

本节内容

定点数

补码乘法运算

补码一位乘法

设机器字长为5位（含1位符号位， $n=4$ ）， $x = -0.1101$ ， $y = +0.1011$ ，采用Booth算法求 $x \cdot y$

$[x]_{\text{补}} = 1.0011$ ， $[-x]_{\text{补}} = 0.1101$ ， $[y]_{\text{补}} = 0.1011$

原码一位乘法：

进行 n 轮加法、移位

每次加法可能 $+0$ 、 $+ [|x|]_{\text{原}}$

每次移位是“逻辑右移”

符号位不参与运算



朋友，过两招？

根据当前MQ中的最低位来确定加什么

MQ中最低位 = 1时， $(ACC) + [|x|]_{\text{原}}$

MQ中最低位 = 0时， $(ACC) + 0$

补码一位乘法：

进行 n 轮加法、移位，最后再多来一次加法

每次加法可能 $+0$ 、 $+ [x]_{\text{补}}$ 、 $+ [-x]_{\text{补}}$

每次移位是“补码的算数右移”

符号位参与运算

根据当前MQ中的最低位、辅助位 来确定加什么

辅助位 - MQ中最低位 = 1时， $(ACC) + [x]_{\text{补}}$

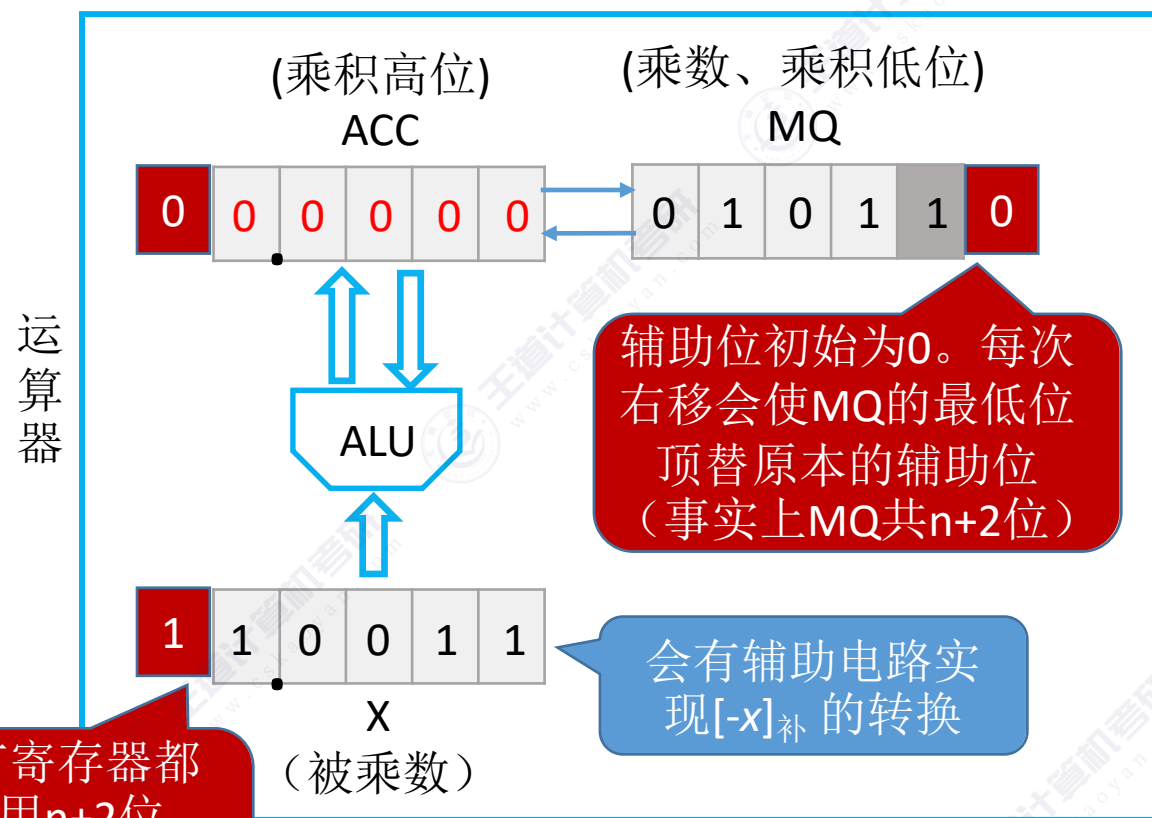
辅助位 - MQ中最低位 = 0时， $(ACC) + 0$

辅助位 - MQ中最低位 = -1时， $(ACC) + [-x]_{\text{补}}$

补码一位乘法

设机器字长为5位（含1位符号位， $n=4$ ）， $x = -0.1101$ ， $y = +0.1011$ ，采用Booth算法求 $x \cdot y$

$[x]_{\text{补}} = 1.0011$ ， $[-x]_{\text{补}} = 0.1101$ ， $[y]_{\text{补}} = 0.1011$



