ГЛАВА iiiiii. Кодирование чисел и арифметика

1. Двоичные числа

Однородное позиционное по основанию 2 с естественным порядком весов представление числа определяется следующим правилом: $(\ldots b_3\, b_2\, b_1\, b_0\, \mathbf{.}\, b_{-1}\, b_{-2}\ldots)_2 =$

$$= \ldots + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 + b_{-1} 2^{-1} + b_{-2} 2^{-2} + \ldots,$$

$b_i \in \{0,1\}$ – бит (bit (BInary digiT)) занимающий разряд с номером $\mathbf{i}$ и с весом 2^i. Очень часто эти понятия (бит и разряд) подменяют друг друга (в английском: разряд – digit, position).

Точка, стоящая между b_0 и b_{-1}, разделяет целую и дробную части числа. Например,

$$(110.101)_2 = 2^2 + 2^1 + 0 + 2^{-1} + 0 + 2^{-3} = (6\ {}^5/_8)_{10}.$$

Такое представление числа для краткости будем называть *двоичным позиционным*.

Существует простая связь между записью чисел по основаниям 2 и 2^k:

$$(\ldots b_3\, b_2\, b_1\, b_0\, \mathbf{.}\, b_{-1}\, b_{-2}\ldots)_2 = (\ldots q_3\, q_2\, q_1\, q_0\, \mathbf{.}\, q_{-1}\, q_{-2}\ldots)_{2^k}$$

где $q_j \in \{0,1,\ldots,2^k-1\}$ цифра разряда $\mathbf{j}$ с весом 2^{kj}

$$q_j = (b_{kj+k-1}\ldots\, b_{kj+1}\, b_{kj})_2$$

В технической документации и текстах программ могут использоваться различные варианты обозначений таких чисел. Для выше приведённого примера, двоичное, четверичное (k=2), восьмеричное (k=3), шестнадцатеричное (k=4) представления могут иметь вид: 110.101b = 12.22f = 6.5o = 6.Ah, соответствнно. Для шестнадцатеричного представления часто используется запись, начинающаяся с нуля, точнее с 0x, позволяющая отличить число от строки символов: 0x6.A = 110.101b.

В табл.1 приведены различные представления одной тетрады (4 бит).

Таблица 1.

Двоичное binary (bin)	Четверичное four	Восьмеричное octal (oct)	Шестнадцатеричное hexadecimal (hex)
0000	00	00	0
0001	01	01	1
0010	02	02	2
0011	03	03	3
0100	10	04	4
0101	11	05	5
0110	12	06	6
0111	13	07	7
1000	20	10	8
1001	21	11	9
1010	22	12	A
1011	23	13	B
1100	30	14	C
1101	31	15	D
1110	32	16	E
1111	33	17	F

При машинных вычислениях длина слов, представляющих числа, фиксирована. Кроме того, форматы с разделительной точкой внутри слова не поддерживаются. Для представления чисел с целой и дробной частью используются два числа, одно из которых либо целое, либо дробь, второе – целое, указывающее место разделительной точки.

Поддерживаются следующие форматы чисел различной, но фиксированной длины:

- целые без знака (только положительные и ноль),
- дробные без знака (только положительные меньше единицы и ноль),
- целые со знаком, в дополнительном коде,
- целые со знаком, в смещённом коде,
- дробные со знаком, в дополнительном коде,
- в формате с плавающей точкой,
- двоично-десятичные числа.

2. Арифметика двоичных чисел

2.1. Дополнительный код числа

Дополнительным кодом (radix complement) кодируются целые и дробные числа со знаком.

2.1.1. Дополнительный код целых чисел. Пусть $\boldsymbol{\alpha}$ – целое число со знаком такое, что $-2^{n-1} \leq \boldsymbol{\alpha} < 2^{n-1}$, тогда число $\boldsymbol{\alpha}$ кодируется n–разрядным двоичным дополнительным кодом $A_Д$ следующим образом:

если $\boldsymbol{\alpha} \geq 0$, то $|A_Д| = |\boldsymbol{\alpha}| = \boldsymbol{\alpha}$,

если $\boldsymbol{\alpha} < 0$, то $|A_Д| = 2^n - |\boldsymbol{\alpha}| = 2^{n-1} + (2^{n-1} - |\boldsymbol{\alpha}|)$,

где $|\boldsymbol{\alpha}|$ – модуль числа,

$|A|$ – номер двоичного набора (номер набора представленный в двоичном позиционном коде и есть сам этот набор).

Например, n = 5

число	6	кодируется	00110,
число	–6	кодируется	11010.

