中国科学院大学计算机组成原理（研讨课）

实 验 报 告

学号：2021K8009929001　　姓名：谢嘉楠　　专业：计算机科学与技术
实验序号：1　　实验名称：基本功能部件——寄存器堆和算术逻辑单元

实验一：寄存器堆($register\ file$)的实现

1） 实验目的：

a） 为实现$MIPS/RISC-V$功能型处理器储备基本功能部件。需要设计包含32个$32-bit$通用寄存器的寄存器堆$register\ file$。

b） 每个寄存器可以“同步写，异步读”，即读操作不受时钟信号约束，而写操作由时钟信号控制。

c） 有2个读端口和1个写端口。

d） 0号寄存器：从0号地址读出的值总是常量$32'b0$，不能向0号地址写入数据。

2） 实验原理：

寄存器堆中每个寄存器是 32 位宽，共有 32 个寄存器，这样的结构可以通过二维寄存器来定义（$reg\ \ [Data\_Width-1:0]\ rf\ [Reg\_Num-1:0]$, 其中$Data_Width$表示每个寄存器的位数，$Reg_Num$表示寄存器的个数）。

为了能够访问任意一个寄存器，我们还需要对32个寄存器编号，给每个寄存器一个 5 位的地址，表示0号到31号寄存器。

因为并不是所有指令都会修改通用寄存器（如跳转指令），故寄存器组需要额外提供写使能信号wen。当$wen=1$（有效）时，才可以向相应的寄存器写入数据。

因为写入操作由时钟信号控制，只有在时钟上升沿时刻才能写入数据，所以写操作需要在$always$块中用阻塞赋值实现。而读出操作是异步的，所以可以直接用组合逻辑实现。

