数字逻辑与计算机组成实验

LAB 10：CPU 数据通路

郑凯琳 205220025
205220025@smail.nju.edu.cn

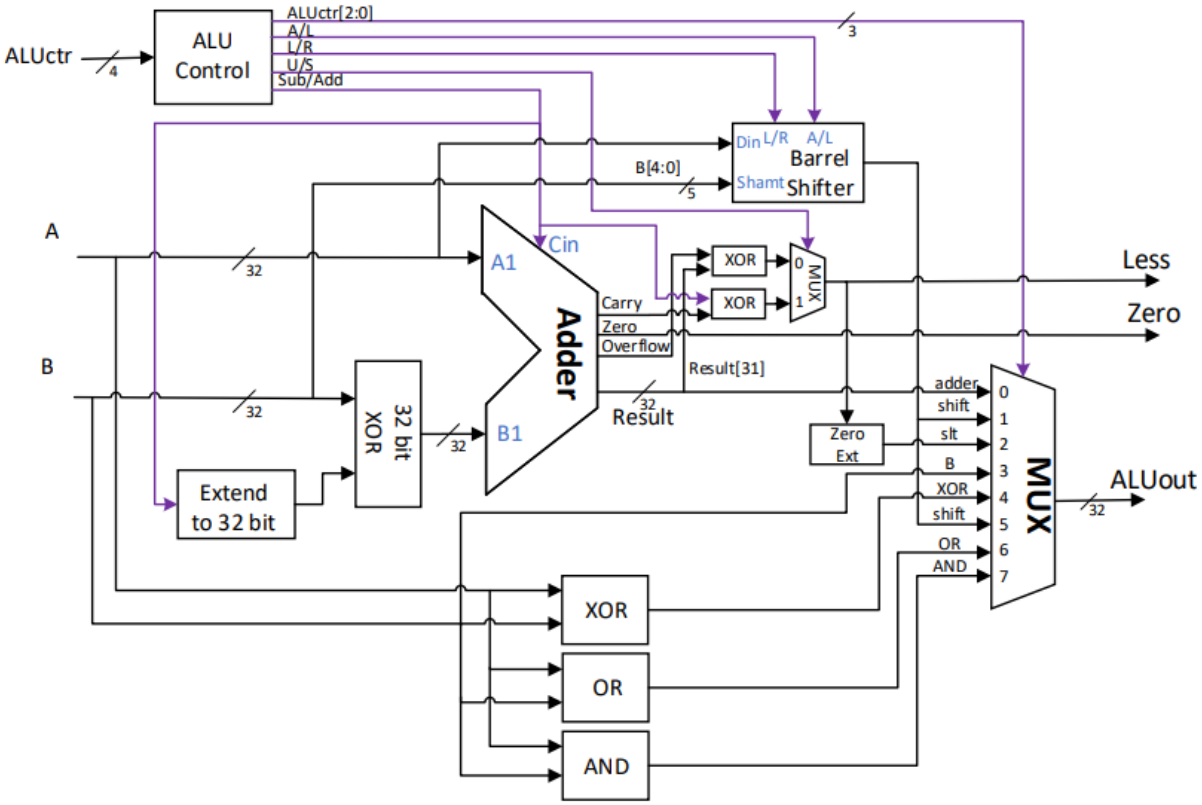
数据通路之 ALU 实现

（一）实验目的

将简版 ALU 改造成可以直接用于 RV32I CPU 内的真·ALU。

（二）实验原理

ALU 是 CPU 中的核心数据通路部件之一，它主要完成 CPU 中需要进行的算术逻辑运算。ALU 对输入数据并行地进行加减法、移位、比较大小、异或等操作。最终 ALUout 输出是通过一个八选一选择器选择不同运算部件的结果，选择器的控制端可以用 ALUctr[2:0] 直接生成。ALU 其他部件的控制信号需要的控制信号包括：A/L 控制移位器进行算术移位还是逻辑移位，L/R 控制是左移还是右移，U/S 控制比较大小是带符号比较还是无符号比较，S/A 控制是加法还是减法。



（三）实现要求

ALUctr[3]	ALUctr[2:0]	ALU 操作
0	000	选择加法器输出，做加法
1	000	选择加法器输出，做减法
×	001	选择移位器输出，左移
0	010	做减法，选择带符号小于置位结果输出，Less 按带符号结果设置
1	010	做减法，选择无符号小于置位结果输出，Less 按无符号结果设置
×	011	选择 ALU 输入 B 的结果直接输出
×	100	选择异或输出
0	101	选择移位器输出，逻辑右移
1	101	选择移位器输出，算术右移
×	110	选择逻辑或输出
×	111	选择逻辑与输出

