, Funct,: 声明了两个输入端口，分别是 6 位的操作码 Op 和功能码 Funct。
input Zero,: 声明了一个输入端口 Zero，可能是用于表示零值的信号。
output MemToReg, MemWrite, PCSrc, ALUSrc, RegDst, RegWrite, Jump,: 声明了多个输出端口，分别为控制信号 MemToReg, MemWrite, PCSrc, ALUSrc, RegDst, RegWrite, Jump。
output [2:0] ALUControl);: 声明了一个 3 位的输出端口 ALUControl，用于输出 ALU 的控制信号。
wire [1:0] ALUOp;: 声明了一个 2 位的线网（wire） ALUOp，用于传递给 MainDec 模块中的 ALUOp 输入端口。
wire Branch;: 声明了一个线网 Branch，用于传递给 MainDec 模块中的 Branch 输入端口。
MainDec MainDec_1(Op, MemToReg, MemWrite, Branch, ALUSrc, RegDst, RegWrite, Jump, ALUOp);: 实例化了一个名为 MainDec_1 的 MainDec 模块，将输入和输出端口相连。
ALUDec ALUDec_1(Funct, ALUOp, ALUControl);: 实例化了一个名为 ALUDec_1 的 ALUDec 模块，将输入和输出端口相连。
assign PCSrc = Branch & Zero;: 使用 assign 语句，将 Branch 和 Zero 进行与运算，并将结果赋给 PCSrc。这个逻辑通常用于确定是否进行分支跳转。

总体而言，这个 Verilog 模块的目的是通过实例化 MainDec 和 ALUDec 模块，并通过逻辑运算，生成用于控制计算机系统中各个功能单元的多个控制信号。其中，MainDec 用于生成主要的控制信号，而 ALUDec 用于生成与 ALU 相关的控制信号。

仿真模块：

```
module sim_con;
    reg [5:0] Op;
    reg [5:0] Funct;
    reg Zero;
    wire MemToReg, MemWrite, PCSrc, ALUSrc, RegDst, RegWrite, Jump;
    wire [2:0] ALUControl;
    wire [1:0] ALUOp;
    wire Branch;

    Controller con (
      .Op(Op),
      .Funct(Funct),
      .Zero(Zero),
      .MemToReg(MemToReg),
      .MemWrite(MemWrite),
      .PCSrc(PCSrc),
      .ALUSrc(ALUSrc),
      .RegDst(RegDst),
      .RegWrite(RegWrite),
      .Jump(Jump),
      .ALUControl(ALUControl)
    );
```

```
    initial begin
      Op = 6'b000000;
      Funct = 6'b100000;
      Zero = 1'b1;
      #10;
      Op = 6'b100011;
      Funct = 6'b100010;
      #10
      Op = 6'b000000;
      Funct = 6'b100100;
      #10
      Op = 6'b101011;
      Funct = 6'b100101;
      #10
      Op = 6'b000100;
      Funct = 6'b101010;
      #10
      Op = 6'b001000;
      Funct = 6'b100000;
      #10
      Op = 6'b000010;
      #10;
      Op = 6'b000000;
      Funct = 6'b100010;
      #10
      Op = 6'b000000;
      Funct = 6'b100100;
      #10
      Op = 6'b000000;
      Funct = 6'b100101;
      #10
      Op = 6'b000000;
      Funct = 6'b101010;
      #10
      Op = 6'b000000;
      Funct = 6'b100000;

    end

endmodule
```

声明了一个名为 sim_con 的 Verilog 模块。
声明了输入和输出端口，其中输入端口包括 Op（6 位操作码）、Funct（6 位功能码）、Zero（零值信号），输出端口包括多个控制信号，例如 MemToReg、MemWrite、PCSrc、ALUSrc 等。
实例化了一个名为 con 的 Controller 模块，并将输入和输出端口相连。
使用 initial 块进行仿真初始化。
分别给输入信号 Op、Funct、Zero 赋初值，并使用 #10 延时 10 个时间单位。
依次改变 Op 和 Funct 的值，模拟不同的输入情况。
总体而言，这个 sim_con 模块用于对 Controller 模块进行仿真测试。通过不同的输入来模拟不同的操作码和功能码，然后观察 Controller 模块的输出信号，以验证其在不同情况下是否产生了正确的控制信号。