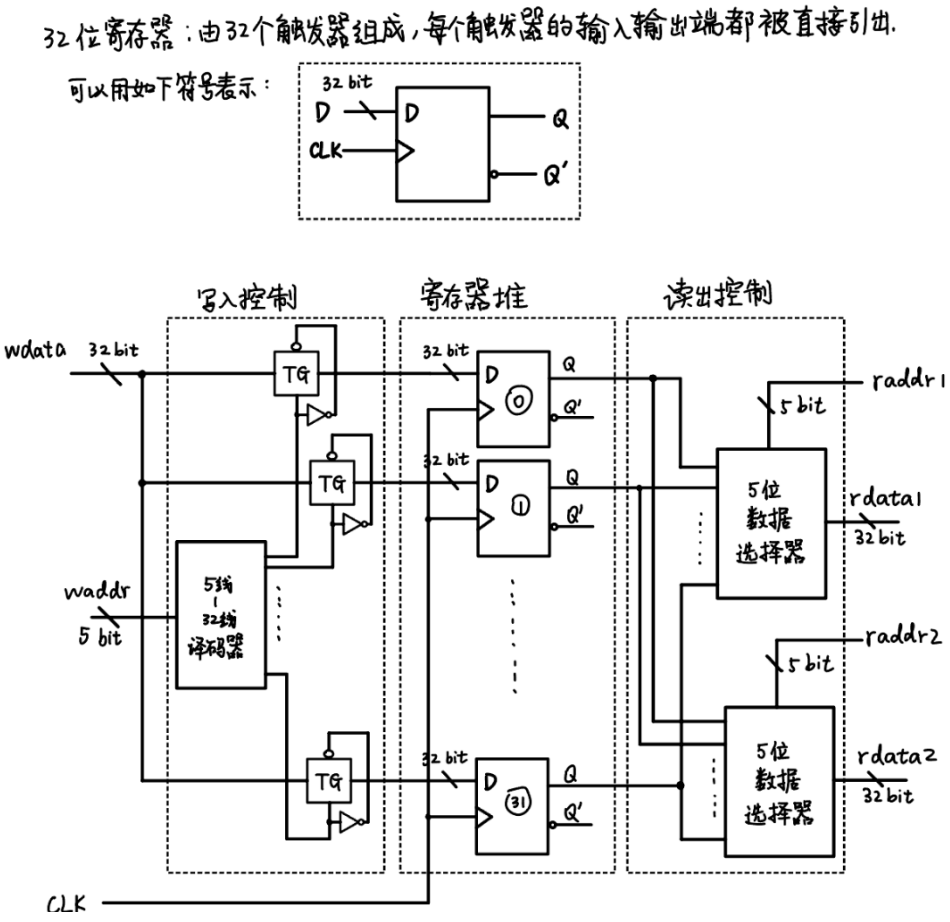
3） RTL代码

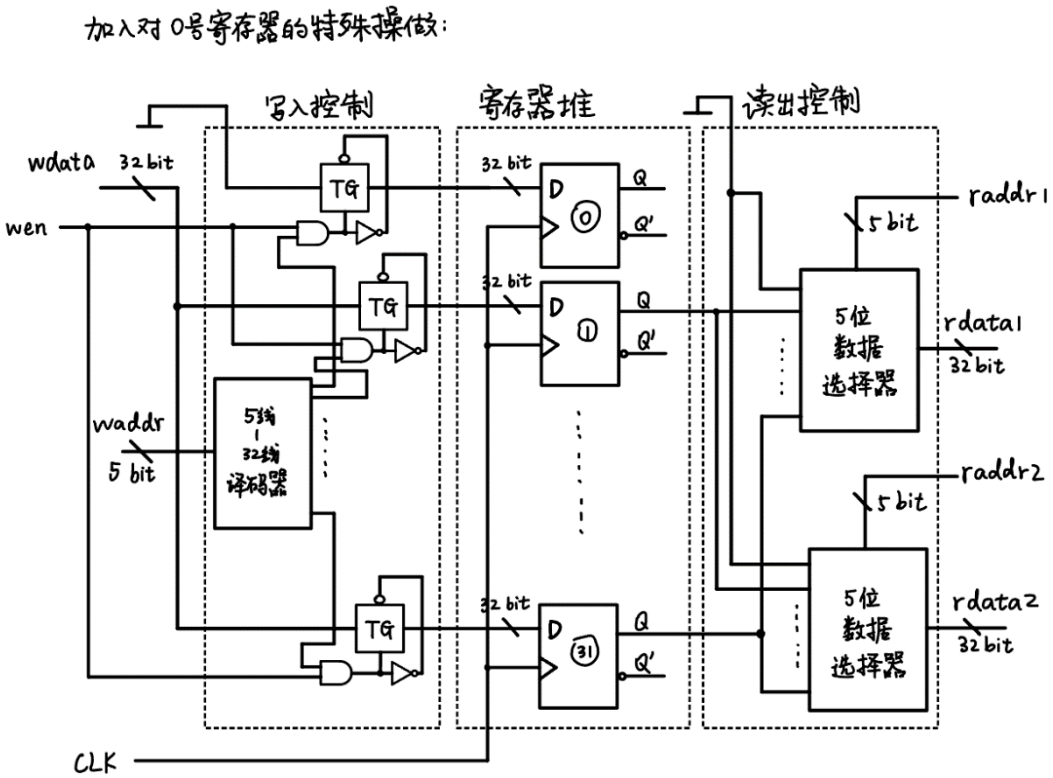
注:$RTL(Register\ Transfer\ Level)$，即寄存器传输级，是指对电路描述的层次。

```
`timescale 10 ns / 1 ns

`define DATA_WIDTH 32
`define ADDR_WIDTH 5

module reg_file(
    input                       clk,
    input  [`ADDR_WIDTH - 1:0]  waddr,
    input  [`ADDR_WIDTH - 1:0]  raddr1,
    input  [`ADDR_WIDTH - 1:0]  raddr2,
    input                       wen,
    input  [`DATA_WIDTH - 1:0]  wdata,
    output [`DATA_WIDTH - 1:0]  rdata1,
    output [`DATA_WIDTH - 1:0]  rdata2
);
    // TODO: Please add your logic design here
```

