电子科技大学 计算机科学与工程 学院

实 验 报 告

（实验）课程名称 计算机组成原理综合实验

指导教师： 陈虹

电子科技大学教务处制表

电 子 科 技 大 学

（实验一）实验报告

学生姓名：黄鑫　学号：2021050901013　　指导教师：陈虹

实验地点：主楼 A2-411　　实验时间：2023.5.20

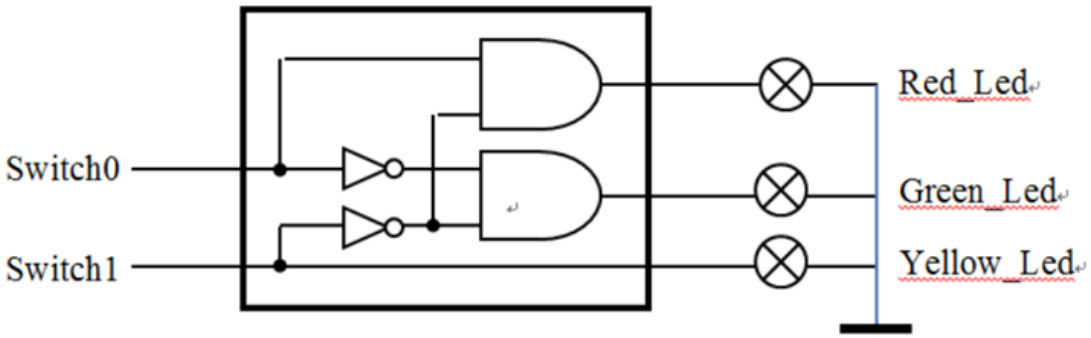
一、实验室名称：主楼 A2-411

二、实验项目名称：Verilog 基本设计

三、实验学时：4 学时

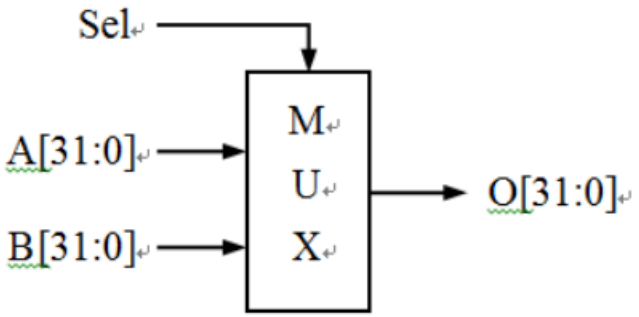
四、实验原理：

1. 交通信号的红绿灯控制

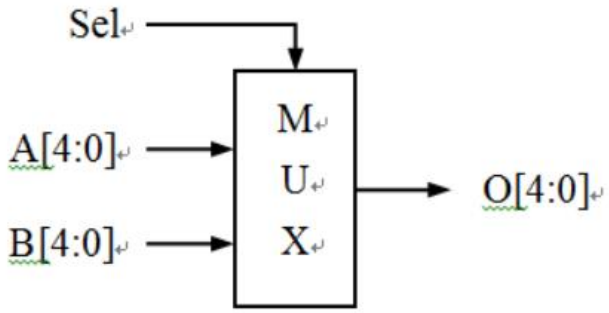


Switch1	Switch0	Red_Led	Green_Led	Yellow_Led
0	0	灭	亮	灭
0	1	亮	灭	灭
1	X	灭	灭	亮

2. 32 位 2 选 1 多路选择器



3. 5 位 2 选 1 多路选择器

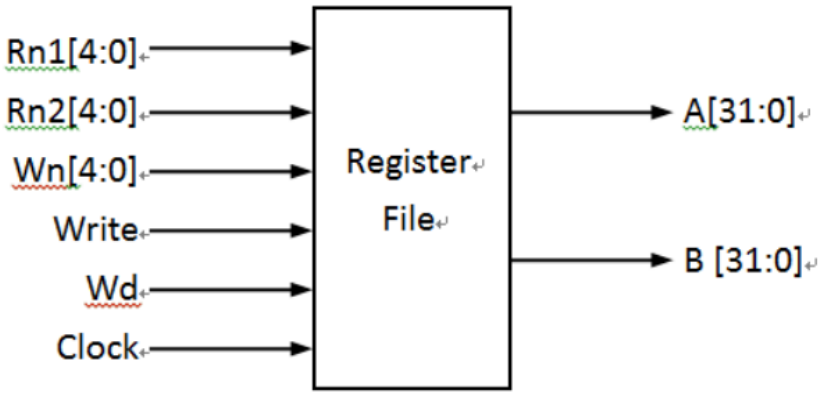


Function:

Sel = 0： O = A

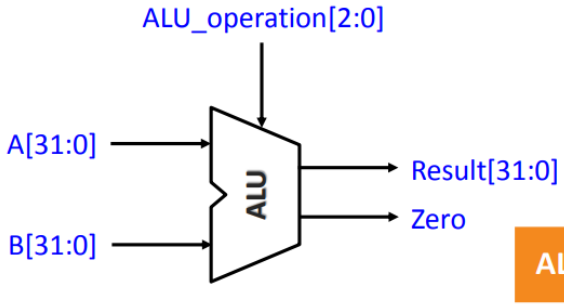
Sel = 1： O = B

4. 32 位寄存器堆



Register File 逻辑结构

5. ALU 的设计



ALU_Operation	功能描述
000	Add（加）
100	Sub（减）
001	And（与）
101	Or（或）
010	Xor（异或）
110	设置高位

五、实验目的：

通过本实验，将学习如何使用 Verilog HDL 硬件描述语言设计硬件电路，并掌握 FPGA 开发板的基本使用方法。在本实验中，将实现以下几个部件的设计：32 位 2 选 1 多路选择器、5 位 2 选 1 多路选择器、32 位寄存器堆和 ALU。通过完成这些设计，将掌握硬件电路设计的基本方法和技能，并加深对所学知识的理解。

首先，将学习如何使用 Verilog HDL 语言描述硬件电路。Verilog HDL 是一种硬件描述语言，用于描述和设计数字电路。将学习如何使用 Verilog HDL 编写电路的结构、行为和数据流描述，并将其应用于设计 32 位 2 选 1 多路选择器、5 位 2 选 1 多路选择器、32 位寄存器堆和 ALU 等部件。

其次，将学习如何使用 FPGA 开发板。FPGA 是一种可编程逻辑器件，可以根据需要配置为不同的电路功能。学习如何通过开发板上的输入和输出接口与设计的电路进行交互。

六、实验内容：

1. 掌握用 Verlog 设计硬件电路的基本方法
2. 开发板的基本使用
3. 基本器件的设计：
   （1）32 位 2 选 1 多路选择器
   （2）5 位 2 选 1 多路选择器
   （3）32 位寄存器堆
   （4）ALU 的设计

七、实验器材（设备、元器件）：

PC、ISE Design Suite 14.7、FPGA 开发板：Spartan6-XC6SLX45

八、实验步骤：

1. 设计一个交通信号的红绿灯控制

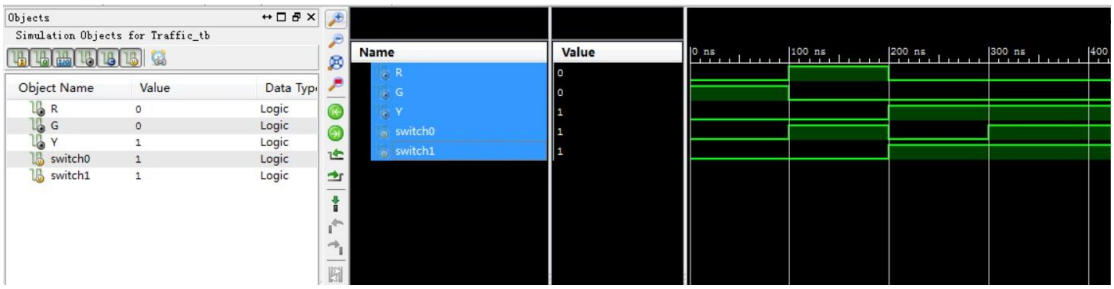
（1）创建工程并创建模块 Controller

```
//////////////////////////////////////////////////////////////////////////////////
module Traffic(
    input switch0,switch1,
    output R,G,Y
    );

    assign Y =  switch1;
    assign R =  switch0 & ~ switch1;
    assign G = ~switch0 & ~ switch1;


endmodule
```

（2）编写测试代码

```
    initial begin
        // Initialize Inputs
        switch0 = 0;
        switch1 = 0;

        // Wait 100 ns for global reset to finish
        #100;
        switch1 = 0;    switch0 = 1;

        #100;
        switch1 = 1;    switch0 = 0;

        #100;
        switch1 = 1;    switch0 = 1;

    end


endmodule
```

