

# 组成原理课程第三次实报告

## 实验名称：寄存器堆实现

学号：22122266 姓名：张恒硕 班次：0416

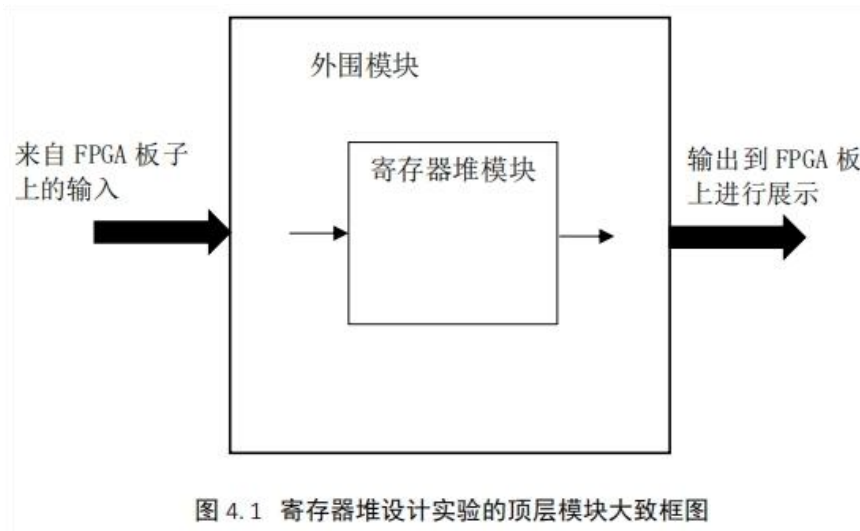
### 一、实验目的

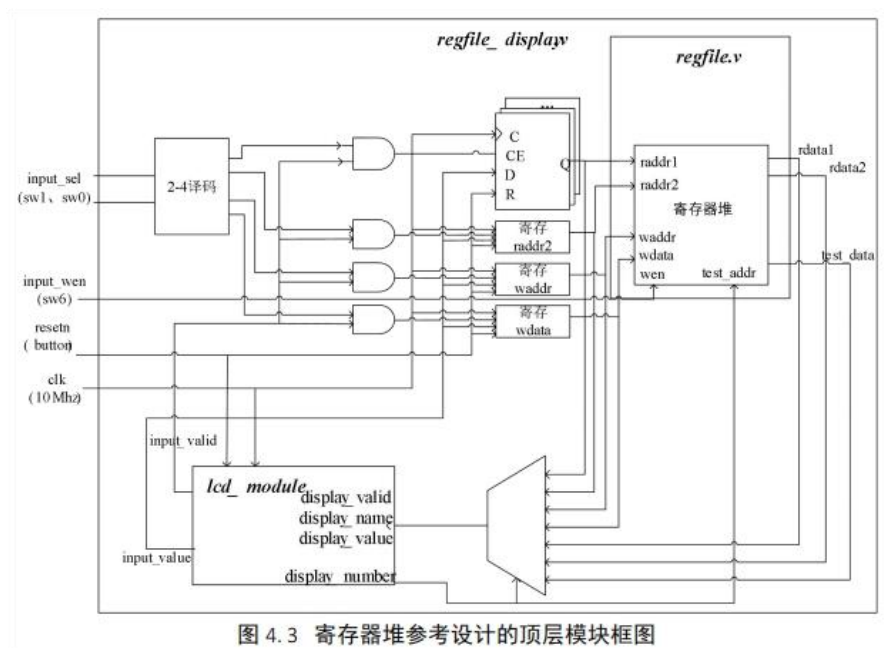
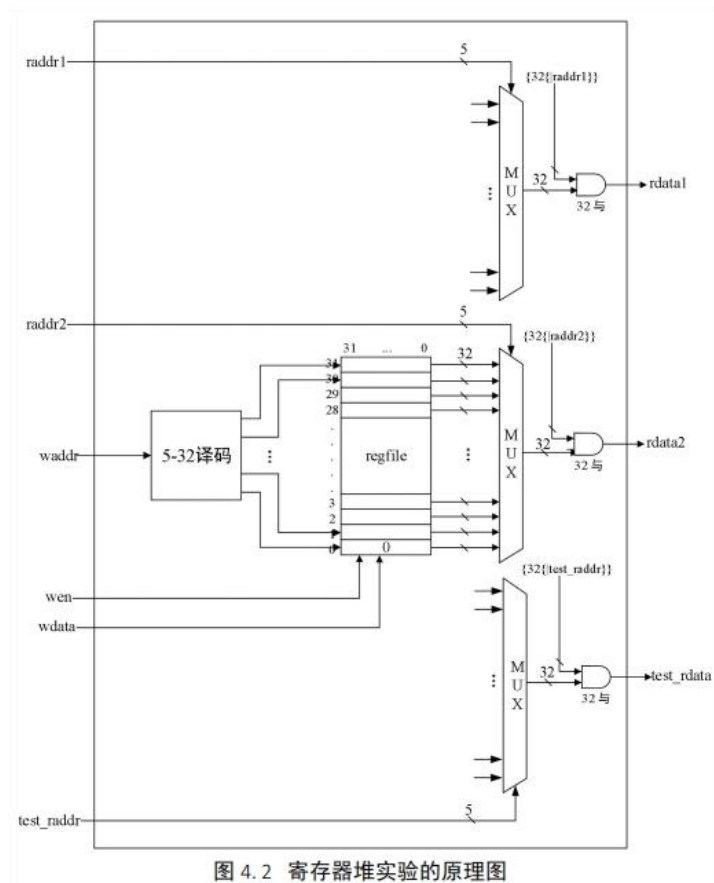
1. 熟悉并掌握 MIPS 计算机中寄存器堆的原理和设计方法。
2. 初步了解 MIPS 指令结构和源/目的操作数的概念。
3. 熟悉并运用 verilog 语言进行电路设计。
4. 为后续设计 cpu 的实验打下基础。