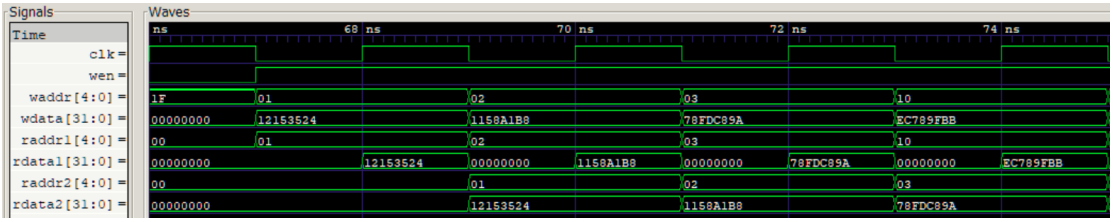
```
    reg [`DATA_WIDTH-1:0] rf [31:0];
    always @(posedge clk)
    begin
        if(wen==1'b1 && waddr!=5'b0) //仅当 wen=1 且 waddr 不等于 0 时，才可以向 waddr 对应的寄存器写入 wdata
        rf [waddr]<= wdata; //时序逻辑使用阻塞赋值
    end

    assign rdata1=(raddr1==0)?0:rf[raddr1];
     assign rdata2=(raddr2==0)?0:rf[raddr2];
    //若 raddr=0，则直接将 32'b0 输出，防止 0 号寄存器未初始化而读出不确定的值
endmodule
```

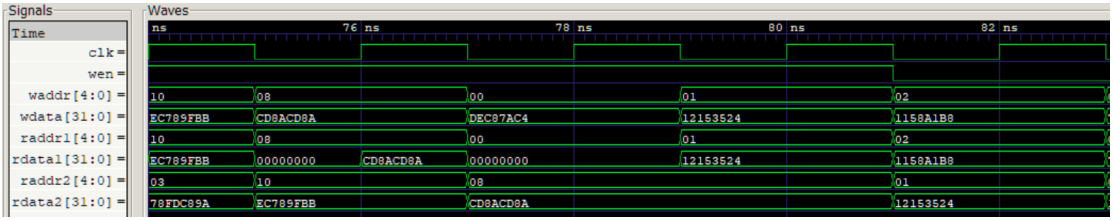
4） 逻辑电路结构图





5） 信号仿真波形





写操作是时钟同步的，寄存器中值的改变仅在时钟上升沿。而读操作是可以时刻进行的，两个输出$rdata$都能够实时地输出对应地址寄存器中的值。

读地址和写地址的改变发生在时钟下降沿，这样保证了在时钟上升沿时写入地址$waddr$的值已经稳定。

输出$rdata$在时钟下降沿和上升沿都有变化。在时钟下降沿变化是因为这时候$raddr$改变（如图中第$69ns$）；而在时钟上升沿变化是因为$waddr$和$raddr$恰是同一个地址，且此时写入$wdata$使寄存器中的值改变（如图中第$70ns$）。

6） 实验中遇到的问题：

a) 开始时我的rtl_chk环节一直报错，显示上次实验的$adder$有语法错误，

```
ERROR: [Synth 8-2715] syntax error near endmodule
[/builds/ucas-cod-2023/xiejianan21/fpga/design/ucas-cod/hardware/sources/example/adder.v:11]
```

问过助教后才知道这一环节会检查所有上传至云平台的文件的语法。😂

b) 然后我又遇到了在*bhv_sim*环节的报错：

```
ERROR: Read at 00, should get 00000000, but get xxxxxxxx.
```

下载波形文件后我发现0号寄存器的值一直是*xxxxxxxx*，这是因为它一直没有被写入过值，因此不确定每一位上是0还是1，处于**高阻态**。分析代码后，我找到了问题所在：0号寄存器不允许被写入的同时，其读出操作也应该做特殊处理，直接将32'*b*0给到输出，这样就避免了因为0号寄存器未初始化而读出不确定的值。

