电子科技大学

计算机专业类课程

实验报告

课程名称：计算机组成原理综合实验

学　　院：计算机科学与工程学院

专　　业：网络空间安全

学生姓名：韩博宇

学　　号：2019040708023

指导教师：米源、吉家成

日　　期：　　2021 年　7 月　1 日

电 子 科 技 大 学

实 验 报 告

实验一

一、实验名称： CPU 基本器件的设计

二、实验学时： 4

三、实验内容和目的：

1.掌握用 Verlog 设计硬件电路的基本方法；

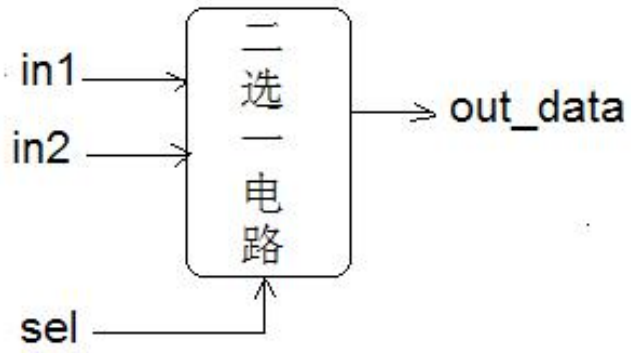
2.开发板的基本使用；

3.基本器件的设计；

- 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
- ALU 的设计
- 符号扩展器的设计

四、实验原理：

1.32 位 2 选 1 多路选择器



输入：两个 32 位数 in1，in2 和一个选择信号 sel

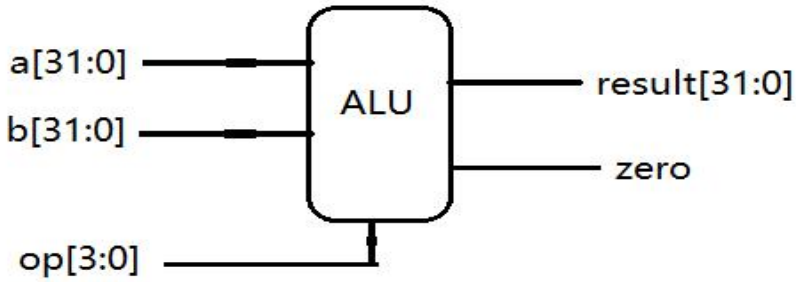
输出：根据选择信号输出 in1 与 in2 中的其中一个

2.符号扩展器

输入：一个 16 位的数据

输出：符号拓展后的 32 位数据

3.32 位 ALU



输入：数据：a[31:0], b[31:0]

操作码：op[3:0]（0000 加；0100 减；0001 与；0101 或；0010 异或；0110 置 b）

输出：运算结果 result[31:0]

零标志 zero（运算结果为 0 时置 1，否则为 0）

五、实验器材（设备、元器件）

ISE Design Suite 14.7

六、实验步骤：

1.32 位 2 选 1 多路选择器

```
module new1(in1,in2,sel,out_data
    );
input [31:0] in1,in2;//两个输入
input sel;//选择信号
output [31:0] out_data;

assign out_data=sel?in1:in2;//sel 为 0 选择 in1，sel 为 1 选择 in2

endmodule
```

2. 符号扩展器

```
module new2(data_16bit,data_32bit
    );
input [15:0] data_16bit;
output [31:0] data_32bit;

//wire [15:0] p;
reg [15:0] p;

always @(data_16bit)
begin
if(data_16bit[15]) p=16'hffff;//输出的高 16 位取决于输入的 16 比特数的最高位
else p=16'h0000;
end

assign data_32bit={p,data_16bit};

endmodule
```

3. 32 位 ALU

```
module new3(a,b,op,result,zero
    );
input [31:0] a,b;//输入两个数
input [3:0] op;//操作码
output [31:0] result;
output zero;//零标志位

reg [31:0] result;
reg zero;

always@(a,b,op)
begin
case(op)
4'b0000:result=a+b;//加
4'b0100:result=a-b;//减
4'b0001:result=a&b;//与
4'b0101:result=a|b;//或
4'b0010:result=a^b;//异或
4'b0110:result=b;//置 b
endcase
end

always@(a,b,op)
begin
if(!result) zero=1;//结果为 0 则置 1
```

```
else zero=0;
end

endmodule
```

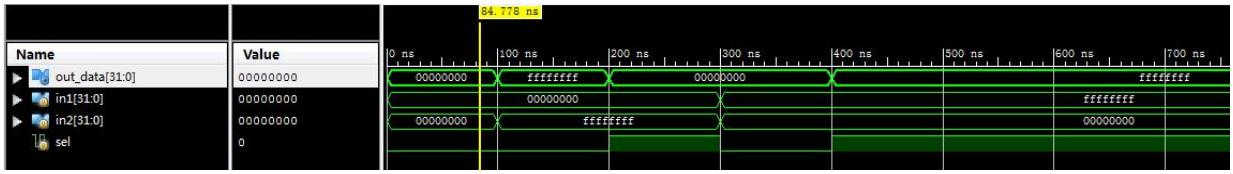
七、实验数据及结果分析：

1. 32 位 2 选 1 多路选择器

```
initial begin
                // Initialize Inputs
                in1 = 0;
                in2 = 0;
                sel = 0;

                // Wait 100 ns for global reset to finish
                #100; in1=32'h00000000;in2=32'hffffffff;sel=0;
                #100; in1=32'h00000000;in2=32'hffffffff;sel=1;
                #100; in1=32'hffffffff;in2=32'h00000000;sel=0;
                #100; in1=32'hffffffff;in2=32'h00000000;sel=1;

                // Add stimulus here

        end
```

预期结果：sel 为 0 时选择 in1，sel 为 1 时选择 in2

测试结果：



由测试结果可知，当 sel 为 0 时，out_data 分别为 00000000 和 ffffffff，与 in1 相同；当 sel 为 1 时，out_data 分别为 ffffffff 和 00000000，与 in2 相同。满足预期结果，测试正确。

