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

学　　院：计算机科学与工程

专　　业：计算机科学与技术

学生姓名：董文龙

学　　号：2018081309003

指导教师：吉家成，米源

日　　期：　　2020 年　10　月　5　日

电 子 科 技 大 学

实 验 报 告

实验一

一、实验名称：多路选择器，符号扩展器以及 ALU 的实现

二、实验学时：4

三、实验内容和目的：

1. 学习并会使用 Verilog 基本语法，学会仿真并在 FPGA 开发板上进行验证，可以读懂程序并实现程序。
2. 实现一个 32 位二选一多路选择器
3. 实现一个符号扩展器（将 16 位带符号数扩展为 32 位）
4. 32 位 ALU（算术逻辑单元）的设计与实现

四、实验原理：

1. 硬件描述语言特点：并发（同时执行相应的语句，与语句的先后顺序无关）

   Verilog HDL 可以执行的语句：

   assign　赋值语句

   always@( )　条件触发语句

   可以包含

   if 语句 1；else 语句 2；

   case ---endcase;

   这条语句中的输出变量，要定义成 reg 型；

   实例化用到的中间变量，要定义成 wire 型；

2. 32 位二选一多路选择器，与二选一多路选择器的原理相似，只是输入输出

的位数不一样，由一个控制信号 sel，来决定输出是哪一个输入

3. **实现一个符号扩展器（将 16 位带符号数扩展为 32 位），只需判断 16 位带符号的第 15 位（因为最低位是第 0 位）是 1 还是 0，如果是 1 则是负数，符号扩展就是全 1，如果是 0 则为正数，符号扩展就是全 0**
4. **32 位 ALU 的实现，通过每一个 op 对应的功能即可设计出 result 与输入的关系。**

五、实验器材（设备、元器件）

1. **安装了 Xilinx ISE Design Suite 13.7 的 PC 机一台**
2. **FPGA 开发板**
3. **计算机与 FPGA 开发板进行连接**

六、实验步骤：

32 位二选一多路选择器：

1. 在 ISE 集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择 New Source 命令，创建一个 Verilog Module 模块，名称为：MUX32_2_1，然后输入其实现代码：

```
module MUX32_2_1(
input [31:0] in1 , in2 ,
input Sel ,
output [31:0] out_data
);
assign out_data = Sel ?in2 :in1;
endmodule
```

2. 在 ISE 集成开发环境中，对模块 MUX32_2_1 进行仿真（Simulation）。首先输入如下测式代码：

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

```
        in2 = 1;
        sel = 0;
        // Wait 100 ns for global reset to finish
        #100; in1=0; in2=1;sel=1;
        #100; in1=1; in2=0;sel=0;
        // Add stimulus here
    end
```

16 位带符号数扩展：

1. 在 ISE 集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择 New Source 命令，创建一个 Verilog Module 模块，名称为：Sign_Extender，然后输入其实现代码：

```
module Sign_Extender(
    input [15:0]   in1 ,
    output [31:0] out_data
    );
    assign out_data={{16{in1[15]}},in1};
endmodule
```

2. 在 ISE 集成开发环境中，对模块 Sign_Extender 进行仿真（Simulation）。首先输入如下测式代码：

```
initial begin
        // Initialize Inputs
        inl = 0;
        // Wait 100 ns for global reset to finish
        #100;
        inl=8;
        // Add stimulus here
        #100;
        inl=-8;
    end
```

ALU（算术逻辑运算单元）：

1. 在 ISE 集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择 New Source 命令，创建一个 Verilog Module 模块，名称为：ALU，然后输入其实现代码：

```
module ALU(a,b,op,zero,result
    );
input [31:0] a,b;
```

```
input [3:0] op;
output [31:0] result;
output zero;
reg [31:0]result;
always @(*)
case(op)
      4'b0000: assign result=a+b;
      4'b0100: assign result=a-b;
      4'b0001: assign result=a&b;
      4'b0101: assign result=a|b;
      4'b0010: assign result=a^b;
      4'b0110: assign result={b[15:0],16'h0};
default:
       result=32'hxxxxxxxx;
endcase
assign zero =~|result;
endmodule
```

2. 在 ISE 集成开发环境中，对模块 ALU 进行仿真（Simulation）。首先输入如下测式代码：

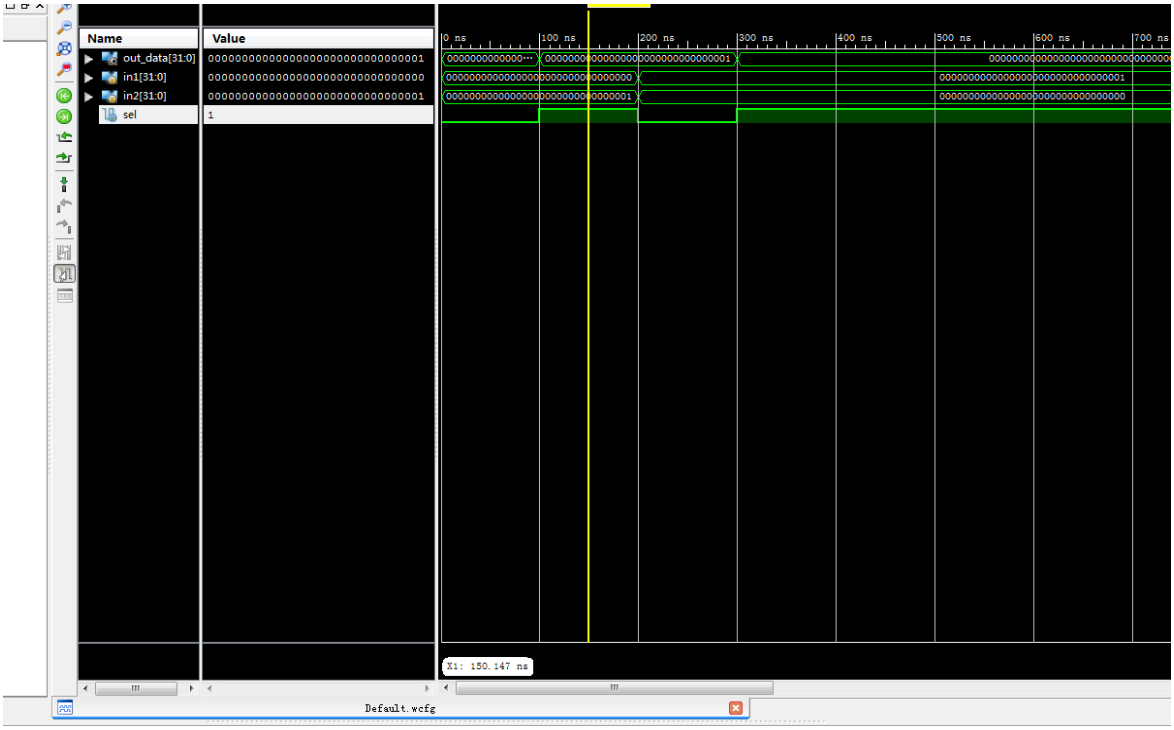
```
initial begin
            // Initialize Inputs
            a = 0;
            b = 0;
            op = 0;
            // Wait 100 ns for global reset to finish
            // Add stimulus here
            #100;a=8;b=3;op=0;
            #100;a=8;b=3;op=4;
            #100;a=8;b=3;op=1;
            #100;a=8;b=3;op=5;
            #100;a=8;b=3;op=2;
            #100;a=8;b=3;op=6;
      end
```

七、实验数据及结果分析：

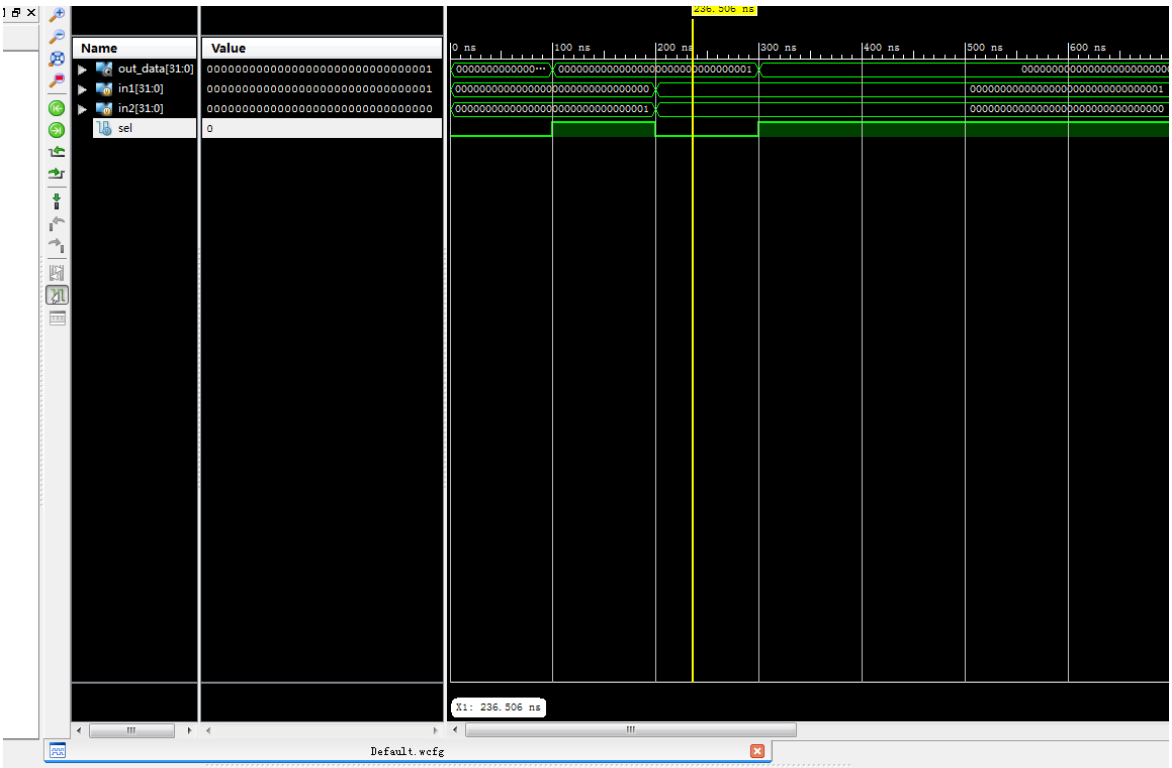
32 位二选一多路选择器：

实验截图:

Sel=1,out_data=in2



Sel=0,out_data=in1

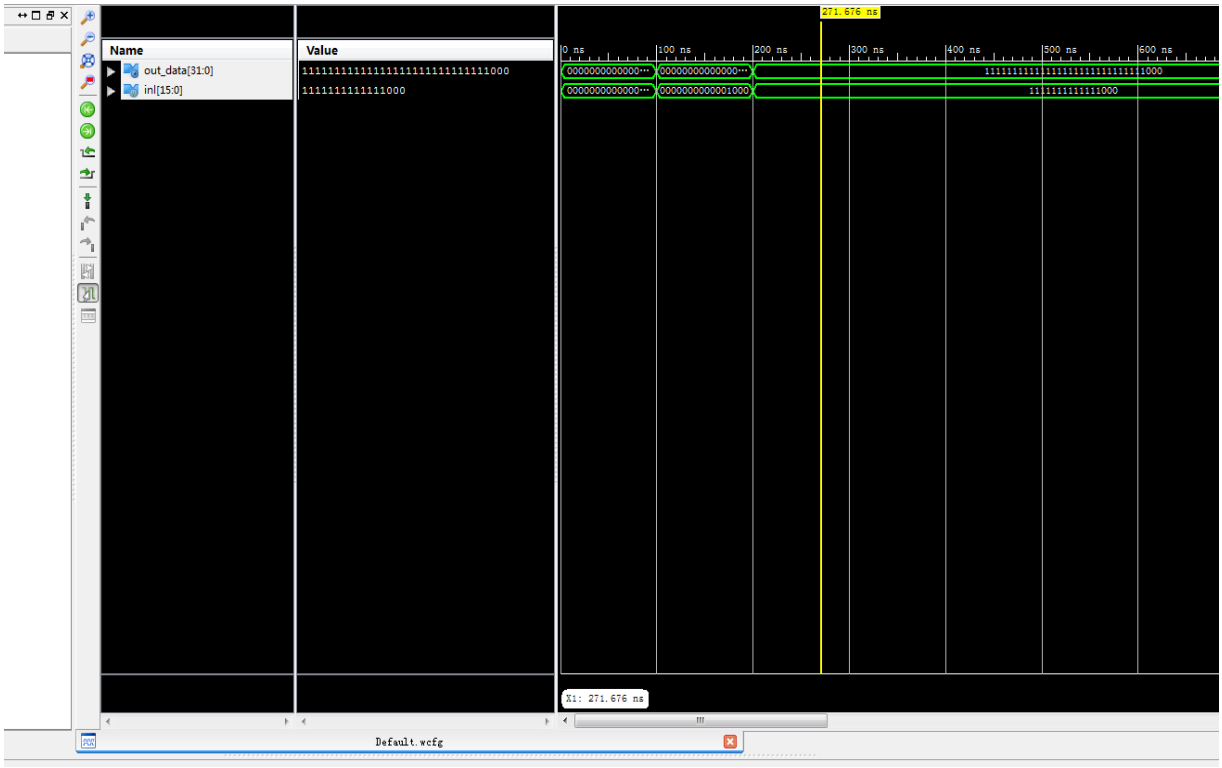


16 位带符号数扩展：

实验截图:

Inl=8，符号扩展应该扩展 0

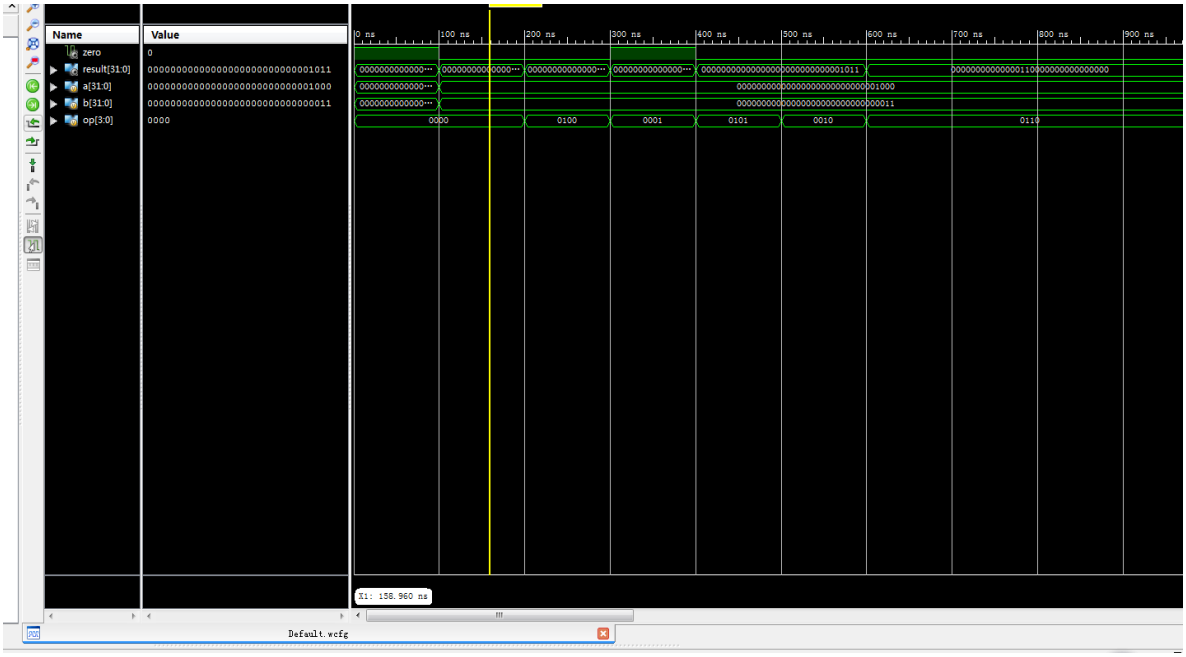
Inl=-8，符号扩展应该扩展 1



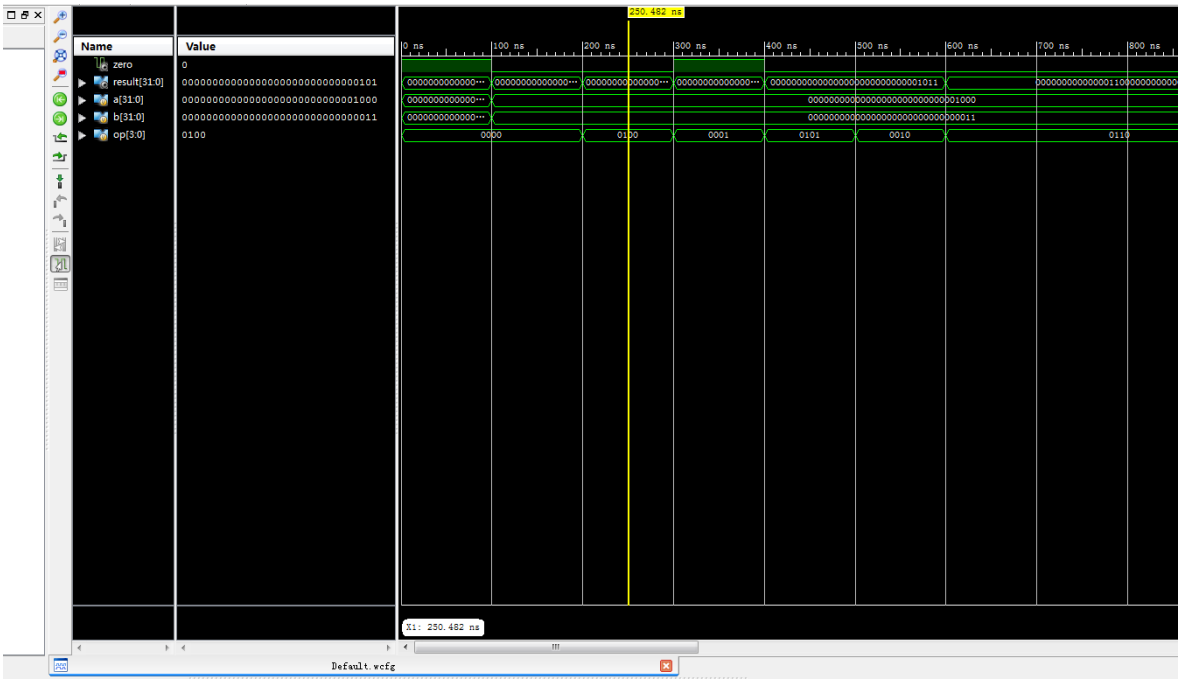
ALU（算术逻辑运算单元）：

实验截图：

Op=0,a=8,b=3,进行加运算：



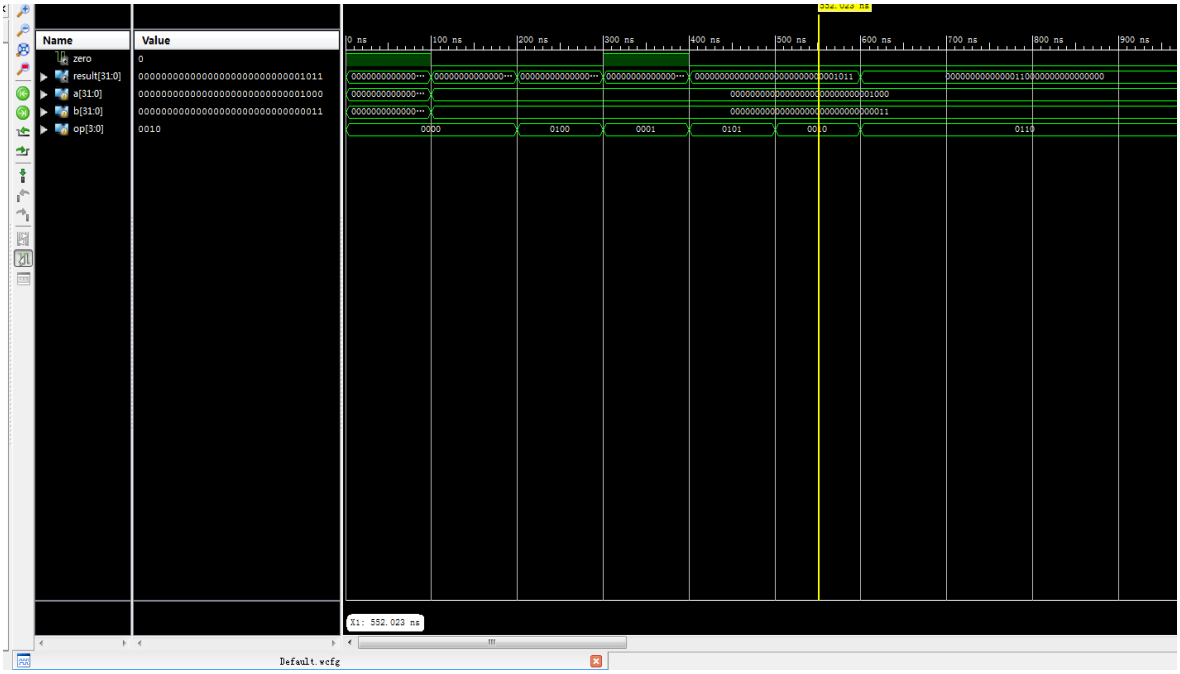
Op=4,a=8,b=3,进行减运算：



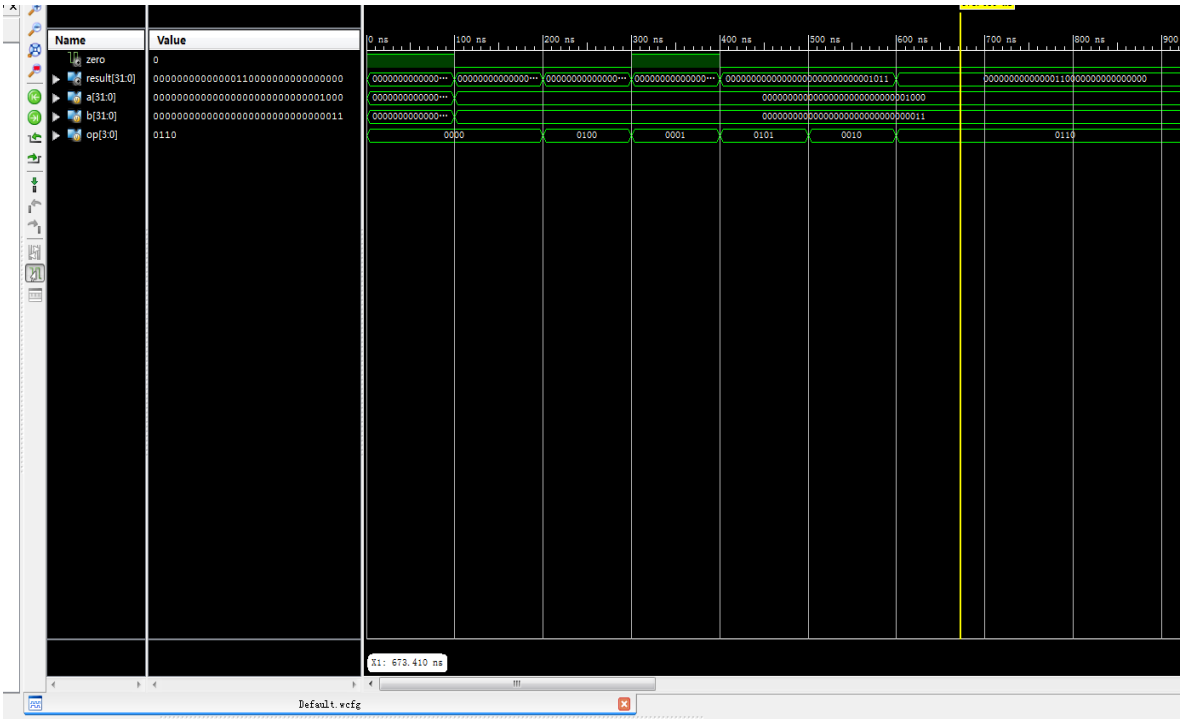
Op=1,a=8,b=3,进行与运算：

Op=5,a=8,b=3,进行或运算：

Op=2,a=8,b=3,进行异或运算：



Op=6,a=8,b=3,将 b 的低 16 位作为结果的高 16 位，结果的低 16 位用 0 填充。



八、实验结论、心得体会和改进建议：

1. 在写代码的时候要逻辑清晰，不可操之过急，要细心仔细，每个模块都在为后面设计 CPU 做铺垫，所以要一步一步认真走过来。
2. 尽量使用简洁的语句来完成程序的要求，尝试一些进阶的方法会加快书写效率。

电 子 科 技 大 学

实 验 报 告

实验二

一、实验名称：控制器（Control Unit）设计与实现

二、实验学时：4

三、实验内容和目的：

1. 完成单周期 CPU 的 Control Unit 元件
2. 更加熟练的掌握 ISE 软件，可以进行模块之间的调用，调用的时候进行实例化。
3. 阅读并分析控制单元代码，掌握其功能与工作原理，针对其每个接口信号，理解其含义与产生逻辑。
4. 分析单周期 CPU 工作原理，掌握各条指令（R 型、lw、sw、beq、lui）在单周期 CPU 中运行时的数据通路。
5. 对控制单元 Verilog 模块进行仿真验证：

   a） 要求仿真内容涵盖 5 种指令（ R 型、lw、sw、beq、lui ）。

   b） 验证并分析仿真结果。

四、实验原理：

1. Control Unit 由 Control 和 ALUop 两部分组成，对两个子模块先进行设计，之后再进行调用。
2. Control 将 op[5:0]字段作为输入，产生 RegDst，RegWrite,ALUsrc MemWrite,MemRead,MemtoReg,Branch,ALUctr[1:0]译码信号.

3. **ALUop 将 func[5:0]，ALUctr[1:0]字段作为输入，产生 ALU_op[2:0]作为译码信号**

五、实验器材（设备、元器件）

1. **安装了 Xilinx ISE Design Suite 13.7 的 PC 机一台**
2. **FPGA 开发板**
3. **计算机与 FPGA 开发板进行连接**

六、实验步骤：

1. **在 ISE 集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择 New Source 命令，创建一个 Verilog Module 模块，名称为：Control，然后输入其实现代码：**

```
module Control(op,RegDst,RegWrite,ALUSrc,
        MemWrite,MemRead,MemtoReg,Branch,ALUctr);
    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[3] & op[2];
    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. **在 ISE 集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择 New Source 命令，创建一个 Verilog Module 模块，名称**

为：**ALUop**，然后输入其实现代码：