*if/else*条件判断语句必须在*always*块中才能使用，比较麻烦。为了更方便，我最后使用了**三目运算符**来实现*rdata*的赋值，最终通过了测试。

实验二：算术逻辑单元(*Arithmetic Logical Unit*)的实现

1) 实验目的

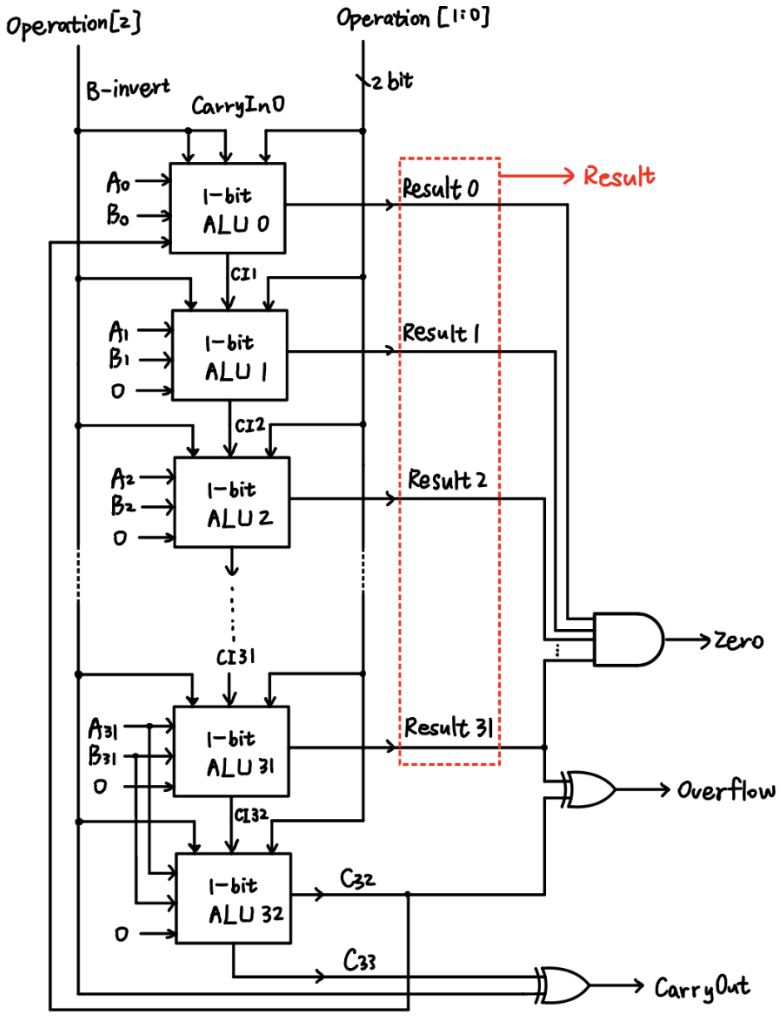
ALUop	功能操作（Operations）	使用相应功能操作的MIPS指令（Instruction）举例
000	逻辑按位与（AND）	and, andi
001	逻辑按位或（OR）	or, ori
010	算术加法（ADD）	add, lw (load word), sw (store word)
110	算术减法（SUB）	sub, beq (branch on equal)
111	有符号数整数比较（SLT） A < B时，Result = 1 A ≥ B时，Result = 0	slt (set on less than)