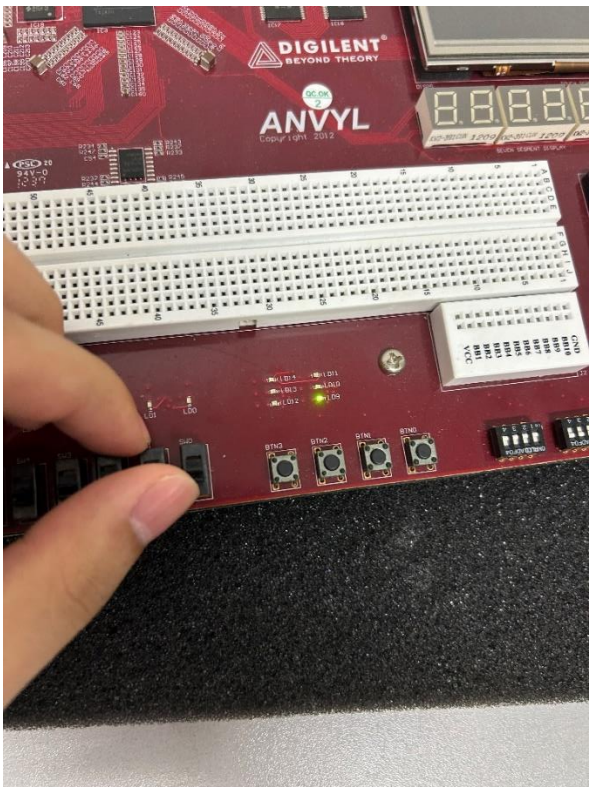
（3）仿真结果

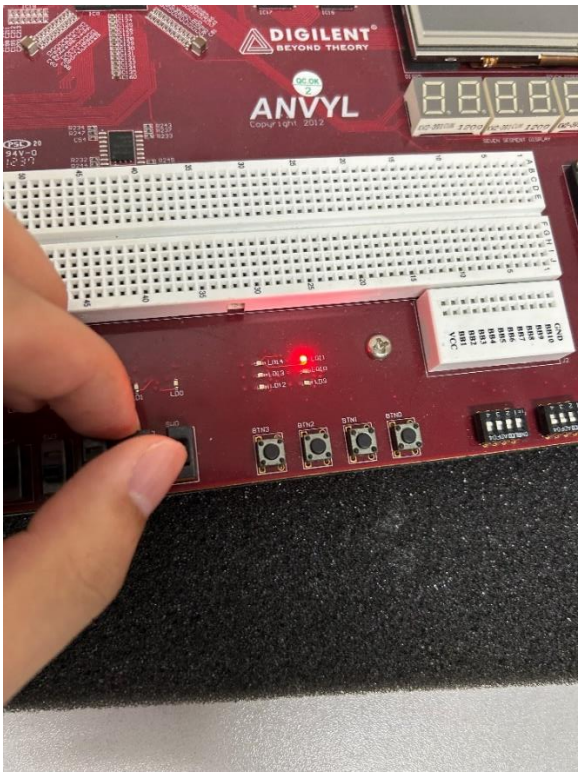


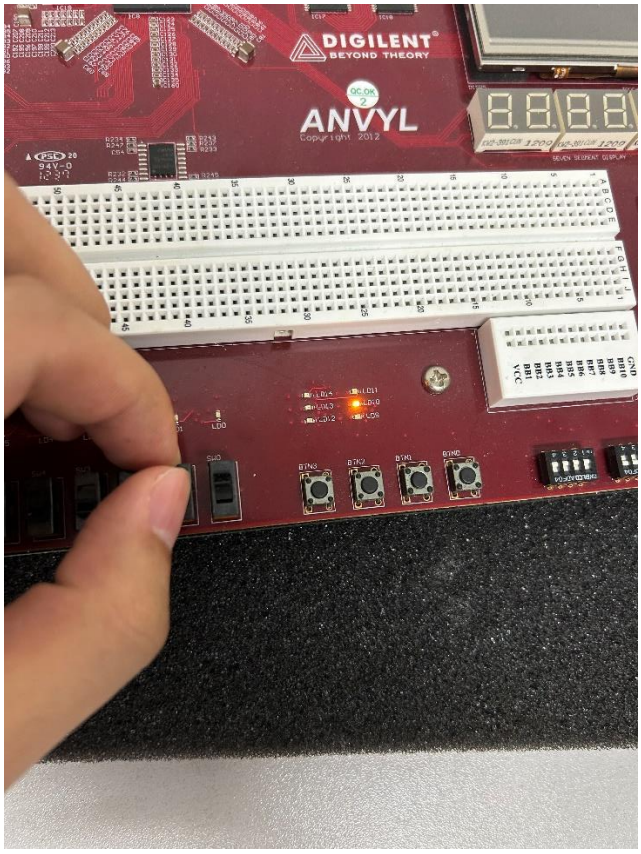
（4）添加约束文件

```
1  #switches
2  NET switch0 LOC = V5;
3  NET switch1 LOC = U4;
4
5  #Leds
6  NET R LOC = T8;
7  NET G LOC = R7;
8  NET Y LOC = U6;
```

（5）生成流代码并下载到开发板


DIGILENT
BEYOND THEORY
ANVYL
Copyright 2012


DIGILENT
BEYOND THEORY
ANVYL
Copyright 2012



2.32 位 2 选 1 多路选择器设计

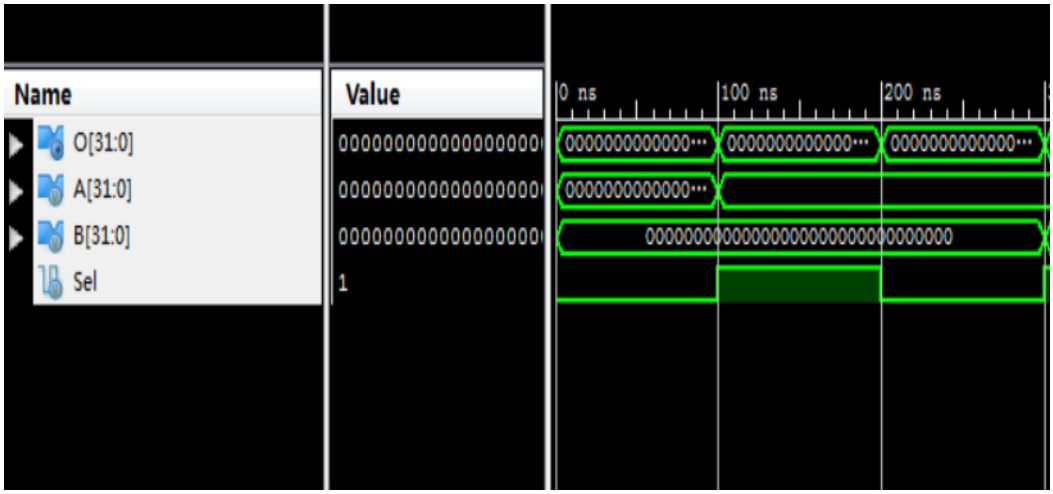
（1）模块代码

```
module MUX32_2_1(
        input[31:0] A,
        input[31:0] B,
        input Sel,
        output[31:0] O
    );
    assign O = Sel? A : B;

endmodule
```

（2）测试代码

```
initial begin
    // Initialize Inputs
    A = 0;
    B = 0;
    Sel = 0;

    // Wait 100 ns for global reset to finish
    #100;
    A=1;B=0;Sel=1;
    #100;
    A=1;B=0;Sel=0;
    #100;
    A=1;B=1;Sel=1;

    // Add stimulus here

end
```

（3）仿真结果



3.5 位 2 选 1 多路选择器的设计

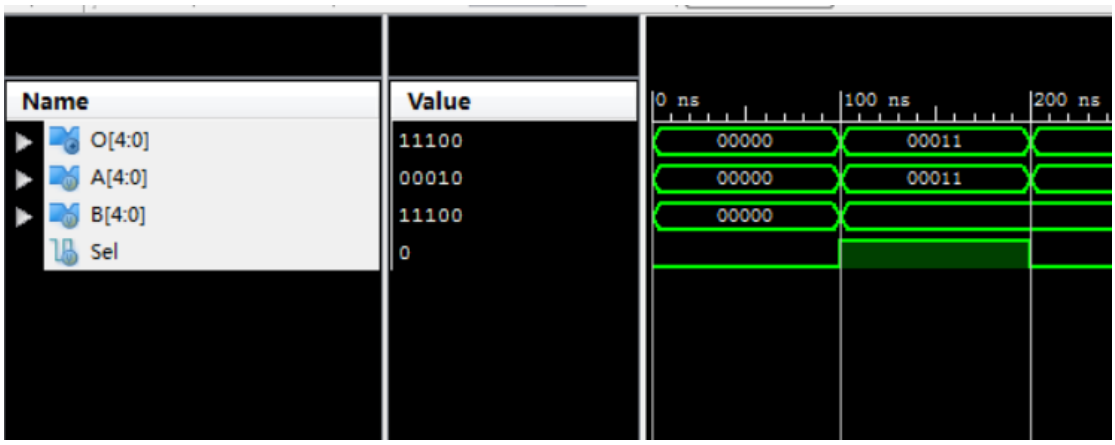
（1）模块代码

```
21  module MUX5_2_1(
22        input[4:0] A,
23        input[4:0] B,
24        input Sel,
25        output[4:0] O
26      );
27     assign O = Sel? A : B;
28
29  endmodule
```

（2）测试代码

```
initial begin
    // Initialize Inputs
    A = 0;
    B = 0;
    Sel = 0;

    // Wait 100 ns for global reset to finish
    #100;
    A=5'b00011;
    B=5'b11100;
    Sel=1;

    #100;
    A=5'b00010;
    B=5'b11100;
    Sel=0;

    // Add stimulus here

end
```

