arXiv:2304.09952v1 [cs.AR] 19 Apr 2023

面向 RISCV 处理器的 Baugh-Wooley 乘法

F.A. Grootjen[0000−0002−0548−5831] 和 N.K. Schauer[0009−0000−2211−8083]

人工智能，拉德堡德大学，荷兰奈梅亨

摘要。本文介绍了一种为 RISCV 处理器高效实现乘法指令的方法。本文不再使用三个分别用于有符号、无符号和混合乘法的预定义 IP 模块，而是提出了对 Baugh-Wooley 乘法算法的一种新颖扩展，使面积和功耗降低约三倍。

关键词：RISCV · CPU · 乘法

1 引言

在 RISC 指令集中，乘法指令可谓有些特立独行。几乎所有指令都只使用一个时钟周期，而乘法指令会占用大量资源，这一点在考察实现的延迟和/或面积时尤为明显。在 RISC-V 指令集架构中，*basic* 指令集中甚至没有乘法指令，因此低端实现可以遵循基本规范 [6]。当然，大多数（更大型的）实现都支持 M 扩展，该扩展提供乘法（和除法）指令。在 M 扩展中，存在无符号、有符号和混合乘法变体。乍看之下，拥有这些不同版本似乎有些奇怪：例如，对于 32 位乘法指令，操作数的符号属性并不影响结果[1]。但这只对结果的*lower 32 bits* 成立；对于较高的 32 位，操作数的符号属性确实会产生影响。大多数现代乘法实现都采用了某种改进形式的 *long multiplication*（见 2.1）。这些实现分为两个不同阶段：

1. 计算部分积 2. 对所有部分积求和

第一阶段相当简单，也很容易并行化；第二阶段则更为复杂。文献中提出了许多第二阶段的解决方案，涵盖了传统的速度与面积权衡：从简单的二叉加法树，到更精巧但占用面积较大的解决方案 [3,4] 和 [5,2]。

本文旨在说明，只需对第 1 阶段稍作调整，就能让第 2 阶段适用于三种乘法形式：有符号、无符号和混合形式。第2节介绍长乘法，并说明其如何处理二进制无符号数列。第 3 节推导了 Baugh 和 Wooley 对有符号数的处理方法。第 4 节说明如何对混合有符号数进行类似调整。最后，第 5 节将这些方法整合起来，并说明如何使用单个乘法器实现所有 RV32 乘法指令。

2 乘法

乘法的数学定义基于 *repeated addition*：

定义 1（乘法）。 *For* $a \in \mathbb{N}$ *and* $b \in \mathbb{Z}$ *the product* $a \cdot b$ *is defined as:*

$$a \cdot b = \underbrace{b + b + \cdots + b}_{a \text{ times}} = \sum_{i=1}^{a} b$$

[1] 本文将说明这一点。令人惊讶的是，这并不像你预想的那样简单。

在此定义中，数 a 称为 *multiplier*，而 b 称为 *multiplicand*。结果 $a \cdot b$ 称为 *product*。请注意，根据定义，当 $a = 0$ 时，乘积为 0。此外，我们还可以通过如下定义将此定义扩展到 $a \in \mathbb{Z}$：$a \cdot b = -((-a) \cdot b)$，其中 $a < 0$。

在计算机中，整数的大小是有限的。在 RV32 指令集中，整数为 32 位。因此，乘法运算接收两个 32 位整数，并产生一个 64 位乘积。mul 指令会存储乘积的低 32 位。至于高 32 位，则要根据数值格式（有符号、无符号或混合格式）使用相应的指令。

整数以二进制位序列的形式存储。无符号数采用以下格式：

定义 **2** *The unsigned number $a \in$ is represented as a*（无符号数）。*N binary sequence $a_{n-1}a_{n-2}\ldots a_1a_0$ with:*

$$a = \sum_{i=0}^{n-1} a_i \cdot 2^i$$

其中，$a_i \in \{0, 1\}$ 和 n 表示数值的位数。请注意，以这种方式表示的最大数值为 $2^n - 1$。对于 32 位数，这个值是 4294967295。

2.1 长乘法

二进制序列的无符号乘法可以使用 *long multiplication*，但要以二进制方式进行。表 1 展示了两个 4 位数 1001（9）和 0101（5）的长乘法。

				1	0	0	1	(=9)
			×	0	1	0	1	(=5)
				1	0	0	1	
		0	0	0	0			
	1	0	0	1				
+	0	0	0	0				
0	0	1	0	1	1	0	1	(=45)