2) *RTL*代码

```
`timescale 10 ns / 1 ns
`define DATA_WIDTH 32

module alu(
    input  [`DATA_WIDTH - 1:0]  A,
    input  [`DATA_WIDTH - 1:0]  B,
    input  [             2:0]  ALUop,
    output                     Overflow,
    output                     CarryOut,
    output                     Zero,
    output [`DATA_WIDTH - 1:0]  Result
);
    // TODO: Please add your logic design here
    wire CarryIn = (ALUop[2]==1); //若要做减法或 SLT，则将进位设为 1
     wire [`DATA_WIDTH :0] A_Extended = {A[`DATA_WIDTH - 1],A};
     //将 A,B 的符号位扩展为双符号位
    wire [`DATA_WIDTH :0] B_Extended = {B[`DATA_WIDTH - 1],B};
    wire [`DATA_WIDTH :0] B_Convert = (ALUop[2]==1)? ~B_Extended : B_Extended;//若要做减法或 SLT，将 B 逐位取反
     wire [`DATA_WIDTH + 1:0] C = A_Extended+B_Convert+CarryIn;
     //将扩展后的 A 和 B 相加的结果放在 C 中，其中 C 的最高位用来存第 32 位的进位
    wire [`DATA_WIDTH - 1:0] Less;
```

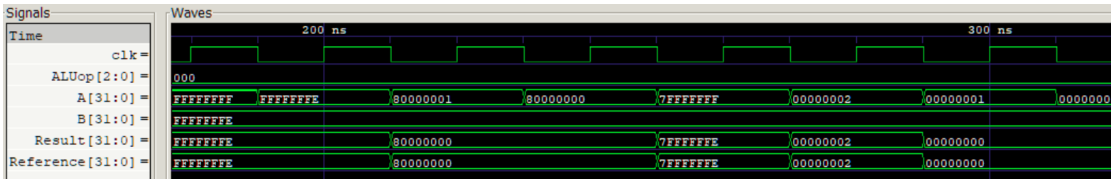
```
    assign Result=({`DATA_WIDTH{ALUop[1:0]==2'b00}}&(A&B))
                |({`DATA_WIDTH{ALUop[1:0]==2'b01}}&(A|B))
                |({`DATA_WIDTH{ALUop[1:0]==2'b10}}&(C[`DATA_WIDTH - 1:0]))
                |({`DATA_WIDTH{ALUop[1:0]==2'b11}}&(Less));
    //对于 ALUop 的每一种输入，例如 ALUop==3'b000，将它判断的结果与相应的计算结果的每一位进行按位与，最后所有情况按位或。这样对于 ALUop 的每一种情况，只有对应的计算结果（&1）可以在 result 上输出，其他计算结果都（&0）被忽略
    assign Zero=(Result==`DATA_WIDTH'b0);
    assign CarryOut=ALUop[2]^C[`DATA_WIDTH+1];
    //无符号数作加法时，最高位产生了进位说明 A+B>=2^n，说明产生了进位；无符号数作减法时，最高位如果没有产生进位就说明 A+2^n-B<2^n，说明有借位
    assign Overflow=C[`DATA_WIDTH]^C[`DATA_WIDTH-1];    //overflow 等于两个符号位的异或
    assign Less={{(`DATA_WIDTH-1){1'b0}},C[`DATA_WIDTH]};
    //Less 等于结果的最高符号位，代表结果真正的符号
endmodule
```