（3）仿真结果



4. 32 位寄存器堆的设计

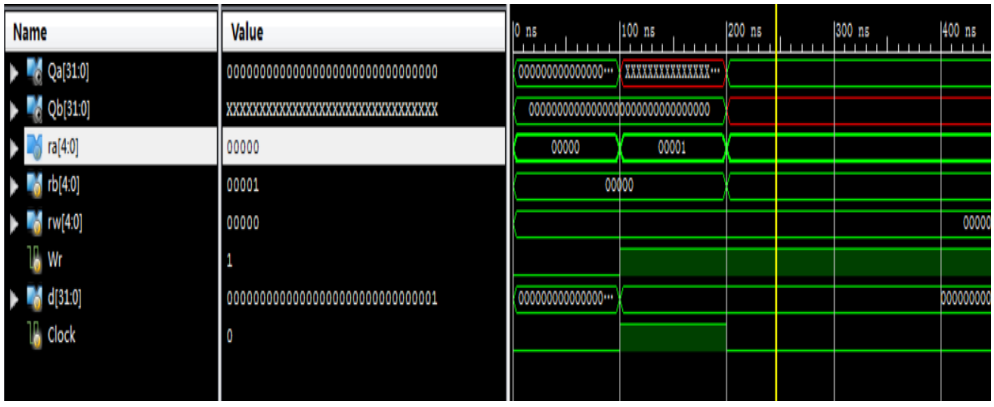
（1）模块代码

```
//////////////////////////////////////////////////////////////////////////////////
module RegFile(
      input[4:0] Rn1,Rn2,Wn,
      input Write,
      input[31:0] Wd,
      output[31:0] A,B,
      input Clock
    );

   reg[31:0] Register[1:31];

   assign A = (Rn1 == 0)? 0 : Register[Rn1];
   assign B = (Rn2 == 0)? 0 : Register[Rn2];

   always @ (posedge Clock) begin
      if ((Write) && (Wn != 0)) Register[Wn] <= Wd;
   end
endmodule
```

（2）测试代码

```
initial begin
    // Initialize Inputs
    ra = 0;
    rb = 0;
    rw = 0;
    Wr = 0;
    d = 0;
    Clock = 0;

    // Wait 100 ns for global reset to finish
    #100;
    ra=1;rb=0;Clock=1;Wr=1;rw=0;d=1;
    #100;
    ra=0;rb=1;Clock=0;Wr=1;rw=0;d=1;

    // Add stimulus here

end
```

（3）仿真结果



5. ALU 的设计

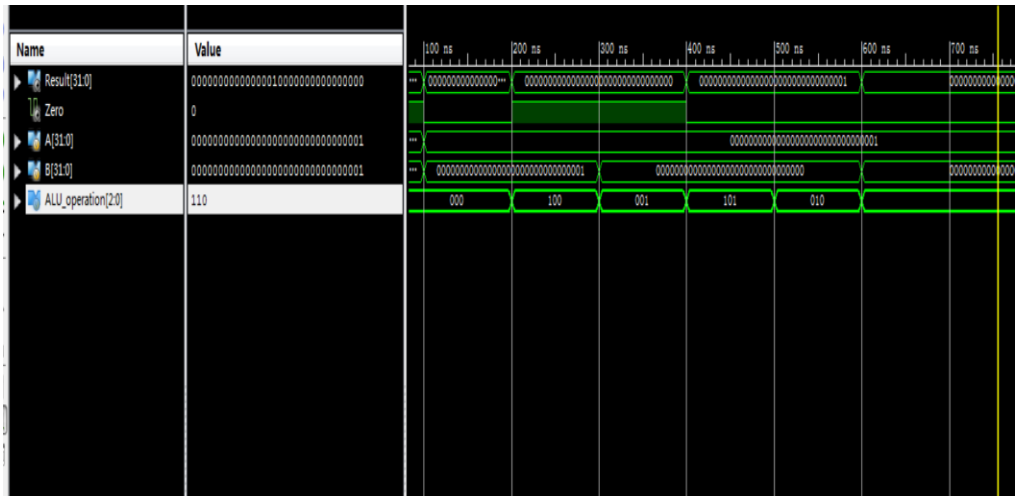
（1）模块代码

```
//////////////////////////////////////////////////////////////////////////////////
module ALU(
   input [31:0] A,B,
   input [2:0] ALU_operation,
   output [31:0] Result,
   output        Zero
    );

assign Result = (ALU_operation==3'b000)?A+B :
                (ALU_operation==3'b100)?A-B :
                (ALU_operation==3'b001)?A&B :
                (ALU_operation==3'b101)?A|B :
                (ALU_operation==3'b010)?A^B :
                (ALU_operation==3'b110)?{B[15:0],16'h0} :
                32'hxxxxxxxx;

assign Zero = ~|Result;


endmodule
```

（2）测试代码

```
initial begin
    // Initialize Inputs
    A = 0;
    B = 0;
    ALU_operation = 0;

    // Wait 100 ns for global reset to finish
    #100;
    A=1;B=1;ALU_operation=3'b000;
    #100;
    A=1;B=1;ALU_operation=3'b100;
    #100;
    A=1;B=0;ALU_operation=3'b001;
    #100;
    A=1;B=0;ALU_operation=3'b101;
    #100;
    A=1;B=0;ALU_operation=3'b010;
    #100;
    A=1;B=1;ALU_operation=3'b110;

    // Add stimulus here
end
```

（3）仿真结果



九、结果分析：

经过实验，成功实现了交通信号的红绿灯控制、32 位 2 选 1 多路选择器、5 位 2 选 1 多路选择器、32 位寄存器堆和 ALU 等部件的设计。在进行测试时，实验结果与预期结果一致。

十、实验结论：

通过本次实验，掌握了使用 Verilog HDL 硬件描述语言设计硬件电路的基本方法，并熟悉了 FPGA 开发板的基本使用。成功设计了交通信号控制、多路选择器和寄存器堆等重要部件。实验结果验证了设计的正确性。

十一、总结及心得体会：

本次实验使我们深入理解了硬件电路设计的重要性，并通过实践掌握了一些基本的电路设计方法和技巧。在设计过程中，我们也发现了一些不足和问题，这将激励我们在未来的学习和实践中不断改进和提高自己的能力。

十二、对本实验过程及方法、手段的改进建议：

为了更好地完成实验，建议在实验前加强对 Verilog 语言的学习和掌握，以提高对硬件描述语言的熟练度。此外，在实验中可以注重调试和测试的过程，以确保设计的正确性和稳定性。同时，鼓励尝试更复杂的电路设计，以拓展设计技能和知识广度。

报告评分：

指导教师签字：

电 子 科 技 大 学

（实验二）实验报告

学生姓名：黄鑫　学号：2021050901013　指导教师：陈虹

实验地点：主楼 A2-411　　实验时间：2023.5.27

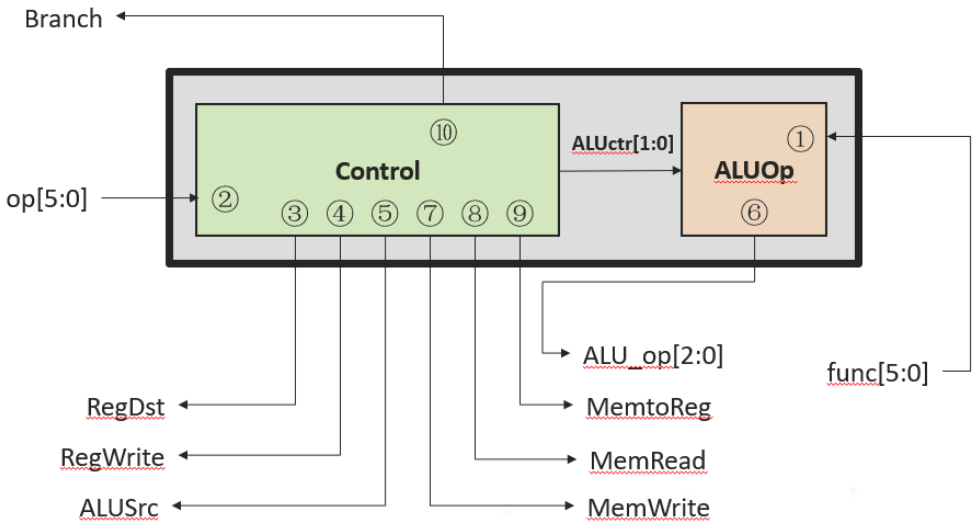
一、实验室名称：主楼 A2-411

二、实验项目名称：控制器与取指电路设计与实现

三、实验学时：4 学时

四、实验原理：

1. 控制器结构



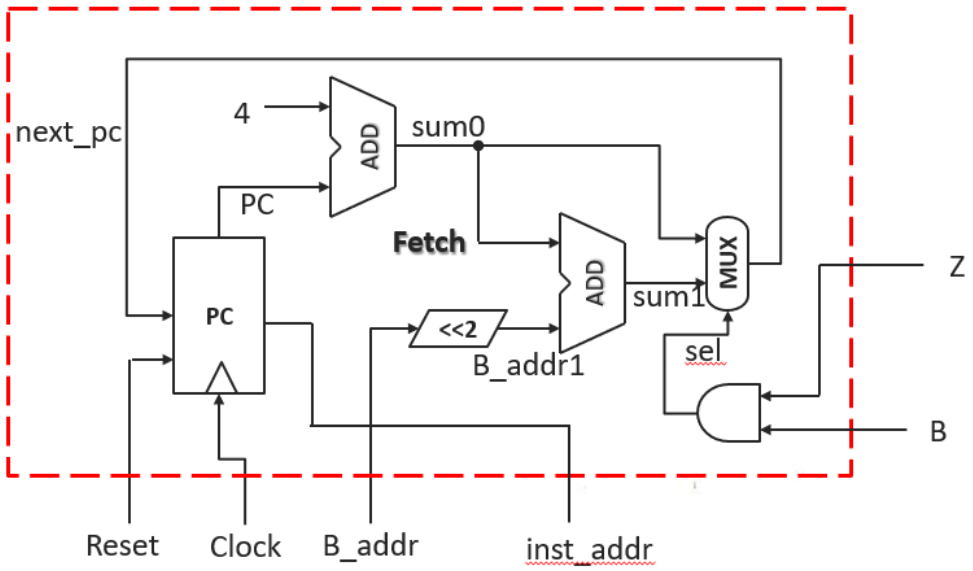
2. 控制单元的输入与输出之间的关系

Input		Oupput							
指令	op[5:0]	RegDst	RegWrite	ALUSrc	MemWrite	MemRead	MemtoReg	Branch	ALUctr[1:0]
RT	000000	1	1	0	0	0	0	0	1 0
lw	100011	0	1	1	0	1	1	0	0 0
sw	101011	x	0	1	1	0	x	0	0 0
beq	000100	x	0	0	0	0	x	1	0 1
lui	001111	0	1	1	0	0	0	0	1 1