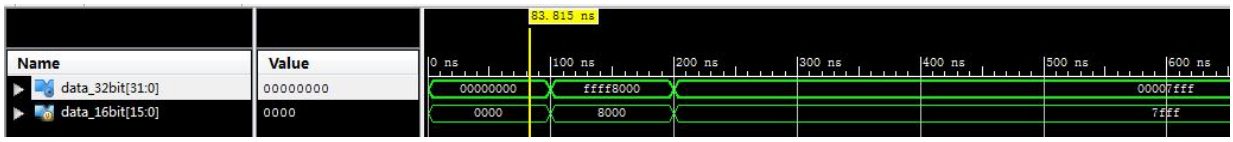
2. 符号扩展器

```
initial begin
                // Initialize Inputs
                data_16bit = 0;

                // Wait 100 ns for global reset to finish
```

```
        #100; data_16bit=16'h8000;
        #100; data_16bit=16'h7fff;

        // Add stimulus here

    end
```

预期结果：当 data_16bit 最高位为 1 时，data_32bit 高 16 位为全 1；当 data_16bit 最高位为 0 时，data_32bit 高 16 位为全 0

测试结果：



由测试结果可知，data_16bit 为 8000 时，data_32bit 为 ffff8000；data_16bit 为 7fff 时，data_32bit 为 00007fff。满足预期结果，测试正确。

3. 32 位 ALU

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

        // Wait 100 ns for global reset to finish
        #100;a=32'h8fffffff;b=32'h70000000;op=4'b0000;
        #100;a=32'hff000000;b=32'hff000000;op=4'b0100;
        #100;a=32'hfff00000;b=32'h000fffff;op=4'b0001;
        #100;a=32'hfff00000;b=32'h000fffff;op=4'b0101;
        #100;a=32'hffffffff;b=32'hf0f0f0f0;op=4'b0010;
        #100;a=32'hffffffff;b=32'h00000000;op=4'b0110;

        // Add stimulus here

    end
```

预期结果：op 为 0000 时执行加操作（结果为 ffffffff），op 为 0100 时执行减操作（结果为 00000000），op 为 0001 时执行与操作（结果为 00000000），op 为 0101 时执行或操作（结果为 ffffffff），op 为 0010 时执行异或操作（结果为 0f0f0f0f），op 为 0110 时执行置 b 操作（结果为 00000000）

测试结果：

由测试结果可知，result 分别为 ffffffff、00000000、00000000、ffffffff、0f0f0f0f、00000000，zero 分别为 0、1、1、0、0、1，成功实现加、减、与、或、异或、置 b 操作。满足预期结果，测试正确。

八、实验结论、心得体会和改进建议：

我未修过数电实验这门课，对 Verilog 的学习只停留在书本，从未实践。通过这门课，我实践编程，让我对 Verilog 语法有了更加深入的认识，使我能够运用软件对程序进行调试，修复 bug，实现简单的功能。

此外，通过这次课，让我对计算机组成原理课堂上学到的知识有了深刻的理解，让我了解了 CPU 的模块化思想，受益匪浅。

希望老师以后能针对每个任务给出一系列标准测试范例，方便代码完成后的检查工作。

电 子 科 技 大 学

实 验 报 告

实验二

一、实验名称：取指电路的设计

二、实验学时：4

三、实验内容和目的：

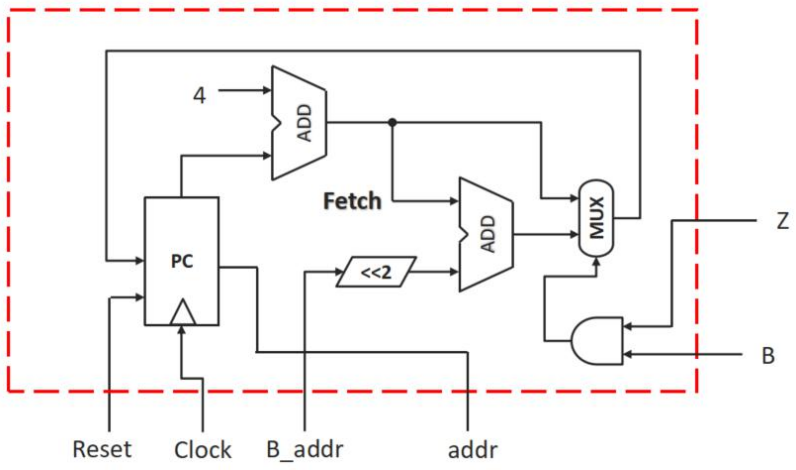
1. 设计并实现取指电路结构中各基本功能部件；
2. 根据取指电路结构图，使用 Verilog HDl 实例化语法实现取指电路功能；
3. 设计实验，验证取指电路在跳转/非跳转情况下可以正常工作。

四、实验原理：

输入端：Reset、Clock、B_addr、Z、B

- Reset 为高电平有效，将指令地址置为 0
- Clock 为时钟信号，上升沿触发
- B_addr 为符号扩展后的立即数地址
- Z 为 ALU 的零标志，用来判断 beq 指令的跳转
- B 为来自 control_unit 的 Branch 标志

输出端：addr



内部设计如上图所示，具体过程为：PC 每次加 4，B_addr 左移两位，并与 PC 自增后的结果相加，送入 MUX 的 B 输入端，PC 自增后的结果送入 MUX 的 A 输入端，根据 B 和 Z 进行与运算后的结果进行选路。选择的结果回到 PC，在 Clock 时钟信号的作用下，输出 addr。如果 Reset 为 1（高电平有效），则 addr 直接置 0。

五、实验器材（设备、元器件）

ISE Design Suite 14.7

六、实验步骤：

取指电路分为加法器、左移部件、多路选择器三部分。

加法器：

```
module ADD32(A,B,C
    );
    input[31:0] A,B;
         output[31:0] C;

         assign C=A+B;

endmodule
```

左移部件：

```
module Left_2_Shifter(d,o
    );
    input[31:0] d;
         output[31:0] o;

    assign o={d[29:0],2'b00};//左移两位

endmodule
```

多路选择器：

```
module MUX32_2_1(A,B,Sel,O
    );

    input[31:0] A,B;
         input Sel;
         output[31:0] O;
```

```
    assign O=Sel?B:A;

endmodule
```

将加法器、左移部件、多路选择器整合到一起，代码如下：

```
module Fetch(B,Z,Reset,Clock,B_addr,addr
    );
          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=Z&B;

