1 booth 编码+华莱士树乘法器

原理

32 位补码乘法公式：

$$[X \times Y]_{补} = [X]_{补} \times (-y_{31} \times 2^{31} + y_{30} \times 2^{30} + \ldots + y_1 \times 2^1 + y_0 \times 2^0)$$

$$= x_{31}x_{30}...x_0 \bullet \overline{y_{32}}y_{31}...y_0$$

即两个 32 位补码乘法可以转换成 32 个部分积之和。

2 位 Booth 编码：

$$-y_{31} \times 2^{31} + y_{30} \times 2^{30} + \ldots + y_1 \times 2^1 + y_0 \times 2^0$$

$$= -y_{31} \times 2^{31} + y_{30} \times 2^{30} + 2y_{29} \times 2^{29} - y_{29} \times 2^{29} + \ldots + y_1 \times 2^1 + y_0 \times 2^0$$

$$= (-2y_{31} + y_{30} + y_{29}) \times 2^{30} - y_{29} \times 2^{29} + \ldots + y_1 \times 2^1 + y_0 \times 2^0$$

$$= (-2y_{31} + y_{30} + y_{29}) \times 2^{30} + (-2y_{29} + y_{28} + y_{27}) \times 2^{28} + ... + (-2y_3 + y_2 + y_1) \times 2^2 - y_1 \times 2 + y_0$$

$$令 - y_1 \times 2 + y_0 = (-2y_1 + y_0 + y_{-1}) \times 2^0 (其中 y_{-1} = 0)$$

$$= \sum_{i=0}^{15} (-2y_{2i+1} + y_{2i} + y_{2i-1}) \times 2^{2i} \qquad (y_{-1} = 0)$$

即通过提前对乘数编码，可使**部分积**个数减半。

部分积可通过下表快速查找：

编码表

y_{2i+1}	y_{2i}	y_{2i-1}	**部分积**
0	0	0	0
0	0	1	$+[x]_{补}$
0	1	0	$+[x]_{补}$
0	1	1	$+2[x]_{补}$
1	0	0	$-2[x]_{补}$
1	0	1	$-[x]_{补}$
1	1	0	$-[x]_{补}$
1	1	1	0

国科大体系结构实验指导书中例子：

比如 0101(5)×1001(-7)，其中$[X]_{补}$=+0101，$2[X]_{补}$=+01010，$-[X]_{补}$=-0101=+1011，$-2[X]_{补}$=-01010=+10110：

				0	1	0	1	
			×	1	0	0	1	0	(补 y_{-1})
+	0	0	0	0	1	0	1		(010,+$[X]_{补}$)
+	1	0	1	1	0				(100,-2$[X]_{补}$)
	1	0	1	1	1	0	1		(结果为-35 补码)

注：部分积是需要左移的

保留进位加法器

我们已经学过可用一位全加器串联形成行波进位加法器，其中全加器的逻辑：

$$s_i = x_i \oplus y_i \oplus c_i$$
$$c_{i+1} = x_i y_i + y_i c_i + x_i c_i$$

$$c_0 = 0$$

若把全加器三个输入中的进位输入替换成第三个加数，且将 s,c 看作两个独立的输出。则可以将三个 n 位补码相加转换成两个(n+1)位补码相加。

即:

$$a_n...a_1a_0 + b_n...b_1b_0 + c_n...c_1c_0 = s_n s_n...s_1 s_0 + c_n...c_1c_0 0$$

其中 $s_n s_n...s_1 s_0$ 中 s_n 表示经过有符号扩展，$c_n...c_1c_0 0$ 中 0 表示左移了一位。

华莱士树：

一层保留进位加法器可将输入减为 2/3，两层就可减为 4/9。通过堆叠成若干层保留进位加法器，直到最后一层**只有 2 个输出**即可构造一棵华莱士树。

32 位无符号乘法转化成 33 位有符号乘法

一个补码表示的数通过有符号扩展不会改变它的值，因此可以将输入的两个 32 位数乘法，统一表示为 33 位的有符号数乘法。最后根据是 mulu 还是 mul 指令，取结果的前 64 位作为输出。

实现

综上，算法步骤：

1） 将输入有符号扩展成 33 位数

2） 通过 booth 编码后可产生 17 个 33 部分积，略微不同于之前的式子，因为此时最高位为第 33 位，即 y_{32}。

$$\overline{y_{32}} y_{31}...y_0 = \sum_{i=0}^{16} (-2y_{2i} + y_{2i-1} + y_{2i-2}) \times 2^{2i-1} \qquad (y_{-1} = y_{-2} = 0)$$