```
module ALUop(func, ALUctr, ALU_op
    );
    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[2] &func[1]) | (ALUctr[1]&ALUctr[0]);
assign ALU_op[0]=(i_Rt & func[2] & ~func[1]);
endmodule
```

3. 在 **ISE** 集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择 **New Source** 命令，创建一个 **Verilog Module** 模块，名称为：**Control_Unit**，然后输入其实现代码：

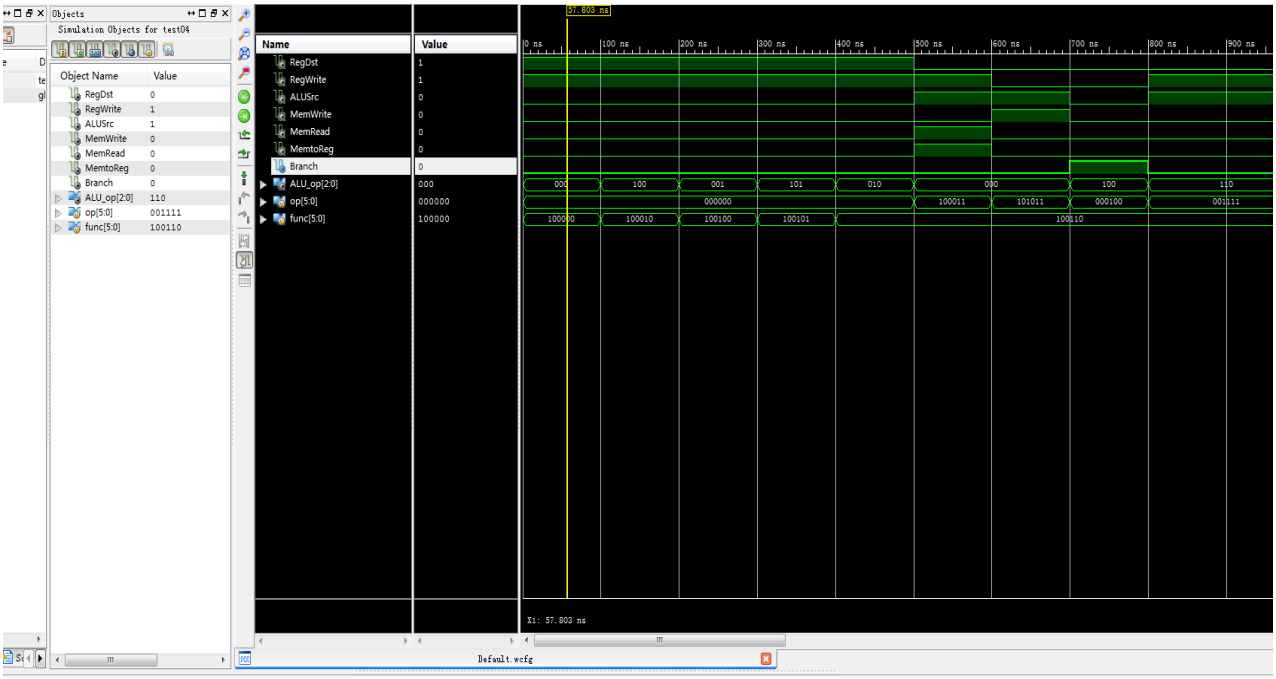
```
module Contro_Unit(op,func,RegDst,RegWrite,ALUSrc,
        MemWrite,MemRead,MemtoReg,Branch,ALU_op);
    input       [5:0] op,func;
    output      RegDst,RegWrite,ALUSrc;
    output      MemWrite,MemRead,MemtoReg;
    output      Branch;
    output      [2:0] ALU_op;
    wire [1:0] ALUctr;
    Control 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
```

4. 在 **ISE** 集成开发环境中，对模块 **Control_Unit** 进行仿真（**Simulation**）。首先输入如下测试代码：

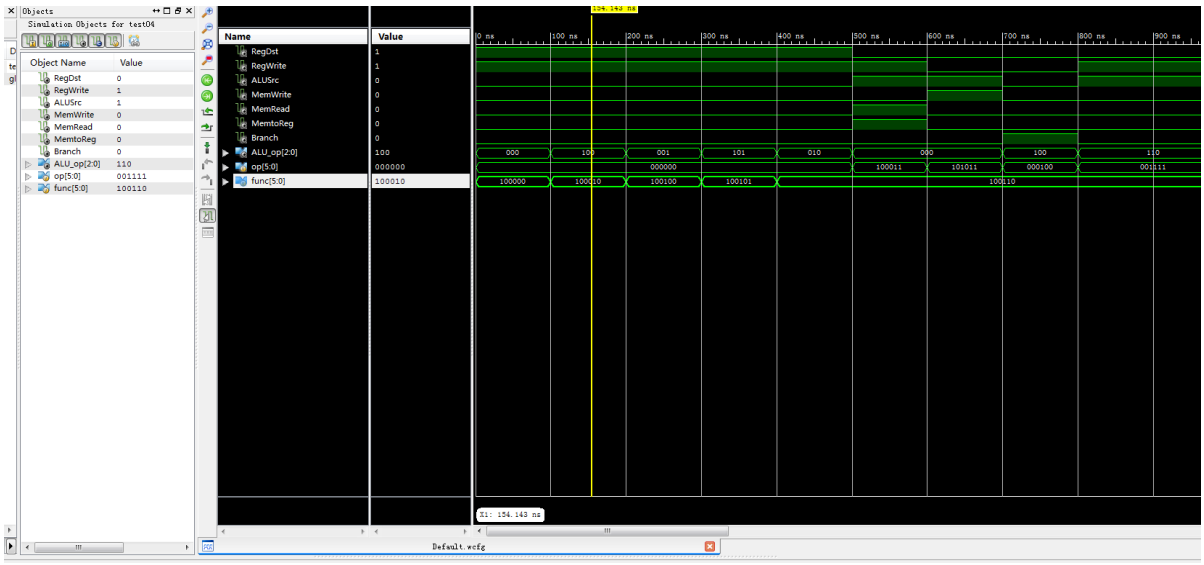
```
initial begin
        // Initialize Inputs
        // Wait 100 ns for global reset to finish
        //R 型指令
        op=6'b000000; func=6'b100000;
        #100; op=6'b000000; func=6'b100010;
       #100; op=6'b000000; func=6'b100100;
        #100; op=6'b000000; func=6'b100101;
        #100; op=6'b000000; func=6'b100110;
        //其他指令
        #100; op=6'b100011;
        #100; op=6'b101011;
        #100; op=6'b000100;
        #100; op=6'b001111;
    end
```

七、实验数据及结果分析：

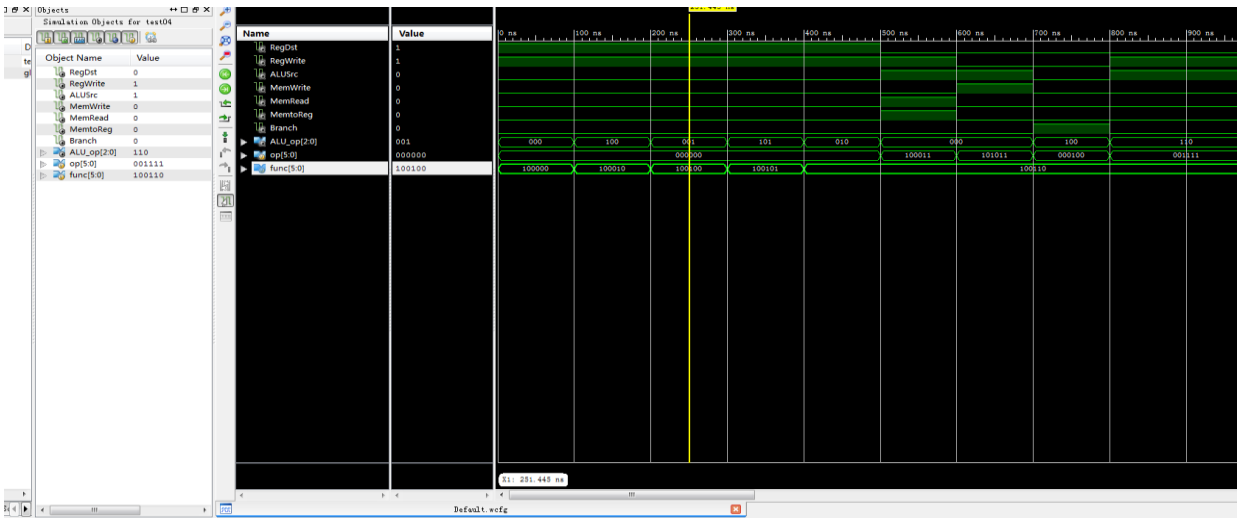
1. R-TYPE（Add）



2. R-TYPE（Sub）

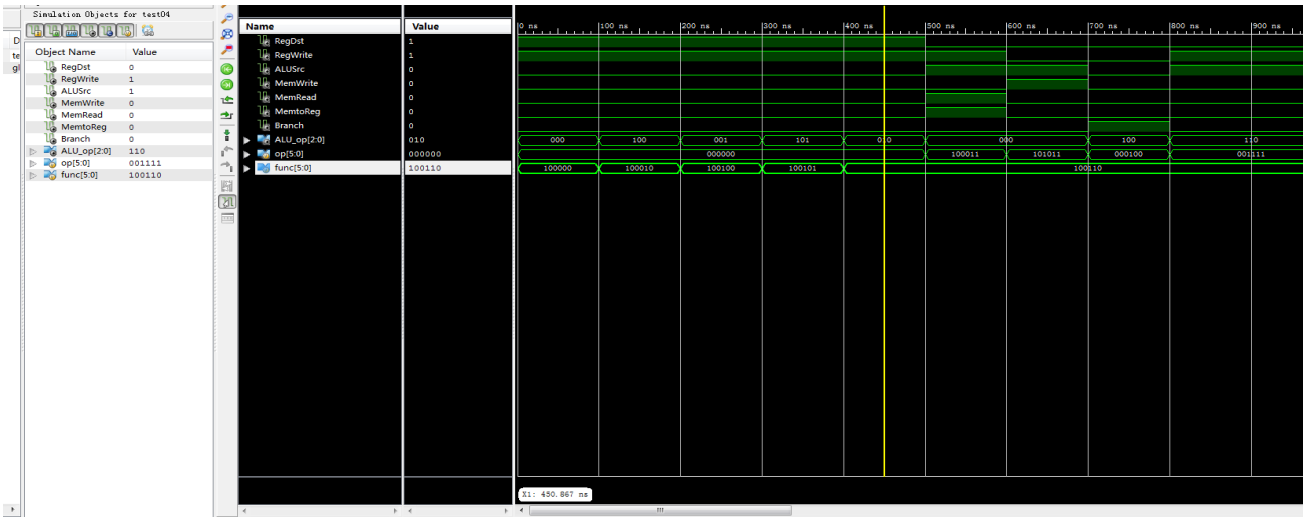


3. R-TYPE（And）

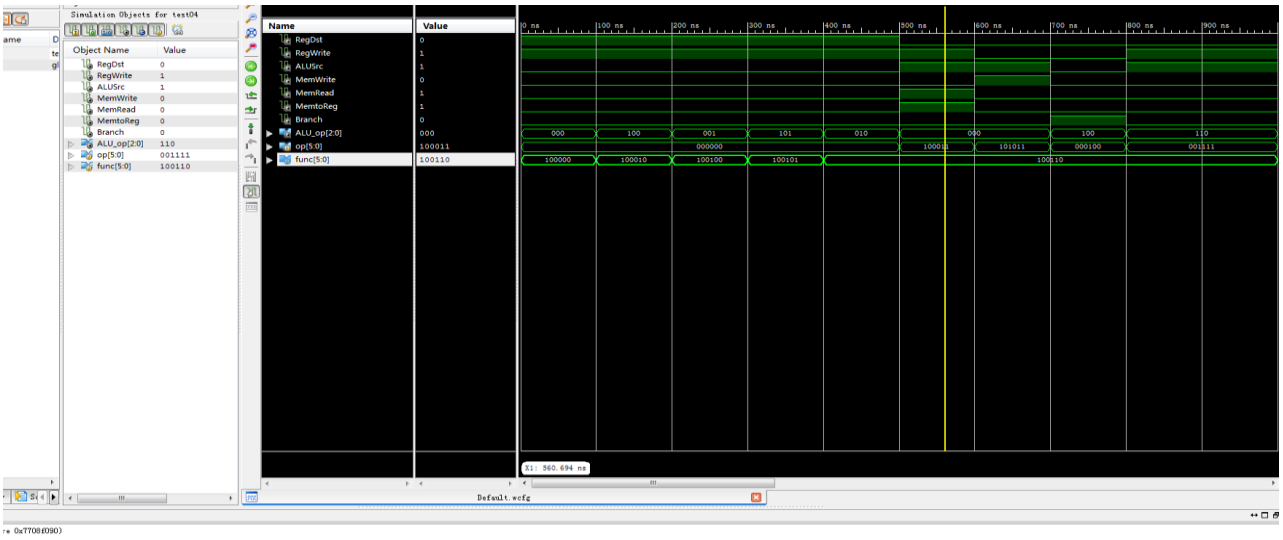


4. R-TYPE（Or）

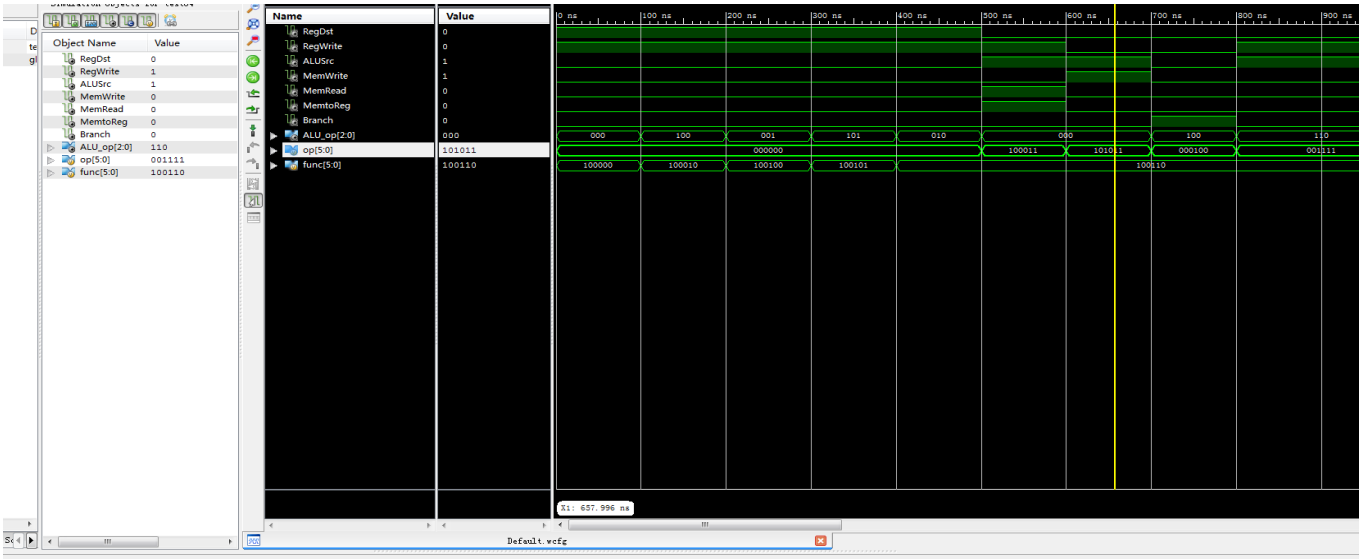
5. R-TYPE（Xor）



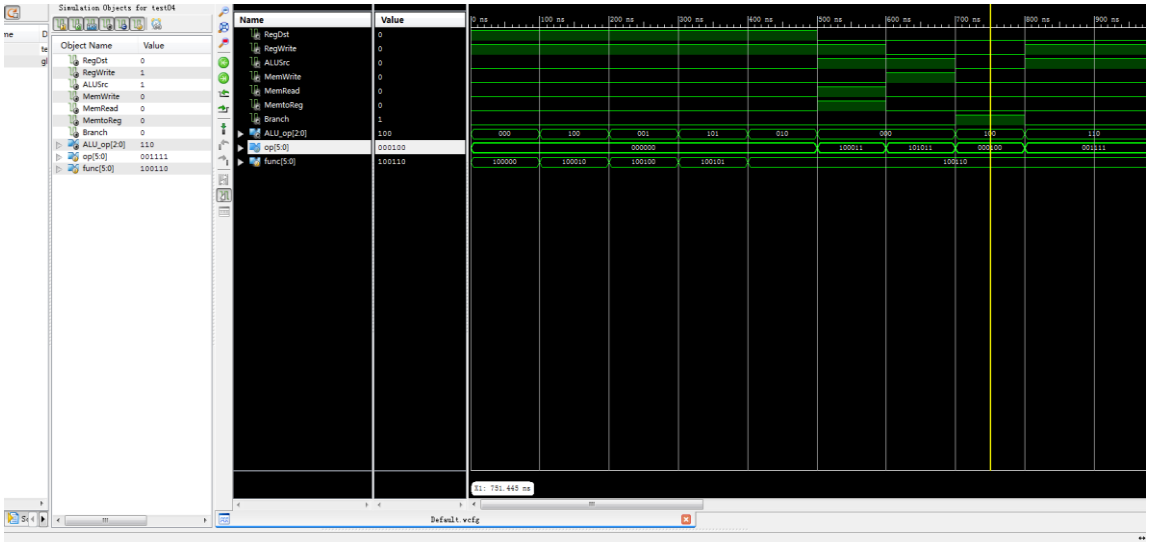
6. Lw（取数指令）



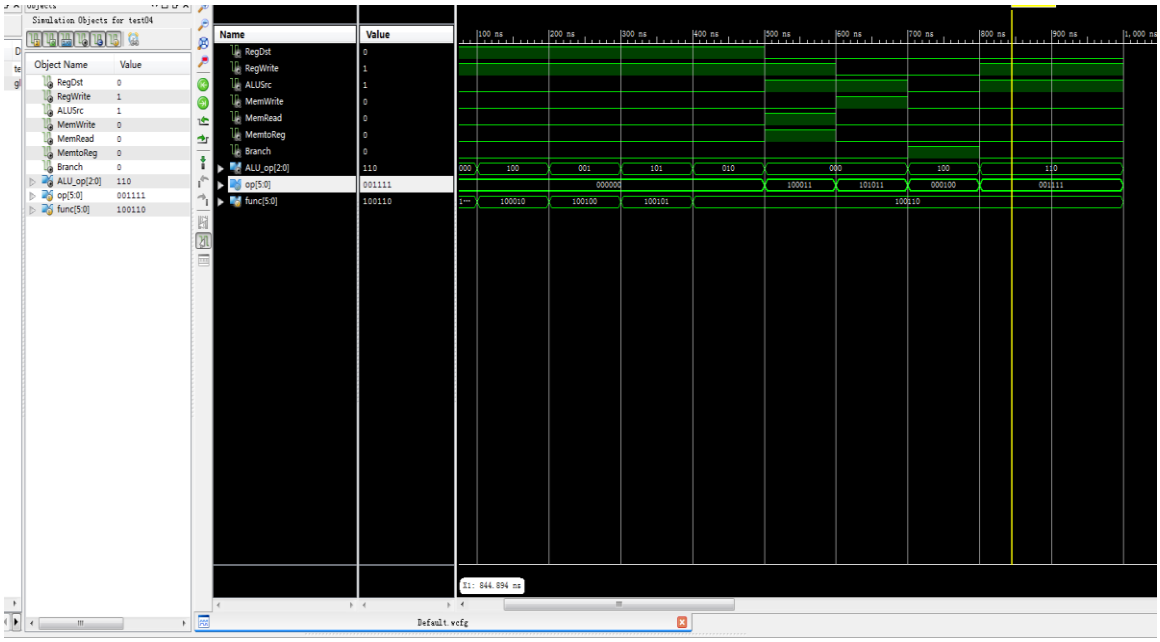
7. Sw（存数指令）



8. Beq（分支指令）



9. Lui（置高位指令）



八、实验结论、心得体会和改进建议：

1. 在写代码的时候要逻辑清晰，不可操之过急，要细心仔细，每个模块都在为后面设计 CPU 做铺垫，所以要一步一步认真走过来。
2. 实验的仿真过程中，可能会产生高阻态的情况，不要急，应该是某个地方的字母敲打错误，需要认真排查。

电 子 科 技 大 学

实 验 报 告

实验三

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. 阅读并分析取指单元代码，掌握其功能与工作原理，针对其每个接口信号，理解其含义与产生逻辑。

2. 针对以下三种情况对取指单元 **Verilog** 模块进行仿真验证：

a）非跳转指令（**beq**）；

b）**beq** 指令跳转不成功；

c）**beq** 指令跳转成功；

对以上情况的仿真结果进行验证分析。

四、实验原理：

1.对于非跳转指令，**PC=PC+4;**

2.对于跳转指令，若 **beq** 指令跳转不成功，**PC=PC+4**

3.对于跳转指令，若 **beq** 指令跳转成功，**PC=PC+4+ Imm*4**

五、实验器材（设备、元器件）

1. 安装了 **Xilinx ISE Design Suite 13.7** 的 **PC** 机一台

2. FPGA 开发板

3. 计算机与 **FPGA** 开发板进行连接

六、实验步骤

1.在 ISE 集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择 New Source 命令，创建一个 Verilog Module 模块，名称为：Fetch，然后输入其实现代码：

