中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2020K8009926006 姓名：游昆霖 专业：计算机科学与技术

实验序号：1 实验名称：基本功能部件设计

注 1：撰写此 Word 格式实验报告后以 PDF 格式保存在~/COD-Lab/reports 目录下。文件命名规则：prjN.pdf，其中“prj”和后缀名“pdf”为小写，“N”为 1 至 4 的阿拉伯数字。例如：prj1.pdf。PDF 文件大小应控制在 5MB 以内。此外，实验项目 5 包含多个选做内容，每个选做实验应提交各自的实验报告文件，文件命名规则：prj5-projectname.pdf，其中“-”为英文标点符号的短横线。文件命名举例：prj5-dma.pdf。具体要求详见实验项目 5 讲义。

注 2：使用 git add 及 git commit 命令将实验报告 PDF 文件添加到本地仓库 master 分支，并通过 git push 推送到 GitLab 远程仓库 master 分支（具体命令详见实验报告）。

注 3：实验报告模板下列条目仅供参考，可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

一、 逻辑电路结构与仿真波形的截图及说明（比如关键 RTL 代码段{包含注释}及其对应的逻辑电路结构图、相应信号的仿真波形和信号变化的说明等）

1、reg_file.v 代码说明

(1) 宏定义及变量定义

```
`define TOP 32
    reg [`DATA_WIDTH -1 : 0] RF [`TOP-1 : 0];
```

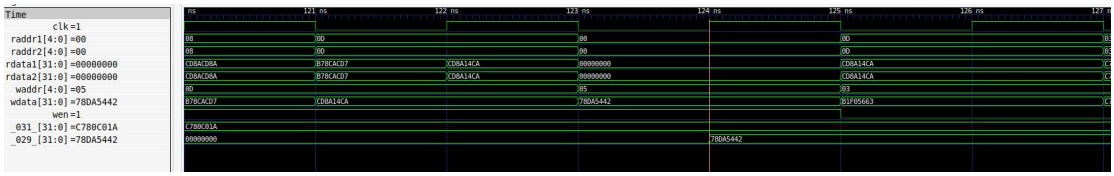
宏定义 TOP 表示寄存器个数，并设置 32 个 32 位寄存器变量。

(2)同步写数据

```
always @ (posedge clk)
    begin
        if(wen & (|waddr)) begin  // |waddr is equal to waddr != 5'b0
            RF[waddr] <= wdata;
        end
    end
```

通过时序逻辑和非阻塞赋值实现同步写数据。同时写数据条件为写使能信号 wen 置高电平，且写地址 waddr 非零，注意此处可用数据按位或简便表示。

相应波形：



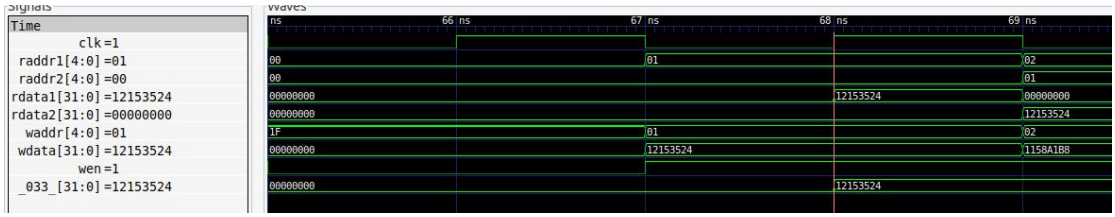
waddr=05时写5号寄存器，对应reference中_029_端口，123ns处waddr和wdata改变，但在124ns时钟上升沿处_029_所存的值才完成更新，说明了“同步写”，waddr=03时对应3号寄存器(reference中_031_端口)，由于126ns时wen为0，不满足写条件，故其值未更新。

(3)异步读数据

```
assign rdata1 = {`DATA_WIDTH {|raddr1}} & RF [raddr1];
assign rdata2 = {`DATA_WIDTH {|raddr2}} & RF [raddr2];
```

