注意事项:

1、本文件总共三个课程设计题目，要求所有同学三题都要做，个人单独完成。

2、每位同学的课程设计报告（三个课程设计的报告合并为一个文件提交）及附件，均提交到本课程的思源空间。

3、课程设计提交截止时间为 2025 年 2 月 17 日零点。

课程设计 1

设计目标:对授课内容的单周期 RISC-V 处理器进行扩展,使之能够支持两个额外的指令:lui 和 xor。图 1 所示是一个完整的单周期处理器，图 2 是控制单元，图 3 是 ALU。

表 1 和表 2 是 Main Decoder 和 ALU Decoder 真值表，表 3 列出的是 ImmSrc 编码，图 4 是 RISC-V 单周期处理器的测试程序。

要求提交设计报告，设计报告中需涵盖的内容：

1、请说明在这个设计过程中你共耗时多久，评估一下这次设计的工作量。

2、在图 1 中标记显示所需的修改。

3、如果修改过，则在图 2 和图 3 中标记所加指令（lui 和 xor）所需的修改。

4、修改 Main Decoder, ALU Decoder 和 ImmSrc 真值表，以支持 lui 和 xor。

5、修改提供的 Verilog 代码 top.v，增加 lui 和 xor 指令。

6、修改提供的验证代码 tb_top.v，能测试 lui 和 xor。

7、提供运行图 4 测试程序的仿真波形(按以下列出的顺序:clk、reset、PC、Instr、SrcA、SrcB、ALUResult、DataAdr、WriteData 和 MemWrite，为了便于阅读，所有波形都以十六进制显示)。观察波形并圈出或高亮显示波形，表明正确的值写入了正确的地址，并确保它是可读的。

