# C1025 Assignment01

## 目的

模拟ALU进行整数和浮点数的四则运算。

## 要求

1. 解压ALUSimulator.zip，修改ALUSimulator/src/ALU.java文件，使用Java实现以下各个方法。
2. 方法说明
3. **public String integerRepresentation (String number, int length)**

生成十进制整数的二进制补码表示。

例：integerRepresentation("9", 8) 返回 "00001001"

**参数：**

* number：十进制整数或浮点数。若为负数；则第一位为“-”；若为正数或 0，则无符号位
* length：二进制补码表示的长度

**返回值：**

* number的二进制表示，长度为length

1. **public String floatRepresentation (String number, int eLength, int sLength)**

生成十进制浮点数的二进制表示。 需要考虑 0、反规格化、正负无穷（“+Inf”和“-Inf”）、 NaN等因素，具体借鉴 IEEE 754。 舍入策略为向0舍入。

例：floatRepresentation("11.375", 8, 11) 返回 "01000001001101100000"

**参数：**

* number：十进制浮点数，包含小数点。若为负数；则第一位为“-”；若为正数或 0，则无符号位
* eLength：指数的长度，取值大于等于 8
* sLength：尾数的长度，取值大于等于 8

**返回值：**

* number的二进制表示，长度为 1+sLength+eLength。从左向右，依次为符号、指数（移码表示）、尾数（首位隐藏）

1. **public String ieee754 (String number, int length)**

生成十进制浮点数的IEEE 754表示，要求调用floatRepresentation实现。

例：ieee754("11.375", 32) 返回 "01000001001101100000000000000000"

**参数：**

* number：十进制浮点数，包含小数点。若为负数；则第一位为“-”；若为正数或 0，则无符号位
* length：二进制表示的长度，为32或64

**返回值：**

* number的IEEE 754表示，长度为length。从左向右，依次为符号、指数（移码表示）、尾数（首位隐藏）

1. **public String integerTrueValue (String operand)**

计算二进制补码表示的整数的真值。

例：integerTrueValue("00001001") 返回 "9"

**参数：**

* operand：二进制补码表示的操作数

**返回值：**

* operand的真值。若为负数；则第一位为“-”；若为正数或 0，则无符号位

1. **public String floatTrueValue (String operand, int eLength, int sLength)**

计算二进制原码表示的浮点数的真值。

例：floatTrueValue("01000001001101100000", 8, 11) 返回 "11.375"

**参数：**

* operand：二进制表示的操作数
* eLength：指数的长度，取值大于等于 8
* sLength：尾数的长度，取值大于等于 8

**返回值：**

* operand的真值。若为负数；则第一位为“-”；若为正数或 0，则无符号位。正负无穷分别表示为“+Inf”和“-Inf”， NaN表示为“NaN”

1. **public String negation (String operand)**

按位取反操作。

例：negation("00001001") 返回 "11110110"

**参数：**

* operand：二进制表示的操作数

**返回值：**

* operand按位取反的结果

1. **public String leftShift (String operand, int n)**

左移操作。

例：leftShift("00001001", 2) 返回 "00100100"

**参数：**

* operand：二进制表示的操作数
* n：左移的位数

**返回值：**

* operand左移n位的结果

1. **public String logRightShift (String operand, int n)**

逻辑右移操作。

例：logRightShift("11110110", 2) 返回 "00111101"

**参数：**

* operand：二进制表示的操作数
* n：右移的位数

**返回值：**

* operand逻辑右移n位的结果

1. **public String ariRightShift (String operand, int n)**

算术右移操作。

例：logRightShift("11110110", 2) 返回 "11111101"

**参数：**

* operand：二进制表示的操作数
* n：右移的位数

**返回值：**

* operand算术右移n位的结果

1. **public String fullAdder (char x, char y, char c)**

全加器，对两位以及进位进行加法运算。

例：fullAdder('1', '1', '0') 返回 "10"

**参数：**

* x：相加的第一位，取0或1
* y：相加的第二位，取0或1
* c：当前进位，取0或1

**返回值：**

* 相加的结果，用长度为2的字符串表示，第1位表示进位，第2位表示和

1. **public String claAdder (String operand1, String operand2, char c)**

8位先行进位加法器。

例：claAdder("00001001", "00000001", '1') 返回 "000001011"

**参数：**

* operand1：8位补码表示的被加数
* operand2：8位补码表示的加数
* c：当前进位，取0或1

**返回值：**

* 长度为9的字符串表示的计算结果，其中第1位是进位，后8位是补码表示的相加结果

1. **public String oneAdder (String operand)**

加一器，实现操作数加1的运算。不允许调用claAdder来实现。

例：oneAdder("00001001") 返回 "000001010"

**参数：**

* operand：二进制补码表示的操作数

**返回值：**

* operand加1的结果，长度为operand的长度加1，其中第1位指示是否溢出（溢出为1，否则为0），其余位为相加结果

1. **public String integerAddition (String operand1, String operand2, int length)**

整数加法，要求调用claAdder方法实现。

例：integerAddition("0100", "0011", 8) 返回 "000000111" （4+3=7）

**参数：**

* operand1：二进制补码表示的被加数
* operand2：二进制补码表示的加数
* length：存放操作数的寄存器的长度，为8的倍数。length不小于操作数的长度，当某个操作数的长度小于length时，需要在高位补符号位

**返回值：**

* 长度为length+1的字符串表示的计算结果，其中第1位指示是否溢出（溢出为1，否则为0），后length位是相加结果

1. **public String signedAddition (String operand1, String operand2, int length)**