3. ALU 控制单元的输入与输出关系

指令	func[5:0]	功能	ALUctr[1:0]	ALU_op[2:0]
R-Type	100000	Add	10	000
	100010	Sub		100
	100100	And		001
	100101	Or		101
	100110	Xor		010
lw	XXXXXX	取数	00	000
sw		存数		
beq		分支	01	100
lui		置高位	11	110

4. 取值电路的设计



五、实验目的：

本实验的目的是设计和实现控制器和取指电路，使其能够对相应的三类指令进行响应。此外，还要设计和实现基本器件，如 32 位加法器和左移两位部件。

六、实验内容：

本实验将使用 Verilog HDL 语言编程，实现控制器和取指电路的功能。首先，需要编写 Verilog 代码来描述控制器和取指电路的结构、行为和数据流。然后，使用 Verilog 设计软件（如 ISE Design Suite 14.7）生成 RTL 层电路，并进行综合。最后，编写测试代码并进行仿真，验证设计的正确性。

七、实验器材（设备、元器件）：

本实验所需的器材和设备包括电脑和 Verilog 设计软件 ISE Design Suite 14.7。

八、实验步骤：

1. ALU 控制单元的设计

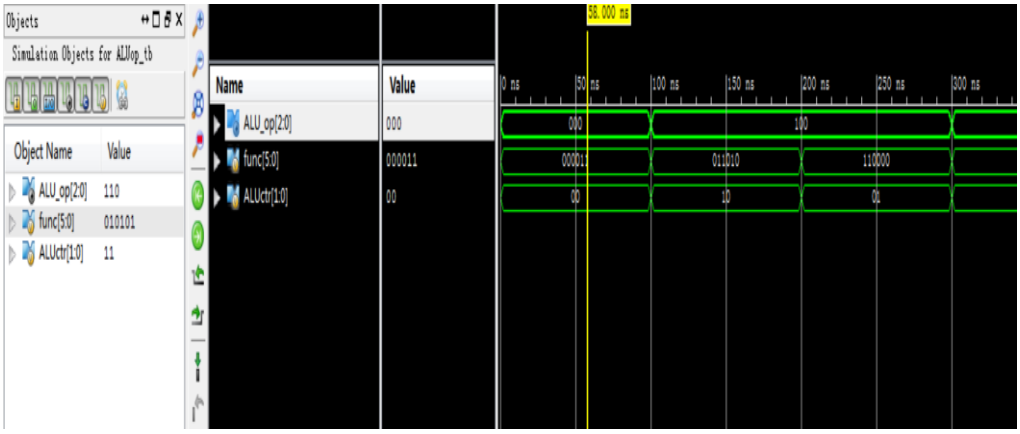
（1）模块代码

```
//////////////////////////////////////////////////////////////////////////////////
module ALUop(
    input [5:0] func,
    input [1:0] ALUctr,
    output [2:0] ALU_op
);
    wire i_Rt =ALUctr[1]&~ALUctr[0];
    assign ALU_op[2] = (i_Rt & ((~func[2] & func[1])|(func[2]&func[0]))) | ALUctr[0];
    assign ALU_op[1] = (i_Rt & ~func[0] & func[1] & func[2]) | (ALUctr[0] & ALUctr[1]);
    assign ALU_op[0] = i_Rt & ~func[1] & func[2];


endmodule
```

（2）测试代码

```
 initial begin
        func = 6'b000011;
        ALUctr = 2'b00;
        #100;
        if (ALU_op !== 3'b000) begin
            $display("Test case 1 failed");
        end

        func = 6'b011010;
        ALUctr = 2'b10;
        #100;
        if (ALU_op !== 3'b101) begin
            $display("Test case 2 fai[o]");
        end

        func = 6'b110000;
        ALUctr = 2'b01;
        #100;
        if (ALU_op !== 3'[o]10) begin
            $display("Test case 3 failed");
        end

        func = 6'b010101;
        ALUctr = 2'b11;
        #100;
        if (ALU_op !== 3'b111) begin
            $display("Test case 4 failed");
        end
```

（3）仿真结果



2.控制单元 Control 的设计

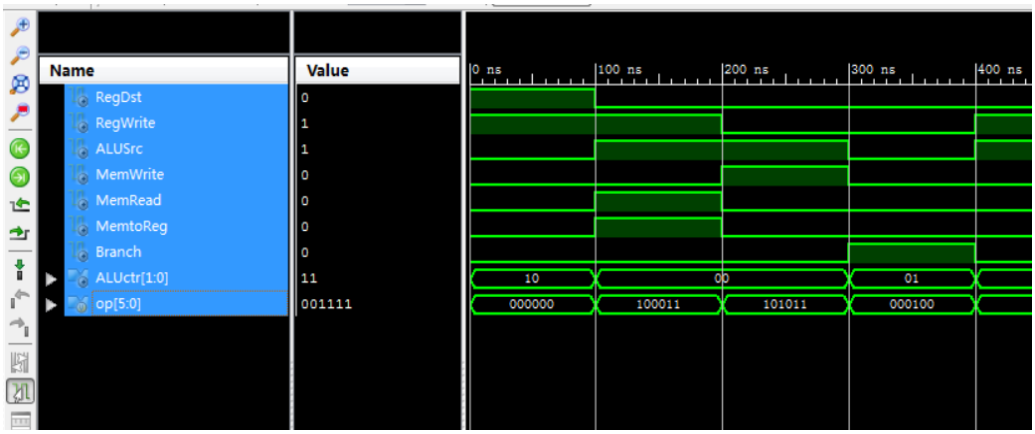
（1）模块代码

```
module Control(
    input [5:0] op,
    output RegDst, RegWrite, ALUSrc,
    output MemWrite, MemRead, MemtoReg,
    output Branch,
    output [1:0]ALUctr
     );

    wire i_Rt  = ~| op;
    wire i_lw  = op[5] & ~op[3];
    wire i_sw  = op[5] & op[3];
    wire i_beq = op[2] & ~op[1];
    wire i_lui = op[2] & op[1];

    assign RegDst     = i_Rt;
    assign RegWrite   = i_Rt | i_lw | i_lui;
    assign ALUSrc     = i_lw | i_sw | i_lui;
    assign MemWrite   = i_sw;
    assign MemRead    = i_lw;
    assign MemtoReg   = i_lw;
    assign Branch     = i_beq;

    assign ALUctr[1] = i_Rt | i_lui;
    assign ALUctr[0] = i_beq | i_lui;

endmodule
```

（2）测试代码

```
initial begin
        // Initialize Inputs
    op = 0;

    // Wait 100 ns for global reset to finish
    #100; op = 6'b100011;
    #100; op = 6'b101011;
    #100; op = 6'b000100;
    #100; op = 6'b001111;


end
```

（3）仿真结果

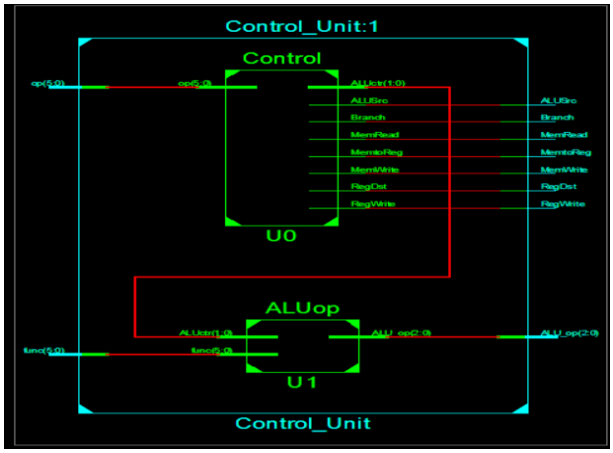


3.控制器的设计

（1）模块代码

```
//////////////////////////////////////////////////////////////////////////////////
module Control_Unit(
    input [5:0] op, func,
    output RegDst, RegWrite, ALUSrc,
    output MemWrite, MemRead, MemtoReg,
    output Branch,
    output [2:0] ALU_op
     );

    wire[1:0] ALUctr;
    Control U0( .op(op),
                .RegDst(RegDst),
                .RegWrite(RegWrite),
                .ALUSrc(ALUSrc),
                .MemWrite(MemWrite),
                .MemRead(MemRead),
                .MemtoReg(MemtoReg),
                .Branch(Branch),
                .ALUctr(ALUctr));

    ALUop U1(.func(func),.ALUctr(ALUctr),.ALU_op(ALU_op));

endmodule
```