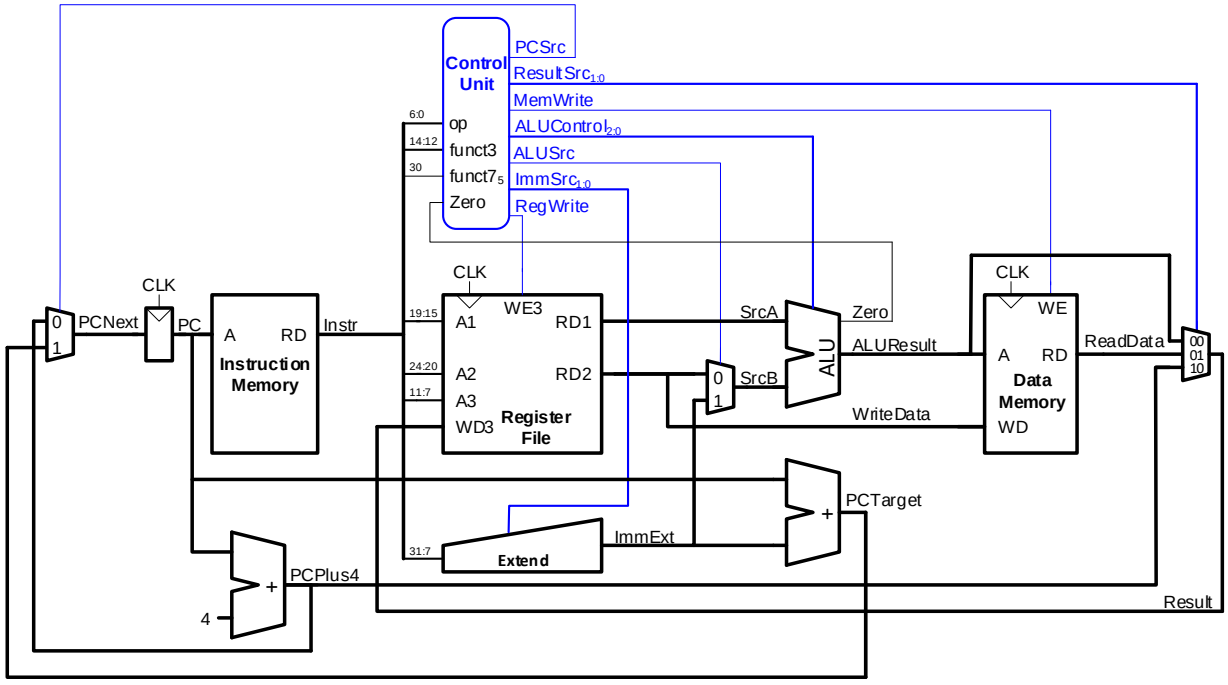

图 1: RISC-V 单周期处理器

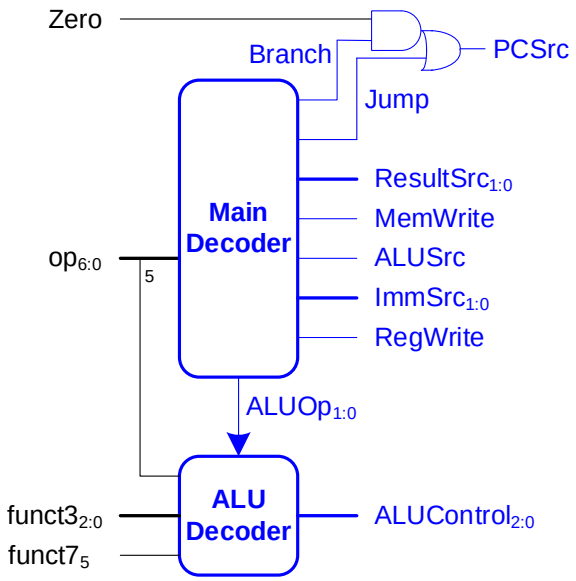

图 2: RISC-V 单周期控制单元

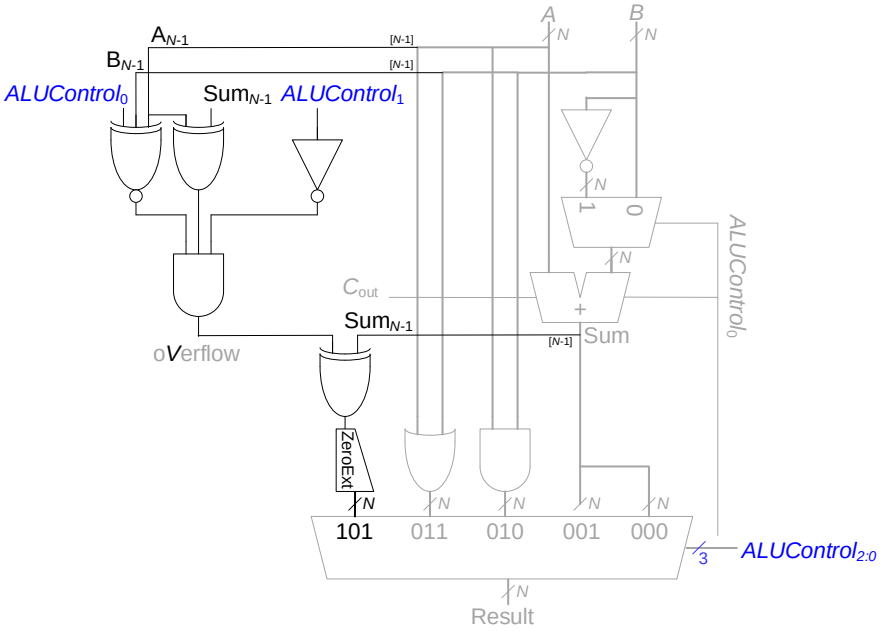

图 3: ALU

表 1. Main Decoder 真值表

Instruction	Opcode	RegWrite	ImmSrc	ALUSrcA	ALUSrcB	MemWrite	ResultSrc	Branch	ALUOp	Jump
lw	0000011	1	00	0	1	0	01	0	00	0
sw	0100011	0	01	0	1	1	xx	0	00	0
R-type	0110011	1	xx	0	0	0	00	0	10	0
beq	1100011	0	10	0	0	0	xx	1	01	0
I-type ALU	0010011	1	00	0	0	0	00	0	10	0
jal	1101111	1	11	x	x	0	10	0	xx	1