```
module Fetch(B,Z,Reset,Clock,B_addr,addr
    );
input B,Z,Reset,Clock;//B(跳转指令标志)，Z(跳转成功标志)，Reset(复位信号)，Clock(时钟信号)
input [31:0] B_addr;//B_addr(偏移地址)
output [31:0] addr;//指令地址
reg [31:0] PC;//程序计数器
    wire [31:0] sum0,B_addr1,sum1,next_pc;//sum0(PC+4),B_addr1(偏移地址左移两位)
                                          //sum1(PC+4+(偏移地址<<2)),next_pc(指令地址)
    wire sel =Z&B;//地址选择信号
    Left_2_Shifter U0(B_addr,B_addr1);
    ADD32 U1(PC,4,sum0);
    ADD32 U2(sum0,B_addr1,sum1);
    MUX32_2_1 M1(sum0,sum1,sel,next_pc);
    assign addr=PC;
    always @(posedge Clock or posedge Reset)
    begin
        if (Reset==1) PC=0;
        else    PC=next_pc;
    end
Endmodule
```

2.创建 Left_2_Shifter 模块，并输入其实现代码：

```
module    Left_2_Shifter (B_addr,B_addr1
);
input [31:0] B_addr;
output [31:0]    B_addr1;
assign B_addr1=B_addr<<2'b10;
endmodule
```

3.创建 ADD32 模块，并输入其实现模块

```
module    ADD32(sum0,B_addr1,sum1
);
input [31:0] sum0;
input [31:0] B_addr1;
output [31:0] sum1;
assign sum1=B_addr1+sum0;
endmodule
```

4.创建 MUX32_2_1 模块，并输入其实现模块

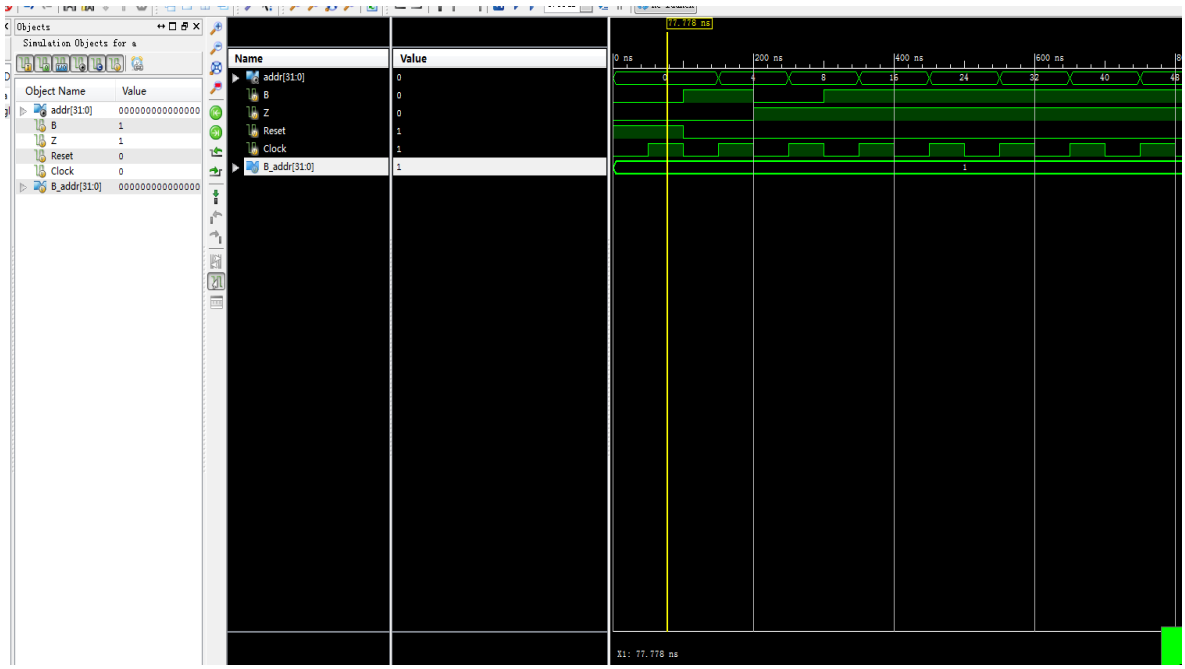
```
module MUX32_2_1(sum0,sum1,sel,next_pc
);
input     [31:0]       sum0,sum1;
input                      sel;
output    [31:0]       next_pc;
reg        [31:0]        next_pc;
always @(sum0,sum1,sel)
     if(!sel) next_pc=sum0;
     else next_pc=sum1;
endmodule
```

5.在 ISE 集成开发环境中，对模块 Fetch 进行仿真（Simulation）。首先输入如下测测试代码：

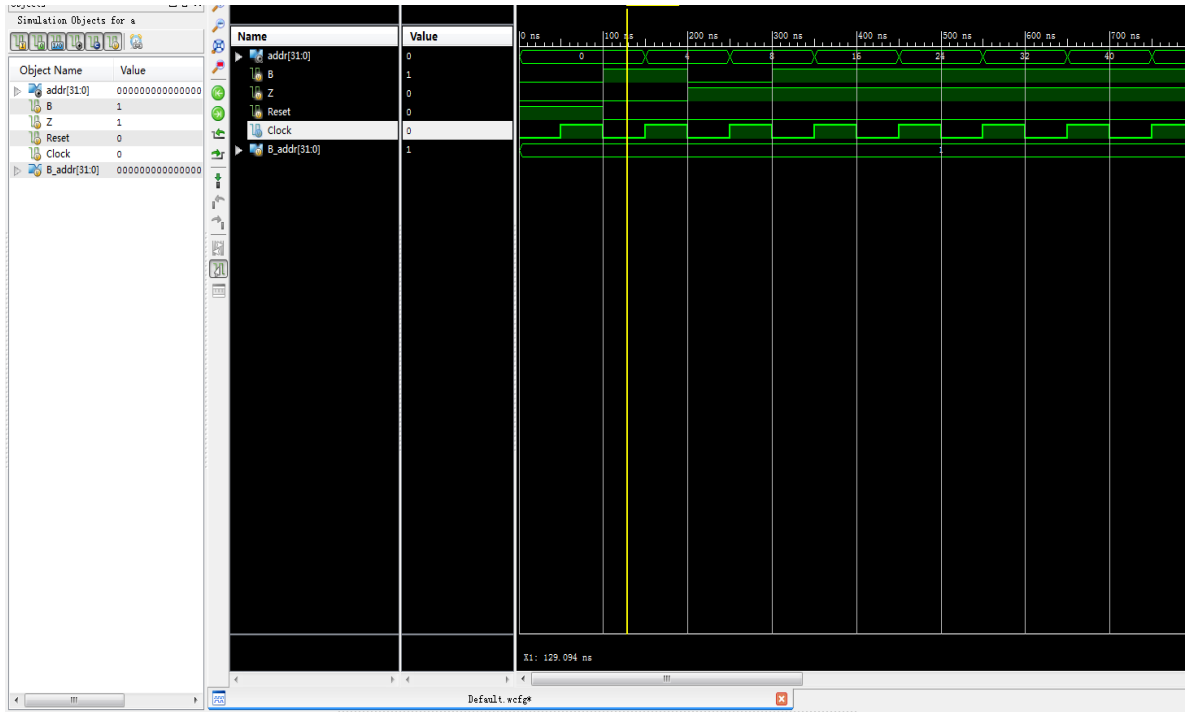
```
initial begin
          // Initialize Inputs
          B = 0;
          Z = 0;
          Reset = 1;
          Clock = 0;
          B_addr = 1;
          // Wait 100 ns for global reset to finish
          #100;
          Reset=0;
          B = 1;
          Z = 0;
          B_addr = 1 ;
          #100;
          B = 0;
          Z = 1;
          B_addr = 1;
          #100;
          B = 1;
          Z = 1;
          B_addr = 1;
          // Add stimulus here
     end
        always #50 Clock =~Clock;
```

七、实验数据及结果分析

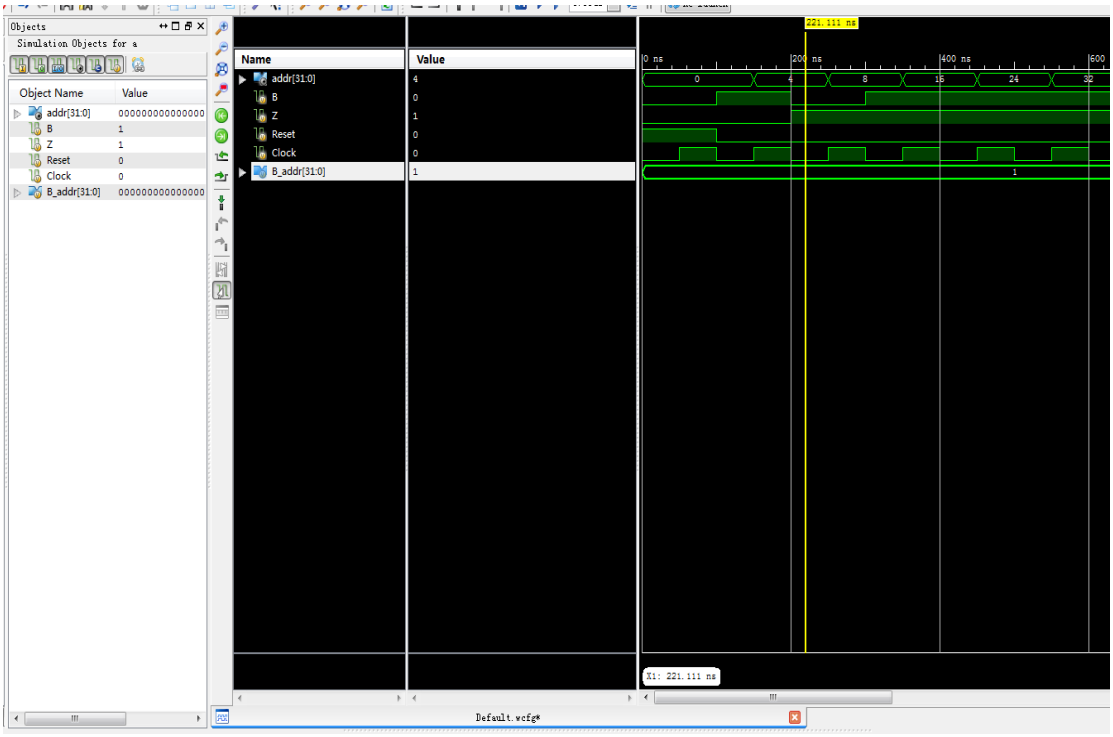
B = 0;Z = 0;Reset = 1;Clock = 0;B_addr = 1;



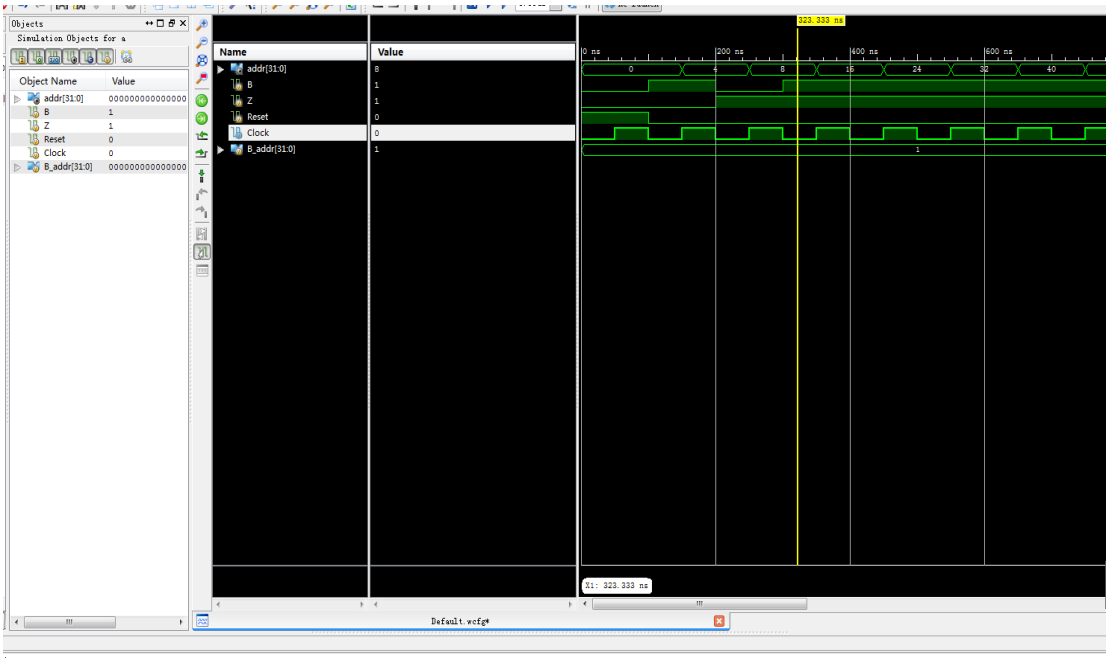
#100;Reset=0;B = 1;Z = 0;B_addr = 1 ;

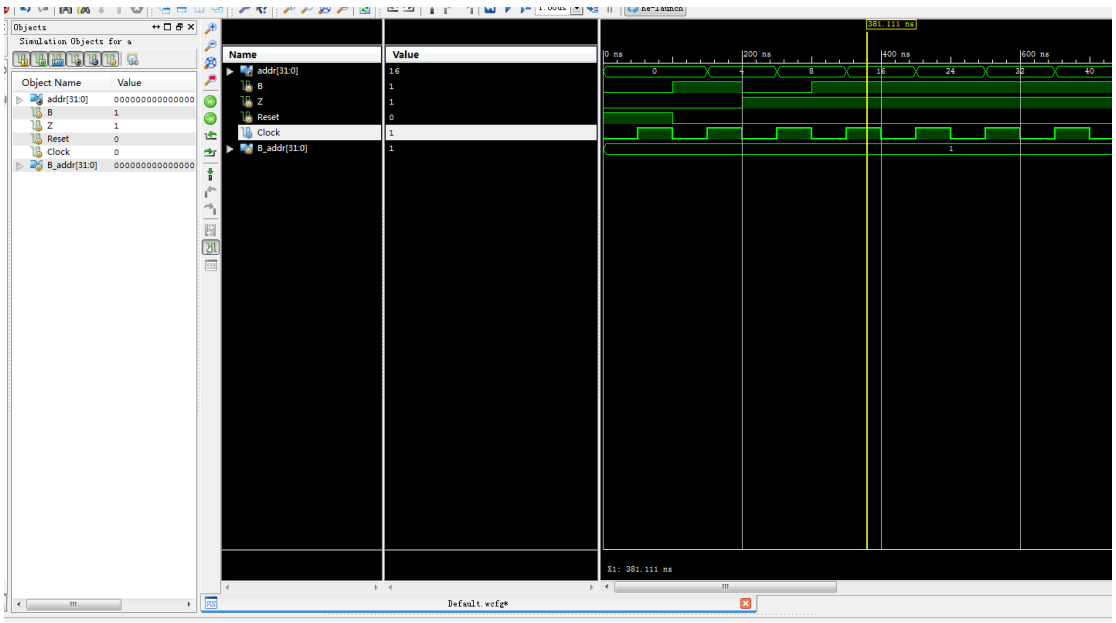


#100;Reset=0;B = 0;Z = 1;B_addr = 1 ;



#100;B = 1;Z = 1;B_addr = 1;





八、实验结论、心得体会和改进建议

1. 在写代码的时候要逻辑清晰，不可操之过急，要细心仔细，每个模块都在为后面设计 **CPU** 做铺垫，所以要一步一步认真走过来。

2.取值部件的 **ADD32**，在 **beq** 成功转移之后，需要调用两次该方法。

3.写代码的时候要认真，否则就会因为字母不正确导致仿真结果出错。

电 子 科 技 大 学

实 验 报 告

实验四

一、实验名称：“Single_Cycle_Computer”

二、实验学时：4

三、实验内容和目的：

1. 在“Single_Cycle_Computer”模块下对单周期 CPU 电路结构中的各模块进行封装。

2. 自行设计测试用指令序列（要求每种类型的指令至少包含一条），填入“INST_ROM”模块（指令存储器）中。

3. 使用 INST_ROM 中的指令序列对 Single_Cycle_Computer 模块进行仿真，验证并分析仿真结果。

选做：

对“Single_Cycle_Computer_IO”模块进行 Synthesize-XST、Implement Design、Generate Programing File 操作，生成*.bit 文件，将文件下载至开发板验证实验结果。

四、实验原理：

1.Single_Cycle_Computer 由 Reset,Clock 信号作为输入，ALU 计算出的结果 Result 作为输出，addr 指令的存储地址也作为输出

2.在 Fetch 取指部件中取出指令，正常取指，执行后 PC+4。跳转指令，立即数左移两位后与 PC+4 的结果相加。

3.在 INST_ROM 指令存储部件中取出指令，进行操作。

4.在 Control_Unit 中得到控制信号

5.五路二选一，根据 Control_Unit 产生的 RegDSt 信号来确定哪 5 位进入 Wn

6. RegFile 寄存器堆，根据 Control_Unit 产生的 RegWrite 信号来决定是否写入寄存器。从 MemtoReg 信号决定后回写到寄存器的数据从 A 输出的和从 B 输出的 32 位结果.

7. Sign_Extender 符号扩展器,对 Inst 的低 16 位进行符号扩展

8.通过 Control_Unit 产生的 ALUsrc 控制信号来进行 32 位 2 路二选一

9. 根据 Control_Unit 产生的 ALU_op 来进行 ALU 运算

10. DATA_RAM，根据 Control_Unit 产生的 MemWrite，MemRead 信号，以及 ALU 产生的 Result 作为 Addr 处理存储器 MemOutdata 作为存储器的输出。

11.根据 Control_Unit 产生的 MemtoReg 信号利用 32 为 2 路二选一来确定最后的 MemtoRegdata

五、实验器材（设备、元器件）

1. 安装了 Xilinx ISE Design Suite 13.7 的 PC 机一台
2. FPGA 开发板
3. 计算机与 FPGA 开发板进行连接

六、实验步骤

1.在 ISE 集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择 New Source 命令，创建一个 Verilog Module 模块，名称为：Single_Cycle_Computer，然后输入其实现代码：