时序输出：
时刻 t=0：
Op = 6'b000000;
Funct = 6'b100000;
Zero = 1'b1;
初始输入。
时刻 t=20：

Op = 6'b100011;
Funct = 6'b100010;
#10: 10 个时间单位之后，Op 和 Funct 的值发生变化。6'b100011: Controls <= 9'b101001000;对 Controls 进行赋值，ALUOp 也被赋值为 00，ALUControl 就赋值为 010，就是 2。
时刻 t=30：

Op = 6'b000000;
Funct = 6'b100100;
#10: 10 个时间单位之后，Op 和 Funct 的值发生变化，其他信号保持不变。6'b000000: Controls <= 9'b110000010; 对 Controls 进行赋值，ALUOp 也被赋值为 10，6'b100100: ALUControl <= 3'b000;ALUControl 就赋值为 000，就是 0。

时刻 t=40：

Op = 6'b101011;
Funct = 6'b100101;
#10: 10 个时间单位之后，Op 和 Funct 的值发生变化，其他信号保持不变。6'b101011: Controls <= 9'b001010000;对 Controls 进行赋值，ALUOp 也被赋值为 00， ALUControl 就赋值为 010，就是 2。

时刻 t=50：

Op = 6'b000100;
Funct = 6'b101010;
#10: 10 个时间单位之后，Op 和 Funct 的值发生变化，其他信号保持不变。6'b000100: Controls <= 9'b000100001;对 Controls 进行赋值，ALUOp 也被赋值为 01， ALUControl 就赋值为 110，就是 6。

以此类推，后续的时刻依次改变 Funct 的值，并保持 Op 的值为 000000，使得 ALU 译码器可以进入 Funct 选择。
Funct 依次为

Funct = 6'b100010;

Funct = 6'b100100;

Funct = 6'b100101;

Funct = 6'b101010;

Funct = 6'b100000;

根据他们的对应关系

6'b100000: ALUControl <= 3'b010;
6'b100010: ALUControl <= 3'b110;
6'b100100: ALUControl <= 3'b000;
6'b100101: ALUControl <= 3'b001;
6'b101010: ALUControl <= 3'b111;

那么结果的 ALUControl 就依次为 6,0,1,7,2；

五、调试和心得体会

实验总结：

Verilog 编程的应用： 通过 Verilog 编程，我学到了如何描述和设计硬件电路。这种硬件描述语言不仅仅是一种编程语言，更是一种将计算机组成原理转化为实际硬件设计的媒介。通过编写 Verilog 代码，我能够具体实现计算机组成原理中的各个模块，包括控制器和 ALU 等。

计算机组成原理的实际应用： 实验使我能够将抽象的计算机组成原理应用于实际的硬件设计。通过实现控制器和 ALU 的模块，我深入理解了 CPU 的基本组成部分，包括指令译码、控制单元和算术逻辑单元。这种实际应用使我更好地理解计算机内部的工作原理。

系统级设计与模块化思维： 实验中的系统级设计方法让我学会了将整个系统分解为多个模块，每个模块专注于特定的功能。这种模块化的设计思维有助于提高系统的可读性、可维护性和可扩展性。通过将系统分解为模块，我更容易理解每个部分的功能和相互关系。

仿真调试的技能： 实验中使用的仿真工具为我提供了验证设计正确性的手段。观察信号波形并进行调试是一项重要的实践技能，这有助于确保硬件系统按照预期工作。在调试过程中，我学到了如何追踪信号并识别潜在问题，这对于工程项目的实际应用至关重要。

心得体会：

实践是理解的最佳途径： 通过亲自实践，我深刻理解了计算机组成原理的概念。在实际编写 Verilog 代码、实现模块、仿真调试的过程中，我更加牢固地掌握了相关知识

团队协作与沟通： 如果实验是在团队中完成的，团队协作与沟通能力也是一个关键点。合理的分工、有效的沟通和团队协作是解决问题和完成复杂项目的重要组成部分。

计算机科学的实际应用： 通过实验，我对计算机科学的实际应用有了更直观的认识。理论知识与实际操作相结合，让我更好地理解了计算机系统的运作原理。

持续学习的重要性： 实验让我认识到计算机科学是一个不断发展的领域，持续学习是保持竞争力的关键。通过这次实验，我意识到对新技术和方法的不断追求是作为计算机科学专业人士的必备素质。

总的来说，这次计算机组成实验是我学习之旅中的一站，让我在硬件层面深入理解了计算机组成原理。这次实验为我今后的学术和职业发展提供了坚实的基础。