表 1. 两个 4 位数的长乘法

二进制序列的长乘法算法非常直观：从乘数的最低有效位开始，仅当该位为 1 时写下被乘数，否则写下零。我们对其他各位重复此过程（位置为 i），并将被乘数向左移位 i 位后写下。这样我们就得到 n 个部分积。最终乘积是所有部分积之和。用数学方法表示长乘法相对容易：

$$a \cdot b = \left(\sum_{i=0}^{n-1} a_i \cdot 2^i\right) \cdot b = b \cdot \sum_{i=0}^{n-1} a_i \cdot 2^i = \sum_{i=0}^{n-1} b \cdot a_i \cdot 2^i$$

如你所见，乘积是 n 个部分积之和。在部分积中，2^i 表示左移，而由于 a_i 只能是 0 或 1，因此乘积 $b \cdot a_i$ 要么全为零，要么就是被乘数本身。

2.2 硬件实现

在硬件中执行上述算法涉及两项任务：

1. 计算 n 个部分积，这些部分积可以并行计算；2. 计算所有部分积之和，例如使用加法器树

对于一个部分积，我们需要计算 $b \cdot a_i \cdot 2^i$。如上所述，2^i 实际上是一个 *fixed* 移位（取决于 i）。由于 $a_i \in \{0, 1\}$ 与 b 的乘法可以在将 a_i 符号扩展为长度为 n的向量后，通过向量按位与运算完成。

2.3 示例

对于4×4位乘法（得到8位结果），无符号乘法方案如下：

$$
\begin{array}{rcccccccc}
 & & & & b_3 & b_2 & b_1 & b_0 \\
 & & & \times & a_3 & a_2 & a_1 & a_0 \\
\hline
 & & & & a_0b_3 & a_0b_2 & a_0b_1 & a_0b_0 \\
 & & & a_1b_3 & a_1b_2 & a_1b_1 & a_1b_0 & \\
 & & a_2b_3 & a_2b_2 & a_2b_1 & a_2b_0 & & \\
+ & a_3b_3 & a_3b_2 & a_3b_1 & a_3b_0 & & & \\
\hline
 & \multicolumn{8}{c}{\cdots}
\end{array}
$$

3 有符号乘法

有符号数以二进制序列存储，但有一个特殊之处：

定义 **3** *The (2's complement) signed number $a \in$ is repre-*（有符号数）。Z *sented as a binary sequence $a_{n-1}a_{n-2}\ldots a_1a_0$ with:*

$$a = -a_0 \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i$$

其中，$a_i \in \{0, 1\}$ 和 n 表示位数。请注意，以这种方式表示的最小负数（最大正数）分别为 -2^{n-1}（$2^{n-1} - 1$）。对于 32 位数，这分别是 -2147483648 和 2147483647。

这种表示二进制补码有符号数的方法有一个优点：普通加法（和减法）仍然可以正确运算。但对于有符号乘法则不然。Baugh 和 Wooley [1] 找到了巧妙的方法，只需稍作修改，就能对有符号数使用无符号乘法。下面推导他们的方法：

$$a \cdot b = \left(-a_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i\right) \cdot \left(-b_{n-1} \cdot 2^{n-1} + \sum_{j=0}^{n-2} b_i \cdot 2^j\right) =$$

$$\underbrace{a_{n-1}b_{n-1} \cdot 2^{2n-2}}_{A} + \underbrace{\sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_ib_j \cdot 2^{i+j}}_{B} \underbrace{- 2^{n-1}\sum_{i=0}^{n-2} a_ib_{n-1} \cdot 2^i}_{X} \underbrace{- 2^{n-1}\sum_{j=0}^{n-2} a_{n-1}b_j \cdot 2^j}_{Y}$$

注意，前两项（A 和 B）为正，因此不会造成问题。后两项为负，因此我们要找出它们对应的二进制补码形式，这样就可以直接将它们相加。

将第一个负项（不含符号）称为 X，第二个负项称为 Y。我们暂时关注 X，因为 Y 的推导类似。所以：

$$X = 2^{n-1}\sum_{i=0}^{n-2} a_ib_{n-1} \cdot 2^i$$

由于我们关注 X 的二进制表示，因此定义：

$$x_i = a_ib_{n-1}$$

假设 X 的位宽为 $2n$ 位，我们可以写出它的二进制表示：

bit position	$2n-1$	$2n-2$	$2n-3$	$2n-4$	$\ldots$	n	$n-1$	$n-2$	$n-3$	$\ldots$	0
bit value X	0	0	x_{n-2}	x_{n-3}	$\ldots$	x_1	x_0	0	0	$\ldots$	0