带符号整数加法，要求调用integerAddition方法实现。

例：signedAddition("1100", "1011", 8) 返回 "010000111" （(-4)+(-3)=-7）

**参数：**

* operand1：二进制原码表示的被加数，其中第1位为符号位
* operand2：二进制原码表示的加数，其中第1位为符号位
* length：存放操作数的寄存器的长度，为8的倍数。length不小于操作数的长度，当某个操作数的长度小于length时，需要在符号位后补0

**返回值：**

* 长度为length+1的字符串表示的计算结果，其中第1位指示是否溢出（溢出为1，否则为0），后length位是相加结果

1. **public String integerSubtraction (String operand1, String operand2, int length)**

整数减法，可调用integerAddition方法实现。

例：integerSubtraction("0100", "0011", 8) 返回 "000000001" （4-3=1）

**参数：**

* operand1：二进制补码表示的被减数
* operand2：二进制补码表示的减数
* length：存放操作数的寄存器的长度，为8的倍数。length不小于操作数的长度，当某个操作数的长度小于length时，需要在高位补符号位

**返回值：**

* 长度为length+1的字符串表示的计算结果，其中第1位指示是否溢出（溢出为1，否则为0），后length位是相减结果

1. **public String integerMultiplication (String operand1, String operand2, int length)**

整数乘法，使用Booth算法实现，可调用integerAddition等方法。

例：integerMultiplication("0100", "0011", 8) 返回 "0000000000001100" （4×3=12）

**参数：**

* operand1：二进制补码表示的被乘数
* operand2：二进制补码表示的乘数
* length：存放操作数的寄存器的长度，为8的倍数。length不小于操作数的长度，当某个操作数的长度小于length时，需要在高位补符号位

**返回值：**

* 长度为2\*length的字符串表示的相乘结果

1. **public String integerDivision (String operand1, String operand2, int length)**

整数的不恢复余数除法，可调用integerAddition等方法实现。

例：integerDivision("0100", "0011", 8) 返回 "0000000100000001" （4÷3=1...1）

**参数：**

* operand1：二进制补码表示的被除数
* operand2：二进制补码表示的除数
* length：存放操作数的寄存器的长度，为8的倍数。length不小于操作数的长度，当某个操作数的长度小于length时，需要在高位补符号位

**返回值：**

* 长度为2\*length的字符串表示的相除结果，其中前length位为商，后length位为余数

1. **public String floatAddition (String operand1, String operand2, int eLength, int sLength, int gLength)**

浮点数加法，可调用intergerAddition等方法实现。

例：floatAddition("00111111010100000", "00111111001000000", 8, 8, 8) 返回 "001000000001110000" （0.8125+0.625=1.4375）

**参数：**

* operand1：二进制表示的被加数
* operand2：二进制表示的加数
* eLength：指数的长度，取值大于等于 8
* sLength：尾数的长度，取值大于等于 8
* gLength：保护位的长度

**返回值：**

* 长度为2+sLength+eLength的字符串表示的相加结果，其中第1位指示是否溢出（溢出为1，否则为0），其余位从左到右依次为符号、指数（移码表示）、尾数（首位隐藏）。舍入策略为向0舍入

1. **public String floatSubtraction (String operand1, String operand2, int eLength, int** **sLength, int gLength)**

浮点数减法，可调用floatAddition方法实现。

例：floatSubtraction("00111111010100000", "00111111001000000", 8, 8, 8) 返回 "000111110010000000"（0.8125-0.625=0.1875）

**参数：**

* operand1：二进制表示的被减数
* operand2：二进制表示的减数
* eLength：指数的长度，取值大于等于 8
* sLength：尾数的长度，取值大于等于 8
* gLength：保护位的长度

**返回值：**

* 长度为2+sLength+eLength的字符串表示的相减结果，其中第1位指示是否溢出（溢出为1，否则为0），其余位从左到右依次为符号、指数（移码表示）、尾数（首位隐藏）。舍入策略为向0舍入

1. **public String floatMultiplication (String operand1, String operand2, int eLength,** **int sLength)**

浮点数乘法，可调用integerAddition等方法实现。

例：floatMultiplication("00111110111000000", "00111111000000000", 8, 8) 返回 "00111110011000000" （0.4375×0.5=0.21875）

**参数：**

* operand1：二进制表示的被乘数
* operand2：二进制表示的乘数
* eLength：指数的长度，取值大于等于 8
* sLength：尾数的长度，取值大于等于 8

**返回值：**

* 长度为1+sLength+eLength的字符串表示的相乘结果,从左到右依次为符号、指数（移码表示）、尾数（首位隐藏）。舍入策略为向0舍入

1. **public String floatDivision (String operand1, String operand2, int eLength, int** **sLength)**

浮点数除法，可调用integerAddition等方法实现。

例：floatDivision("00111110111000000", "00111111000000000", 8, 8) 返回 "00111111011000000" （0.4375÷0.5=0.875）

**参数：**

* operand1：二进制表示的被除数
* operand2：二进制表示的除数
* eLength：指数的长度，取值大于等于 8
* sLength：尾数的长度，取值大于等于 8

**返回值：**

* 长度为1+sLength+eLength的字符串表示的相乘结果,从左到右依次为符号、指数（移码表示）、尾数（首位隐藏）。舍入策略为向0舍入

## 注意事项

1. 请在ALU.java文件开头的注释中填写自己的学号和姓名；
2. 不得修改上述方法的方法名和参数，允许在ALU.java文件中创建新的方法，但是不允许创建新的文件；
3. 所有方法必须采用指定算法实现；
4. 不需要进行输入检查，假设调用各个方法时所传入的参数都是符合要求的，例如在方法3) public String ieee754 (String number, int length)中，参数length取32或64，则不考虑length取其他值的情况。
5. 最后两个函数选做，完成的同学有附加分。