          Left_2_Shifter U0(B_addr,B_addr1);

    ADD32 U1(PC,4,sum0);  //PC+4
    ADD32 U2(sum0,B_addr1,sum1); //PC+4+ B_addr
    MUX32_2_1 M1(sum0,sum1,sel,next_pc);  //选路

    assign addr=PC;

    always @(posedge Clock )
          begin
             if (Reset==1) PC=0;
                   else PC = next_pc;
          end

          initial PC=0;

endmodule
```

七、实验数据及结果分析：

测试代码如下：

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

                // Wait 100 ns for global reset to finish
```

```
        #100;Reset=1;Clock=1;B_addr=32'h00000000;Z=0;B=0;
        #100;Reset=1;Clock=0;B_addr=32'h00000000;Z=0;B=0;
        #100;Reset=0;Clock=1;B_addr=32'h00000008;Z=0;B=0;
        #100;Reset=0;Clock=0;B_addr=32'h00000008;Z=0;B=0;
        #100;Reset=0;Clock=1;B_addr=32'h00000008;Z=1;B=1;
        #100;Reset=0;Clock=0;B_addr=32'h00000008;Z=1;B=1;

        // Add stimulus here

    end
```

由测试结果可知，Reset=1 时，PC 置 0，此时 addr=32’h00000000；Z=B=0,顺序取指，addr=32’h00000004；Z=B=1,执行跳转，addr=32’h00000028。满足预期结果，测试正确。

八、实验结论、心得体会和改进建议：

通过这次课，我对 Verilog 语法掌握更加深入了，在编程的过程中，遇到了一些错误，通过查阅资料，顺利的解决，很有成就感。

此外，这节课让我对取指电路的自增、跳转有了更为直观的理解，对其内部实现也有了深入地认识。

本次实验测试时由于时钟信号设置错误，导致我误以为前面 Verilog 编程出现错误。希望老师以后在讲解阶段可以讲一下测试时时钟信号该如何设置。

电 子 科 技 大 学

实 验 报 告

实验三

一、实验名称： 控制部件的设计

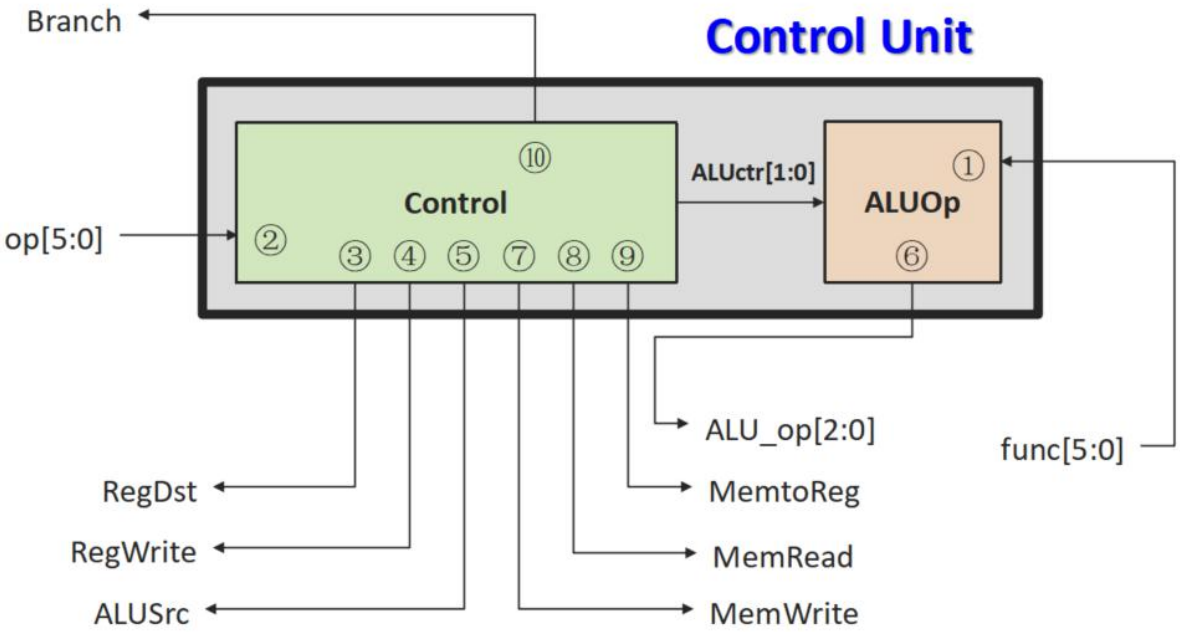
二、实验学时： 4

三、实验内容和目的：

1. 使用 case 语句设计并实现控制单元 Control_Unit。
2. 设计实验，验证控制单元可以根据指令给出相应控制信号（9 种指令均需要验证）。

四、实验原理：

控制部件主要负责指令译码，产生控制信号，控制 CPU 其它部件如 ALU、程序计数器等的工作。控制器包括主控模块和 ALU 控制子模块，其原理图如下：



各指令对应的 Control_Unit 输出如下：

Input		Ouppput							
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

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

其中，Op 和 func 确定指令功能（如 Op 全 0 为 R 型指令，通过 func 确定具 体的某一种指令）。Overflow 和 Zero 在某些指令中作为判断条件。控制器输出信号繁多，包括对取值电路、ALU、寄存器和存储器等的控制，在此不一一赘述。

五、实验器材（设备、元器件）

ISE Design Suite 14.7

六、实验步骤：

Control_Unit 由两个部件组成：Control 部件和 ALUop 部件。Control 部件的输入为 op

操作码，输出为各标志信号和传递给 ALUop 部件的 ALUctr 标志。ALUop 部件的输入为 func 字段和来自 Control 部件的 ALU_ctr，输出为 ALU_op。