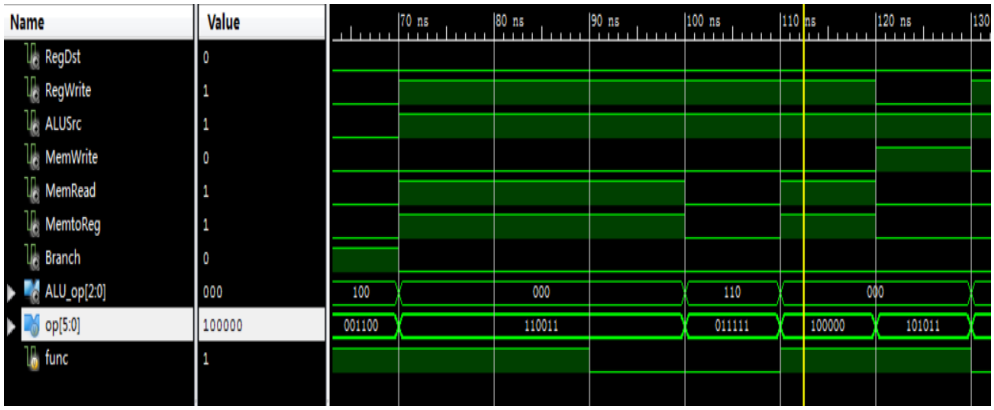
（2）RTL 层电路



（3）测试代码

```
    initial begin
        op = 6'b000000;
        func = 1'b0;

        #10 op = 6'b111111;
        func = 1'b1;

        #10 op = 6'b010101;
        func = 1'b0;

        #10 op = 6'b101010;
        func = 1'b1;

        #10 op = 6'b001100;
        #20 func = 1'b1;

        #10 op = 6'b110011;
        #20 func = 1'b0;

        #10 op = 6'b011111;
        func = 1'b0;

        #10 op = 6'b100000;
        func = 1'b1;

        #10 op = 6'b101011;
        func = 1'b1;

        #10 op = 6'b010110;
        func = 1'b0;

    end
endmodule
```

（4）仿真结果



4. 左移两位部件的设计

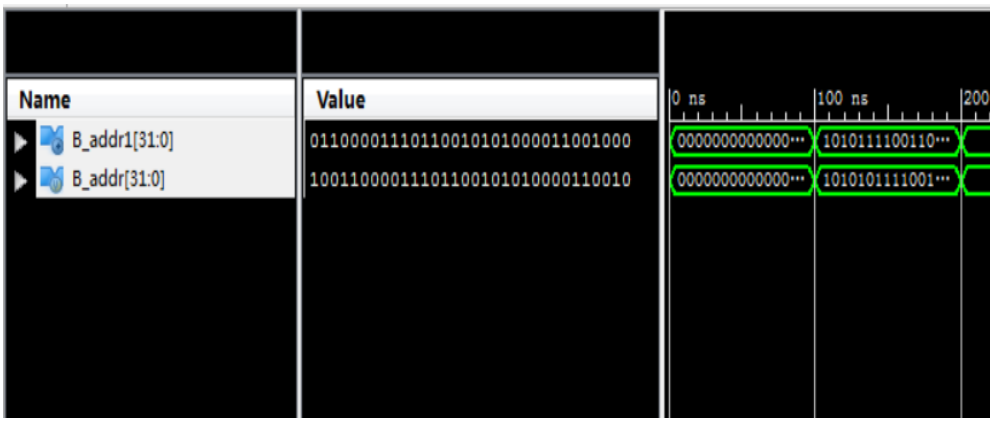
（1）模块代码

```
module Left_2_Shifter(B_addr, B_addr1);
   input    [31:0] B_addr;
   output   [31:0] B_addr1;

 assign  B_addr1[31:0] = { B_addr[29:0], 2'b00 };

endmodule
```

（2）测试代码

```
initial begin
 B_addr = 0;
 #100;

 B_addr = 32'hABCDEF01;
 #100;

 B_addr = 32'h98765432;
 #100;


end
```

（3）仿真结果



5.32 位加法器的设计

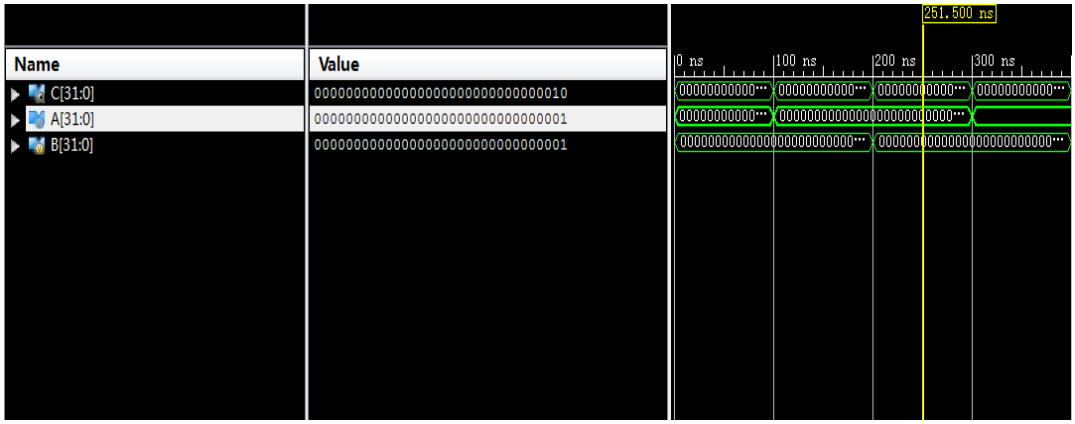
（1）模块代码

```
//////////////////////////////////////////////
module ADD32(a, b, result);
  input  [31:0] a, b;
  output [31:0] result;

  assign result = a + b;
endmodule
```

（2）测试代码

```
40
41    initial begin
42       // Initialize Inputs
43       a = 0;
44       b = 0;
45
46       // Wait 100 ns for global reset to finish
47       #100;
48       a=1;b=1;
49          #100;
50       a=0;b=1;
51          #100;
52       a=1;b=0;
53          #100;
54       a=0;b=0;
55       // Add stimulus here
56
57    end
```

（3）仿真结果

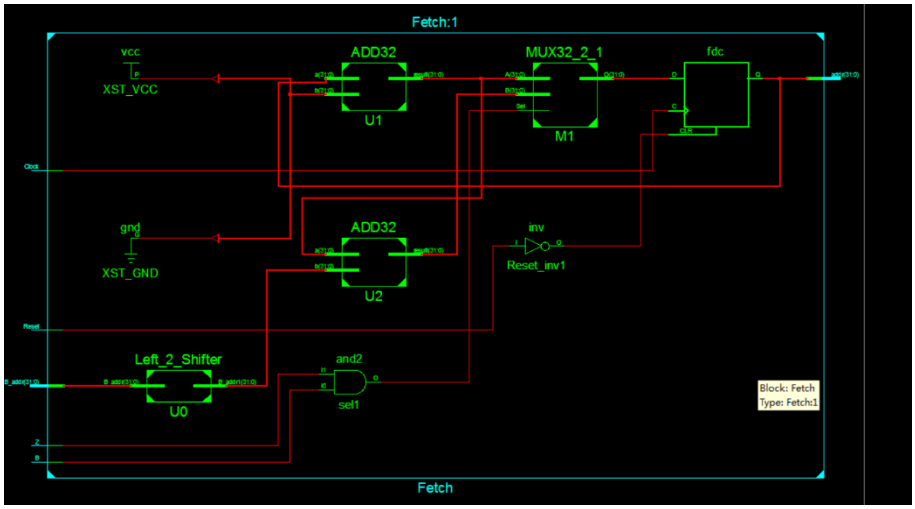


6. 取指电路的设计

（1）模块代码

```
module Fetch(
   input B, Z, Reset, Clock,
   input [31:0] B_addr,
   output[31:0] addr
    );
   reg [31:0] PC;
   wire[31:0] sum0, B_addr1, sum1, next_pc;
   wire sel = B & Z;

   Left_2_Shifter U0(.B_addr(B_addr), .B_addr1(B_addr1));
   ADD32 U1(.a(PC), .b(4), .result(sum0));
   ADD32 U2(.a(sum0), .b(B_addr1), .result(sum1));
   MUX32_2_1 M1(.A(sum0), .B(sum1), .Sel(sel), .O(next_pc));

   assign addr = PC;
   always @(posedge Clock or negedge Reset) begin
      if(Reset == 0)
         PC = 0;
      else
         PC = next_pc;
   end

endmodule
```

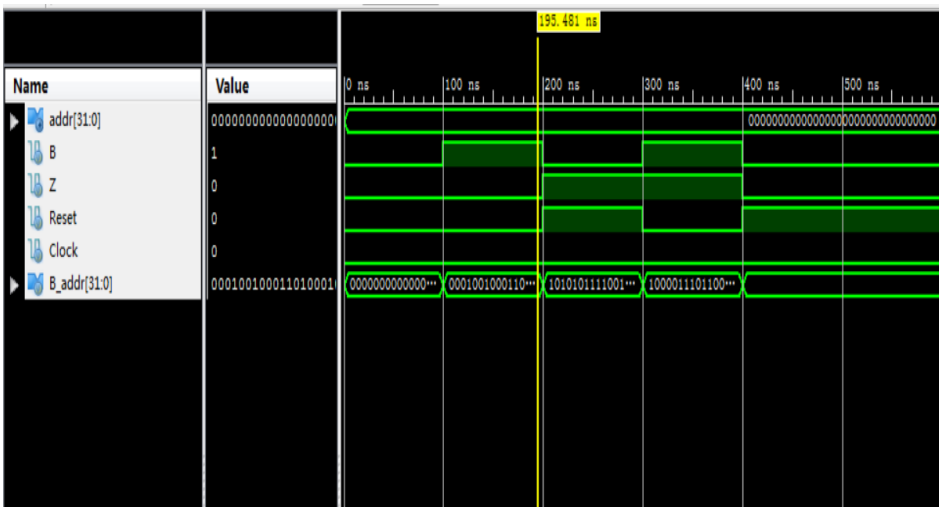
（2）RTL 层电路



（3）测试代码

```
initial begin
   // Initialize Inputs
   B = 0;
   Z = 0;
   Reset = 0;
   Clock = 0;
   B_addr = 0;

   // Wait 100 ns for global reset to finish
   #100;
    B = 1;
 Z = 0;
 Reset = 0;
 B_addr = 32'h12345678;
 #100;

 B = 0;
 Z = 1;
 Reset = 1;
 B_addr = 32'hABCDEF01;
 #100;

 B = 1;
 Z = 1;
 Reset = 0;
 B_addr = 32'h87654321;
 #100;

 B = 0;
 Z = 0;
 Reset = 1;
 B_addr = 32'hFEDCBA09;

   // Add stimulus here
```