表 2. ALU Decoder 真值表

$ALUOp_{1:0}$	$funct3_{2:0}$	$\{op_5, funct7_5\}$	$ALUControl_{2:0}$	Operation
00	x	x	000	Add
01	x	x	001	Subtract
10	000	00, 01, 10	000	Add
	000	11	001	Subtract
	010	x	101	SLT
	110	x	011	OR
	111	x	010	AND

表 3. *ImmSrc* 编码

ImmSrc	*ImmExt*	Type	Description
00	{{20{*Instr*[31]}}, *Instr*[31:20]}	I	12-bit signed immediate
01	{{20{*Instr*[31]}}, *Instr*[31:25], *Instr*[11:7]}	S	12-bit signed immediate
10	{{20{*Instr*[31]}}, *Instr*[7], *Instr*[30:25], *Instr*[11:8], 1'b0}	B	13-bit signed immediate
11	{{12{*Instr*[31]}}, *Instr*[19:12], *Instr*[20], *Instr*[30:21], 1'b0}	J	21-bit signed immediate

```
#       RISC-V Assembly         Description              Address   Machine
main:   addi x2, x0, 5          # x2 = 5                 0         00500113
        addi x3, x0, 12         # x3 = 12                4         00C00193
        addi x7, x3, -9         # x7 = (12 - 9) = 3      8         FF718393
        or   x4, x7, x2         # x4 = (3 OR 5) = 7      C         0023E233
        and  x5, x3, x4         # x5 = (12 AND 7) = 4    10        0041F2B3
        add  x5, x5, x4         # x5 = (4 + 7) = 11      14        004282B3
        beq  x5, x7, end        # shouldn't be taken     18        02728863
        slt  x4, x3, x4         # x4 = (12 < 7) = 0      1C        0041A233
        beq  x4, x0, around     # should be taken        20        00020463
        addi x5, x0, 0          # shouldn't happen       24        00000293
around: slt  x4, x7, x2         # x4 = (3 < 5)  = 1      28        0023A233
        add  x7, x4, x5         # x7 = (1 + 11) = 12     2C        005203B3
        sub  x7, x7, x2         # x7 = (12 - 5) = 7      30        402383B3
        sw   x7, 84(x3)         # [96] = 7               34        0471AA23
        lw   x2, 96(x0)         # x2 = [96] = 7          38        06002103
        add  x9, x2, x5         # x9 = (7 + 11) = 18     3C        005104B3
        jal  x3, end            # jump to end, x3 = 0x44 40        008001EF
        addi x2, x0, 1          # shouldn't happen       44        00100113
end:    add  x2, x2, x9         # x2 = (7 + 18)  = 25    48        00910133
        sw   x2, 0x20(x3)       # mem[100] = 25          4C        0221A023
done:   beq  x2, x2, done       # infinite loop          50        00210063
```

图 4. RISC-V 测试程序

课程设计 2

一、 用 verilog HDL 语言搭建一个以 ARM Cortex-M0 为处理器核的嵌入式 SOC 系统,系统包含以下几个部分：

(1)ARM Cortex-M0 核
(2)AHB 总线译码器
(3)AHB 总线从设备多路复用器
(4)片上存储器外设
(5)LED 外设
(6)七段数码管
(7)定时器
(8)UART

二、 具体实验要求

系统时钟设置为 20MHz；目标 FPGA 选择 Xilinx Artix-7 系列，型号：xc7a75tffg484-1。

（1）实验一

在系统 reset 的时候直接实现依次点亮 1 个灯、点亮两个灯、……、点亮八个灯（即输出的 8 位宽的 LED 信号依次为 8’b0000_0001、8’b0000_0011、……、8’b1111_1111，在报告中截屏确认）