### 二、实验内容说明

1. 学习 MIPS 计算机中寄存器堆的设计及原理，包括寄存器数量、有无特殊设置寄存器、mips 指令索引方法等。
2. 设计实验方案，包括 1 个写端口和 2 个读端，32 个 32 位地址，画出结构框图。
3. 使用 verilog 编写相应代码。
4. 仿真编写的代码，得到正确的波形图。
5. 将以上设计作为一个单独的模块，设计一个外围模块去调用之。外围模块中需调用封装好的 LCD 触摸屏模块，显示寄存器堆的读写端口地址和数据，能扫描出所有寄存器的值显示在 LCD 触摸屏上，并需利用触摸功能输入寄存器堆的读写地址和写数据。
6. 将编写的代码进行综合布局布线，并下载到实验箱中的 FPGA 板上进行演示。
7. 重复上述步骤，通过修改代码获得 16 个 32 位地址的寄存器，并仿真、演示。

### 三、实验原理图





以上给出了寄存器堆的原理。其实现的是 32 个 32 位地址的寄存器。

通过修改译码器为 4-16 译码,并将地址减少到 16 个,可实现 16 个 32 位地址的寄存器。

#### 四、实验步骤

以下给出了 32 个 32 位地址的寄存器的代码。

```
module regfile(
    input          clk,
```

```

input          wen,
input  [4 :0]  raddr1,
input  [4 :0]  raddr2,
input  [4 :0]  waddr,
input  [31:0]  wdata,
output reg [31:0] rdata1,
output reg [31:0] rdata2,
input  [4 :0]  test_addr,
output reg [31:0] test_data
);
reg [31:0] rf[31:0];

// three ported register file
// read two ports combinationally
// write third port on rising edge of clock
// register 0 hardwired to 0

always @(posedge clk)
begin
    if (wen)
    begin
        rf[waddr] <= wdata;
    end
end

//读端口 1
always @(*)
begin
    case (raddr1)
        5'd1 : rdata1 <= rf[1 ];
//省略中间部分
        5'd31: rdata1 <= rf[31];
        default : rdata1 <= 32'd0;
    endcase
end

//读端口 2
always @(*)
begin
    case (raddr2)
        5'd1 : rdata2 <= rf[1 ];
//省略中间部分
        5'd31: rdata2 <= rf[31];
        default : rdata2 <= 32'd0;
    endcase
end

```

```

end
    //调试端口，读出寄存器值显示在触摸屏上
    always @(*)
    begin
        case (test_addr)
            5'd1 : test_data <= rf[1 ];
            //省略中间部分
            5'd31: test_data <= rf[31];
            default : test_data <= 32'd0;
        endcase
    end
Endmodule