```
module Single_Cycle_Computer(
    input Reset , Clock,              //CPU 输入输出接口：Reset（复位信号）,Clock（时钟信号），
    output [31:0] Result,addr         //Result（CPU 运算结果），addr（指令存储器地址）,都是 32 位
```

```
    );
////////////////////////////////////////////////////////////////////////////////
//存储器写信号，存储器读信号，存储器回寄存器信号
    wire MemWrite, MemRead , MemtoReg;
//instruction ROMd Inst(即 32 位长度的指令),从存储器往寄存器回写的值，从寄存器往存储器写的值，
    wire [31:0] Inst , MemOutData , MemtoRegdata ,Result;
//五路二选一即选择 rs 还是 rt 寄存器，寄存器写信号，参加 ALU 运算的来源（是通过 B 过来还是经过扩展而来），
//Branch 分支指令，ALU 运算得到的指令
    wire RegDst , RegWrite , ALUSrc   ,Branch , Zero;
//用来区分 ALU 的运算方式
    wire [2:0] ALU_op;
//得到的由 rs 还是 rt 进行 5 路二选一得到
    wire [4:0] Wn;
//寄存器堆从 A 出去的值，寄存器堆从 B 出去的值，符号扩展完成的值即（Inst 的低 16 位符号扩展完成），
    wire [31:0] A , B , Ext_Imm , ALU_B ;
//取指部件，首先取出指令，正常取指，执行后 PC+4。跳转指令，立即数左移两位后与 PC+4 的结果相加。
   Fetch U0(Branch,Zero,Reset,Clock,Ext_Imm,addr);
//指令存储部件，在这里取出指令，取出的指令在 Inst 中
   INST_ROM U7(addr,Inst);
//控制信号的产生，对于 Inst 提供的指令，产生相应的信号，对应的字段信号为 op 字段（不同的操作类型
//对应不同的操作 R 型的操作字段为全 0，lw 为 100011，sw 为 101011，beq 为 000100，lui 为 001111），
//func 字段来确定指令的功能。剩下的均为产生的控制信号。
    Control_Unit U1( .op(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)   );
   //五路二选一，根据 Control_Unit 产生的 RegDSt 信号来确定哪 5 位进入 Wn
    MUX5_2_1 U2( Inst[20:16] , Inst[15:11] , RegDst , Wn );
   //寄存器堆，根据 Control_Unit 产生的 RegWrite 信号来决定是否写入寄存器。从 MemtoReg 信号决定后
   //回写到寄存器的数据从 A 输出的和从 B 输出的 32 位结果
    RegFile U3(Inst[25:21] , Inst[20:16] ,Wn ,RegWrite , MemtoRegdata , A ,B , Clock);
    //对 Inst 的低 16 位进行符号扩展
    Sign_Extender U4( Inst[15:0] , Ext_Imm);
    //通过 Control_Unit 产生的 ALUSrc 控制信号来进行 32 位 2 路二选一，得到的结果赋值给 ALU_B 与 A 进行 ALU 运算
    MUX32_2_1 U5( B , Ext_Imm , ALUSrc , ALU_B );
     //根据 Control_Unit 产生的 ALU_op 来进行 ALU 运算，得到的结果 Result 位输出
     ALU U6 ( A , ALU_B , ALU_op , Result , Zero);
   //根据 Control_Unit 产生的 MemWrite，MemRead 信号，以及 ALU 产生的 Result 作为 Addr 处理存储器
```

```
    //MemOutdata 作为存储器的输出。
      DATA_RAM U8(Clock, MemOutData,B,Result ,MemWrite, MemRead);
      //根据 Control_Unit 产生的 MemtoReg 信号来确定最后的 MemtoRegdata
      MUX32_2_1 U9(Result , MemOutData , MemtoReg , MemtoRegdata);
endmodule
```

2.按照实验三的方法，将其他的模块调入进来，来完成整个 Single_Cycle_Computer

3.在 ISE 集成开发环境中，对模块 Single_Cycle_Computer 进行仿真（Simulation）。首先输入如下测式代码：

```
    initial begin
        // Initialize Inputs
        Reset = 1;
        Clock = 0;
        // Wait 100 ns for global reset to finish
        #100;
        Reset=0;
        // Add stimulus here
    end
       always #50 Clock=~Clock;
```

4.将文件生成为二进制文件，下载到开发板上，对数码管进行观察。

七、实验数据及结果分析

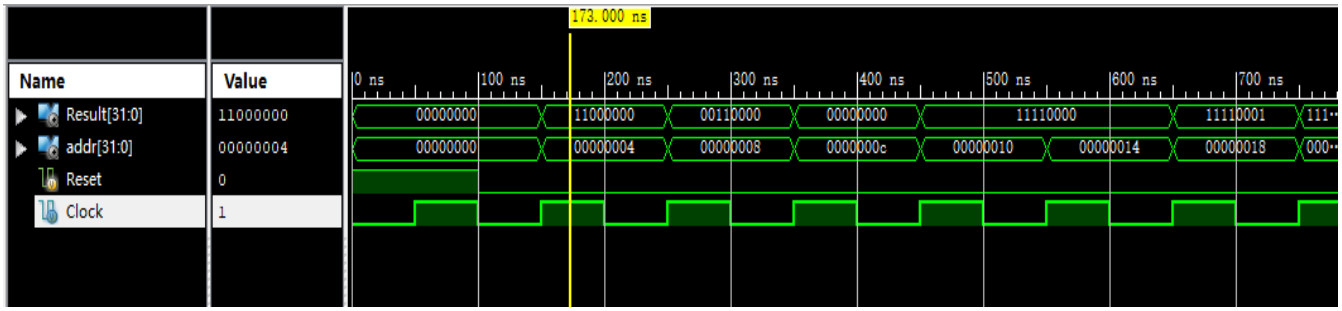
设计的指令序列：