所有寄存器都统一用 $n+2$ 位，因此采用双符号位补码运算

补码一位乘法：

进行 n 轮加法、移位，最后再多来一次加法

每次加法可能 $+0$ 、 $+[x]_{\text{补}}$ 、 $+[-x]_{\text{补}}$

每次移位是“补码的算数右移”

符号位参与运算

根据当前MQ中的最低位、辅助位 来确定加什么

辅助位 - MQ中最低位 = 1时， $(ACC) + [x]_{\text{补}}$

辅助位 - MQ中最低位 = 0时， $(ACC) + 0$

辅助位 - MQ中最低位 = -1时， $(ACC) + [-x]_{\text{补}}$

补码一位乘法（手算模拟）

设机器字长为5位（含1位符号位， $n=4$ ）， $x = -0.1101$ ， $y = +0.1011$ ，采用Booth算法求 $x \cdot y$

$[x]_{\text{补}} = 11.0011$ ， $[-x]_{\text{补}} = 00.1101$ ， $[y]_{\text{补}} = 0.1011$

	(高位部分积)	(低位部分积/乘数)	说明
	00.0000	0.1011 0 丢失位	起始情况
正数算数右移	$+[-x]_{\text{补}}$ 00.1101	辅助位	$Y_4Y_5=10$ ， $Y_5-Y_4=-1$ ，则 $+[-x]_{\text{补}}$
	00.1101		
右移	00.0110	----- 10.101 10	右移部分积和乘数
	$+0$ 00.0000		$Y_4Y_5=11$ ， $Y_5-Y_4=0$ ，则 $+0$
	00.0110		
右移	00.0011	----- 010.10 110	右移部分积和乘数
负数算数右移	$+ [x]_{\text{补}}$ 11.0011		$Y_4Y_5=01$ ， $Y_5-Y_4=1$ ，则 $+ [x]_{\text{补}}$
	11.0110		
右移	11.1011	----- 0010.1 0110	右移部分积和乘数
	$+ [-x]_{\text{补}}$ 00.1101	原符号位	$Y_4Y_5=10$ ， $Y_5-Y_4=-1$ ，则 $+ [-x]_{\text{补}}$
	00.1000		
最后多一次加法	右移 00.0100	----- 00010. 10110	右移部分积和乘数
	$+ [x]_{\text{补}}$ 11.0011		$Y_4Y_5=01$ ， $Y_5-Y_4=1$ ，则 $+ [x]_{\text{补}}$
	11.0111		构成 $[x \cdot y]_{\text{补}}$

n轮加法、算数右移，加法规则如下：

辅助位 - MQ中最低位 = 1时， $(ACC) + [x]_{\text{补}}$

辅助位 - MQ中最低位 = 0时， $(ACC) + 0$

辅助位 - MQ中最低位 = -1时， $(ACC) + [-x]_{\text{补}}$

补码的**算数右移**：

符号位不动，数值位右移，正数右移补0，负数右移补1（符号位是啥就补啥）

注：一般来说，Booth算法的被乘数、部分积采用双符号位补码

$[x \cdot y]_{\text{补}} = 11.01110001$

即 $x \cdot y = -0.10001111$

知识点回顾

部分积、被乘数、乘数都可采用双符号位原码，也可用单符号位原码（手算时乘数的符号位可不写）

部分积、被乘数采用双符号位补码；乘数采用单符号位补码，并在末位添个0

原码一位乘法：

符号位通过异或确定，数值位由被乘数和乘数的绝对值进行 n 轮加法、移位

每次加法可能 $+0$ 、 $+ [|x|]_{\text{原}}$

每次移位是“**逻辑右移**”

乘数的符号位不参与运算

MQ中最低位 = 1时， $(ACC) + [|x|]_{\text{原}}$

MQ中最低位 = 0时， $(ACC) + 0$

补码一位乘法（Booth算法）：

符号位、数值位都是由被乘数和乘数进行 n 轮加法、移位，**最后再多来一次加法**

每次加法可能 $+0$ 、 $+ [x]_{\text{补}}$ 、 $+ [-x]_{\text{补}}$

每次移位是“补码的**算数右移**”

乘数的符号位参与运算

辅助位 - MQ中“最低位” = 1时， $(ACC) + [x]_{\text{补}}$

辅助位 - MQ中“最低位” = 0时， $(ACC) + 0$

辅助位 - MQ中“最低位” = -1时， $(ACC) + [-x]_{\text{补}}$



朋友，过两招？