（2）实验二

控制 8 位的数码管显示“1A2B3C4D”（在报告中截图数码管控制信号 seg）

（3）实验三

配置定时器相关寄存器，设置定时器的时钟为系统时钟的 16 分频时钟，并使得定时器以周期模式运行，load 寄存器初始值为 32’d 5000。

当计数器计到零时触发中断，中断服务程序中实现：点亮一个 LED 灯一段时间并且熄灭。（在报告中截图相关的寄存器配置信号、LED 信号、中断相关信号）

（4）实验四

实现 UART 串口控制器，以 4800 的波特率实现数据 8’b0000_0001,8’b0000_0010,8’b0000_0100,……,8’b1000_0000 的传输。（在报告中截图波特率设置的相关代码以及发送数据、接收数据、中间的串行传输数据）

三、 实验指导

（1） 参考书籍：《Arm Cortex-M0 全可编程 Soc 原理及实现》

电子版链接：https://pan.baidu.com/s/18eg-qLyn6ed_LWAoveSZHg

书籍示例对应代码：http://www.edawiki.com/index.php?doc-view-499.htm

（2） Arm Cortex-M0 开源 IP

链接：https://developer.arm.com/ip-products/designstart/eval

（3） 实验流程：

1）Vivado 中建立工程，将搭建好的 SOC 系统导入其中
2）在 Keil 中编译汇编或者 C 语言程序，生成十六进制可执行文件.hex
3）Vivado 中将.hex 文件初始化为 memory 文件
4）Vivado 中仿真实现实验要求

（具体流程参见上述参考书籍）

四、 作业提交

（1） 提交材料

所有源代码（只要.v 的 verilogHDL 代码文件，不要 vivado 工程文件）打包为一个压缩文件，注明 VLSI 课程设计 2 以及学号姓名，以附件形式提交到思源空间。

（2） 设计报告要求

报告中列出设计流程，简述设计思想，完成设计的软件仿真，截图 vivado 仿真结果，并列出 vivado 中 FPGA 的资源消耗情况。

课程设计 3

一、 硬件密码锁：

密码锁共有 12 个键， 0-9 的数字键， *为取消键， #为确定键

开锁时，需要输入 4 位正确密码后，按#号键确定，密码锁可以打开，注意这里只要最后按#键前 4 位正确即可，密码门打开后 30 秒回到初始态。

如果连续 3 次输错密码，密码门自动死锁 3 分钟。

密码门有一个六位超级密码 230419，输入后可以用户重置并且设置四位开锁密码， 用于设置新的开锁密码时，需要连续输入两次并按#确认，两次必须相同。否则设置失败。

无论是开锁还是设置密码可以按*号来取消。

二、 输入/输出定义如下：

```
module lock(
    input wire clk,
    input wire clr,
    input wire [3:0] din,    //数字键 0-9,需要 4bit
    input wire confirm,      //确定键'#'
    input wire cancel,       //取消键'*'
    output reg unlock_ok,    //成功开锁状态时,输出 1
    output reg reset_ok,     //重设密码成功时,输出 1
    output reg locking       //输错密码 3 次进入锁定状态时,输出 1
);
```

此外，定义一些变量用于保存密码/计时等

```
//初始密码 1234
    reg [3:0] passwd0 = 1;
    reg [3:0] passwd1 = 2;
    reg [3:0] passwd2 = 3;
    reg [3:0] passwd3 = 4;

    //超级密码 230419
    reg [3:0] superwd0 = 2;
    reg [3:0] superwd1 = 3;
    reg [3:0] superwd2 = 0;
    reg [3:0] superwd3 = 4;
    reg [3:0] superwd4 = 1;
    reg [3:0] superwd5 = 9;

    //新密码
    reg [3:0] newpasswd0 = 0, newpasswd1 = 0, newpasswd2 = 0, newpasswd3 = 0;

    //开启状态保持时间
    reg [5:0] open_time = 0;
    //连续输错密码次数
    reg [5:0] wrong_count = 0;
    //锁定状态保持时间
    reg [5:0] lock_time = 0;
```