Control 部件：

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

        reg RegDst,RegWrite,ALUSrc,MemWrite,MemRead,MemtoReg,Branch;
        reg[1:0] ALUctr;
        always @ (op)//根据 op 的值，确定各标志信号的取值，表中 x 随意取值
        begin
                case (op)
                //RT
                6'b000000:
                begin
                     RegDst=1;RegWrite=1;ALUSrc=0;MemWrite=0;
                    MemRead=0;MemtoReg=0;Branch=0;ALUctr[0]=0;ALUctr[1]=1;
                end
                //lw
                6'b100011:
                begin
                     RegDst=0;RegWrite=1;ALUSrc=1;MemWrite=0;
                    MemRead=1;MemtoReg=1;Branch=0;ALUctr[0]=0;ALUctr[1]=0;
                end
                //sw
                6'b101011:
                begin
                     RegDst=0;RegWrite=0;ALUSrc=1;MemWrite=1;
                    MemRead=0;MemtoReg=0;Branch=0;ALUctr[0]=0;ALUctr[1]=0;
                end
                //beq
                6'b000100:
                begin
                     RegDst=0;RegWrite=0;ALUSrc=0;MemWrite=0;
                    MemRead=0;MemtoReg=0;Branch=1;ALUctr[0]=1;ALUctr[1]=0;
                end
                //lui
                6'b001111:
                begin
                     RegDst=0;RegWrite=1;ALUSrc=1;MemWrite=0;
                    MemRead=0;MemtoReg=0;Branch=0;ALUctr[0]=1;ALUctr[1]=1;
                end
                default://其他情况
```

```
                begin
                    RegDst=0;RegWrite=0;ALUSrc=0;MemWrite=0;
                    MemRead=0;MemtoReg=0;Branch=0;ALUctr[0]=0;ALUctr[1]=0;
                end
                endcase
        end
endmodule
```

ALUop 部件：

```
module ALUop(func,ALUctr,ALU_op
    );
        input[5:0] func;
        input[1:0] ALUctr;
        output[2:0] ALU_op;

        reg[2:0] ALU_op;
        always @ (ALUctr or func)
        begin
        if(ALUctr==2'b10)
        begin
                case (func)
                        //Add
                        6'b100000:
                                begin
                                        ALU_op[2]=0;ALU_op[1]=0;ALU_op[0]=0;
                                end
                        //Sub
                        6'b100010:
                                begin
                                        ALU_op[2]=1;ALU_op[1]=0;ALU_op[0]=0;
                                end
                        //And
                        6'b100100:
                                begin
                                        ALU_op[2]=0;ALU_op[1]=0;ALU_op[0]=1;
                                end
                        //Or
                        6'b100101:
                                begin
                                        ALU_op[2]=1;ALU_op[1]=0;ALU_op[0]=1;
                                end
                        //Xor
                        6'b100110:
```

```
                                begin
                                        ALU_op[2]=0;ALU_op[1]=1;ALU_op[0]=0;
                                end
                        default:
                                begin
                                        ALU_op[2]=1;ALU_op[1]=1;ALU_op[0]=1;
                                end
                endcase
        end
        else
        begin
                case (ALUctr)
                2'b00:
                        begin
                                ALU_op[2]=0;ALU_op[1]=0;ALU_op[0]=0;
                        end
                2'b01:
                        begin
                                ALU_op[2]=1;ALU_op[1]=0;ALU_op[0]=0;
                        end
                2'b11:
                        begin
                                ALU_op[2]=1;ALU_op[1]=1;ALU_op[0]=0;
                        end
                default:
                        begin
                                ALU_op[2]=0;ALU_op[1]=0;ALU_op[0]=0;
                        end
                endcase
        end
        end

endmodule
```

将 Control 部件和 ALUop 部件整合到一起：

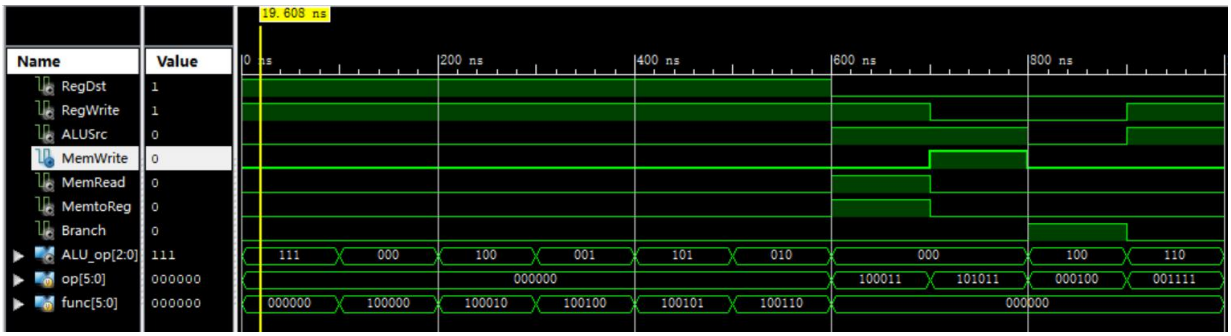
```
module
Control_Unit(op,func,RegDst,RegWrite,ALUSrc,MemWrite,MemRead,MemtoReg,Branch,ALU_op
    );
        input[5:0] func,op;
        output[2:0] ALU_op;
        output RegDst,RegWrite,ALUSrc,MemWrite,MemRead,MemtoReg,Branch;

        wire [1:0] ALUctr;

//实例化 Control 模块
   Control U0( .op(op),
```

```
                    .RegDst(RegDst) ,
                     .RegWrite(RegWrite) ,
                     .ALUSrc(ALUSrc) ,
                    .MemWrite(MemWrite),
                     .MemRead(MemRead) ,
                     .MemtoReg(MemtoReg) ,
                     .Branch(Branch) ,
                     .ALUctr(ALUctr));

//实例化 ALUop 模块
        ALUop U1(.func(func),.ALUctr(ALUctr),.ALU_op(ALU_op));

endmodule
```

七、实验数据及结果分析：

下面对 9 种不同的指令进行测试，代码如下：

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

                // Wait 100 ns for global reset to finish
                #100;op=6'b000000;func=6'b100000;//Add
                #100;op=6'b000000;func=6'b100010;//Sub
                #100;op=6'b000000;func=6'b100100;//And
                #100;op=6'b000000;func=6'b100101;//Or
                #100;op=6'b000000;func=6'b100110;//Xor
                #100;op=6'b100011;func=6'b000000;//lw
                #100;op=6'b101011;func=6'b000000;//sw
                #100;op=6'b000100;func=6'b000000;//beq
                #100;op=6'b001111;func=6'b000000;//lui

                // Add stimulus here

        end
```