（4）仿真结果



九、结果分析：

通过本实验成功实现了控制器和取指电路，并进行了相应的测试。测试结果与理论分析一致，验证了设计的正确性。

十、实验结论：

本次实验中，使用 Verilog HDL 语言编写了控制器和取指电路的代码，并生成了综合后的 RTL 层电路。通过对电路进行仿真测试，验证了电路的功能与设计的一致性。实验结果表明所编写的代码具有正确性和可行性。

十一、总结及心得体会：

通过本次实验，深入学习了 Verilog HDL 语言的应用，并了解了控制器和取指电路的基本原理和设计方法。实践中，掌握了电路设计和仿真的基本技巧，并通过解决问题的过程提升了自己的实践能力。

十二、对本实验过程及方法、手段的改进建议：

建议在实验中加入更多基本器件的设计，可以根据要求自行选择设计相应的基本器件，以拓展对各器件原理和联系的理解，并学习取指电路的封装等相关知识。此外，可以进一步加强对 Verilog HDL 语言的学习，提高代码的质量和效率。

报告评分：

指导教师签字：

电 子 科 技 大 学

（实验三）实验报告

学生姓名：黄鑫　　学号：2021050901013　　指导教师：陈虹

实验地点：主楼 A2-411　　实验时间：2023.6.3

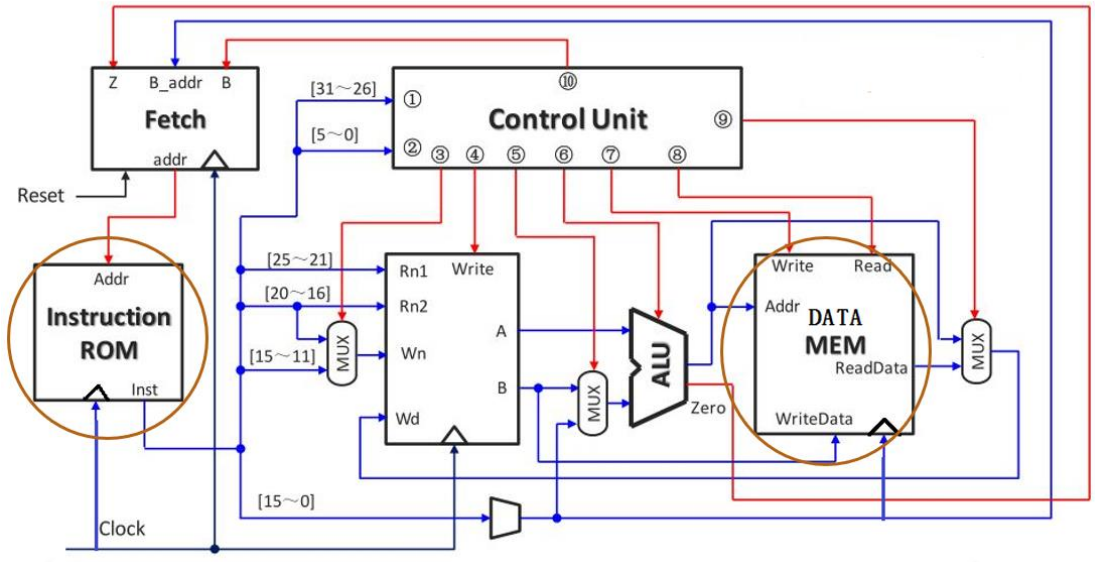
一、实验室名称：主楼 A2-411

二、实验项目名称：单周期 CPU 的设计与实现

三、实验学时：4 学时

四、实验原理：

1.单周期计算机结构图



2. Instruction ROM 指令要求

在指令存储器中赋值指令为：在数据存储器Data Mem的0、1、2号单元分别存放了三个32位操作数，完成0号和1号单元相加，与2号单元比较，如果相等，设置3号单元的值为和；否则设置为0。

```
Add $a1, $0, $0
Lw   $s1, 0($a1)
Lw   $s2, 4($a1)
Add $s1, $s1, $s2
Lw   $s2, 8($a1)
Beq $s1, $s2, 1
Sw   $0, 12($a1)
Sw   $s1, 12($a1)
```

3. 指令编码表

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	功能
add	000000	rs	rt	rd	00000	100000	寄存器加
sub	000000	rs	rt	rd	00000	100010	寄存器减
and	000000	rs	rt	rd	00000	100100	寄存器与
or	000000	rs	rt	rd	00000	100101	寄存器或
xor	000000	rs	rt	rd	00000	100110	寄存器异或

指令	[31:26]	[25:21]	[20:16]	[15:0]	功能
lw	100011	rs	rt	immediate	取字数据
sw	101011	rs	rt	immediate	存字数据
beq	000100	rs	rt	immediate	相等转移
lui	001111	00000	rt	immediate	设置高位

五、实验目的：

本实验的目的是让学生掌握单周期 CPU 的工作原理，以及控制器、运算器等部件设计的基本方法和技能。通过实验，学生将加深对所学知识的理解和掌握。

六、实验内容：

1.从单周期计算机的结构图中除去指令存储器（Instruction ROM）和数据存储器（DATA MEM），将剩余的电路封装成一个单周期的CPU 数据通路（Data_Flow）模块。

2.设计一个指令存储器，将以下指令存储在其中：在数据存储器（Data Mem）的 0、1、2 号单元分别存放了三个 32 位操作数，完成 0 号和 1 号单元的相加，并将结果与 2 号单元的值进行比较。如果相等，将结果存储到 3 号单元；否则将 3 号单元的值设置为 0。

3.设计一个数据存储器，由于需要保存并写入数据，所以应设置 32 个 reg 型变量。

七、实验器材（设备、元器件）：

电脑、Verilog 设计软件 ISE Design Suite 14.7

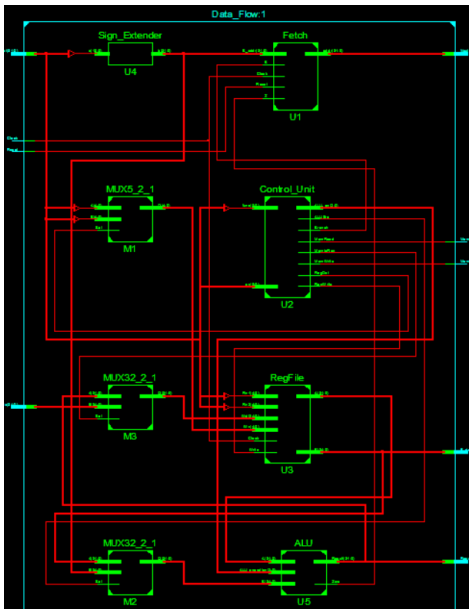
八、实验步骤：

1.Data_Flow 的设计

（1）模块代码

```
module Data_Flow(
    input Reset,
    input Clock,
    input [31:0] Inst,
    input [31:0] Data,
    output MemWrite,
    output MemRead,
    output [31:0] Result,
    output [31:0] B_data,
    output [31:0] NextPC
     );

    wire Z, B;
    wire [31:0] B_addr;

    Fetch U1 (.B(B),
              .Z(Z),
              .Reset(Reset),
              .Clock(Clock),
              .B_addr(B_addr),
              .addr(NextPC));

    wire RegDst,RegWrite,ALUSrc,ExtOp,MemtoReg;
    wire [2:0] ALU_op;
    Control_Unit U2 (.op(Inst[31:26]),
                     .func(Inst[5:0]),
                     .RegDst(RegDst),
                     .RegWrite(RegWrite),
                     .ALUSrc(ALUSrc),
                     .MemWrite(MemWrite),
                     .MemRead(MemRead),
                     .MemtoReg(MemtoReg),
                     .Branch(B),
                     .ALU_op(ALU_op));

    wire [4:0] Rn1,Rn2,Wn;
    wire [31:0] Wd,busA,busB;
    MUX5_2_1 M1 (.A(Inst[20:16]),
                 .B(Inst[15:11]),
                 .Sel(RegDst),
                 .O(Wn));

    RegFile U3 (.Rn1(Inst[25:21]),
                .Rn2(Inst[20:16]),
                .Wn(Wn),
                .Write(RegWrite),
                .Wd(Wd),
                .A(busA),
                .B(busB),
                .Clock(Clock));
    assign B_data = busB;

    wire [31:0] EX_imm,busB_or_imm;
    Sign_Extender U4 (.a(Inst[15:0]),
                      .b(EX_imm)
                      );
    assign B_addr = EX_imm;

    MUX32_2_1 M2(.A(busB),
                 .B(EX_imm),
                 .Sel(ALUSrc),
                 .O(busB_or_imm));

    wire [31:0] ALU_result;
    ALU U5 (.A(busA),
            .B(busB_or_imm),
            .ALU_operation(ALU_op),
            .Result(ALU_result),
            .Zero(Z));
    assign Result = ALU_result;

    MUX32_2_1 M3(.A(ALU_result),
                 .B(Data),
                 .Sel(MemtoReg),
                 .O(Wd));
```