3） 通过构造 6 层的华莱士树便可将输出减为 2 个，最后通过一个 64 位的加法器即可完成乘法的计算。

优化点：计算过程中的中间数不必使用 6 6 位，使用 6 5 位的数即可装下。原因：

有符号数相乘最大值：$(-2^{31})*(-2^{31})=2^{62}$

无符号数相乘最大值：$(2^{32}-1)*(2^{32}-1)<2^{65-1}-1$

都可用 6 5 位有符号数装下。

树的结构图

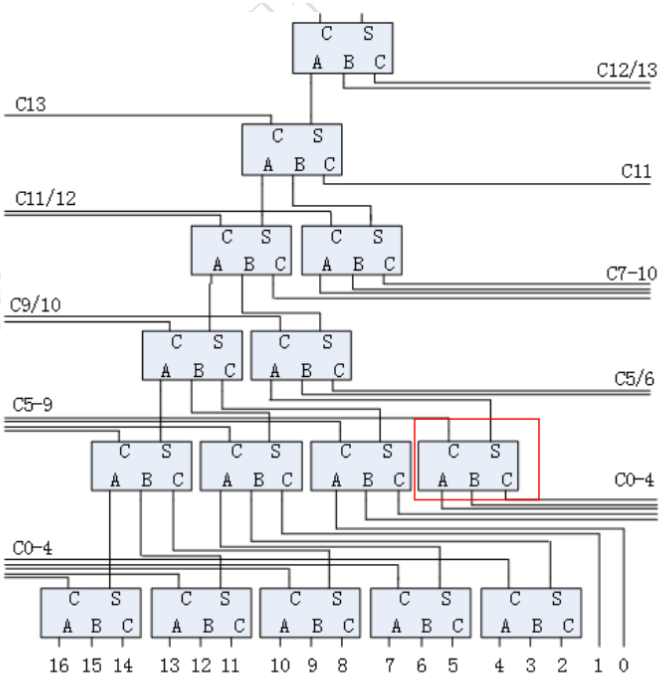

图 3　17 个数相加的 1 位华莱士树

来自国科大实验指导书

2 除法器

下图为软硬件接口第五版 127 页给出的除法器的结构。

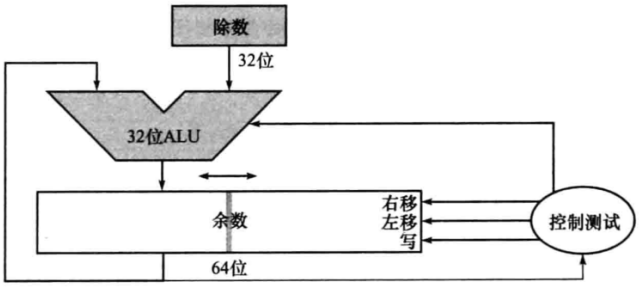

图 3-11 除法器的一种改进版本。除数寄存器、ALU、商寄存器都是 32 位，只有余数寄存器为 64 位。同图 3-8 相比，ALU 和除数寄存器都是位宽减半，余数进行左移。这个结构将商寄存器和余数寄存器的右半部分进行了拼接（正如图 3-5 中的那样，余数寄存器应该是 65 位从而保证加法器的进位不会丢失。）□

Radix-2 除法器

下图为一篇文献里作为对比模型的，对经典 radix-2 算法稍作改进（通过多路选择器**恢复余数**）的版本。(A High-Performance Data-Dependent Hardware Divider; Rainer Trummer, Peter Zinterhof, Roman Trobec)

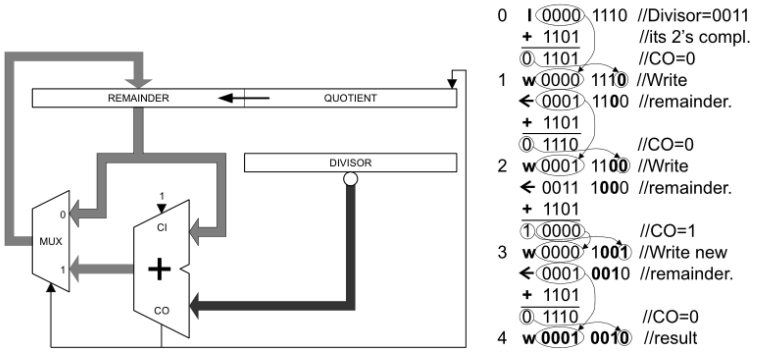

Figure 1: Schematic of Radix-2 divider (left) and demo division $0111_2/0011_2$ in binary (right).

后者结构结构更加简单清晰一些，只需要一个加法器，多路选择器，以及一个 64 位移位寄存器和一个 33 位除数寄存器

经典的 radix-2 除法器为恢复余数的，在一次减法余数小于 0 后，需要通过一次加法来恢复余数。

加减交替除法器也不需要恢复余数。

这里的除法器虽然结果上恢复了余数，但通过多路选择器可以实现并行（融合到移位寄存器里的多路选择器），因此速度和加减交替差不多，但结构更简单。

此除法器，每次迭代可得出 1 位商，共 32 次迭代可得到结果。添加到 alu 后，从输入到输出，最后需要 33 个周期。（除法器时钟对 cpu 时钟取了反，若不取反，需要 34 个周期）

Self-align 除法器

下图为上述同一篇文献里另一个除法器——Self Align 自对齐除法器。该除法器的迭代次数和输入有关，更确切的讲与输入两个数的位数（不包含前导零）之差有关。推导的近似公式：