X 的二进制补码可通过将所有位取反后加 1 得到：

bit position	$2n-1$	$2n-2$	$2n-3$	$2n-4$	$\dots$	n	$n-1$	$n-2$	$n-3$	$\dots$	0
bit value $-X$	1	1	$\overline{x}_{n-2}$	$\overline{x}_{n-3}$	$\dots$	$\overline{x}_1$	$\overline{x}_0$	1	1	$\dots$	1+1

请注意，位值行尚未完成；位位置 0 处的加法会产生进位，并一路向上传递到位置 $n-1$:

bit position	$2n-1$	$2n-2$	$2n-3$	$2n-4$	$\dots$	n	$n-1$	$n-2$	$n-3$	$\dots$	0
bit value $-X$	1	1	$\overline{x}_{n-2}$	$\overline{x}_{n-3}$	$\dots$	$\overline{x}_1$	$\overline{x}_0+1$	0	0	$\dots$	0

同样，我们将求出 $-Y$:

bit position	$2n-1$	$2n-2$	$2n-3$	$2n-4$	$\dots$	n	$n-1$	$n-2$	$n-3$	$\dots$	0
bit value $-Y$	1	1	$\overline{y}_{n-2}$	$\overline{y}_{n-3}$	$\dots$	$\overline{y}_1$	$\overline{y}_0+1$	0	0	$\dots$	0

将 $-X$ 和 $-Y$ 相加时，有几个值得注意的位位置:

位位置	$2n-1$	$2n-2$	n	$n-1$
位值 $-X$	1	1	$\overline{x}_1$	$\overline{x}_0+1$
位值 $-Y$	1	1	$\overline{y}_1$	$\overline{y}_0+1$
位值 $-X-Y$	1	0	$\overline{x}_1+\overline{y}_1+1$	$\overline{x}_0+\overline{y}_0$

因此，加上 $-X-Y$ 可以通过将取反后的 x 和 y 位相加来完成，随后再将位位置为 $2n-1$ 和 n 的位相加。

3.1 示例

对于 4x4 转 8 位，有符号乘法方案如下:

			b_3	b_2	b_1	b_0	
		$\times$	a_3	a_2	a_1	a_0	
	a_3b_3						A
				a_0b_2	a_0b_1	a_0b_0	B
			a_1b_2	a_1b_1	a_1b_0		B
		a_2b_2	a_2b_1	a_2b_0			B
	$\overline{a_3b_2}$	a_3b_1	a_3b_0				Y
	a_2b_3	a_1b_3	a_0b_3				X
$+1$		1					X,Y
		$\dots$					

将各项合并后得到:

				b_3	b_2	b_1	b_0
			$\times$	a_3	a_2	a_1	a_0
			1	$\overline{a_0b_3}$	a_0b_2	a_0b_1	a_0b_0
			$\overline{a_1b_3}$	a_1b_2	a_1b_1	a_1b_0	
		$\overline{a_2b_3}$	a_2b_2	a_2b_1	a_2b_0		
$+1$	a_3b_3	$\overline{a_3b_2}$	a_3b_1	$\overline{a_3b_0}$			
			$\dots$				

将此方案与无符号方案（见第2节）进行比较，很容易看出二者的相似之处。如果只考虑输出的低 n 位，两种方案会产生相同的输出。乍看之下，这并不明显，因为无符号方案使用 a_0b_3 和 a_3b_0，而有符号方案使用 $\overline{a_0b_3}$ 和 $\overline{a_3b_0}$。不过，两者的和（而非进位）相同，见下表:

a_0b_3	a_3b_0	进位		$\overline{a_0b_3}$	$\overline{a_3b_0}$	进位	和	
0	0	0	0	1	1	1	0	
0	1	0	1	1	0	0	1	
1	0	0	1	0	1	0	1	
1	1	1	0	0	0	0	0	

4 混合乘法

RISCV 指令集 [6] 有一条特殊指令，它接受一个 *unsigned* 乘数和一个 *signed* 被乘数：

MULHSU rd, rs1, rs2

将有符号操作数 rs1（被乘数）与无符号操作数 rs2（乘数）相乘，并将乘积的高 32 位存入 rd。我们将尝试推导一种适用于混合操作数的 Baugh-Wooley 乘法扩展。令 a 为无符号乘数，b 为有符号被乘数。

$$a \cdot b = \left(\sum_{i=0}^{n-1} a_i \cdot 2^i\right) \cdot \left(-b_{n-1} \cdot 2^{n-1} + \sum_{j=0}^{n-2} b_i \cdot 2^j\right) =$$

$$\underbrace{\sum_{i=0}^{n-1}\sum_{j=0}^{n-2} a_i b_j \cdot 2^{i+j}}_{B} \underbrace{- 2^{n-1} \sum_{i=0}^{n-1} a_i b_{n-1} \cdot 2^i}_{X}$$