```

代码先声明了时钟输入信号 `clk`、使能信号 `wen`、读端口地址输入 `raddr1` 和 `raddr2`、写端口地址输入 `waddr`、写端口数据输入 `wdata`、读端口数据输出 `rdata1` 和 `rdata2`、调试端口地址输入 `test_addr`、调试端口数据输出 `test_data`。

其后，针对不同的 `clk` 和 `wen` 信号，实现了不同的逻辑：

写端口逻辑：`clk` 上升沿且 `wen` 为高时，将 `wdata` 写入到 `waddr` 指定的寄存器。

读端口 1 逻辑：组合逻辑，根据 `raddr1` 读取对应寄存器的内容，读端口 2 则读取不同地址。

调试端口逻辑：根据 `test_addr` 读取对应寄存器的内容，并显示在触摸屏上。

以下给出了 16 个 32 位地址的寄存器的代码，对比上述代码的修改部分已标红。

```

regfile.v
module regfile(
    input          clk,
    input          wen,
    input  [3 :0]  raddr1,
    input  [3 :0]  raddr2,
    input  [3 :0]  waddr,
    input  [31:0]  wdata,
    output reg [31:0] rdata1,
    output reg [31:0] rdata2,
    input  [3 :0]  test_addr,
    output reg [31:0] test_data
);
    reg [31:0] rf[15:0];

    // three ported register file
    // read two ports combinationally
    // write third port on rising edge of clock
    // register 0 hardwired to 0

    always @(posedge clk)
    begin
        if (wen)
        begin

```

```

        rf[waddr] <= wdata;
    end
end

//读端口 1
always @(*)
begin
    case (raddr1)
        5'd1 : rdata1 <= rf[1 ];
//省略中间部分
        5'd15: rdata1 <= rf[15];
        default : rdata1 <= 16'd0;
    endcase
end
//读端口 2
always @(*)
begin
    case (raddr2)
        5'd1 : rdata2 <= rf[1 ];
//省略中间部分
        5'd15: rdata2 <= rf[15];
        default : rdata2 <= 16'd0;
    endcase
end
//调试端口，读出寄存器值显示在触摸屏上
always @(*)
begin
    case (test_addr)
        5'd1 : test_data <= rf[1 ];
//省略中间部分
        5'd15: test_data <= rf[15];
        default : test_data <= 16'd0;
    endcase
end
endmodule

```

regfile\_display.v

```

module regfile_display(
    //时钟与复位信号
    input clk,
    input resetn,    //后缀“n”代表低电平有效

    //拨码开关，用于产生写使能和选择输入数
    input wen,
    input [1:0] input_sel,

```

```

//led 灯，用于指示写使能信号，和正在输入什么数据
output led_wen,
output led_waddr,    //指示输入写地址
output led_wdata,    //指示输入写数据
output led_raddr1,   //指示输入读地址 1
output led_raddr2,   //指示输入读地址 2

//触摸屏相关接口，不需要更改
output lcd_rst,
output lcd_cs,
output lcd_rs,
output lcd_wr,
output lcd_rd,
inout[15:0] lcd_data_io,
output lcd_bl_ctr,
inout ct_int,
inout ct_sda,
output ct_scl,
output ct_rstn
);
//-----{LED 显示}begin
    assign led_wen      = ~wen;
    assign led_raddr1 = ~(input_sel==2'd0);
    assign led_raddr2 = ~(input_sel==2'd1);
    assign led_waddr  = ~(input_sel==2'd2);
    assign led_wdata  = ~(input_sel==2'd3);
//-----{LED 显示}end

//-----{调用寄存器堆模块}begin
    wire [31:0] test_data;
    wire [3 :0] test_addr;

    reg  [3 :0] raddr1;
    reg  [3 :0] raddr2;
    reg  [3 :0] waddr;
    reg  [31:0] wdata;
    wire [31:0] rdata1;
    wire [31:0] rdata2;
    regfile rf_module(
        .clk    (clk    ),
        .wen    (wen    ),
        .raddr1 (raddr1),
        .raddr2 (raddr2),