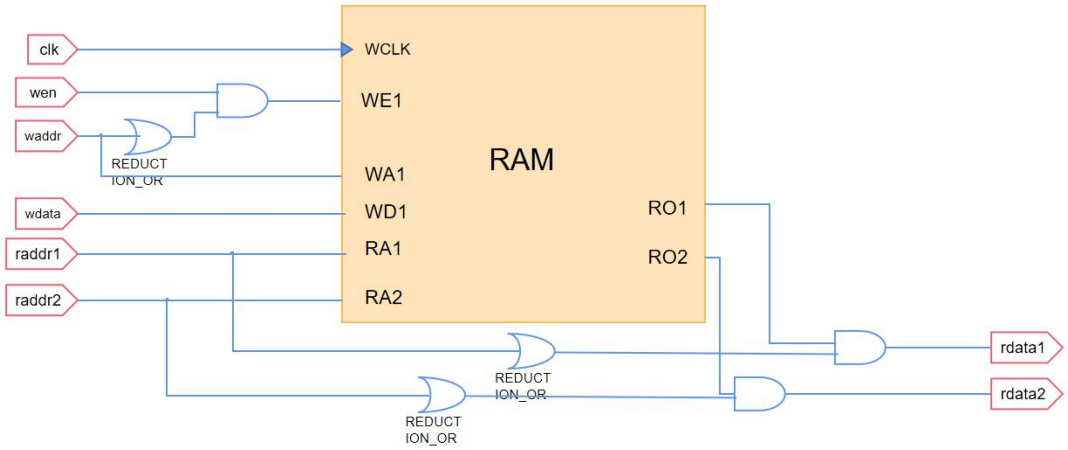
通过组合逻辑保证异步读数据。同时为保证读0号寄存器结果恒为0，通过|raddrN判断地址是否为0，并将其位拓展到32位，与从对应寄存器读取数据进行按位与，确保结果正确性。

相应波形：



68ns处，raddr2 = 00，对应0号寄存器，读出数据为全0；raddr1=01，对应1号寄存器，读出数据为12153524。Reference中的_033_端口对应1号寄存器，在67ns处waddr和wdata改变后，在68ns时钟上升沿才完成了数据更新，说明了“同步写”，数据更新后，rdata1立即读出数据，说明了“异步读”。

逻辑电路结构图：



2、alu.v 代码说明（只展示关键代码及对应逻辑电路结构）

(1) 宏定义及变量定义

```
//ALUop operators
    `define AND 3'b000
    `define OR  3'b001
    `define ADD 3'b010
    `define SUB 3'b110
    `define SLT 3'b111
...
//wire virables for direct results
    wire [`DATA_WIDTH -1 :0] R_AND;
    wire [`DATA_WIDTH -1 :0] R_OR;
    wire [`DATA_WIDTH -1 :0] R_ADD;
    wire [`DATA_WIDTH -1 :0] R_SUB;
    wire [`DATA_WIDTH -1 :0] R_SLT;

//wire virable for carryout
    wire flag;
    wire Inverse;

//wire virable for overflow
    wire Compare;

//add 1 more bit for completement form
    wire [`DATA_WIDTH :0]    A_tmp;
    wire [`DATA_WIDTH :0]    B_tmp;
```

宏定义五种操作对应的名称及操作码。变量定义通过不同逻辑得到的五种结

果。（注意，只在对应操作下保证所得结果正确。）变量定义求 carryout 过程的中间变量 flag 和 Inverse，求 Overflow 过程中间变量 Compare 及操作数 A、B 对应的转化数。

(2) A、B 转化数说明

```
//completement of A / B    :
    assign Inverse = ALUop[2] ; //sub or slt
    assign A_tmp = {1'b0,A};
    assign B_tmp = {1'b0, Inverse ? ~B : B };
    //note that the inverse result exclude the highest bit
```

当加法操作时 A、B 转化数均在最高位位拓展一位 0 以得到进位信号；减法或比较操作时 B 转化数除最高位外取反并加 1，(将加 1 的操作转移到转化数加法器，详见部分二问题 4）得与 B 相加为 0X1_0000_0000 的数，(注意其与补码有一定区别，详见部分二问题 2)，下述求借位信号逻辑。