```
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'h3c011100;      //lui R1,0x1100;ALU=0x11000000
      assign ram[5'h02]=32'h3c020011;      //lui R2,0x0011;ALU=0x00110000
      assign ram[5'h03]=32'h00221824;      //and R3,R1,R2; ALU=0x00000000
      assign ram[5'h04]=32'h00221826;      //xor R3,R1,R2; ALU=0x11110000
      assign ram[5'h05]=32'h00221820;      //add R3,R1,R2;ALU=0x11110000
      assign ram[5'h06]=32'hac610001;      //sw R1,1(R3) ALU=0x11110001
      assign ram[5'h07]=32'h8c640002;      //lw R4,2(R3) ALU=0x11110002
      assign ram[5'h08]=32'h10220001;      //beq R1,R2,1 ALU=R1-R2=0x10ef0000
      assign Inst=ram[addr[6:2]];        //以字节为单位进行寻址，32 位指令占用 4 字节，PC+4
```

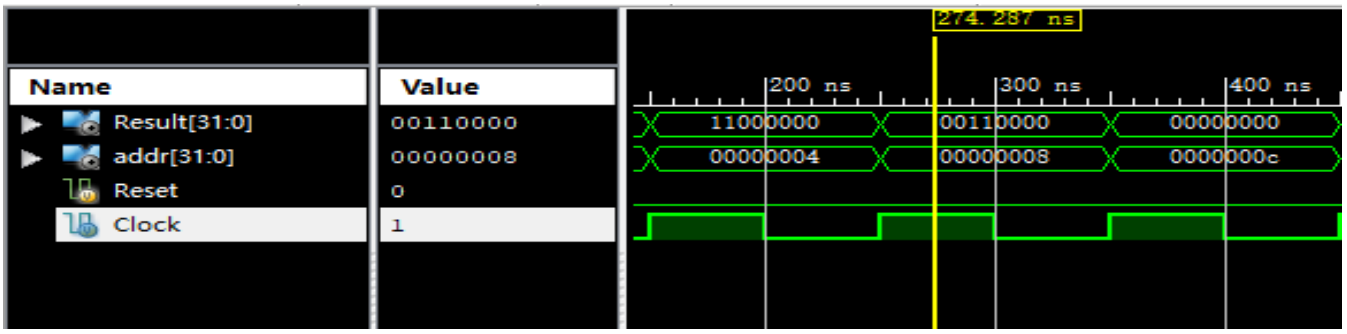
指向下一条指令地址。
endmodule

仿真截图：

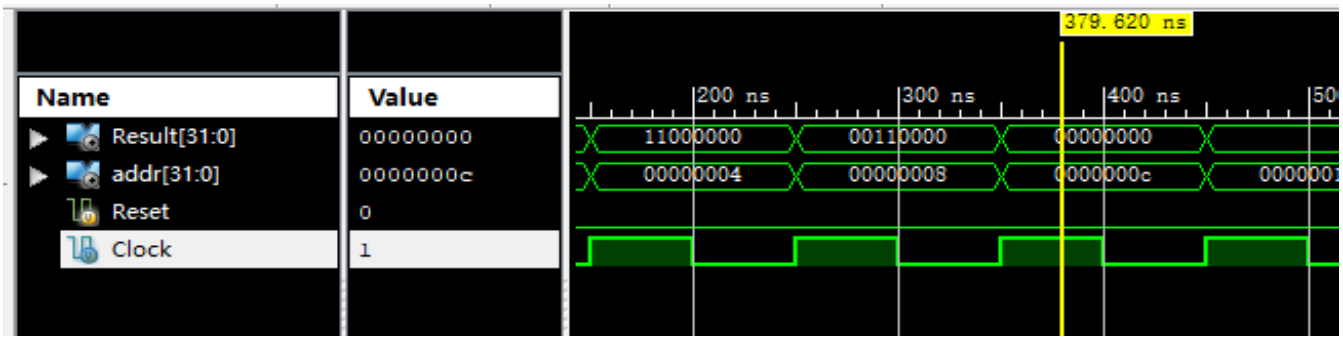
32'h 3c011100 ==》lui R1,0x1100,ALU=11000000,PC=4:



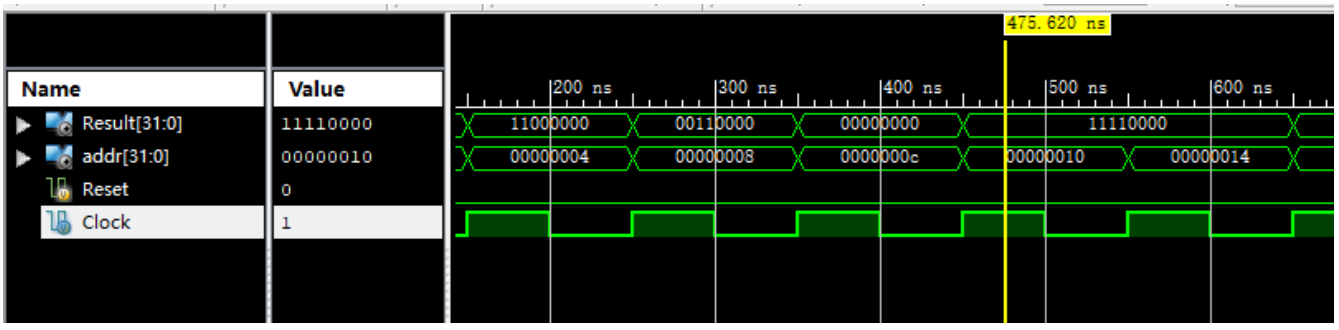
32'h 3c020011 ==》lui R2,0x0011;ALU=0x00110000，PC=8：



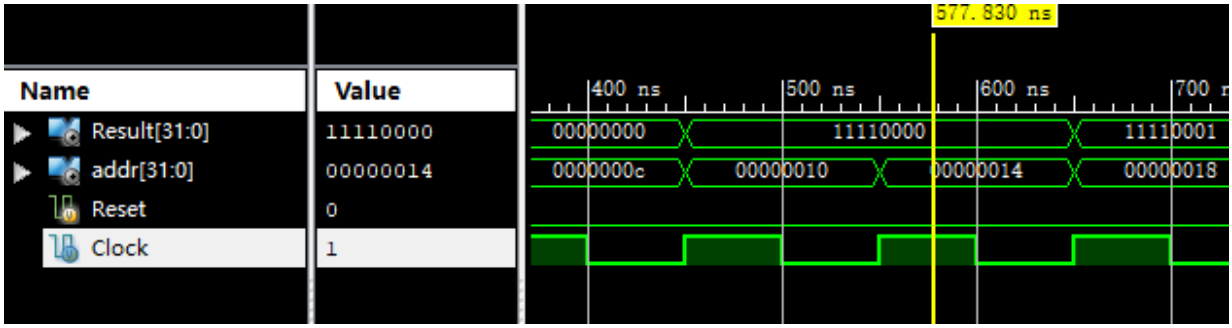
32'h 00221824 ==》and R3,R1,R2; ALU=0x00000000，PC=12:



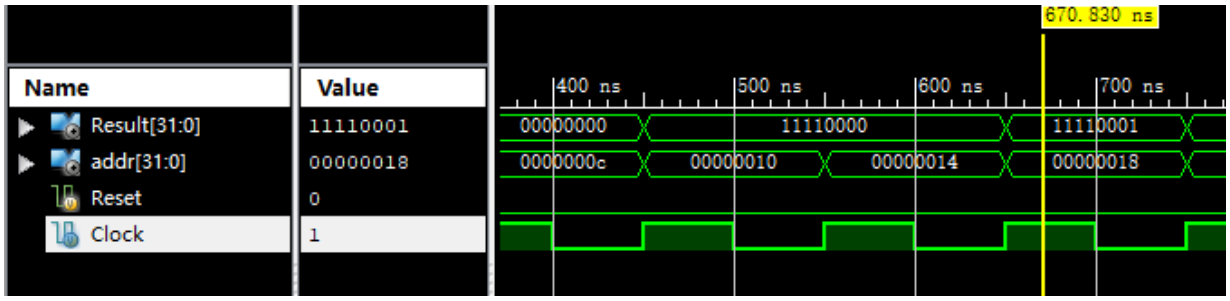
32'h00221826 ==》xor R3,R1,R2; ALU=0x11110000，PC=16:



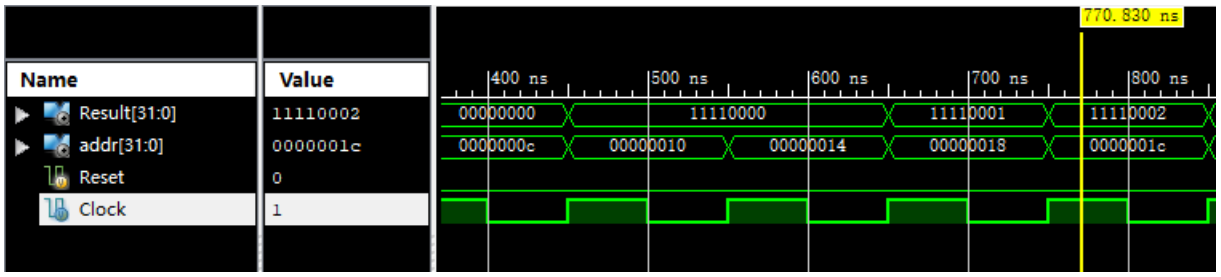
32'h00221820 ==》add R3,R1,R2；ALU=0x11110000， PC=20:



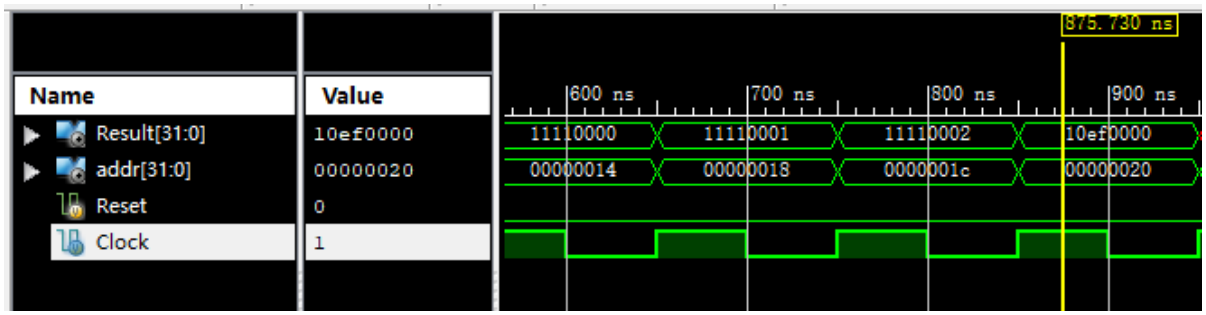
32'hac610001 ==》sw R1,1(R3) ALU=0x11110001,PC=24:



32'h8c640002 ==》lw R4,2(R3) ALU=0x11110002,PC=28:



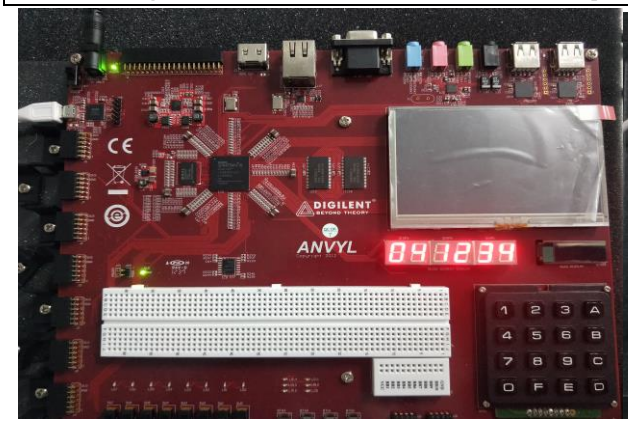
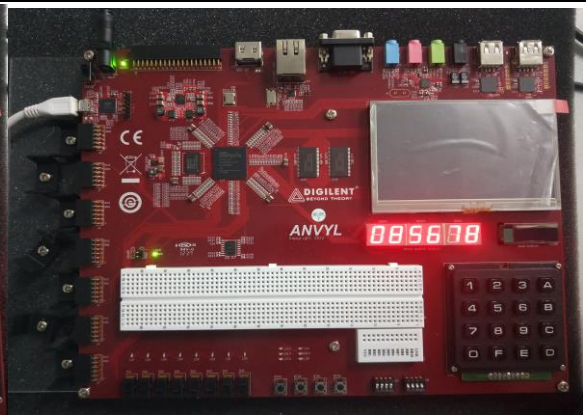
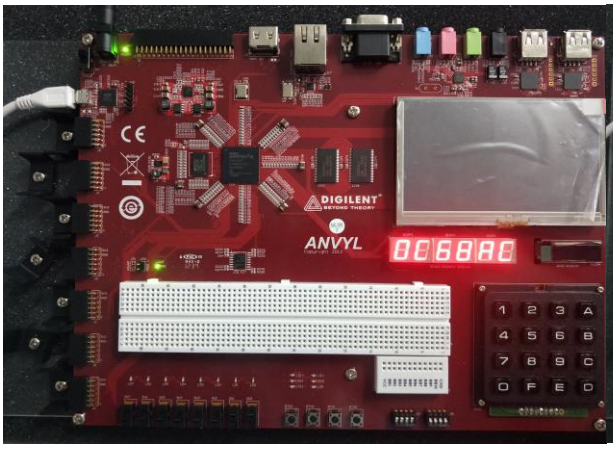
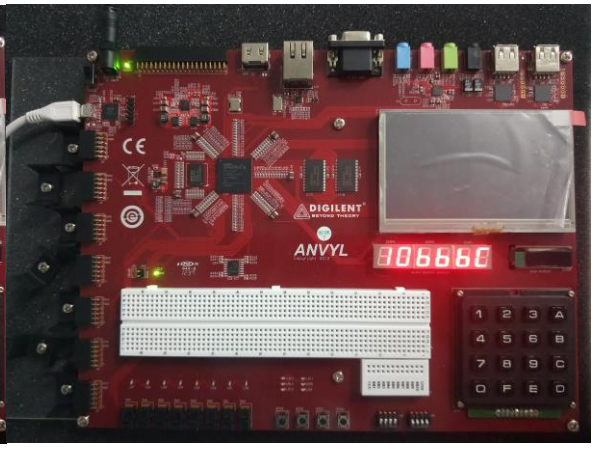
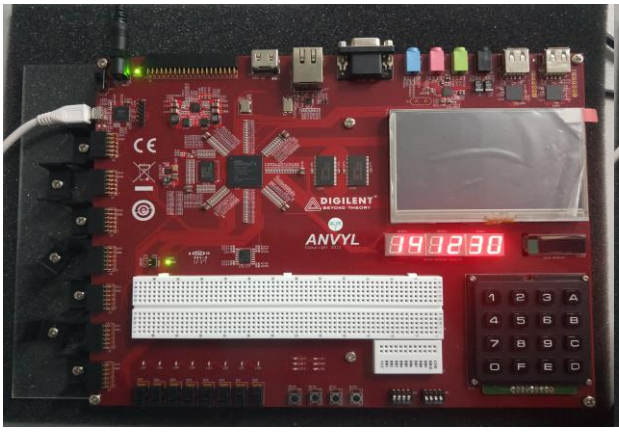
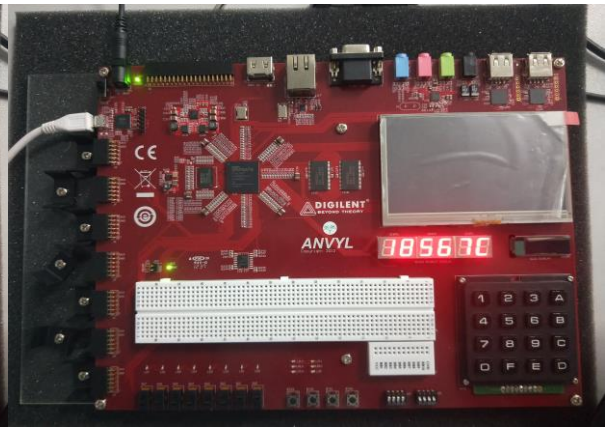
32'h10220001 ==》beq R1,R2,1 ALU=R1-R2=0x10ef0000,PC=32:

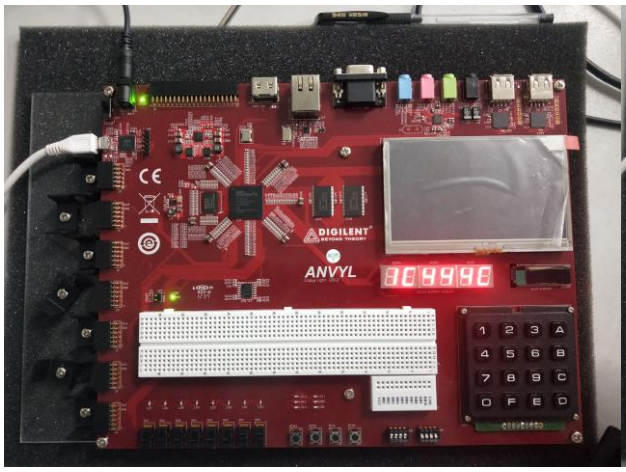


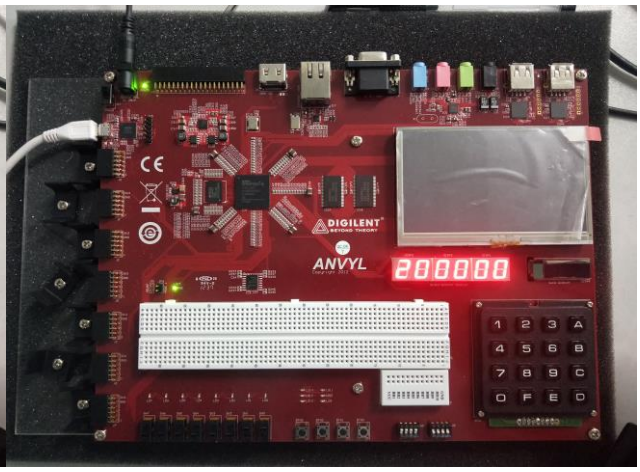
开发板（此截图是按照教师设计的指令进行拍照）：

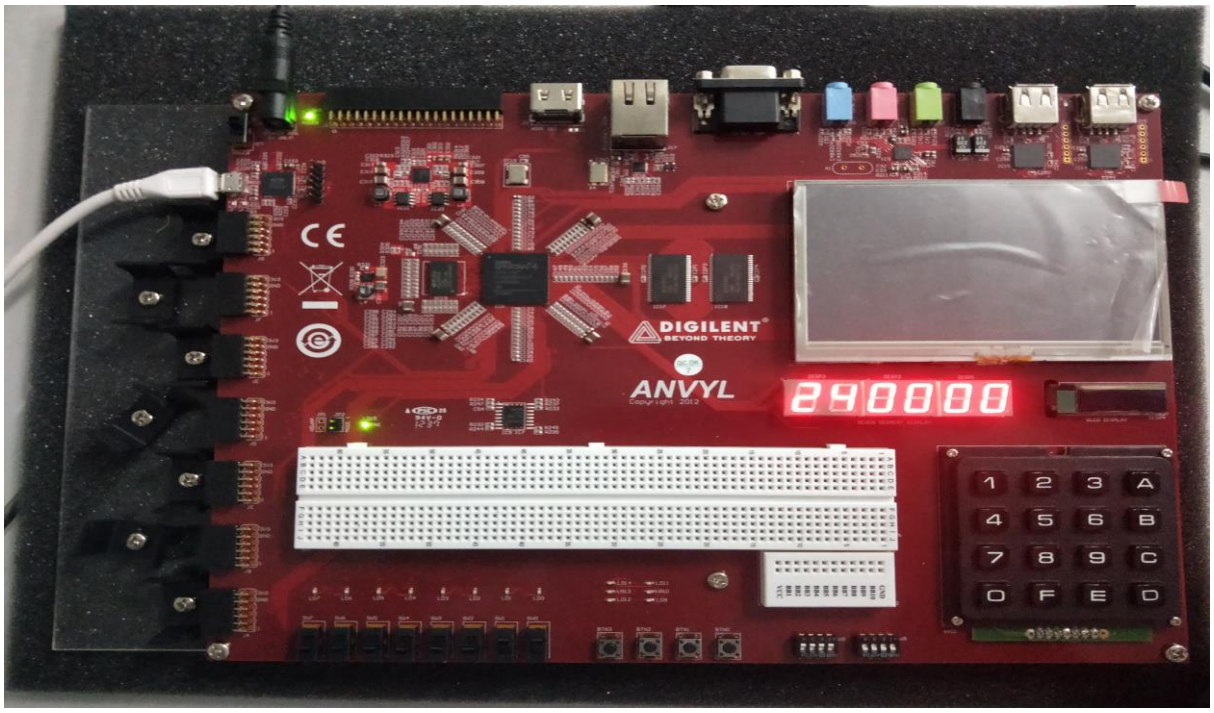
```
// assign ram[5'h00]=0;                    //
//assign ram[5'h01]=32'h3c011234;      //lui R1,0x1234;   ALU=0x12340000
//assign ram[5'h02]=32'h3c025678;      //lui R2,0x5678;   ALU=0x56780000
//assign ram[5'h03]=32'h00221820;      //add R3,R1,R2;   ALU=0x68ac0000
```