Бит, занимающий в дополнительном коде старший разряд, для n–разрядных целых – это разряд с номером (n–1) является индикатором знака. При этом 0 – число не отрицательное, 1 – число отрицательное. Старший разряд, для n–разрядных целых – это разряд с номером (n–1). Если придать старшему разряду отрицательный вес (-2^{n-1}), то всегда $\boldsymbol{\alpha} = -b_{n-1}2^{n-1} + |a_Д|$, где $|a_Д|$ – номер набора без старшего разряда. Такая интерпретация полезна при умножении чисел в дополнительном коде (смотри ниже).

Отрицательных чисел в дополнительном коде на одно больше чем положительных:

максимальное n–разрядное положительное 011…1 = $2^{n-1}-1$,

минимальное n–разрядное отрицательное 100…0 = -2^{n-1}.

Дополнительный код обладает замечательным свойством, если $\boldsymbol{\alpha} \leftrightarrow A_Д$, $\boldsymbol{\beta} \leftrightarrow B_Д$, $\boldsymbol{\delta} \leftrightarrow C_Д$, $\boldsymbol{\alpha}+\boldsymbol{\beta}=\boldsymbol{\delta}$, $-2^{n-1} \leq \boldsymbol{\delta} < 2^{n-1}$, то суммируя дополнительные коды на каноническом n–разрядном сумматоре получим $A_Д+B_Д=C_Д$. Поэтому такой сумматор называют ещё сумматором дополнительных кодов.

2.1.2. Приведём пример умножения «столбиком» двух отрицательных чисел в дополнительном коде. Следует обратить внимание на цифры, выделенные жирным шрифтом и объяснить их происхождение.

				1	0	1	1	$А_Д \leftrightarrow \boldsymbol{\alpha} = -5$
				1	0	1	1	$В_Д \leftrightarrow \boldsymbol{\alpha} = -5$
				0	0	0	0	0 – частичное произведение
			+	1	0	1	1	$А_Д \bullet b_0$
			1	1	0	1	1	1 – частичное произведение
		+	1	0	1	1		$А_Д \bullet b_1 2$
		1	1	0	0	0	1	2 – частичное произведение
	+	0	0	0	0			$А_Д \bullet b_2 2^2$
	1	1	1	0	0	0	1	3 – частичное произведение
+	**0**	**1**	**0**	**1**				$А_Д \bullet(-b_3 2^3)$ умножение на старший разряд, разряд знака
0	0	0	1	1	0	0	1	2n – разрядный результат = +25

2.1.3. Изменение знака. Пусть $\boldsymbol{\alpha} \leftrightarrow А_Д$, $(-\boldsymbol{\alpha}) \leftrightarrow (-А)_Д$, n–разрядные дополнительные коды чисел с противоположными знаками, можно записать:

$$А_Д + (-А)_Д = 2^n$$

$$А_Д + \overline{А_Д} = 2^n - 1$$

где $\overline{А_Д}$ – означает инверсию всех разрядов набора $А_Д$.
(Последнее равенство справедливо для любых наборов.)

Отсюда: $(-А)_Д = \overline{А_Д} + 1$

Теперь операция вычитания сводится к операции сложения следующим образом:

$\boldsymbol{\alpha} \leftrightarrow А_Д$, $\boldsymbol{\beta} \leftrightarrow В_Д$, $\boldsymbol{\delta} \leftrightarrow С_Д$, $\boldsymbol{\alpha - \beta = \delta}$, $-2^{n-1} \le \boldsymbol{\delta} < 2^{n-1}$

$$А_Д + \overline{В_Д} + 1 = С_Д$$

2.1.4. Дополнительный код дробных чисел. Дополнительный код двоичных дробей называют «дополнением до двух» (twos complement), так как знаковый разряд дроби имеет вес $2^0=1$, или –1, в зависимости от интерпретации. Соответственно для двоичной n–разрядной дроби (со знаком та же дробь на один разряд больше) с весами разрядов $(q_0 .q_{-1}\ q_{-2}\ldots,\ q_{-n}) == q_0+q_{-1}2^{-1} +q_{-2}2^{-2} +\ldots+q_{-n}2^{-n}$, для дополнительных кодов выполняется равенство $А_Д+(-А)_Д=2$.