ALU 的输出包括运算结果及 less 和 zero 指示位:

- less 指示只需要在比较大小时正确即可，
- zero 指示需要在所有运算中均正确指示 ALU 输出是否为 0。
- 在做比较的时候，zero 指示两个输入是否相等。

（四）实验过程

数字抽象

1. 输入:
   - dataa [31:0] —— ALU 操作数 A
   - datab [31:0] —— ALU 操作数 B，移位时低 5 位为移位量
   - ALUctr [3:0] —— ALU 控制信号
2. 输出:
   - less —— less 指示，小于时置 1 控制信号
   - zero —— zero 指示，结果为 0 时置 1
   - aluresult [31:0] —— ALU 输出结果

设计思路 & 设计代码

一个 always 语句块

选择加法器（加法 / 减法）

1. 通过检查 ALUctr 的值来确定执行的操作是加法还是减法。

如果 ALUctr 为 4'b1000，则表示执行减法运算，否则执行加法运算。

2. 在执行减法运算时，将 addsub 设置为 1'b1，表示进行减法操作。
3. 为了执行减法，需要将操作数 B 取反（求补码），然后与操作数 A 进行相加。这里使用了 {32{addsub}} ^ datab 的方式，将 addsub 复制为 32 位，与操作数 B 进行按位异或操作，得到无进位的运算结果 t_no_Cin。
4. 然后，将操作数 A、无进位运算结果和进位标志 addsub 一起进行加法运算，并将结果存储在{cf, aluresult}中。
5. 溢出标志 of 的计算是通过检查操作数 A 的最高位和无进位运算结果的最高位是否不相等来确定的。如果它们不相等，则表示发生了溢出。
6. 进位标志 cf 根据 addsub 进行异或运算，以调整加法和减法的进位标志。
7. 零标志位 zero 是通过检查运算结果 aluresult 的每一位是否都为 0 来确定的。

左移

1. 使用了两个 case 语句来处理左移操作。
    a. 第一个 case 语句处理无符号左移（ALUctr 为 4'b0001），
    b. 第二个 case 语句处理带符号左移（ALUctr 为 4'b1001）。
2. 在无符号左移操作中，操作数 A（dataa）的值将向左移动 datab[4:0]位。
   这是通过将操作数 A 左移指定的位数来实现的，结果存储在 aluresult 中。
3. 在带符号左移操作中，同样将操作数 A 的值向左移动 datab[4:0]位。
   对于带符号数，左移时最高位（符号位）要保持不变，因此符号位左侧的位数将被填充为 0。结果同样存储在 aluresult 中。

减法

1. **选择带符号小于置位结果输出，less 按带符号结果设置**

操作数 A（dataa）和操作数 B（datab）相减，结果存储在 res 中。然后根据结果的值进行判断：

- 如果结果为 0，将 zero 置为 1，表示结果为零；
- 如果结果的最高位（符号位）为 1，将 less 置为 1，表示结果为负数（小于零）；
- 否则，将 less 置为 0，表示结果为正数（大于等于零）。
  最后，将 less 赋值给 aluresult。

2. **选择无符号小于置位结果输出，less 按无符号结果设置**

操作数 A（dataa）和操作数 B（datab）相减，结果存储在 res 中。然后根据结果的值进行判断：

- 如果结果为 0，将 zero 置为 1，表示结果为零；
- 如果结果小于 0，将 less 置为 1，表示结果为负数（小于零）；
- 否则，将 less 置为 0，表示结果为正数（大于等于零）。
  最后，将 less 赋值给 aluresult。

逻辑右移

操作数 A（dataa）按照操作数 B（datab）指定的位数进行逻辑右移，结果存储在 aluresult 中。

逻辑右移是通过将 A 的二进制表示向右移动 B 位，并在左侧插入 0 来实现的。

算术右移

同样对操作数 A 进行右移运算，结果存储在 res 中。然后进行以下操作：

- 定义变量 a，其值为(2 ^ datab - 1)左移(32 - datab)位。这个操作是为了创建一个掩码，用于屏蔽右移操作中新增的符号位。
- 将 a 与 res 相加，得到最终结果。这是因为算术右移需要将右移产生的新位补充为符号位的值，所以需要将掩码与右移结果相加。
- 最后，将 res 赋值给 aluresult。

另一个 always 语句块

输出 dataab

1. 如果 dataa 等于 datab，则将 zero 置为 1，否则置为 0。
2. 将 datab 的值存储在 aluresult 中。

异或

1. 如果 dataa 等于 datab，则将 zero 置为 1，否则置为 0。
2. 将 dataa 与 datab 进行异或操作，并将结果存储在 aluresult 中。