```
//assign ram[5'h04]=32'h00221822;     //sub R3,R1,R2;   ALU=0xbbbc0000
//assign ram[5'h05]=32'h00221824;     //and R3,R1,R2;   ALU=0x12300000
//assign ram[5'h06]=32'h00221825;     //or R3,R1,R2;    ALU=0x567c0000
//assign ram[5'h07]=32'h00221826;     //xor R3,R1,R2;   ALU=0x444c0000
//assign ram[5'h08]=32'h00631826;     //xor R3,R3,R3;   ALU=0x00000000
//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'h1021fff4;     //beq R1,R1,0xfff4
```







八、实验结论、心得体会和改进建议

1.在实现 Single_Cycle_Computer 的时候，进行模块调用时候要进行实例化。

2.阅读并分析每个元器件的功能，搞清楚每一个单元的作用是非常有必要的。

3.写的时候要对每条指令的数据通路十分了解，这样才能对内部的数据流向有更加清楚的认识，写起来更加的速度。

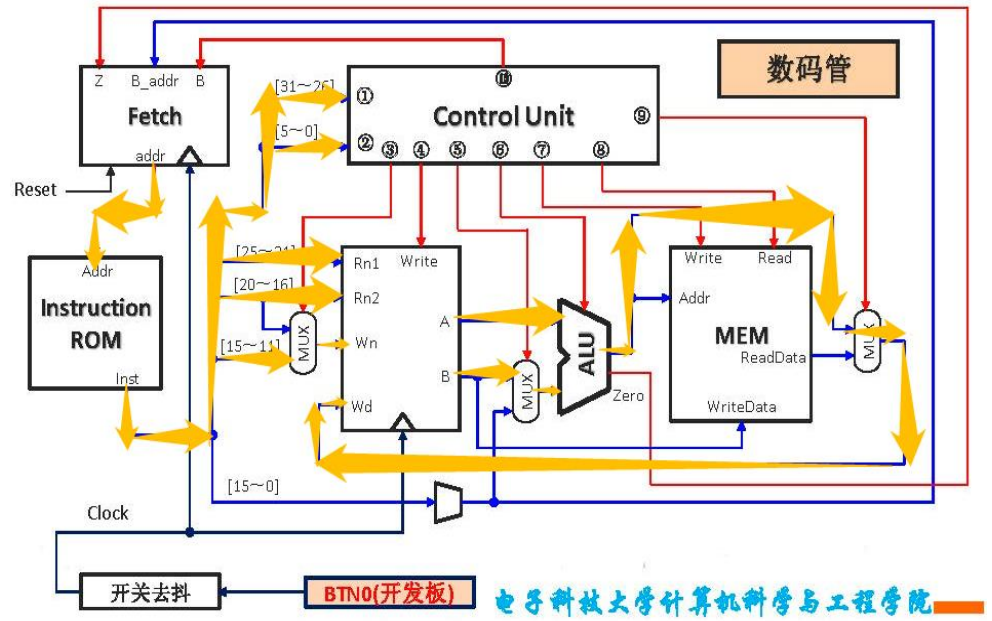
电 子 科 技 大 学

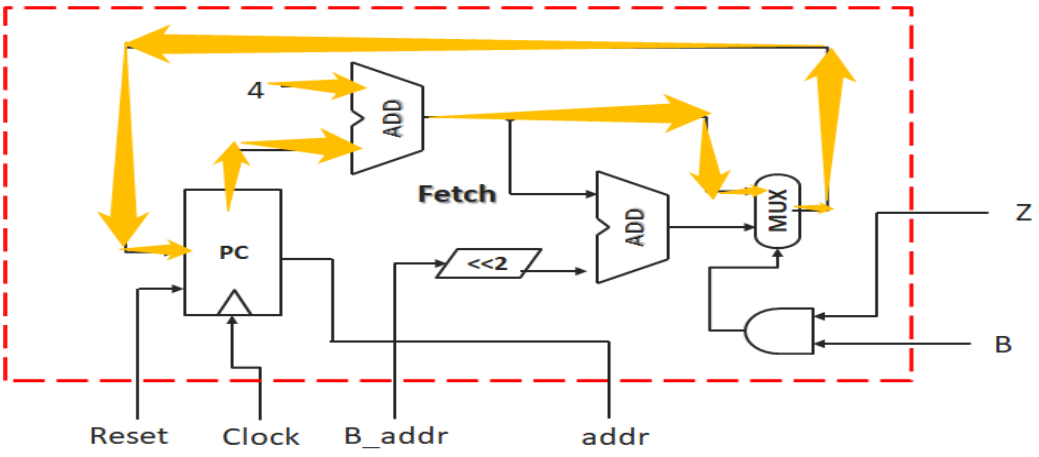
实 验 报 告

课后作业

指令分析以及数据通路：

1. R-TYPE 的 Add 指令，R[rd]R[rs]+R[rt]





Func：Add 指令对应的 func 为 100000。

Op：由于指令为 R 型，所以 op 字段置为 000000，

RegDst：因为是向 rd 寄存器写内容，所以控制信号为 1。

RegWrite：因为进行的操作是向寄存器写，所以控制信号为 1.

ALUSrc：因为此时数据是从 B 中送出，所以二选一多路选择器的控制信号为 0。

ALUop[2:0]:执行的为加法指令，控制信号为 000。

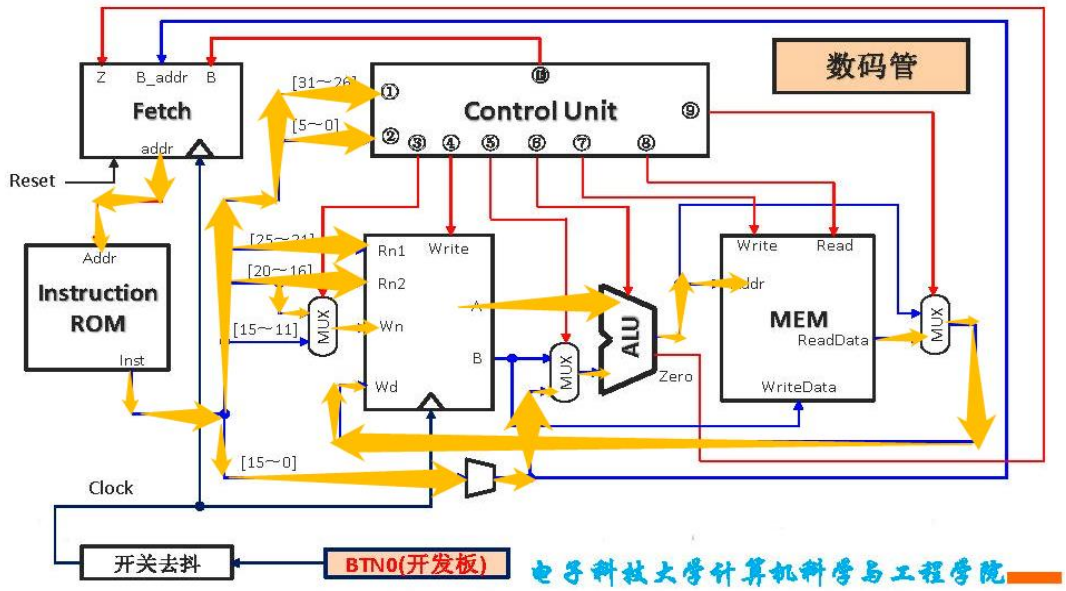
MemWrite：因为此时并不需要向存储器中写入数据且防止数据写入，此时控制信号为 0。

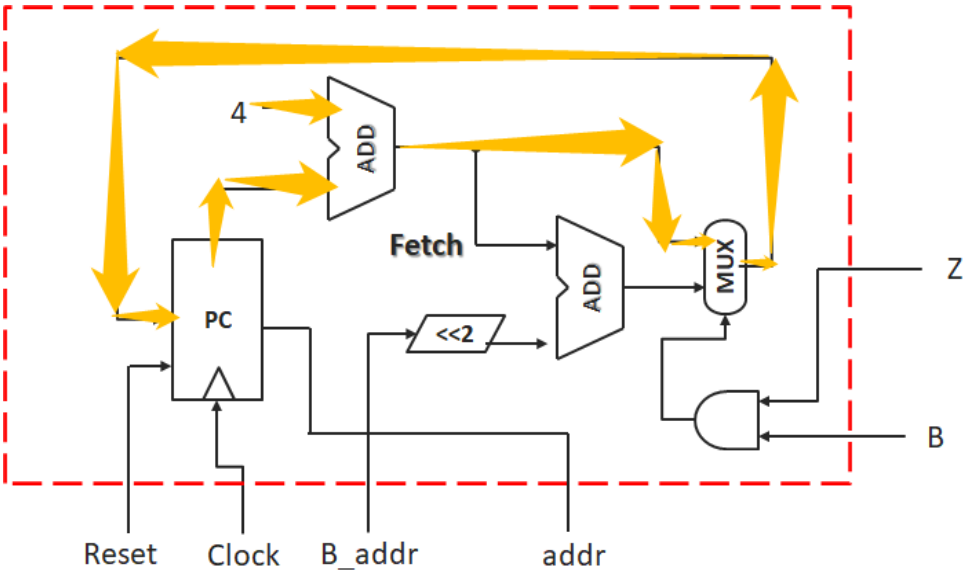
MemRead：因为此时不需要从存储器中读数据，此时控制信号为 0。

MemtoReg：因为此时是从寄存器那条线回写到 rd 寄存器中，此时控制信号为 0。

Branch：因为此操作为 PC 正常工作，不为跳转指令，只需做 PC=PC+4。所以 Branch 的信号的为 0。

2. Lw 指令，R[rt]←Data Memory{R[rs]+SignExt[imm16]}





Func：非 R 型指令，与 func 字段无关。

Op：指令为 lw，op 字段置为 100011，

RegDst：因为是向 rt 寄存器写内容，所以控制信号为 0。

RegWrite：因为进行的操作是向寄存器写，所以控制信号为 1.

ALUSrc：因为此时数据是从扩展后的数据中送出，所以二选一多路选择器的控制信号为 1。

ALUop[2:0]:执行的为加法指令，控制信号为 000。

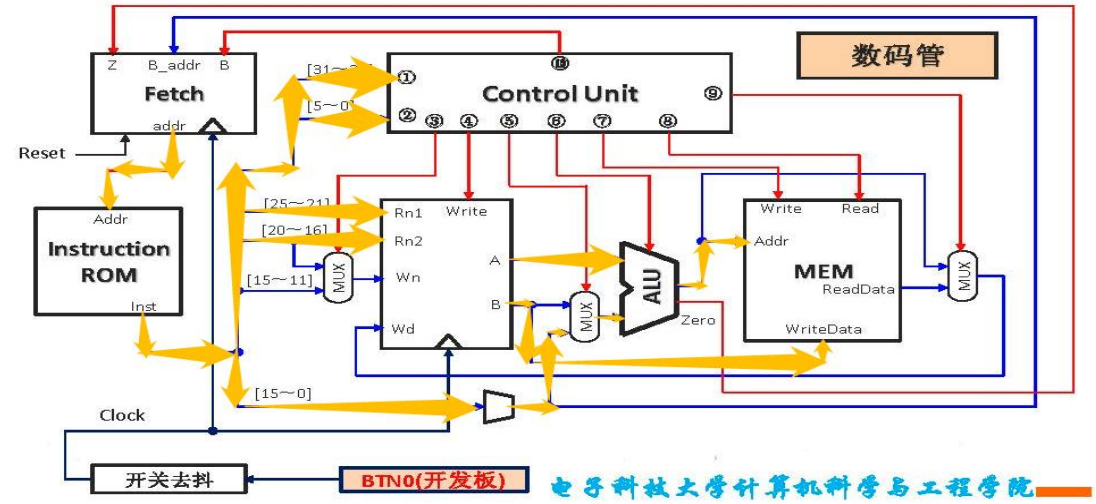
MemWrite：因为此时并不需要向存储器中写入数据且防止数据写入，此时控制信号为 0。

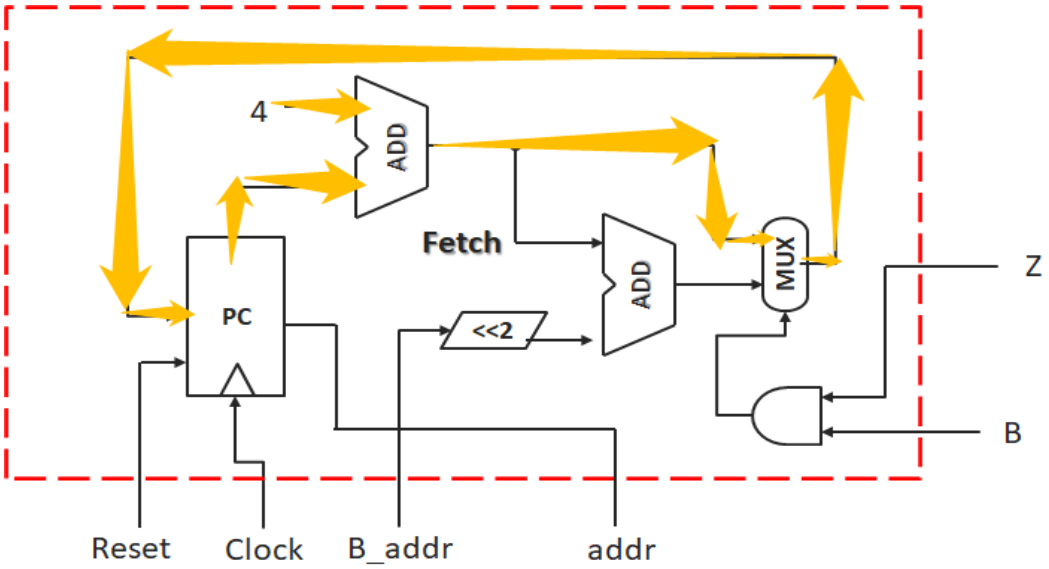
MemRead：因为此时需要从存储器中读数据，此时控制信号为 1。