由测试结果可知，5 条 R-type 型指令的 op 相同（000000），各标志信号的取值也相同，但 func 不同，输出的 ALU_op 不同，分别为 000、100、001、101、010。满足预期结果，测试正确。

lw 指令和 sw 指令的 ALU_op 相同（000），而 op 不同（lw 为 100011，sw 为 101011）;beq 指令的 ALU_op 为 100，op 为 000100；lui 指令的 ALU_op 为 110，op 为 001111。满足预期结果，测试正确。

八、实验结论、心得体会和改进建议：

通过这次课，我对 Verilog 中 case 语句的掌握更加深入了，能够独立调试程序，顺利解决遇到的问题，收获很大。

此外，我更加全面的认识了 CPU 中控制部件的原理，能够实现根据输入的不同指令，进行译码操作，生成不同的标志信号。

电 子 科 技 大 学

实 验 报 告

实验四

一、实验名称：单周期 CPU 的封装

二、实验学时：4

三、实验内容和目的：

1. 将单周期 CPU 结构中的各部件进行封装，构成一个完整的单周期 CPU。

   端口信号要求：

   ```
   input  Clock, Reset;
   output  [31:0]  addr, result;
   ```

   //[31:0] addr：指令地址；

   //[31:0] result：ALU 运算结果；

2. 设计一个指令序列，将其写入指令存储器，运行单周期 CPU 进行仿真分析：

   a）使用 lui 指令将操作数据预先写入寄存器组；

   b）设计出的指令序列需要满足以下要求：

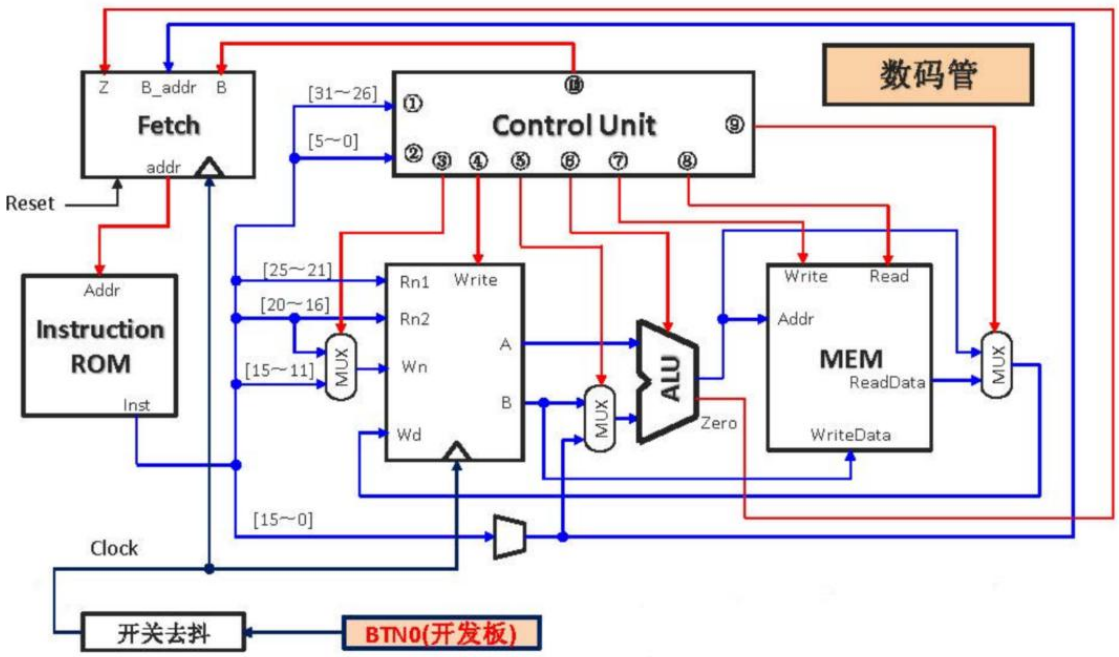
   需要包含指令集中每种类型的指令（R 型，存储器操作，跳转指令）。

   指令序列中的跳转指令要涵盖跳转成功、跳转失败两种情况。

   c）对指令序列执行结果进行演算，与仿真波形对比，分析验证实验结果。

四、实验原理：

冯诺依曼体系计算机的组成包括运算器、控制器、存储器、输入设备、输出设备，其中运算器和控制器合称中央处理器（CPU）。将单周期 CPU 与 DATA_MEM 和 IO 设备相连，即可组成一个单周期计算机，结构图如下：



将实验 1-3 做好的 Fetch（取指部件）、control_unit（控制单元）、ALU、Registerfiles、符号扩展器和老师提供的 Instruction ROM（指令存储器）、Data MEM（数据存储器）组装起来，就形成了一个简单的单周期 CPU。

输入：时钟信号 Clock 和重置信号 reset

输出：addr（指令地址）和 result（ALU 运算结果）

五、实验器材（设备、元器件）

ISE Design Suite 14.7

六、实验步骤：

将

1.多路选择器模块

① 5 位多路选择器：

```
module MUX5_2_1(A,B,Sel,O
    );

    input[4:0] A,B;
         input Sel;
         output[4:0] O;

    assign O=Sel?B:A;
```

```
endmodule
```

② 32 位多路选择器：

```
module MUX32_2_1(A,B,Sel,O
    );

    input[31:0] A,B;
         input Sel;
         output[31:0] O;

    assign O=Sel?B:A;

endmodule
```

2. 符号扩展器模块

```
module Sign_Extender(a,b
    );
         input[15:0] a;
    output[31:0] b;

    assign b={a[15]?16'hffff:16'h0,a};
endmodule
```