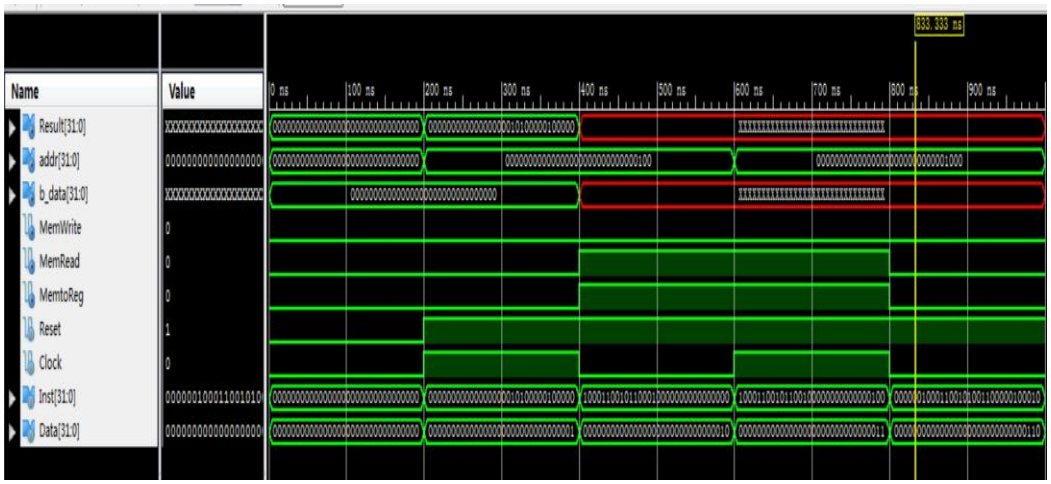
（2）RTL 层电路



（3）测试代码

```
    initial begin
    // 初始化输入信号
    Reset = 0;
    Clock = 0;
    Inst = 0;
    Data = 0;

    // 等待全局复位完成，持续100ns
    #200;
    Clock = ~Clock;
    Inst = 32'h00002820;
    Data = 32'h00000001;
    Reset = 1;
    #200;
    Clock = ~Clock;
    Inst = 32'h8CB10000;
    Data = 32'h00000002;
    Reset = 1;
    #200;
    Clock = ~Clock;
    Inst = 32'h8CB20004;
    Data = 32'h00000003;
    Reset = 1;
    #200;
    Clock = ~Clock;
    Inst = 32'h02329822;
    Data = 32'h00000006;
    Reset = 1;

    end

endmodule
```

（4）仿真结果



2.指令存储器的设计

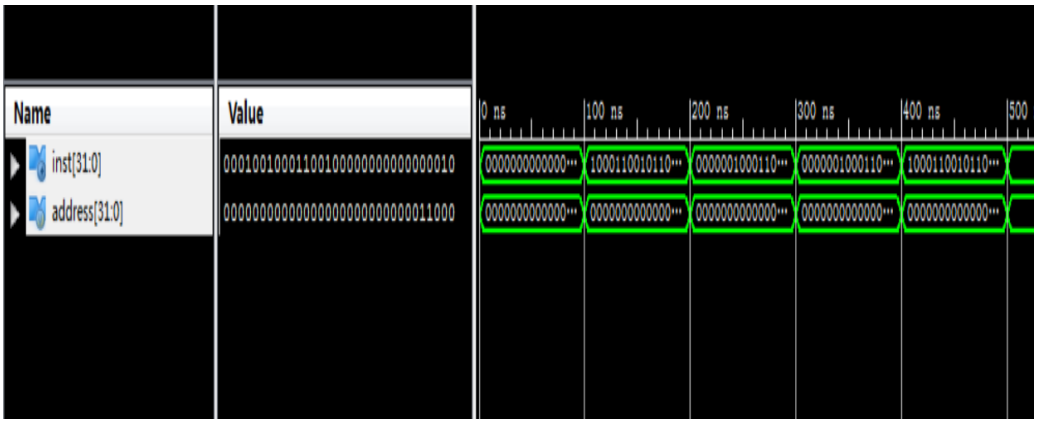
（1）模块代码

```
//////////////////////////////////////////////////////////////////////////////////
module Inst_mem(
    input  [31:0] address,
     output [31:0] inst
);

    reg [31:0] ram [0:31];

    initial begin
        ram[5'h00] = 32'h00002820; //add $a1, $0, $0
        ram[5'h01] = 32'h8CB10000; //lw  $s1, 0($a1)
        ram[5'h02] = 32'h8CB20004; //lw  $s2, 4($a1)
        ram[5'h03] = 32'h02329822; //sub $s3, $s1, $s2
        ram[5'h04] = 32'h02328830; //add $s1, $s1, $s2
        ram[5'h05] = 32'h8CB20008; //lw  $s2, 8($a1)
        ram[5'h06] = 32'h12320002; //beq $s1, $s2, 2
        ram[5'h07] = 32'hACB3000C; //sw  $s3, 12($a1)
        ram[5'h08] = 32'hACB1000C; //sw  $s1, 12($a1)
    end

    assign inst = ram[address[6:2]];

endmodule
```

（2）测试代码

```
initial begin
    // Initialize Inputs
    address = 0;

    // Wait 100 ns for global reset to finish


    #100; address = 32'h00000008;
    #100;address = 32'h00000000;
    #100; address = 32'h00000010;

    #100; address = 32'h00000014;

    #100; address = 32'h00000018;

end
```

（3）仿真结果



3. 数据存储器的设计

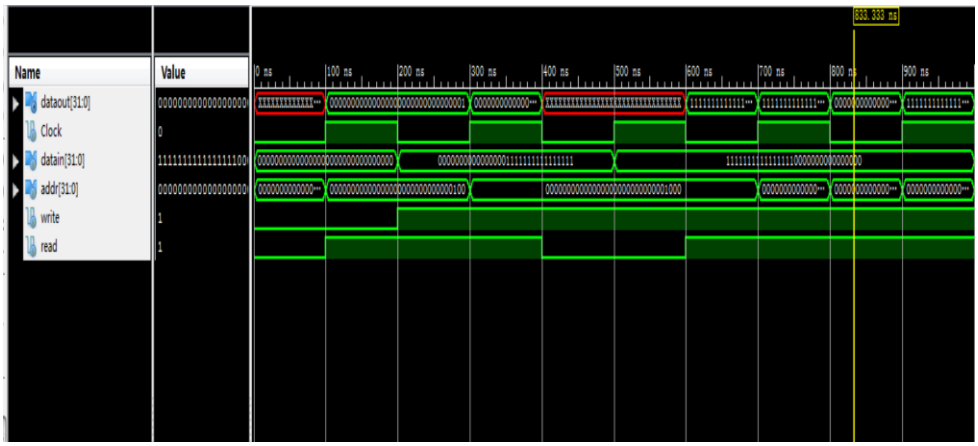
（1）模块代码

```
//
//////////////////////////////////////////////////////////////////////////////////
module DATA_RAM(
   input         Clock,
   output [31:0] dataout,
   input  [31:0] datain,
   input  [31:0] addr,
   input         write,
   input         read
);
   reg [31:0] ram[0:31];

   assign dataout = read ? ram[addr[6:2]] : 32'hxxxxxxxx;

   always @(posedge Clock) begin
      if (write)
         ram[addr[6:2]] <= datain;
   end
   integer i;
   initial begin

      for (i = 0; i <= 31; i = i + 1) begin
         ram[i] <= i * i;
      end
   end
endmodule
```

（2）测试代码

```
   ''
   initial begin
      // Initialize Inputs
      Clock = 0;
      datain = 0;
      addr = 0;
      write = 0;
      read = 0;

      // Wait 100 ns for global reset to finish
      #100; Clock = ~Clock; read = 1; addr = 32'h00000004;
      #100; Clock = ~Clock; write = 1; datain = 32'h0000ffff;
      #100; Clock = ~Clock; addr = 32'h00000008;
      #100; Clock = ~Clock; read = 0;
      #100; Clock = ~Clock; datain = 32'hffff0000;
      #100; Clock = ~Clock; read = 1;
      #100; Clock = ~Clock; addr = 32'h0000000C;

      #100; Clock = ~Clock; read = 1; addr = 32'h00000010;

      #100; Clock = ~Clock; read = 1; addr = 32'h00000014;

      #100; Clock = ~Clock; write = 1; datain = 32'h12345678;


   end

endmodule
```

（3）仿真结果



九、结果分析：

在本实验中，所设计的单周期 CPU 能够在一个时钟周期内完成一

条指令的执行。通过对三类共计 11 条指令的测试，我们验证了指令执行结果与预期结果的一致性。这表明我们的 CPU 数据通路模块和存储器设计是正确的，能够正确地执行各种指令。

十、实验结论：

通过本次实验，我们成功地封装了一个单周期的 CPU 数据通路模块，并设计了一个指令存储器和一个数据存储器，用于实现特定的功能。在实验过程中，我们验证了 CPU 的正常运行和指令执行的正确性。这为我们进一步研究和设计更复杂的计算机结构奠定了基础。

十一、总结及心得体会：

在本次实验中，我遇到了一些问题和挑战，例如在设计指令存储器时需要注意指令的格式和编码，确保指令能够正确地被读取和执行；同时，需要注意指令的顺序和执行时间，确保指令能够按照正确的顺序执行。

十二、对本实验过程及方法、手段的改进建议：

尽管本实验成功地实现了单周期 CPU 的设计，但单周期计算机具有一定的局限性。为了更深入地学习和掌握更高效的计算机结构，我建议在今后的实验中可以尝试设计和实现更复杂的多周期 CPU 或流水线 CPU。这样可以更好地理解和应用计算机体系结构中的各种优化技术，提高计算机的性能和效率。此外，还可以探索其他类型的存储器和高级指令集，以扩展实验的范围和挑战。