```

```

        .waddr (waddr ),
        .wdata (wdata ),
        .rdata1(rdata1),
        .rdata2(rdata2),
        .test_addr(test_addr),
        .test_data(test_data)
    );
//-----{调用寄存器堆模块}end

//-----{调用触摸屏模块}begin-----//
//-----{实例化触摸屏}begin

    reg          display_valid;
    reg [39:0] display_name;
    reg [31:0] display_value;
    wire [5 :0] display_number;
    wire          input_valid;
    wire [31:0] input_value;

    lcd_module lcd_module(
        .clk          (clk          ),    //10Mhz
        .resetn       (resetn       ),

        //调用触摸屏的接口
        .display_valid (display_valid ),
        .display_name  (display_name  ),
        .display_value (display_value ),
        .display_number (display_number),
        .input_valid   (input_valid   ),
        .input_value   (input_value   ),

        //lcd 触摸屏相关接口，不需要更改
        .lcd_rst       (lcd_rst       ),
        .lcd_cs        (lcd_cs        ),
        .lcd_rs        (lcd_rs        ),
        .lcd_wr        (lcd_wr        ),
        .lcd_rd        (lcd_rd        ),
        .lcd_data_io    (lcd_data_io   ),
        .lcd_bl_ctr     (lcd_bl_ctr    ),
        .ct_int         (ct_int        ),
        .ct_sda         (ct_sda        ),
        .ct_scl         (ct_scl        ),
        .ct_rstn        (ct_rstn       )
    );

```

```

//-----{实例化触摸屏}end

//-----{从触摸屏获取输入}begin
    //16个寄存器显示在7~22号的显示块，故读地址为（display_number-1）
    assign test_addr = display_number-5'd7;
    //当 input_sel 为 2'b00 时，表示输入数为读地址 1，即 raddr1
    always @(posedge clk)
    begin
        if (!resetn)
        begin
            raddr1 <= 5'd0;
        end
        else if (input_valid && input_sel==2'd0)
        begin
            raddr1 <= input_value[3:0];
        end
    end
end

//当 input_sel 为 2'b01 时，表示输入数为读地址 2，即 raddr2
always @(posedge clk)
begin
    if (!resetn)
    begin
        raddr2 <= 5'd0;
    end
    else if (input_valid && input_sel==2'd1)
    begin
        raddr2 <= input_value[3:0];
    end
end

//当 input_sel 为 2'b10 时，表示输入数为写地址，即 waddr
always @(posedge clk)
begin
    if (!resetn)
    begin
        waddr <= 5'd0;
    end
    else if (input_valid && input_sel==2'd2)
    begin
        waddr <= input_value[3:0];
    end
end
end

```



```

//当 input_sel 为 2'b11 时，表示输入数为写数据，即 wdata
always @(posedge clk)
begin
    if (!resetn)
    begin
        wdata <= 32'd0;
    end
    else if (input_valid && input_sel==2'd3)
    begin
        wdata <= input_value;
    end
end
end
//-----{从触摸屏获取输入}end

//-----{输出到触摸屏显示}begin
always @(posedge clk)
begin
    if (display_number >6'd6 && display_number <6'd23 )
    begin //块号 7~38 显示 32 个通用寄存器的值
        display_valid <= 1'b1;
        display_name[39:16] <= "REG";
        display_name[15: 8] <= {4'b0011,4'b0000};
        display_name[7 : 0] <= {4'b0011,test_addr[3:0]};
        display_value      <= test_data;
    end
    else
    begin
        case(display_number)
            6'd1 : //显示读端口 1 的地址
            begin
                display_valid <= 1'b1;
                display_name <= "RADD1";
                display_value <= raddr1;
            end
            6'd2 : //显示读端口 1 读出的数据
            begin
                display_valid <= 1'b1;
                display_name <= "RDAT1";
                display_value <= rdata1;
            end
            6'd3 : //显示读端口 2 的地址
            begin
                display_valid <= 1'b1;
                display_name <= "RADD2";
            end
        endcase
    end
end
end