3. 取指电路模块

① 加法器：

```
module ADD32(A,B,C
    );
    input[31:0] A,B;
         output[31:0] C;

         assign C=A+B;

endmodule
```

② 左移部件：

```
module Left_2_Shifter(d,o
    );
    input[31:0] d;
         output[31:0] o;
```

```
    assign o={d[29:0],2'b00};//左移两位

endmodule
```

③ 多路选择器：

```
module MUX32_2_1(A,B,Sel,O
    );

    input[31:0] A,B;
         input Sel;
         output[31:0] O;

    assign O=Sel?B:A;

endmodule
```

④ 将加法器、左移部件、多路选择器整合到一起：

```
module Fetch(B,Z,Reset,Clock,B_addr,addr
    );
         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=Z&B;

         Left_2_Shifter U0(B_addr,B_addr1);

    ADD32 U1(PC,4,sum0);  //PC+4
    ADD32 U2(sum0,B_addr1,sum1); //PC+4+ B_addr
    MUX32_2_1 M1(sum0,sum1,sel,next_pc);  //选路

    assign addr=PC;

    always @(posedge Clock )
         begin
            if (Reset==1) PC=0;
                 else PC = next_pc;
         end

         initial PC=0;

endmodule
```

4. 控制部件模块

①Control 部件：

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

        reg RegDst,RegWrite,ALUSrc,MemWrite,MemRead,MemtoReg,Branch;
        reg[1:0] ALUctr;
        always @ (op)//根据 op 的值，确定各标志信号的取值，表中 x 随意取值
        begin
                case (op)
                //RT
                6'b000000:
                begin
                     RegDst=1;RegWrite=1;ALUSrc=0;MemWrite=0;
                    MemRead=0;MemtoReg=0;Branch=0;ALUctr[0]=0;ALUctr[1]=1;
                end
                //lw
                6'b100011:
                begin
                     RegDst=0;RegWrite=1;ALUSrc=1;MemWrite=0;
                    MemRead=1;MemtoReg=1;Branch=0;ALUctr[0]=0;ALUctr[1]=0;
                end
                //sw
                6'b101011:
                begin
                     RegDst=0;RegWrite=0;ALUSrc=1;MemWrite=1;
                    MemRead=0;MemtoReg=0;Branch=0;ALUctr[0]=0;ALUctr[1]=0;
                end
                //beq
                6'b000100:
                begin
                     RegDst=0;RegWrite=0;ALUSrc=0;MemWrite=0;
                    MemRead=0;MemtoReg=0;Branch=1;ALUctr[0]=1;ALUctr[1]=0;
                end
                //lui
                6'b001111:
                begin
                     RegDst=0;RegWrite=1;ALUSrc=1;MemWrite=0;
                    MemRead=0;MemtoReg=0;Branch=0;ALUctr[0]=1;ALUctr[1]=1;
                end
```

```
                default://其他情况
                begin
                     RegDst=0;RegWrite=0;ALUSrc=0;MemWrite=0;
                    MemRead=0;MemtoReg=0;Branch=0;ALUctr[0]=0;ALUctr[1]=0;
                end
                endcase
        end
endmodule
```

② ALUop 部件：

```
module ALUop(func,ALUctr,ALU_op
    );
        input[5:0] func;
        input[1:0] ALUctr;
        output[2:0] ALU_op;

        reg[2:0] ALU_op;
        always @ (ALUctr or func)
        begin
        if(ALUctr==2'b10)
        begin
                case (func)
                        //Add
                        6'b100000:
                                begin
                                        ALU_op[2]=0;ALU_op[1]=0;ALU_op[0]=0;
                                end
                        //Sub
                        6'b100010:
                                begin
                                        ALU_op[2]=1;ALU_op[1]=0;ALU_op[0]=0;
                                end
                        //And
                        6'b100100:
                                begin
                                        ALU_op[2]=0;ALU_op[1]=0;ALU_op[0]=1;
                                end
                        //Or
                        6'b100101:
                                begin
                                        ALU_op[2]=1;ALU_op[1]=0;ALU_op[0]=1;
                                end
                        //Xor
                        6'b100110:
                                begin
                                        ALU_op[2]=0;ALU_op[1]=1;ALU_op[0]=0;
```

```
                        end
                default:
                        begin
                                ALU_op[2]=1;ALU_op[1]=1;ALU_op[0]=1;
                        end
        endcase
    end
    else
    begin
        case (ALUctr)
        2'b00:
                begin
                        ALU_op[2]=0;ALU_op[1]=0;ALU_op[0]=0;
                end
        2'b01:
                begin
                        ALU_op[2]=1;ALU_op[1]=0;ALU_op[0]=0;
                end
        2'b11:
                begin
                        ALU_op[2]=1;ALU_op[1]=1;ALU_op[0]=0;
                end
        default:
                begin
                        ALU_op[2]=0;ALU_op[1]=0;ALU_op[0]=0;
                end
        endcase
    end
    end

endmodule
```

③ 将 Control 部件和 ALUop 部件整合到一起：

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

    wire [1:0] ALUctr;

//实例化 Control 模块
```

```
   Control U0( .op(op),
                   .RegDst(RegDst) ,
                    .RegWrite(RegWrite) ,
                    .ALUSrc(ALUSrc) ,
                   .MemWrite(MemWrite),
                    .MemRead(MemRead) ,
                    .MemtoReg(MemtoReg) ,
                    .Branch(Branch) ,
                    .ALUctr(ALUctr));

//实例化 ALUop 模块
        ALUop U1(.func(func),.ALUctr(ALUctr),.ALU_op(ALU_op));

endmodule
```

5. 存储器模块

```
module DATA_RAM(
                input        Clock,
                output[31:0] dataout,
                input [31:0] datain,
                input [31:0] addr,
                input        write , 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) ram [i] = i * i;
        end

endmodule
```