逻辑或

1. 如果 dataa 等于 datab，则将 zero 置为 1，否则置为 0。
2. 将 dataa 与 datab 进行逻辑或操作，并将结果存储在 aluresult 中。

逻辑与

1. 如果 dataa 等于 datab，则将 zero 置为 1，否则置为 0。
2. 将 dataa 与 datab 进行逻辑与操作，并将结果存储在 aluresult 中。

数据通路之数据存储实现

（一）实验目的

实现支持按字节，半字和字访问的数据存储器，按 RISC-V 的存储小端方式实现。

（二）实现要求

	指令	MemOP	操作
lb	rd,imm12(rs1)	000	R[rd] ← SEXT(M_{1B}[R[rs1] + SEXT(imm12)])
lh	rd,imm12(rs1)	001	R[rd] ← SEXT(M_{2B}[R[rs1] + SEXT(imm12)])
lw	rd,imm12(rs1)	010	R[rd] ← M_{4B}[R[rs1] + SEXT(imm12)]
lbu	rd,imm12(rs1)	100	R[rd] ← {24'b0, M_{1B}[R[rs1] + SEXT(imm12)]}
lhu	rd,imm12(rs1)	101	R[rd] ← {16'b0, M_{2B}[R[rs1] + SEXT(imm12)] }
sb	rs2,imm12(rs1)	000	M_{1B}[R[rs1] + SEXT(imm12)] ← R[rs2][7:0]
sh	rs2,imm12(rs1)	001	M_{2B}[R[rs1] + SEXT(imm12)] ← R[rs2][15:0]
sw	rs2,imm12(rs1)	010	M_{4B}[R[rs1] + SEXT(imm12)] ← R[rs2]

实现一个能够支持 RV32I 中的 lb,lh,lw,lbu,lhu 及 sb,sh,sw 等指令的存储器。

该存储器具有独立的读取时钟 rdclk 和写入时钟 wrclk，均为上升沿有效，读取和写入在上升沿到达后立刻生效。

（三）实验过程

数字抽象

3. 输入：
    - addr [31:0] —— 读写地址
    - datain [31:0] —— 存储器写入的数据
    - rdclk —— 读取时钟，上升沿有效
    - wrclk —— 写入时钟，上升沿有效
    - memop [2:0] —— 内存操作控制位
    - we —— 写使能，高电平有效
4. 输出：
    - dataout [31:0] —— 存储器读取结果

设计思路 & 设计代码

在给定的代码中，有两行声明了一些变量，分别是：

寄存器变量：

- opcode：无符号（unsigned）寄存器，位宽为 32 位，用于存储指令的操作码字段。

操作码通常用于指示指令的类型和操作类型。

- rd：无符号（unsigned）寄存器，位宽为 32 位，用于存储指令的目标寄存器（Destination Register）字段。
  目标寄存器通常用于指定指令的结果应该存储在哪个寄存器中。
- rs：无符号（unsigned）寄存器，位宽为 32 位，用于存储指令的源寄存器（Source Register）字段。
  源寄存器通常用于指定指令的操作数或数据来源所在的寄存器。
- funct：无符号（unsigned）寄存器，位宽为 32 位，用于存储指令的功能码字段。
  功能码通常用于指示指令的具体功能或操作类型，与操作码一起协助指令的执行。
- imm：有符号（signed）寄存器，位宽为 32 位，用于存储指令的立即数（Immediate）字段。
  立即数是一种用于指令操作的常数或即时值，它可以直接参与指令的运算或操作。
- temp：有符号（signed）寄存器，位宽为 32 位，用于临时存储数据。
  在代码中，它被用于暂存从输入端口传入的数据，以在时钟边沿触发时进行处理和解析。

always 块：

1. 使用 posedge rdclk 和 posedge we 作为时钟边沿触发器。
2. 如果 we 为 1（写使能信号有效），将 datain 的值存储在 temp 中。
3. 如果 we 为 0（写使能信号无效），根据 temp 的值提取出 opcode、rd、rs、funct 和 imm 的值。

case 语句根据 memop 的值执行不同的内存操作：

- 3'b000：字节加载（lb）操作，
  根据 addr 的值判断偏移，从 temp 中提取相应字节并扩展成 32 位数据。
- 3'b001：半字加载（lh）操作，
  根据 addr 的值判断偏移，从 temp 中提取相应半字并扩展成 32 位数据。
- 3'b010：字加载（lw）操作，
  根据 addr 和 temp 的值进行特殊处理，然后将 temp 的值写入 dataout。
- 3'b100：无符号字节加载（lbu）操作，
  将 temp 中的字节数据零扩展成 32 位数据。
- 3'b101：无符号半字加载（lhu）操作，
  根据 addr 的值判断偏移，从 temp 中提取相应半字并零扩展成 32 位数据。