相应逻辑电路结构图见五种操作赋值说明（加减法操作）部分。

(3) CarryOut、Overflow 信号说明

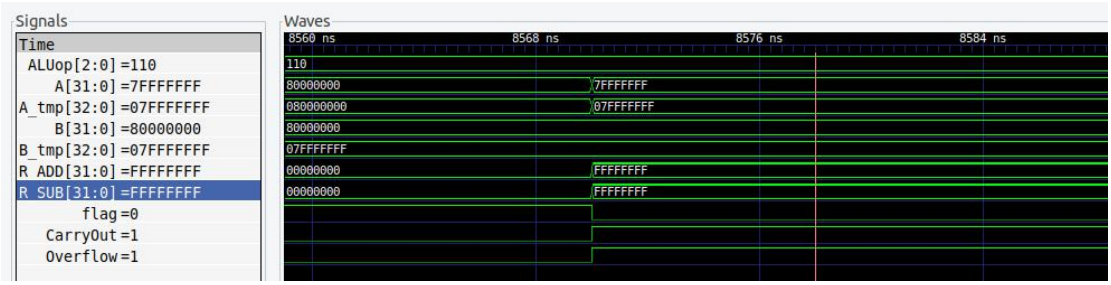
```
assign {flag , R_ADD} = A_tmp + B_tmp  + Inverse ;
...
//CarryOut  -> unsigned num
//ADD: carryout is the same as actual
//SUB: when A=B the result is 0x1 0000 0000
    //when A < B, Completement of B < completement of A, so the highest bit of A-B is
0, however borrow actually happen, vice versa;
    //so carryout = ~flag
//Overflow  -> signed num
//overflow cases:
    // If A > 0 and B > 0 but A + B < 0
    // If A < 0 and B < 0 but A + B >= 0
    // If A > 0 and B < 0 but A - B < 0
    // If A < 0 and B > 0 but A - B >= 0
//Overflow only consider ADD or SUB, but we want to use result for slt, so set compare
    assign CarryOut = (ALUop == `ADD)? flag : ~flag;
    assign Compare = (~A[`DATA_WIDTH-1] & ~B[`DATA_WIDTH-1] & R_ADD[`DATA_WIDTH-1] &
ALUop== `ADD)|
                     (A[`DATA_WIDTH-1] & B[`DATA_WIDTH-1] & ~R_ADD[`DATA_WIDTH-1] &
```

```
ALUop== `ADD)|
                        (~A[`DATA_WIDTH-1] & B[`DATA_WIDTH-1] & R_SUB[`DATA_WIDTH-1] &
(ALUop==`SLT|ALUop== `SUB))|
                        (A[`DATA_WIDTH-1] & ~B[`DATA_WIDTH-1] & ~R_SUB[`DATA_WIDTH-1] &
(ALUop==`SLT|ALUop== `SUB));
    assign Overflow = (ALUop == `ADD | ALUop==`SUB ) & Compare;
```

flag 记录 A、B 转化数相加后的最高位。由于 CarryOut 针对无符号数，当操作为加法时，flag 自然表示进位；当操作为减法或比较时，简记此时 B_tmp 为△B，详见(2)。由于 A+△A=0X1_0000_0000，则当 A≥B 时，有△A≤△B，即 A+△B≥0X1_0000_0000，最高位为 1，然而从真值看实质未发生借位，当 A<B 时情况类似，因此操作为减法或比较时进位信号为 flag 的反。