6. 指令寄存器模块

```
module INST_ROM(
         input [31:0] addr,
```

```
        output [31:0] Inst
    );

        wire [31:0] ram [31:0];

        assign ram[5'h00]=0;                                    //
        assign ram[5'h01]=32'h3c011234;    //lui R1,0x1234
        assign ram[5'h02]=32'h3c025678;    //lui R2,0x5678
        assign ram[5'h03]=32'h00221820;    //add R3,R1,R2
        assign ram[5'h04]=32'h00221822;    //sub R3,R1,R2
        assign ram[5'h05]=32'h00221824;    //and R3,R1,R2
        assign ram[5'h06]=32'h00221825;    //or R3,R1,R2
        assign ram[5'h07]=32'h00221826;    //xor R3,R1,R2
        assign ram[5'h08]=32'h00631826;    //xor R3,R3,R3
        assign ram[5'h09]=32'hac610001;    //sw R1,1(R3)
        assign ram[5'h0a]=32'h8c640001;    //lw R4,1(R3)
        assign ram[5'h0b]=32'h10220000;    //beq R1,R2,0
        assign ram[5'h0c]=32'h1021fffb;    //beq R1,R1,0xfffb

        assign Inst=ram[addr[6:2]];

endmodule
```

7. RegFile 模块

```
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];

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

        //Write data

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

```
        end

endmodule
```

8. ALU 模块

```
module ALU(A,B,ALU_operation,Result,Zero
    );

        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
```

9. CPU 封装

```
module MIPS_CPU(Clock,Reset,addr,result
    );
        input Clock,Reset;//时钟信号和置位
        output [31:0] addr,result;//题目要求的输出

        wire [31:0] inst,muxout3,Aout,Bout,Extnum,muxout2,dataout,Wd;
        wire [2:0] ALU_op;
        wire [4:0] muxout1;
        wire RegDst,RegWrite,ALUSrc,MemWrite,MemRead,MemtoReg,Branch,Z;

        //实例化
        INST_ROM inst_rom(
                .addr(addr),
                .Inst(inst)
        );

        Fetch fetch(
```

```
        .option(inst[31:26]),
        .func(inst[5:0]),
        .regDst(RegDst),
        .regWrite(RegWrite),
        .ALUSrc(ALUSrc),
        .MemWrite(MemWrite),
        .MemRead(MemRead),
        .MemtoReg(MemtoReg),
        .Branch(Branch),
        .ALU_op(ALU_op)
);

MUX_5_2_1 regdst(
        .a(inst[20:16]),
        .b(inst[15:11]),
        .sign(RegDst),
        .c(muxout1)
);

RegFile regfiles(
        .Rn1(inst[25:21]),
        .Rn2(inst[20:16]),
        .Wn(muxout1),
        .Wd(muxout3),
        .Write(RegWrite),
        .A(Aout),
        .B(Bout),
        .Clock(Clock)
);

Sign_Extender signextend(
        .immediate(inst[15:0]),
        .result(Extnum)
);

MUX_32_2_1 ALUmemchoose(
        .a(result),
        .b(dataout),
        .sign(MemtoReg),
        .c(muxout3)
);

Control_Unit control(
        .B_addr(Extnum),
```

```
                .Z(Z),
                .B(Branch),
                .addr(addr),
                .reset(Reset),
                .Clock(Clock)
        );

        ALU alu(
                .rega(Aout),
                .regb(muxout2),
                .ALUopcode(ALU_op),
                .result(result),
                .zero(zero)
        );

        DATA_RAM data_ram(
                .Clock(Clock),
                .dataout(dataout),
                .datain(Bout),
                .addr(result),
                .write(MemWrite),
                .read(MemRead)
        );

endmodule
```

七、实验数据及结果分析：

测试代码如下：

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

                // Wait 100 ns for global reset to finish
                #100;Clock = ~Clock; Reset = 1;// Reset 高电平有效
                #100;Clock = ~Clock; Reset = 1;
                #100;Clock = ~Clock; Reset = 0;
                #100;Clock = ~Clock;
                #100;Clock = ~Clock;
                //均为“#100;Clock = ~Clock;”，已省略
                #100;Clock = ~Clock;
                #100;Clock = ~Clock;

                // Add stimulus here
```

```
    end
```

结果分析：

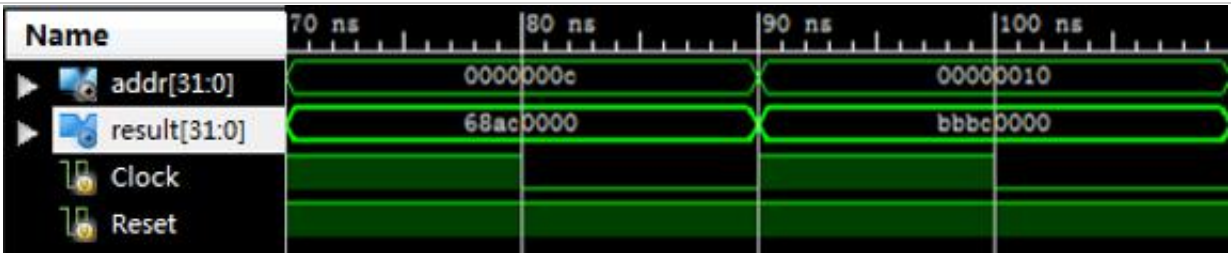
① lui 语句：

```
    assign ram[5'h01]=32'h3c011234;    //lui R1,0x1234
    assign ram[5'h02]=32'h3c025678;    //lui R2,0x5678
```

该指令不转跳，PC 依次加 4（00000004H 和 00000008H），Result 输出是 12340000H 和 56780000H。满足预期结果，测试正确。

② add 和 sub 语句：

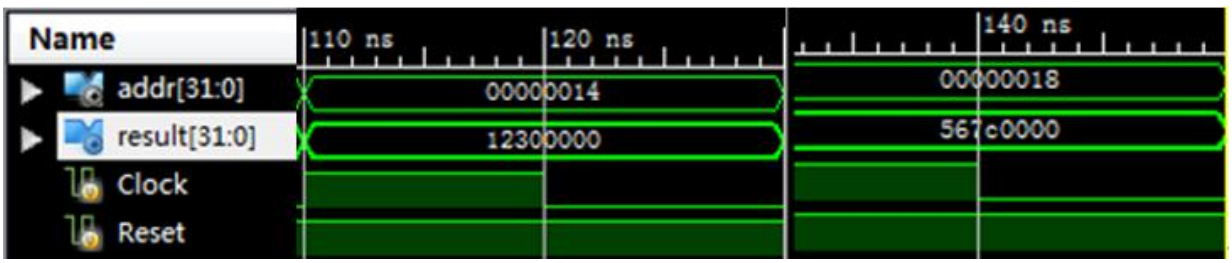
```
    assign ram[5'h03]=32'h00221820;    //add R3,R1,R2
    assign ram[5'h04]=32'h00221822;    //sub R3,R1,R2
```