计算 a/b

迭代次数= max{2*(a 的有效位数-b 的有效位数)+1, 1}

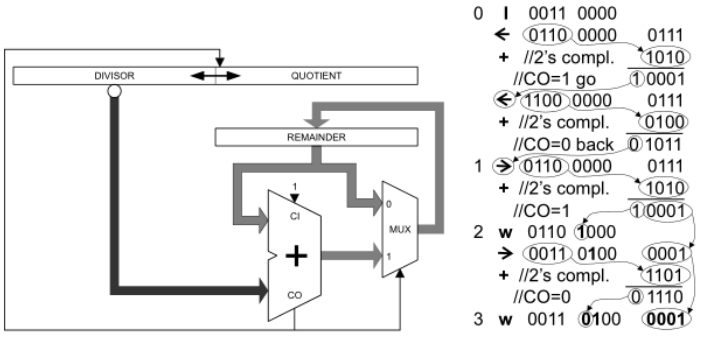

Figure 2: Schematic of Self-Aligning divider (left) and demo division $0111_2/0011_2$ in binary (right). Notation is the same as in Figure 1.

最后实现的代码计算周期（近似） = 3+max{2*(a 的有效位数-b 的有效位数), 0 }

测试

若被除数和除数都均匀地从 0~2^32-1 中取，则上式期望周期为 4.33。但这是没有意义的，因为实际中不可能会服从这个分布。a 更多情况下会大于 b，且 a，b 的范围也不一致。计算机仿真结果

rng1 = 1, 2**8 - 1

rng2 = 1, 2**16 - 1

rng3 = 1, 2**24 -1

rng4 = 1, 2**32 -1

a b 范围	平均周期	最大周期
rng2, rng1	18.97	35
rng3, rng2	19.00	51
rng4, rng3	19.01	62
rng4, rng2	35.00	67

下图为那篇论文里测试得到的数据。每一个数值代表相对于经典的 R2 (radix-2)算法速度上的提升。RT 即通过多路选择器对 R2 算法的改进版，SA 即上面的自对齐除法器，可以看到 SA 相对于 R2 有差不多 2 倍的提升。

Average Speedup				
Divider	16 bit	32 bit	64 bit	128 bit
RT	1.2	1.2	1.1	1.1
SA	2.1	2.7	3.2	3.5
DA	3.5	4.7	5.6	6.2
HA	4.2	6.0	7.3	8.2

Table 3: Average speedup of four dividers compared to R2.

加入了自己写的乘法器和除法器(self align)后，频率从 45MHz 提升到了 50MHz

性能测试得分：

原得分（45Mhz，乘法使用乘号，除法使用自己动手写 cpu 给的除法器）

序号	测试程序	myCPU 上板计时(16进制) 数码管显示	gs132 上板(16进制) 数码管显示	T_{gs132}/T_{mycpu}
cpu_clk : sys_clk		40MHz : 100MHz	50MHz : 100MHz	-
1	bitcount	ce2c0	13CF7FA	24.59840138
2	bubble_sort	445ff8	7BDD47E	28.98476908
3	coremark	e16c7b	10CE6772	19.08585075
4	crc32	71a2bd	AA1AA5C	23.95079673
5	dhrystone	2064a6	1FC00D8	15.68242579
6	quick_sort	580674	719615A	20.64613456
7	select_sort	45b1e9	6E0009A	25.25293523
8	sha	4ccaf1	74B8B20	24.31927107
9	stream_copy	5a493	853B00	23.61041721
10	stringsearch	45cabb	50A1BCC	18.48507187

性能分	22.139

改进后得分（50Mhz，self align 除法+booth 乘法（无流水））

序号	测试程序	myCPU 上板计时(16进制) 数码管显示	gs132 上板(16进制) 数码管显示	T_{gs132}/T_{mycpu}
cpu_clk : sys_clk		50MHz : 100MHz	50MHz : 100MHz	-
1	bitcount	bbc64	13CF7FA	27.00846417
2	bubble_sort	3dcf1a	7BDD47E	32.06378283
3	coremark	cee5c0	10CE6772	20.79486169
4	crc32	61d830	AA1AA5C	27.81623858
5	dhrystone	1d5072	1FC00D8	17.32957445
6	quick_sort	508802	719615A	22.56738336
7	select_sort	3ec5c6	6E0009A	28.03776938
8	sha	454576	74B8B20	26.95983185
9	stream_copy	52b02	853B00	25.77989312
10	stringsearch	404bc2	50A1BCC	20.06516746

性能分	24.458

时序（与 ip 对比）

Self align 除法 + 直接乘号	50MHz	wns = -0.08
Self align 除法 + 乘法 ip	50MHz	wns = 0.091
Self align 除法 + booth 乘法	50MHz	wns = 0.033
Self align 除法 + booth 乘法（3 级流水，周期数 3）	50MHz	wns = 0.075

加上李果得到的结果

除法 ip + booth 乘法	50MHz	wns = -0.039
除法 ip + 乘法 ip(5 级流水)	50MHz	wns = 0.11
除法 ip + 乘法 ip(5 级流水)	55MHz	wns = -0.571

3 axi 总线的 brust 传输