第一项（B）为正，因此没有问题。第二项 X 为负，因此我们将求出它的二进制补码表示，以便将其相加。由于我们需要 X 的二进制表示，定义：

$$x_i = a_i b_{n-1}$$

假设 X 的宽度为 $2n$ 位，我们可以写出它的二进制表示：

bit position	$2n-1$	$2n-2$	$2n-3$	$\ldots$	n	$n-1$	$n-2$	$n-3$	$\ldots$	0
bit value X	0	x_{n-1}	x_{n-2}	$\ldots$	x_1	x_0	0	0	$\ldots$	0

X 的二进制补码可以通过将所有位取反并加 1 来计算：

bit position	$2n-1$	$2n-2$	$2n-3$	$\ldots$	n	$n-1$	$n-2$	$n-3$	$\ldots$	0
bit value $-X$	1	$\overline{x}_{n-1}$	$\overline{x}_{n-2}$	$\ldots$	$\overline{x}_1$	$\overline{x}_0$	1	1	$\ldots$	1+1

注意，位值行并非最终结果；位位置 0 处的加法会产生一个进位，并一路传递到位置 $n-1$：

bit position	$2n-1$	$2n-2$	$2n-3$	$\ldots$	n	$n-1$	$n-2$	$n-3$	$\ldots$	0
bit value $-X$	1	$\overline{x}_{n-1}$	$\overline{x}_{n-2}$	$\ldots$	$\overline{x}_1$	$\overline{x}_0 + 1$	0	0	$\ldots$	0

4.1 示例

对于 4x4 位乘法，得到 8 位结果时，混合乘法方案如下：

$$
\begin{array}{rcccccc}
 & & b_3 & b_2 & b_1 & b_0 & \\
 & \times & a_3 & a_2 & a_1 & a_0 & \\
\hline
 & & & a_0b_2 & a_0b_1 & a_0b_0 & B \\
 & & a_1b_2 & a_1b_1 & a_1b_0 & & B \\
 & a_2b_2 & a_2b_1 & a_2b_0 & & & B \\
 a_3b_2 & a_3b_1 & a_3b_0 & & & & B \\
\overline{a_3b_3} & \overline{a_2b_3} & \overline{a_1b_3} & \overline{a_0b_3} & & & X \\
+1 & & & 1 & & & X \\
\hline
 & & \circ\ \circ & & & & \\
 & & \circ & & & &
\end{array}
$$

将各项合并后得到：

$$
\begin{array}{rcccccc}
 & & b_3 & b_2 & b_1 & b_0 \\
 & \times & a_3 & a_2 & a_1 & a_0 \\
\hline
 & & \overline{a_0b_3} & a_0b_2 & a_0b_1 & a_0b_0 \\
 & \overline{a_1b_3} & a_1b_2 & a_1b_1 & a_1b_0 & \\
\overline{a_2b_3} & a_2b_2 & a_2b_1 & a_2b_0 & & \\
1\ \overline{a_3b_3} & a_3b_2 & a_3b_1 & a_3b_0 & & \\
+ & & & 1 & & \\
\hline
 & & \cdots & & &
\end{array}
$$

最后一行（在位位置 $n-1$ 加上 1）有些不太理想。如果我们计划使用树形加法器完成所有加法，这一行就会导致额外的一次（非并行）操作。不过有个技巧可以让这个方案更加紧凑：

$$
\begin{array}{rcccccc}
 & & b_3 & b_2 & b_1 & b_0 \\
 & \times & a_3 & a_2 & a_1 & a_0 \\
\hline
 & & \overline{a_0b_3} & a_0b_3 & a_0b_2 & a_0b_1 & a_0b_0 \\
 & & \overline{a_1b_3} & a_1b_2 & a_1b_1 & a_1b_0 & \\
 & \overline{a_2b_3} & a_2b_2 & a_2b_1 & a_2b_0 & & \\
+1\ \overline{a_3b_3} & a_3b_2 & a_3b_1 & a_3b_0 & & & \\
\hline
 & & \cdots & & & &
\end{array}
$$