该指令不转跳，PC 依次加 4（0000000cH 和 00000010H），result 是 12340000H+56780000H=68AC0000H 和 12340000H-56780000H=BBBC0000H。满足预期结果，测试正确。

③ and 和 or 语句：

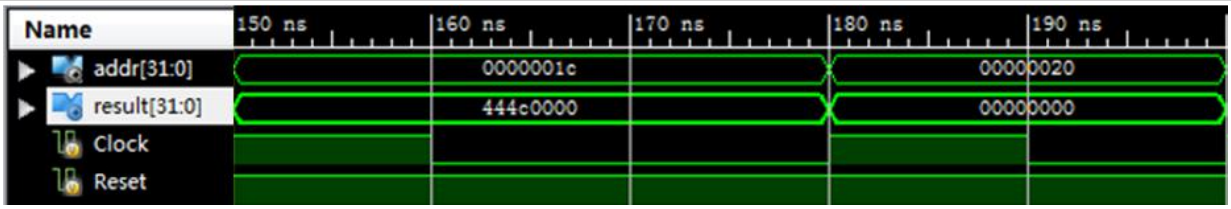
```
    assign ram[5'h05]=32'h00221824;    //and R3,R1,R2
    assign ram[5'h06]=32'h00221825;    //or R3,R1,R2
```



该指令不转跳，PC 依次加 4（00000014H 和 00000018H），result 是 12340000H and 56780000H=12300000H 和 12340000H or 56780000H=567C0000H。满足预期结果，测试正确。

④ xor 语句：

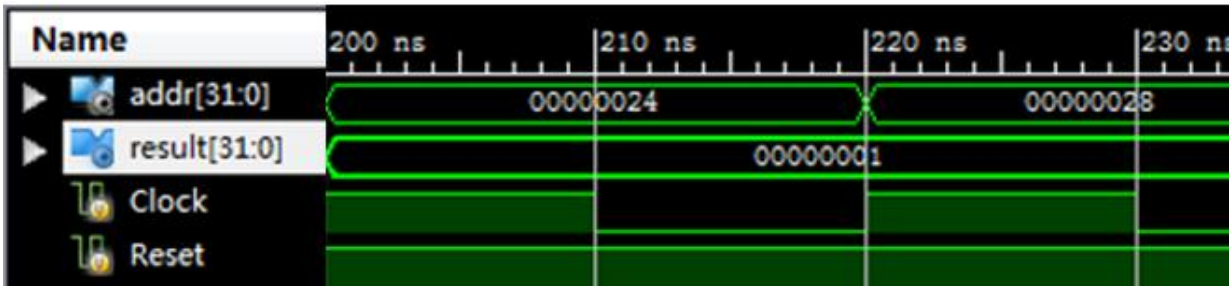
```
assign ram[5'h07]=32'h00221826;    //xor R3,R1,R2
assign ram[5'h08]=32'h00631826;    //xor R3,R3,R3
```



该指令不转跳，PC 依次加 4（0000001cH 和 00000020H），result 是 12340000H xor 56780000H=444C0000H 和 00000000H（任何数与自身的异或都为 0）。满足预期结果，测试正确。

⑤ lw 和 sw 语句：

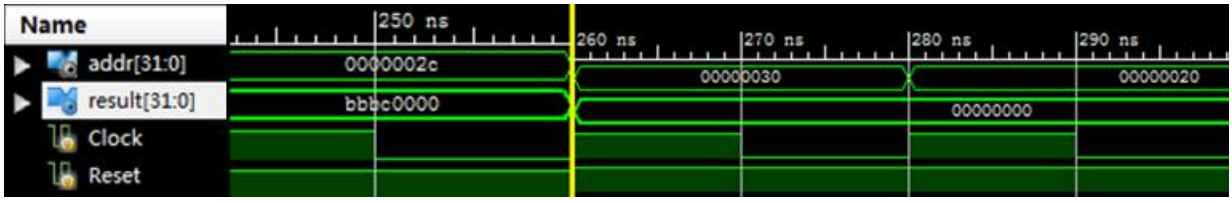
```
assign ram[5'h09]=32'hac610001;    //sw R1,1(R3)
assign ram[5'h0a]=32'h8c640001;    //lw R4,1(R3)
```



该指令不转跳，PC 依次加 4（00000024H 和 00000028H），result 是立即数 00000001H。满足预期结果，测试正确。

⑥ beq 语句

```
assign ram[5'h0b]=32'h10220000;    //beq R1,R2,0
assign ram[5'h0c]=32'h1021fffb;    //beq R1,R1,0xfffb
```



beq 语句先让 A 与 B 相减，若结果为 0 则转跳，如果不为 0 则顺序执行。

第一条 beq 语句，相减后不为 0，不转跳，PC 加 4（0000002cH），下条指令为 00000030H。满足预期结果，测试正确。

第二条 beq 语句，相减后为 0，转跳，通过计算向前转跳 4 条指令，为 00000020H。满

足预期结果，测试正确。

八、实验结论、心得体会和改进建议：

通过这次的实验，我成功地完成了完整 CPU 的制作与封装。从开始学习 Verilog，到实现各个部件，再到最终封装，我遇到过许许多多的问题，感谢老师和同学为我耐心解答，让我的能力有了提升。

在此也想提出一些建议，希望这门课可以根据计算机组成原理课程教学进度在学期中开课，不要集中在学期末。我是转专业选手，本学期课程加上补课课程共有 9 门期末考试，期末时备考、上课同时进行压力有点大。

希望今后能够有更多的实践课程，提升我们的动手能力，也希望合理安排上课时间并适当增加该课程时长，让我们能有更加全面的锻炼。