```

```

        display_value <= raddr2;
    end
    6'd4 : //显示读端口 2 读出的数据
    begin
        display_valid <= 1'b1;
        display_name <= "RDAT2";
        display_value <= rdata2;
    end
    6'd5 : //显示写端口的地址
    begin
        display_valid <= 1'b1;
        display_name <= "WADDR";
        display_value <= waddr;
    end
    6'd6 : //显示写端口写入的数据
    begin
        display_valid <= 1'b1;
        display_name <= "WDATA";
        display_value <= wdata;
    end
    default :
    begin
        display_valid <= 1'b0;
        display_name <= 40'd0;
        display_value <= 32'd0;
    end
endcase
end
end
//-----{输出到触摸屏显示}end
//-----{调用触摸屏模块}end-----//
endmodule

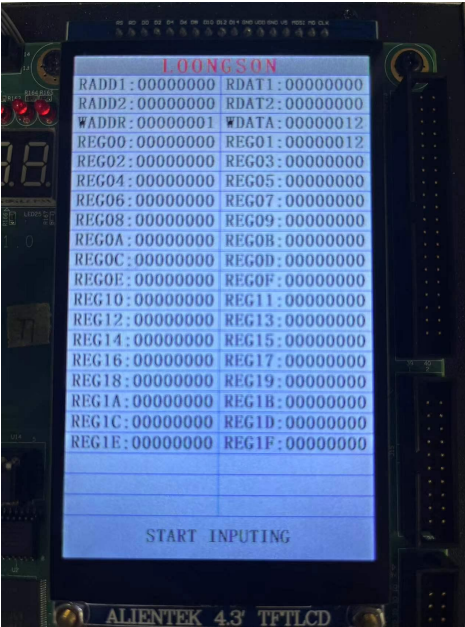
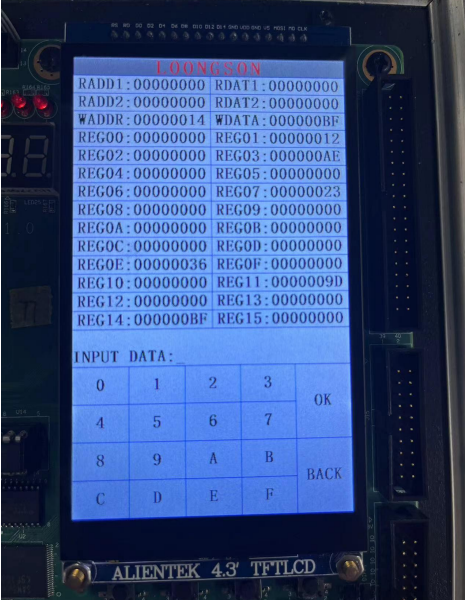
```

在修改过程中，主要将 32 个地址修改为 16 个，并把地址编码由 5 位改成 4 位。

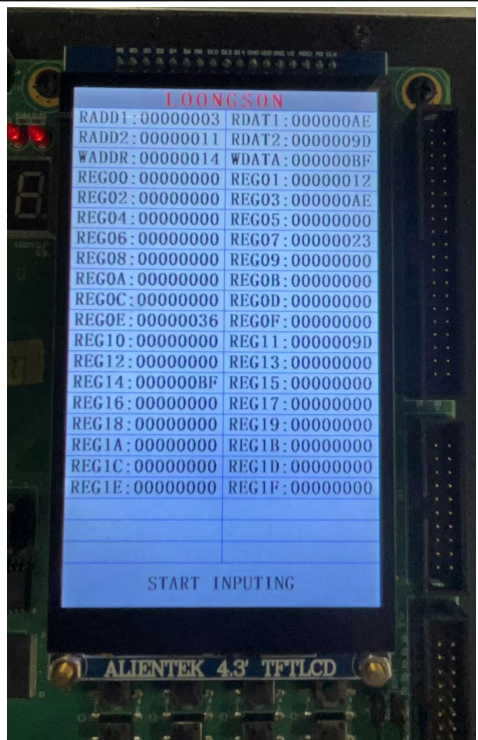
## 五、实验结果分析

### 32 个 32 位地址寄存器

步骤	图像	操作内容
----	----	------

<p>输入样例</p>		<p>将地址 1 的数据设为 12。</p>
<p>输入结果</p>		<p>地址 1 数据为 12。地址 3 数据为 AE。 地址 7 数据为 23。地址 E 数据为 36。 地址 11 数据为 9D。地址 14 数据为 BF。</p>

读数据样例



地址 3 读出数据 AE。地址 11 读出数据 9D。

实验中，给出了 32 个 32 位地址寄存器的演示过程，通过写操作，输入数据与其地址，再用读操作，显示数据与其地址。

16 个 32 位地址寄存器

步骤	图像	操作内容
置零	A screenshot of a Loongson development board's 4.3-inch TFTLCD display. The screen shows a list of 32 registers (RADD1 to REG1F) arranged in two columns. Each register entry includes its name, address, and data value. The data values are mostly 00000000, but RADD1 is 00000000, RADD2 is 00000000, WADDR is 00000000, and REG11 is 00000000. At the bottom of the screen, it says 'START INPUTING'. The board itself is visible in the background, showing various components and connectors.	将寄存器置零