С дробями, в значительной мере, можно обращаться как с целыми числами. Двоичная n–разрядная дробь $0.q_{-1}q_{-2}\ldots q_{-j}\ldots q_{-n}$ может быть представлена как пара целых чисел $(B/2^n)$, где $B=b_{n-1}b_{n-2}\ldots b_{n-j}\ldots b_0$, $(b_{n-j}=q_{-j})$, или как несокращаемая дробь:

$0.q_{-1}q_{-2}\ldots q_{-(k-1)}10\ldots0 = (b_{k-1}b_{k-2}\ldots b_1 1)/2^k$, $(b_{k-j}=q_{-j})$,
где последняя значащая 1 в дроби стоит на (–k)-ом месте, тогда числитель дроби – нечётное число, простые делители знаменателя – только двойки. Например,

$$0.010010000 = 9/32$$
$$1.101110000 = -9/32.$$

2.1.5. Умножение дробей в дополнительном коде. Буквальное повторение примера п.2.1.2 с теми же двоичными наборами

$$\begin{array}{r l}
1.\ 0\ 1\ 1 & A_Д \leftrightarrow \boldsymbol{\alpha}= -5/8 \\
1.\ 0\ 1\ 1 & B_Д \leftrightarrow \boldsymbol{\alpha}= -5/8 \\
\hline
\mathbf{0}\ \ \mathbf{0.}\ 0\ 1\ 1\ 0\ 0\ 1 &
\end{array}$$

Дает правильный результат, но с двумя знаковыми разрядами. Разумеется, таким будет формат результата при умножении любых наборов, интерпретируемых как двоичные дроби в дополнительном коде. При умножении n–разрядных дробей точный результат, в общем случае, это – 2n–разрядная дробь. Умножение вместе со знаковыми разрядами дает результат на два разряда больше. Обычно при умножении дробей n младших разрядов отбрасывается, за исключением, может быть, одного, двух (с номерами –(n+1), –(n+2)), участвующих в округлении старших n разрядов.

2.2. Сложение в дополнительном коде

Канонический сумматор, складывая n–разрядные двоичные наборы, фактически складывает номера этих наборов. Рассмотрим подробнее, что происходит при сложении, если наборы интерпретируются как дополнительные коды целых чисел.

$\boldsymbol{\alpha}\leftrightarrow A_Д$, $\boldsymbol{\beta}\leftrightarrow B_Д$, $\boldsymbol{\delta}\leftrightarrow C_Д$, $\boldsymbol{\alpha}+\boldsymbol{\beta}=\boldsymbol{\delta}$,

2.2.1.1. $\boldsymbol{\alpha}\geq 0$, $\boldsymbol{\beta}\geq 0$, $\boldsymbol{\delta}<2^{n-1}$,
тогда $A_Д+B_Д= |A_Д| + |B_Д|= \boldsymbol{\alpha}+\boldsymbol{\beta}=\boldsymbol{\delta}=|C_Д|$.

Например,

$$\begin{array}{l l}
0010 & +2 \\
0101 & +5 \\
\hline
0111 & +7
\end{array}$$

2.2.1.2. $\boldsymbol{\alpha}>0$, $\boldsymbol{\beta}>0$, $\boldsymbol{\delta}\geq 2^{n-1}$ (разумеется, $2^n > \boldsymbol{\delta}$)
тогда $A_Д+B_Д= |A_Д| + |B_Д|= \boldsymbol{\alpha}+\boldsymbol{\beta}=\boldsymbol{\delta}=2^{n-1}+(\boldsymbol{\delta}-2^{n-1})$ – набор с 1 в старшем разряде. Складывая положительные числа, получили отрицательное число. Это является индикатором *переполнения* (overflow).

Например,
$$\begin{array}{rr} 0011 & +3 \\ 0101 & +5 \\ \hline 1000 & -8 \end{array}$$

2.2.2.1. $\boldsymbol{\alpha}<0$, $\boldsymbol{\beta}<0$, $|\boldsymbol{\delta}| \le 2^{n-1}$,
тогда $A_Д+B_Д = |A_Д|+|B_Д| = 2^{n-1}+(2^{n-1}-|\boldsymbol{\alpha}|)+2^{n-1}+(2^{n-1}-|\boldsymbol{\beta}|)=$
$= 2^n+2^{n-1}+(2^{n-1}-|\boldsymbol{\delta}|)=2^n+|C_Д|$,

Например,
$$\begin{array}{rr} 1110 & -2 \\ 1011 & -5 \\ \hline {}_1 1001 & -7 \end{array} \qquad \begin{array}{rr} 1101 & -3 \\ 1011 & -5 \\ \hline {}_1 1000 & -8 \end{array}$$