很容易看出最后这一步是正确的：对于所有 a_0b_3 的值，它都等于 $\overline{a_0b_3}+1$（不考虑进位）。进位本身等于 $\overline{a_0b_3}$。

		$\overline{a_0b_3}+1$	
a_0b_3	$\overline{a_0b_3}$	进位和	
0	1	1	0
1	0	0	1

5 合并

假设我们有二进制信号 s、u 和 m，分别表示有符号、无符号和混合乘法（可能通过解复用器对 RISC-V 乘法指令进行解码得到，见表 2），现在可以将所有实现合并为单个乘法器 [2]。本质上，部分积有 3 种不同类型：第一个、最后一个以及中间的所有部分积。我们将使用 $\oplus$ 表示按位异或运算符。未提及的位位置均视为 0。

[2] 请注意，我们实际上并未使用 u 信号，而是利用了这样一个事实：对于无符号乘法，s 和 m 均为 0。

指令说明	s	u	m
`mulh`	有符号的		1 0 0
无符号高位乘法	无符号	0	1 0
有符号×无符号高位乘法	混合		0 0 1

表 2. RV32 乘法指令及其解复用信号

5.1 第一个部分积

result bit	$s + m\overline{a_0 b_{n-1}}$	$s \oplus a_0 b_{n-1}$	$a_0 b_{n-2}$	$\ldots$	$a_0 b_1$	$a_0 b_0$
bit position	n	$n-1$	$n-2$	$\ldots$	1	0

如你所见，对于从 0 到 $n-2$（含）的位位置，第一部分积是标准形式（未改变）。在有符号乘法中，位位置 $n-1$ 取反；而在混合符号乘法中，位位置 n 为 1，位位置 $a_0 b_{n-1}$ 则取反。

5.2 中间部分积

中间部分积（从 1 到 $n-2$）会根据其（固定）编号进行移位。

result bit	$(s+m) \oplus a_i b_{n-1}$	$a_i b_{n-2}$	$\ldots$	$a_i b_1$	$a_i b_0$
bit position	$n-1+i$	$n-2+i$	$\ldots$	i+1	i

除了移位之外，位值均保持不变。执行有符号乘法或混合符号乘法时，只有最高有效位会被取反。

5.3 最后一个部分积

result bit	$s+m$	$m \oplus a_{n-1} b_{n-1}$	$s \oplus a_{n-1} b_{n-2}$	$\ldots$	$s \oplus a_{n-1} b_1$	$s \oplus a_{n-1} b_0$
bit position	$2n-1$	$2n-2$	$2n-3$	$\ldots$	n-2	n-1

对于最后一个部分积，在进行有符号乘法时，我们必须将其位值取反。然而，只有在进行混合符号乘法时，才应将最高有效位取反。最后，在有符号乘法和混合符号乘法中，结果前都应加上一个 1（位位置 $2n-1$）。

6 结论

我们证明了，只需对部分积进行一些小的修改，所有 RV32 乘法都可以由单个乘法器处理，从而使实现复杂度大约降低三倍。这对于多核和类向量实现尤其有益。

参考文献

1. Baugh, C., Wooley, B：《二进制补码并行阵列乘法算法》。IEEE计算机学报 C-22(12), 1045–1047 (1973)。https://doi.org/10 223648 2 1109 1973 . /T-C. . . Dadda, L.：《并行乘法器的一些方案》。Alta Frequenza 34(5), 349–356 (1965年5月)

3. Sakthikumaran, S., Salivahanan, S., Bhaaskaran, V.S.K., Kavinilavu, V., Brindha, B., Vinoth, C.：《一种极快且低功耗的进位选择加法器电路》。载于：2011年第3届国际电子与计算机技术会议。第1卷，第273–276页 (2011)。https://doi.org/10 5941604 4 1109 2011 . /ICECTECH. . . Swathi, P., Kumar, R., Ramana, B.：《采用改进型进位旁路加法的低密度、低延迟优化多操作数二进制加法器》。International Journal of Engineering in Advanced Research Science and Technology 03(01), 285–294 (2021年9月) 5. Wallace, C.S.：《一种快速乘法器的构想》。IEEE Transactions on Electronic Computers EC-
13 1 14 17 1964 10 1109 1964 263830 ( ), – (February)。
6
https://doi.org/ . /PGEC. . . Waterman, A., Lee, Y., Patterson, D.A., Asanovic, K.：《RISC-V指令集手册，第一卷：用户级指令集架构，版本2.1》。技术报告 UCB/EECS-2016-118，电气工程与计算机科学系，加州大学伯克利分校 (2016年5月)，http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html