3） 逻辑电路结构图

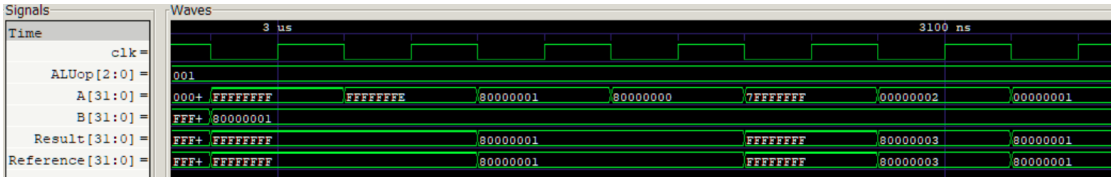


4） 信号仿真波形

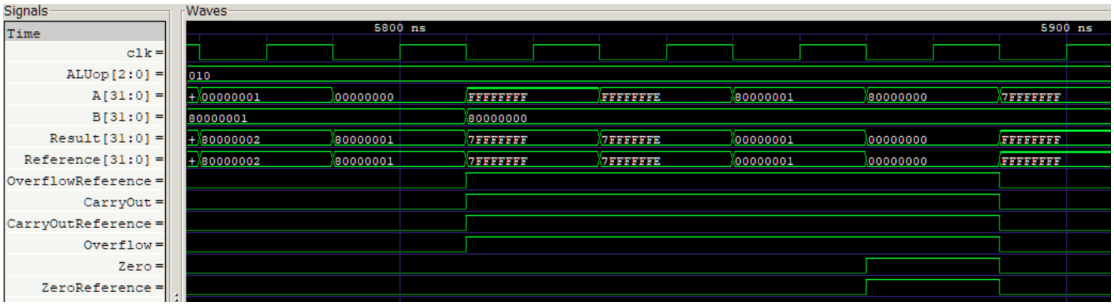
按位与(*AND*)操作：



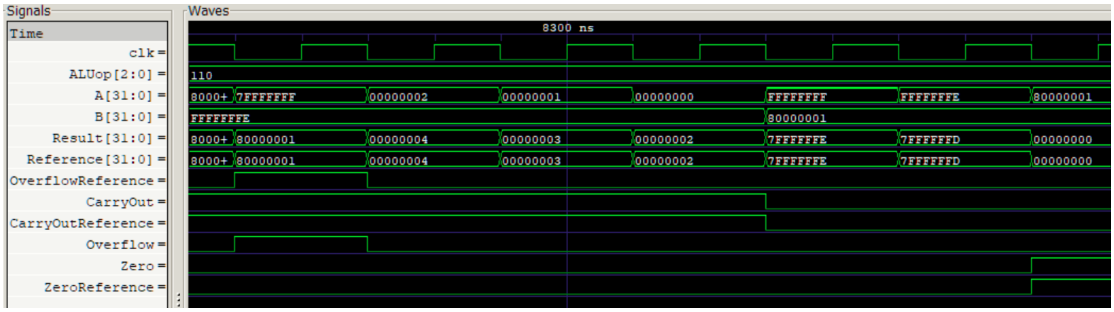
按位或(OR)操作：



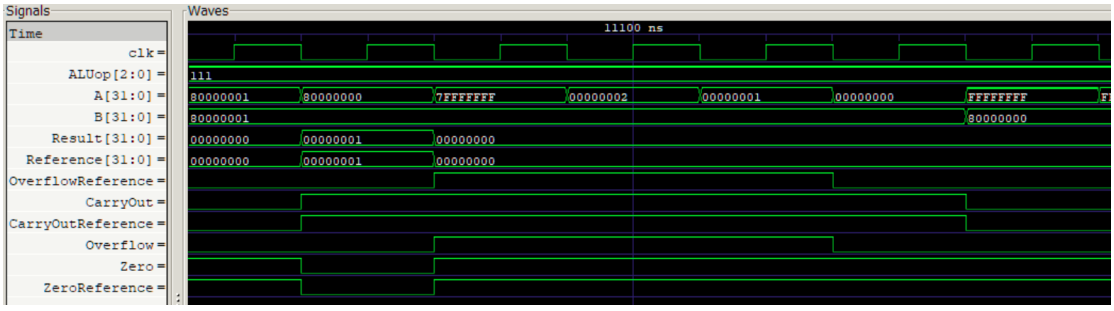
加法(ADD)操作：



减法(SUB)操作：



有符号数整数比较(SLT)操作：



在波形图中，各个输出信号与参考值始终保持一致，说明电路功能正确。

实验心得：

这次实验难度不大。但在在实现这些基本功能部件时，我发现自己对$verilog$语言的掌握和使用还是不太熟练，理解不够深入，感觉上学期数字电路课程中关于$verilog$语言的知识点有些忘记了。不过通过这次实验，我对于一些基本的语法，尤其是组合逻辑赋值语句的

使用方法有了更多的了解和实践，并且对于根据错误案例、观察波形分析问题有了更多的经验。