2.2.2.2. $\boldsymbol{\alpha}<0$, $\boldsymbol{\beta}<0$, $|\boldsymbol{\delta}| > 2^{n-1}$, (разумеется, $2^n \ge |\boldsymbol{\delta}|$)
тогда $A_Д+B_Д = |A_Д|+|B_Д| = 2^n - |\boldsymbol{\alpha}| + 2^n - |\boldsymbol{\beta}| = 2^n + 2^n - |\boldsymbol{\delta}|$
$2^{n-1} > (2^n - |\boldsymbol{\delta}|) \ge 0$, т.е. получили набор с 0 в старшем разряде. Складывая отрицательные числа, получили положительный результат. Это является индикатором переполнения.
Например,
$$\begin{array}{rr} 1101 & -3 \\ 1010 & -6 \\ \hline {}_1 0111 & +7 \end{array}$$

2.2.2.3. $\boldsymbol{\alpha}\ge 0$, $\boldsymbol{\beta}\le 0$, $\boldsymbol{\delta}=\boldsymbol{\alpha}+\boldsymbol{\beta}= |\boldsymbol{\alpha}|-|\boldsymbol{\beta}|$
тогда $A_Д+B_Д = |A_Д|+|B_Д| = |\boldsymbol{\alpha}|+2^n-|\boldsymbol{\beta}|)=2^n + \boldsymbol{\delta}$
если $\boldsymbol{\delta} \ge 0$ (разумеется, $\boldsymbol{\delta} < 2^{n-1}$), то $A_Д+B_Д = 2^n + \boldsymbol{\delta} = 2^n + |C_Д|$.

Например,
$$\begin{array}{rr} 0111 & +7 \\ 1010 & -6 \\ \hline {}_1 0001 & +1 \end{array}$$
если $\boldsymbol{\delta}<0$, ($|\boldsymbol{\delta}|\le 2^{n-1}$), то $A_Д+B_Д=2^n-|\boldsymbol{\delta}|=2^{n-1}+(2^{n-1}-|\boldsymbol{\delta}|)=|C_Д|$.

Например,
$$\begin{array}{rr} 0110 & +6 \\ 1001 & -7 \\ \hline 1111 & -1 \end{array}$$

2.2.3. Изменение знака числа выполняется как операция вычитания и также требует проверки на переполнение.

$$(-A)_Д = 0_Д + \overline{A_Д} + 1$$

Например, (0000) – (1000) = (0000) + $\overline{(1000)}$ + 1

$$
\begin{array}{r l}
0000 & \\
0111 & \\
\underline{1} & \\
1000 & \text{переполнение}
\end{array}
$$

Это единственный случай переполнения при изменении знака.

2.2.4. Механизм сложения дробей в дополнительном коде аналогичен, только интерпретация весов разрядов другая.

2.3. Сложение и сравнение чисел без знака

При сложении чисел без знака признаком переполнения является единица в самом старшем разряде суммы – выходном переносе.

При сравнении n–разрядных чисел без знака на n–разрядном сумматоре выполняется операция вычитания в дополнительных кодах без использования знаковых разрядов (их значения предопределены). Значение выходного переноса определяет знак результата.

X + $\overline{\text{Y}}$ +1. Например:

1101	D	1001	9	1101	D
0110	9	0010	D	0010	D
1		1		1	
10100	+4	**0**1100	–4	**10000**	+0

2.4. Сложение и вычитание в прямом коде

Прямой код (direct code), прежде всего, используется в формате с плавающей точкой. К основным разрядам, которые являются двоичным позиционным представлением модуля (абсолютного значения) числа, добавляется ещё один двоичный разряд, который кодирует алгебраический знак числа; ему не присваивается никакого определённого веса, поэтому и расположение его может быть произвольным. Возможна такая трактовка бита знака **s**, если |A| модуль, то число $A=(-1)^S \bullet |A|$. При арифметических манипуляциях с прямыми кодами возможно появление как «минус нуля», так и «плюс нуля».

Результат сложения или вычитания исходных чисел в прямом коде должен быть представлен в прямом коде. В тоже время операндами сумматора должны быть дополнительные коды, выход сумматора – дополнительный код. Поэтому следует минимизировать количество преобразований кодов.