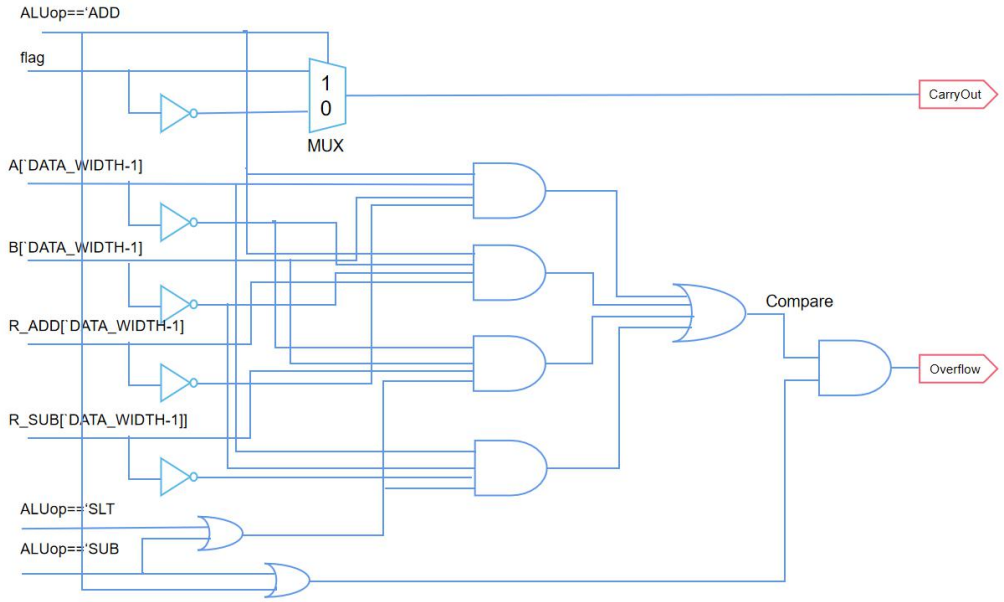
Overflow 针对有符号数的加减法操作。由于该结果在比较操作时也需要使用，且判定逻辑与减法类似，因此设置变量 Compare 表示两数加减不符合预期的情形，四种情形如注释所示，通过与或操作获得 Overflow 的值。

相应波形：



上图红线所示周期，A 为 0X7FFFFFFF，B 为 0X80000000，操作为 SUB。由于 CarryOut 针对无符号数，Overflow 针对有符号数。当 A、B 均为无符号数时，B 取反并拓展 1 位零与 A 相加的拓展位 flag 为 0，CarryOut 为 1，产生借位。当 A、B 均为有符号数时，相减的真值应当为正，但是 R_SUB 表示为负，因此产生 Overflow 信号。

逻辑电路结构图：



(4) 五种操作对应赋值说明

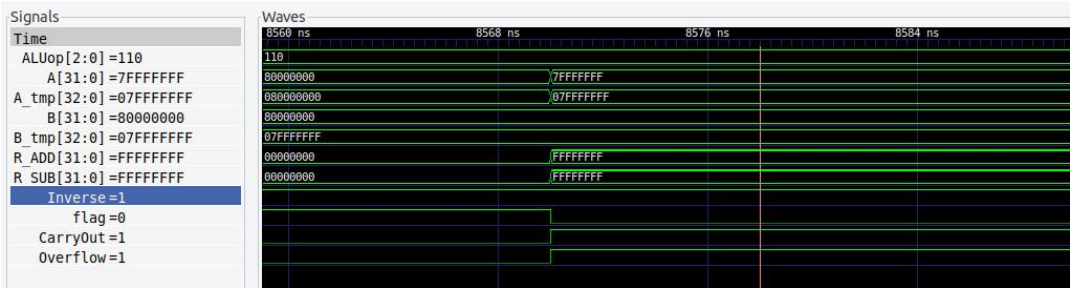
```
//result of OP AND / OR
    assign R_AND = A & B;
    assign R_OR  = A | B;
...
//result of OP ADD / SUB
    assign {flag , R_ADD} = A_tmp + B_tmp  + Inverse ;
    assign R_SUB = R_ADD;
...
//result of OP SLT
//Sigend num:   compare means the result is the inverse of expected, so we use xor to
get result
    assign R_SLT[0] = Compare ^ R_SUB[`DATA_WIDTH-1];
    assign R_SLT[`DATA_WIDTH-1 :1] = 31'b0;
```