MemtoReg：因为此时从存储器那条线回写到 rt 寄存器中，此时控制信号为 1。

Branch：因为此操作为 PC 正常工作，不为跳转指令，只需做 PC=PC+4。所以 Branch 的信号的为 0。

3. Sw 指令，M{R[rs]+SignExt[imm16]}←R[rt]





Func：非 R 型指令，与 func 字段无关。

Op：指令为 sw，op 字段置为 101011，

RegDst：因为不向寄存器写内容，控制信号为 x。且由于 RegWrite 置为了 0，所以为无关项。

RegWrite：因为进行的操作不需要向寄存器写，所以控制信号为 0。

ALUSrc：因为此时数据是从扩展后的数据中送出，所以二选一多路选择器的控制信号为 1。

ALUop[2:0]:执行的为加法指令，控制信号为 000。

MemWrite：因为此时需要向存储器中写入数据，此时控制信号为 1。

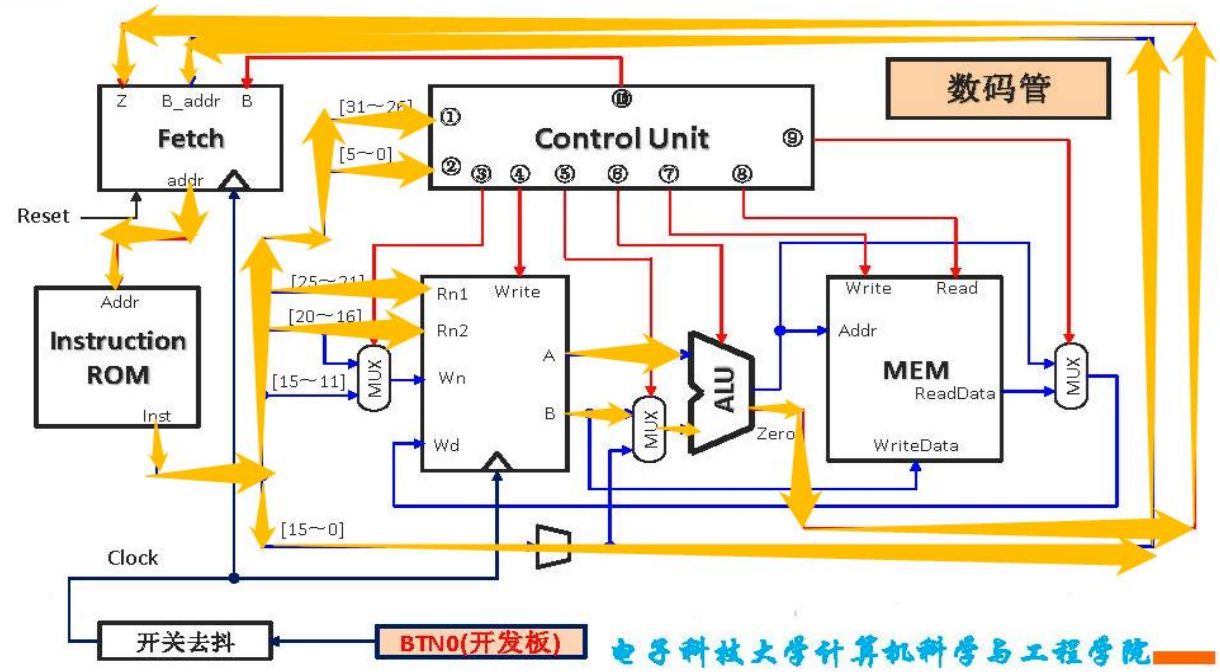
MemRead：因为此时不需要从存储器中读数据，此时控制信号为 0。

MemtoReg：因为此时是不需要回写到寄存器中的，此时控制信号为 x。且由于 RegWrite 置为了 0，所以为无关项。

Branch：因为此操作为 PC 正常工作，不为跳转指令，只需做 PC=PC+4。所以 Branch 的信号的为 0。

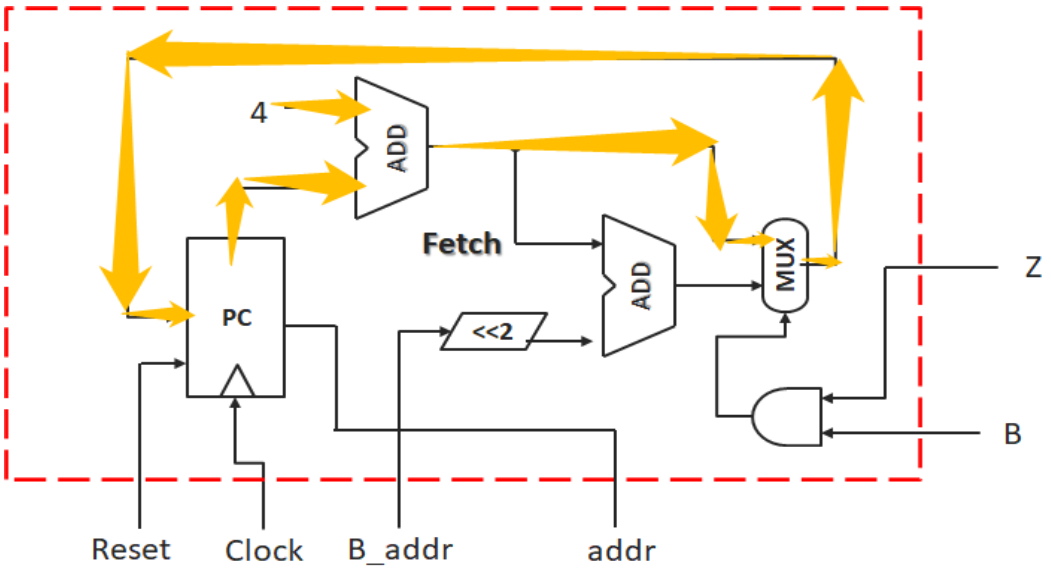
4. Beq 指令，if(R[rs]-R[rt]==0) then Zero←1；else Zero←0

If（Zero==1） then PC=PC+4+SignExt[Imm16]*4，else PC=PC+4

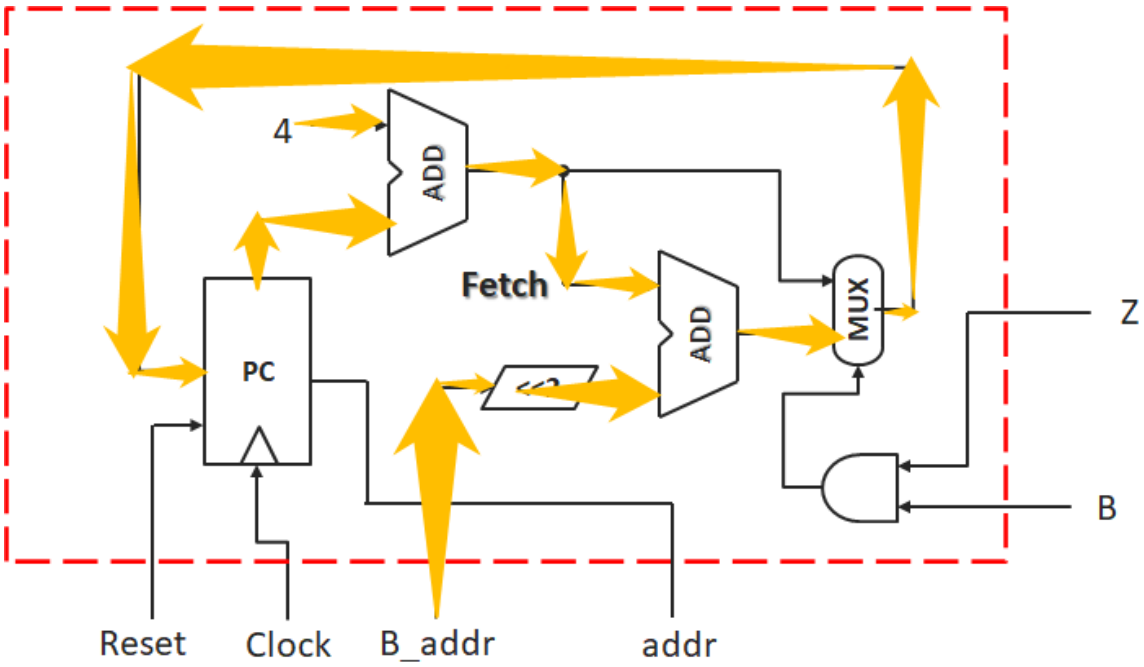


取指部件：

Zero=0:



Zero=1:



Func：非 R 型指令，与 func 字段无关。

Op：指令为 Beq，op 字段置为 000100，

RegDst：因为不向寄存器写内容，所以控制信号为 x。且由于 RegWrite 置为了 0，所以为无关项。

RegWrite：因为进行的操作不需要向寄存器写，所以控制信号为 0。

ALUSrc：因为此时数据是从 B 的数据中送出，所以二选一多路选择器的控制信号为 0。

ALUop[2:0]:执行的为减法指令，控制信号为 100。

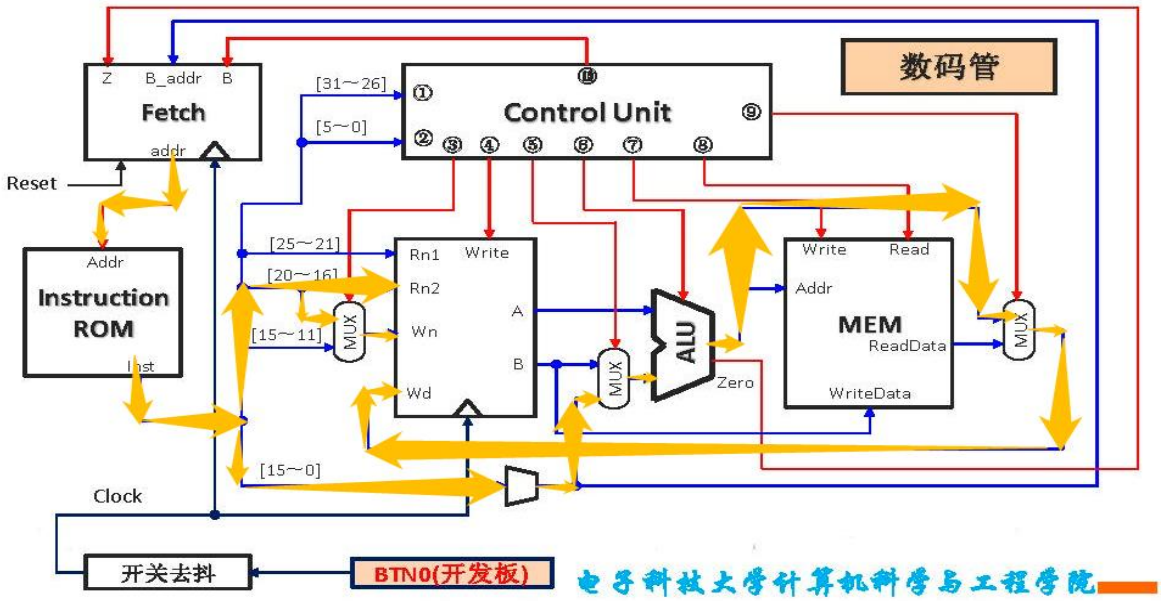
MemWrite：因为此时不需要向存储器中写入数据，此时控制信号为 0。

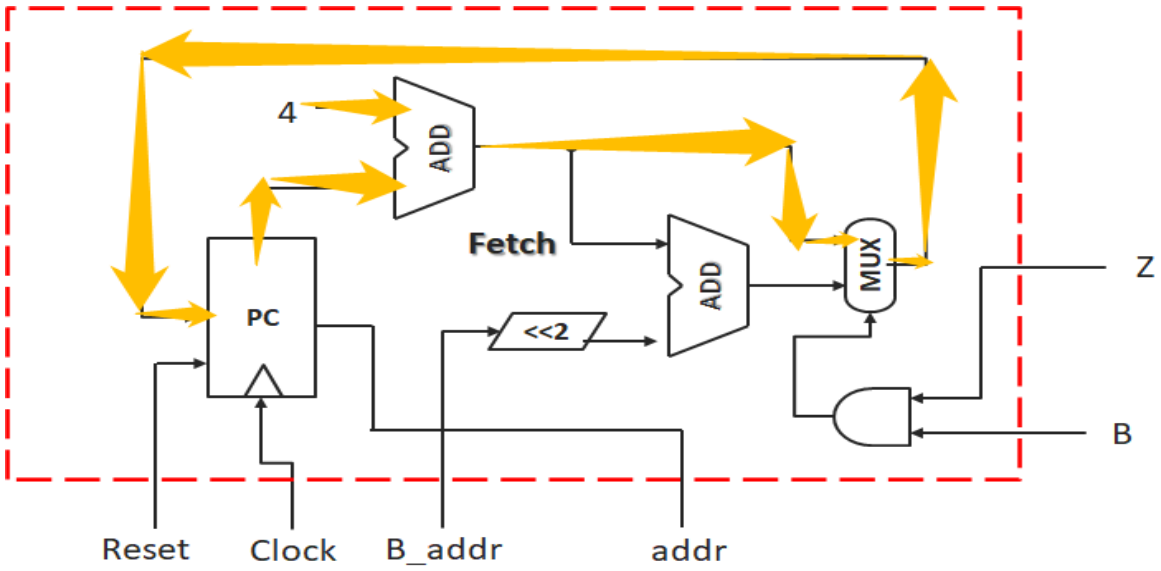
MemRead：因为此时不需要从存储器中读数据，此时控制信号为 0。

MemtoReg：因为此时是不需要回写到寄存器中的，此时控制信号为 x。且由于 RegWrite 置为了 0，所以为无关项。

Branch：因为此操作为跳转指令，Branch 的信号为 1。

5. Lui 指令,R[rt]←{Imm16,0000H}





Func：非 R 型指令，与 func 字段无关。

Op：指令为 lui，op 字段置为 001111，而且 rs 设置为 00000

RegDst：因为不需要寄存器写内容，所以控制信号为 0。

RegWrite：因为进行的操作需要向寄存器写，所以控制信号为 1。

ALUSrc：因为此时数据是从扩展的数据中送出，所以二选一多路选择器的控制信号为 1。

ALUop[2:0]:执行的为减法指令，控制信号为 110。

MemWrite：因为此时不需要向存储器中写入数据，此时控制信号为 0。

MemRead：因为此时不需要从存储器中读数据，此时控制信号为 0。

MemtoReg：因为此时是从寄存器那条线回写到寄存器中的，此时控制信号为 0。

Branch：因为此操作不为跳转指令，Branch 的信号为 0。