具体功能使用两个状态机实现。

具体功能使用两个状态机实现。
对于"输入密码——判定密码——开锁/死锁"部分功能，划分一个状态机 S。
对于"输入超级密码——重设密码"功能，划分一个状态机 T。
状态机 S 共有 6 个状态：
S0 - S3: 当前已输入了正确密码的前 0/1/2/3 位。
S4: 已输入正确密码，等待确认。
Open: 已开启，开启状态保持 30 秒。
Lock： 已死锁，死锁状态保持 3 分钟。
状态机 S 定义部分代码如下：

```
    //状态定义
    reg[3:0] present_state_s, next_state_s;//当前状态，下一状态
     parameter S0 = 3'b0, S1 = 3'b1, S2 = 3'b10, S3 = 3'b11;
     parameter S4 = 3'b100;        //等待键入确定键'#'
     parameter Open = 3'b101;      //开启状态
     parameter Lock = 3'b110;      //输错 3 次密码,锁定状态
```

S0-S3 的状态转移为常见的序列检测思路，根据当前输入的数字 din 是否与保存在 passwd0/1/2/3 中的正确密码一致，决定下一个状态为 S_{i+1} 还是 S0。此外，在每个状态中检测确定键和取消键，确定键和取消键都会使状态跳转到 S0，而此时按下确定键会使得 wrong_count(记录的连续输错次数)加 1。如果正确输入了四位密码，进入 S4，此时按下确认键即可跳转到 Open 状态，否则回到 S0。
在 Open 状态开始时，输出 unlock_ok 设为 1，将 reg [5:0] open_time 赋值为保持开启状态的时钟周期数，这里为方便仿真验证设置为 3 个周期。每个时钟上升沿 open_time 递减，直到 3 个周期后为 0，回到 S0 状态。
如果连续输错次数 wrong_count 达到 3，则跳转至 Lock 状态，输出 locking 设为 1，将 reg [5:0] lock_time 赋值为保持死锁状态的时钟周期数，这里为方便仿真验证设置为 3 个周期。每个时钟上升沿 lock_time 递减，直到 3 个周期后为 0，回到 S0 状态。

状态机 T 共有 22 个状态：
T0 - T11: 检测 12 位序列 230419230419 已经正确输入到哪一位
T11 - T15: 输入新密码第 1/2/3/4 位
T16 - T19: 重复新密码的第 1/2/3/4 位
T20: 等待确认。
OK: 重设密码成功。
T0-T11 的状态转移为常见的序列检测思路，T11-T15 将输入的数字暂存在 reg[3:0] newpasswd0/1/2/3 中，T16-T19 分别检测第二次输入的密码是否与寄存器中的密码相同，如果相同，进入 T20 等待按下确认键。
以上各状态中，超级密码出错/再次输出密码不一致/按下取消键都会回到 T0 状态。
T20 时按下确认键进入 OK 状态，将 newpasswd 中的密码复制到 passwd 中，并重置连续输错次数 wrong_count 为 0。
根据以上描述编写密码锁的 HDL 代码，并利用以下 tb 程序验证正确性。

```
`timescale 1ns / 1ps
module tb_lock();
reg clk;
reg clr;
reg cancel;       //取消键'*'
reg [3:0] din;    //数字键
reg confirm;      //确定键'#'
wire unlock_ok;   //开锁,输出 1
wire reset_ok;    //成功重设密码，输出 1
wire locking;     //输错密码锁定状态,输出 1
```

```
initial begin
    clk <= 0;
    confirm <= 0;
    din <= 0;
    cancel <= 0;
    clr <= 1;
    #30
    clr <= 0;

    //依次输入 1 2 3 4 确认,成功开锁
    din <= 1;
    #20
    din <= 2;
    #20
    din <= 3;
    #20
    din <= 4;
    #20
    din <= 0;
    confirm <= 1;

    #20
    confirm <= 0;
    #20
    #20
    #20
    #20

    //依次输入 1 2 3 5 确认,无法开锁
    din <= 1;
    #20
    din <= 2;
    #20
    din <= 3;
    #20
    din <= 5;
    #20
    din <= 0;
    confirm <= 1;


    #20
    confirm <= 0;
    din <= 1;
    #20
    din <= 0;
    #20
    #20
```