对于 AND/OR/ADD/SUB 操作，结果只需考虑 32 位的直接操作结果，特别的，不需考虑有符号数加减法的溢出情况。由于减法时会先将 B 取补得到 B 的转化数，同样使用加法器运算，因此转化数运算逻辑与 ADD 相同，且直接赋值可以减少一个加法器的使用。

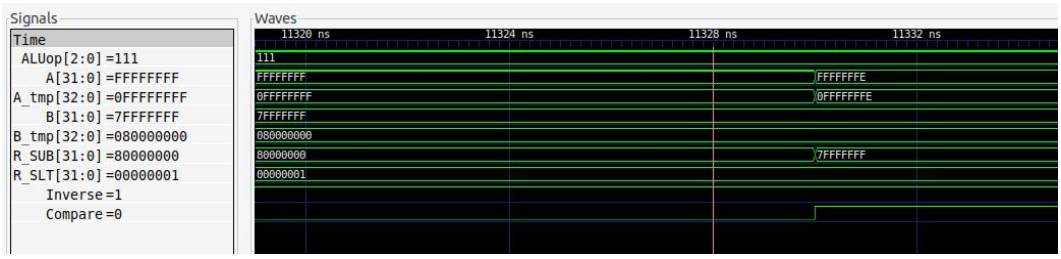
对于 SLT 操作，只需利用减法逻辑判断即可，A<B 也即相减结果真值为负。Compare 信号为高表示减法结果符号与真值不同。因此当 Compare 为 0、

R_SUB 最高位为 1 或 Compare 为 1，R_SUB 最高位为 0 时真值为负，SLT 最低位置 1。从结果上考虑只需将 SLT 最低位赋值为 Compare 和 R_SUB 最高位的异或值，其余位取零即可。

相应波形（部分）：

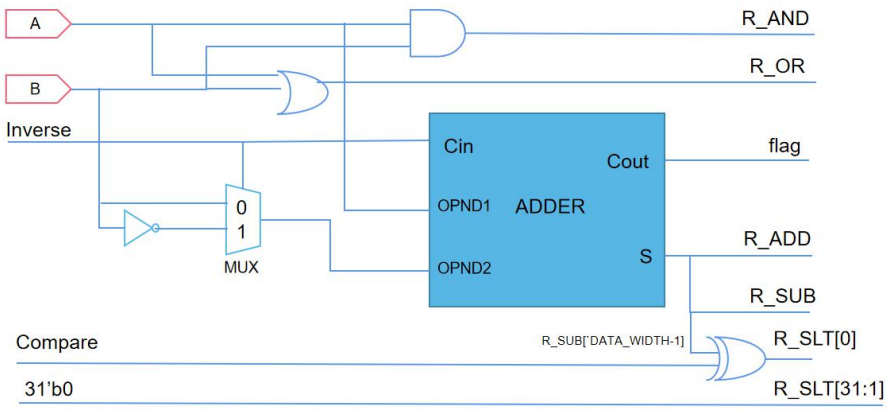


红线所示周期处操作为 SUB，R_SUB 为 A、B 转化数相加结果（其中+1 独立为+Inverse）的最后 32 位，与预期结果一致。



图中所示操作为 SLT，针对有符号数比较。11328ns 处 A<B，相减结果也为负，与预期相符，Compare 为 0；11332ns 处 A<B，但相减结果为正，与预期不符，Compare 为 1，上述两个周期对应 R_SLT 最低位均为 1。