输入样例



LOONGSON

RADD1:00000000	RDAT1:00000000
RADD2:00000001	RDAT2:00000000
WADDR:0000000F	WDATA:00000033
REG00:00000000	REG01:00000000
REG02:00000000	REG03:00000000
REG04:00000000	REG05:00000000
REG06:00000000	REG07:00000000
REG08:00000000	REG09:00000000
REG0A:00000000	REG0B:00000000
REG0C:00000000	REG0D:00000000
REG0E:00000000	REG0F:00000033

START INPUTING

HUAWEI Mate 40 Pro  
Ultra Vision Cine Camera | LEICA

将地址 0F 的数据设为 33。



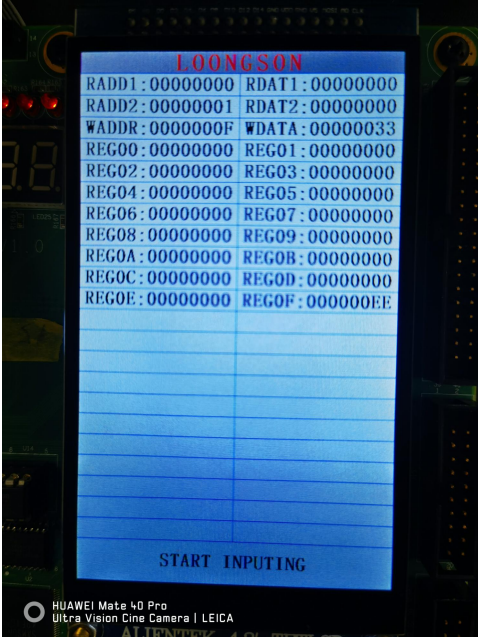
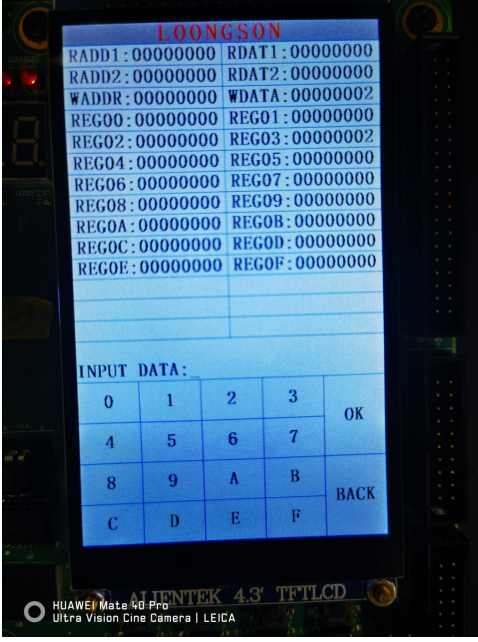
LOONGSON

RADD1:00000007	RDAT1:00000000
RADD2:00000000	RDAT2:00000000
WADDR:0000000E	WDATA:00007777
REG00:00000000	REG01:00000000
REG02:00000000	REG03:00000002
REG04:00000000	REG05:00000000
REG06:00000000	REG07:00000000
REG08:00000000	REG09:00000000
REG0A:00000000	REG0B:00000000
REG0C:00000000	REG0D:00000000
REG0E:00007777	REG0F:00000000

START INPUTING

HUAWEI Mate 40 Pro  
Ultra Vision Cine Camera | LEICA

将地址 0E 的数据设为 7777。

<p>读数据样例 (测试写但不更新的操作效果)</p>		<p>地址 1 读出数据 0。将地址 0F 的数据设为 33，但不更新。</p>
		<p>将地址 0 的数据设为 2，但不更新。</p>
<p>实验中，给出了 16 个 32 位地址寄存器的演示过程。</p>		

六、总结感想

本次实验实现了寄存器堆实现这一内容，并对其进行了改动。在本次实验中，更多地学习了寄存器的实现原理和其代码，提升了相关能力。在修改代码的反复尝试中，加深了对其的理解。