报告评分：

指导教师签字：

电 子 科 技 大 学

（实验四）实验报告

学生姓名：黄鑫　学号：2021050901013　　指导教师：陈虹

实验地点：主楼 A2-411　　实验时间：2023.6.10

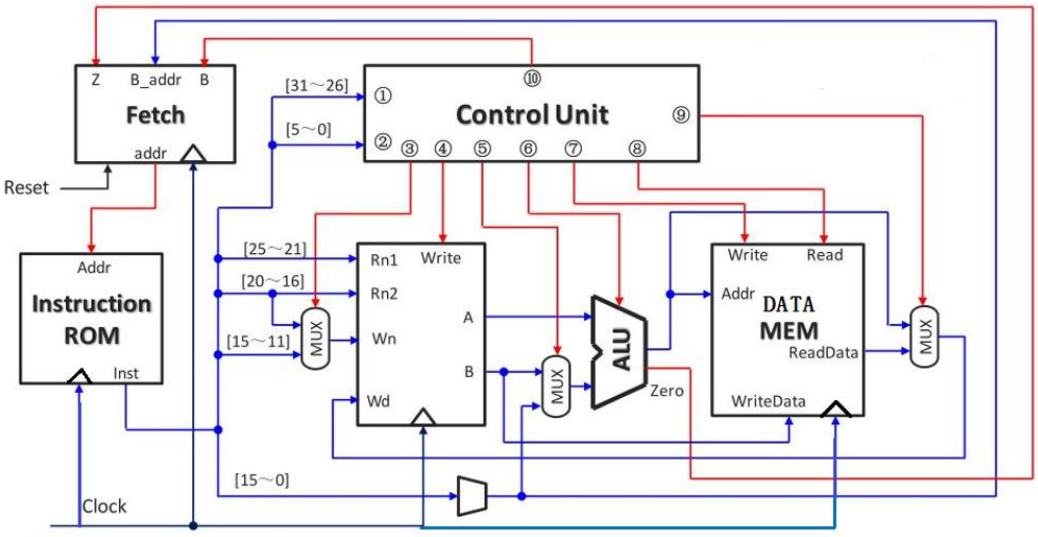
一、实验室名称：主楼 A2-411

二、实验项目名称：单周期计算机的设计与实现

三、实验学时：4 学时

四、实验原理：

单周期计算机的原理图如下。



五、实验目的：

本实验旨在通过实践，使学生掌握单周期 CPU 的工作原理，并学习控制器、运算器等部件设计的基本方法和技能。通过使用硬件描述语言 Verilog 和 ISE 工具软件进行软件设计与仿真，学生将培养分析

和设计 CPU 的能力，并加深对所学知识的理解和掌握。

六、实验内容：

使用指令存储器 Instruction ROM、数据存储器 DATA MEM 以及单周期的 CPU（MIPS_CPU）模块封装成一个单周期计算机。

模块的输入/输出信号如下：

输入信号：

1. Reset—复位信号
2. Clock—时钟信号

输出信号：

1. Inst — 指令码（32 位）
2. Pc — 指令寄存器的值（32 位）
3. Aluout — ALU 运算器输出（32 位）
4. B_data — 数据存储器 Data Mem 的数据输入端（32 位）

通过完成本实验，学生将深入了解单周期 CPU 的工作原理，掌握控制器和运算器等部件的设计方法。同时，学生将通过使用 Verilog 硬件描述语言和 ISE 工具软件进行软件设计和仿真，提高他们的分析和设计 CPU 的能力。

七、实验器材（设备、元器件）：

电脑、Verilog 设计软件 ISE Design Suite 14.7

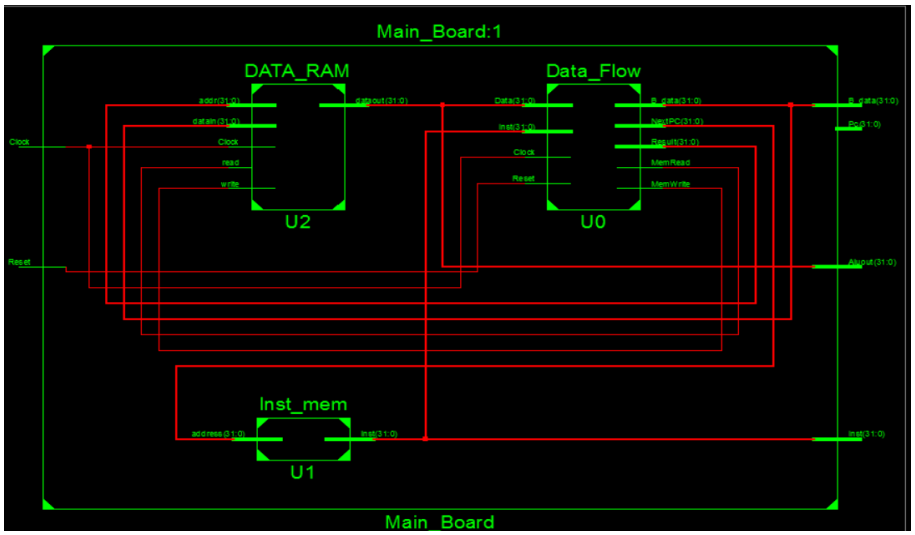
八、实验步骤：

1.单周期计算机的设计

（1）模块代码

```
module Main_Board(
   input Reset,Clock,
   output [31:0] Inst,
   output [31:0] Pc,
   output [31:0] Aluout,
   output [31:0] B_data
    );

   wire [31:0] addr_FTI;
   wire MemWrite,MemRead;
   wire [31:0] Result;
   wire [31:0] NextPC;

   Data_Flow U0(.Reset(Reset),
                .Clock(Clock),
                .Inst(Inst),
                .Data(Aluout),
                .MemWrite(MemWrite),
                .MemRead(MemRead),
                .Result(Result),
                .B_data(B_data),
                .NextPC(NextPC));

   Inst_mem U1(.address(NextPC),
               .inst(Inst));

   DATA_RAM U2(.Clock(Clock),
               .dataout(Aluout),
               .datain(B_data),
               .addr(Result),
               .write(MemWrite),
               .read(MemRead));

endmodule
```

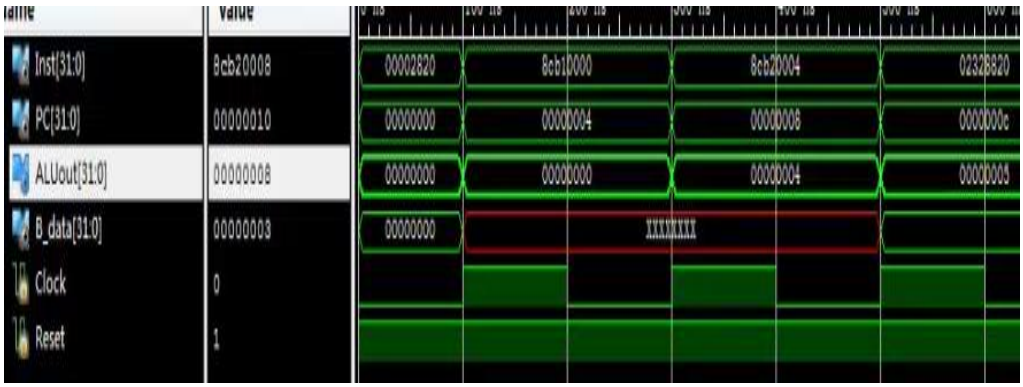
（2）RTL 层电路



（3）测试代码

```
  initial begin
     // Initialize Inputs
     Reset = 0;
     Clock = 0;

     // Wait 100 ns for global reset to finish
     #100;
     Reset = 1;
     Clock = 0;
     #100;
     Reset = 0;
     Clock = 1;
     #100;
     Reset = 0;
     Clock = 0;
     #100;
     Reset = 1;
     Clock = 1;

     // Add stimulus here

  end

endmodule
```

（4）仿真结果



九、结果分析：

通过对复位和时钟信号的控制，我们可以观察到指令寄存器中的指令被自动取出并执行相关操作。在设计的单周期 CPU 中，能够在一个时钟周期内完成相应指令的执行。经过测试，我们发现指令执行结果与预期结果是一致的，表明我们的 CPU 模块工作正常并能够正确执行指令。

十、实验结论：

通过本次实验，我们成功地封装了一个单周期计算机模块，其中包括指令存储器（Instruction ROM）、数据存储器（DATA MEM）以及单周期 CPU（MIPS_CPU）。我们使用 Reset 和 Clock 输入信号来控制计算机模块的复位和时钟信号，同时输出了指令码、指令寄存器的值、ALU 运算器输出以及数据存储器 Data Mem 的数据输入端。这些输出信号能够帮助我们进行计算机的调试和优化工作。

十一、总结及心得体会：

通过这次实验，我进一步理解了 CPU 的工作原理及实现机制，并模拟了单周期计算机自动取出指令和操作数进行工作的过程。实验过程中，我学会了使用 Verilog 硬件描述语言和 ISE 工具软件进行 CPU

模块的设计和仿真。通过观察输出信号的变化，我能够判断 CPU 的工作状态和指令执行结果的正确性。

然而，本次实验还存在一些不足之处。虽然我们在 ISE 平台上进行了仿真测试，但尚未将设计的单周期计算机模块下载到实际的开发板进行板级验证。这是下一步可以改进和拓展的方向。

十二、对本实验过程及方法、手段的改进建议：

为了进一步提升实验的可靠性和完整性，可以设计一些其他指令并进行测试，以更全面地验证 CPU 的功能和性能。此外，可以考虑将设计的单周期计算机模块下载到实际的开发板上，进行实际的硬件验证和测试。这样可以更好地验证设计的正确性，并发现和解决可能存在的硬件问题。

报告评分：

指导教师签字：