逻辑电路结构图：



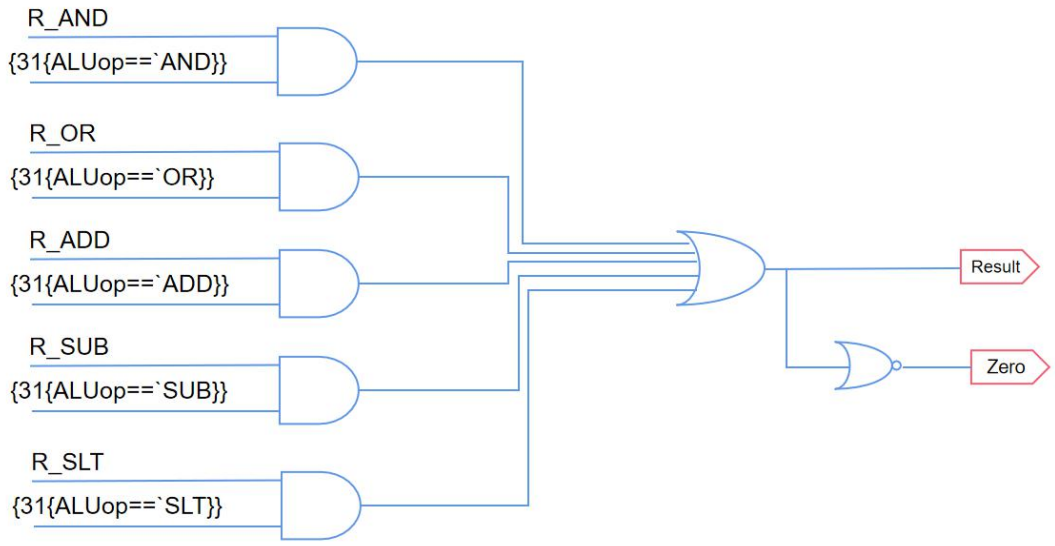
(5) 生成最终输出结果及 Zero 信号

```
//final result and zero check
    assign Result = ({32{ALUop == `AND}} & R_AND ) |
                    ({32{ALUop == `OR }} & R_OR  ) |
                    ({32{ALUop == `ADD}} & R_ADD ) |
                    ({32{ALUop == `SUB}} & R_SUB ) |
                    ({32{ALUop == `SLT}} & R_SLT );

    assign Zero = ~|Result;
```

通过操作数选择不同操作逻辑所得结果，并通过 Result 与 0 比较得到 Zero 信号。

逻辑电路结构图：



二、 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出现的逻辑 bug，逻辑仿真和 FPGA 调试过程中的难点等）

问题 1：ALU 中位拼接位宽错误

查看云平台语法检查 log，发现在进行位拼接，未指定 1 位 0 的位数，vivado 默认未指定整数位宽为 32，经检查后指定位宽，解决错误。

问题 2：ALU 中位拼接求值错误

为进行减法及求 Overflow 等值，设置 B_tmp，减法或比较运算时表示

与 B 相加为 0X1_0000_0000 的值（以下简记为△B)。但在检查波形时发现，{1’b0, ALUop[2] ? ~B : B} + ALUop[2] 与 {1’b0, ALUop[2]? ~B+1 :B} 二者在对 0 求△B 时，前者符合预期值 0X1_0000_0000，而后者得到 0X0_0000_0000，不符合预期。这是因为当+1 发生在位拼接过程中时，对 32’0 进行取反加 1 时高位进位会因位数限制被舍去，因此应当采用前一种写法。

问题 3：运算优先级及括号嵌套问题

编写代码过程中同级运算容易造成理解歧义，例如&和|运算看似同级，但是&优先执行，因此应当增加括号规范逻辑顺序。同时由于多层括号嵌套容易降低代码可读性，应当通过适当缩进，使同级逻辑缩进相同，从而凸显逻辑层次，并便利错误查找。

问题 4：电路资源消耗的精简

在考虑对 B 求补的转化数加法中如果将+1 置于转化数生成过程，将产生额外的一个加法器，消耗资源。在经过实验课助教老师的讲解后，将其作为 A、B 转化数的加法器的 Cin，节约了一个加法器，减少电路资源消耗。

三、 对讲义中思考题（如有）的理解和回答

本次实验无思考题。

四、 在课后，你花费了大约_____5_____小时完成此次实验。

五、 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）

实验难度适中，在与同学交流实验项目心得时发现部分同学不理解 ALU 部分输出信号的意义，建议在实验讲解中适当结合后续指令应用 ALU 输出信号进行判断的例子，（例如分支指令中对于 SLT 结果，Zero、Overflow 等信号的应用），有助于同学们更好理解 ALU 的作用。