#20
#20

//依次输入 230419 230419 6789 6789 确认
//密码从默认的 1234 修改为 6789
din <= 2; #20
din <= 3; #20
din <= 0; #20
din <= 4; #20
din <= 1; #20
din <= 9; #20
din <= 2; #20
din <= 3; #20
din <= 0; #20
din <= 4; #20
din <= 1; #20
din <= 9; #20
din <= 6; #20
din <= 7; #20
din <= 8; #20
din <= 9; #20
din <= 6; #20
din <= 7; #20
din <= 8; #20
din <= 9; #20
confirm <= 1; #20
confirm <= 0; din <= 0; #20

//连续输入 3 次错误密码：
//1234 # 1234 # 1234 #
din <= 1; confirm <= 0; #20
din <= 2; #20
din <= 3; #20
din <= 4; #20
din <= 0; confirm <= 1; #20
din <= 1; confirm <= 0; #20
din <= 2; #20
din <= 3; #20
din <= 4; #20
din <= 0; confirm <= 1; #20
din <= 1; confirm <= 0; #20
din <= 2; #20
din <= 3; #20
din <= 4; #20
din <= 0; confirm <= 1; #20

//连续输错三次后，进入 lock 状态，无法开锁

```
    din <= 6; confirm <= 0; #20
    din <= 7; #20
    din <= 8; #20
    din <= 9; #20
    din <= 0; confirm <= 1; #20

    //中途取消 test
    din <= 6; confirm <= 0; #20
    din <= 7; #20
    din <= 8; #20
    cancel <= 1; din <= 0; #20
    din <= 9; cancel <= 0; #20
    din <= 0; confirm <= 1; #20
    confirm <= 0;


end

always #10 clk = ~clk;  //晶振周期 20ns


lock test_lock(
   .clk(clk),
    .clr(clr),
    .din(din),
    .confirm(confirm),
    .cancel(cancel),
    .unlock_ok(unlock_ok),
    .reset_ok(reset_ok),
    .locking(locking)
);

endmodule
```

设计报告提供运行以上测试程序的仿真波形图。观察波形并圈出或高亮显示波形，截图，依次要求符合以下预期。
第一部分：依次输入 1、2、3、4、Confirm，输出 unlock_ok，保持三个周期。
第二部分：依次输入 1、2、3、5、Confirm，回到状态 S0，wrong_count = 1
第三部分：依次输入 230419 230419 6789 6789、Confirm，密码从 1234 修改为 6789。
此时不关心状态机 S 的状态。按下确认键后，重设密码成功，则 wrong_count 会清零。
第四部分：依次输入 1234 confirm 1234 confirm 1234 confirm wrong_count 达到 3，进入死锁状态，持续三个周期，在此期间输入 6789Confirm，也无法开锁。
第五部分：依次按下 6 7 8 取消 9 确认。可见按下取消键后回到了 S0 状态。

为了便于阅读，所有波形都以 10 进制显示。观察波形并圈出或高亮显示波形，表明正确的输入和输出。所有源代码（只要.v 的 verilogHDL 代码文件）打包为一个压缩文件，注明 VLSI 课程设计 3 以及学号姓名，以附件形式提交到思源空间。