Независимо от знаков чисел и знака операции число X участвует в сложении всегда как |X|. Код второго слагаемого Y зависит от знаков чисел и знака операции и равен либо |Y| либо $(-|Y|)_Д$. Код результата необходимо преобразовать, если сумма отрицательна.

$$X + \overline{Y} + 1 = S,\ \text{если } S<0, \text{ то } \overline{S} + 1 = |S|$$

Проще сделать следующим образом

$$X + \overline{Y} = S,\ \text{если } S<0, \text{ то } \overline{S} = |S|$$

Это следует из следующих соображений:

обозначим $X + \overline{Y} = W$, тогда $X + \overline{Y} + 1 = W + 1$

$$\overline{W+1} + W+1 = 2^n - 1$$

$$\overline{W} + W = 2^n - 1$$

отсюда $\overline{W+1} + 1 = \overline{W}$

Если $S \geq 0$, то $X + \overline{Y} + 1 = |S|$

Сложение на сумматоре можно выполнять без знаковых разрядов, поскольку все знаки предопределены. Выходной перенос в одном случае, когда оба операнда сумматора неотрицательные, будет индикатором переполнения, в другом – индикатором знака. Требуется также вычислить знак результата.

Сведём эти соображения в следующие таблицы и алгоритм, где
K – знак операции.
Z_X – знак числа X.
Z_Y – знак числа Y.
iY – признак преобразования кода числа, функция от K, Z_X, Z_Y.
P – выходной перенос сумматора.
Z_R – знак результата. Это значение проще вычислять отдельно при разных значениях признака iY (разных ветвях алгоритма).

K	Z_X	Z_Y	iY	P	Z_R
+	+	+	0		+
+	–	–	0		–
–	+	–	0		+
–	–	+	0		–
+	+	–	1	0	–
+	+	–	1	1	+
+	–	+	1	0	+
+	–	+	1	1	–
–	+	+	1	0	–
–	+	+	1	1	+
–	–	–	1	0	+
–	–	–	1	1	–

Если кодировать знак плюс – 0, знак минус – 1, тогда карты Карно:

iY \ Z_XZ_Y / K	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$iY = K \oplus Z_X \oplus Z_Y$

Z_R \ Z_XZ_Y / K	00	01	11	10
0	0	x	x	**1**
1	x	0	x	**1**

При iY=0

$Z_R = Z_X$

Z_R \ Z_XZ_Y / KP	00	01	11	10
00	x	**1**	x	0
01	x	0	x	**1**
11	0	x	**1**	x
10	**1**	x	0	x

При iY=1

$Z_R = \overline{Z_X \oplus P}$

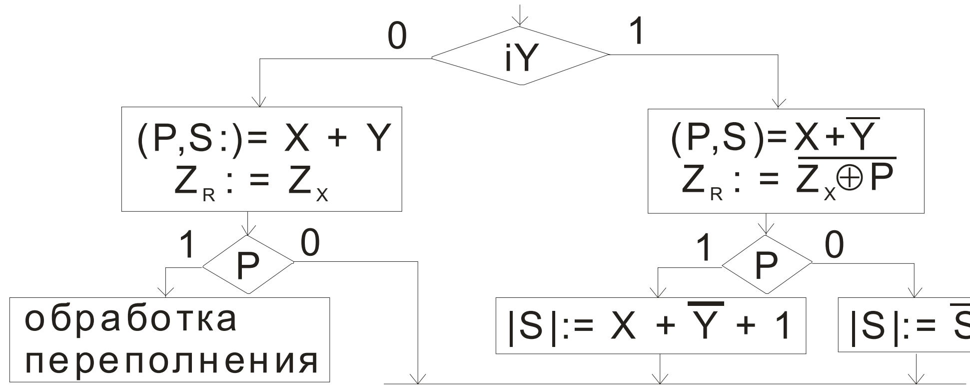

Рис.1. Алгоритм сложения/вычитания прямых кодов

Несмотря на существенно более сложный алгоритм в сравнении с дополнительными кодами, есть и преимущества – разрядность сумматора на единицу меньше.

2.5. Смещённый код

Смещённый (bias) код, прежде всего, используется для кодирования целых чисел со знаком, определяющих порядок числа в формате с плавающей точкой.

Номера n-разрядных наборов, кодирующих целые числа со знаком $\boldsymbol{\alpha}$ в смещенном коде A_C определяются следующим образом:

$|A_C| = C + \boldsymbol{\alpha}$, где константа (смещение) $C>0$, $2^n > (C+\boldsymbol{\alpha}) \geq 0$

Если для n–разрядного кода выбрать $C=2^{n-1}$, то для $(-2^{n-1} \leq \boldsymbol{\alpha} < 2^{n-1})$

$|A_C| = 2^{n-1} + \boldsymbol{\alpha}$, , если $\boldsymbol{\alpha} \geq 0$, то $|A_C| = 2^{n-1} + |\boldsymbol{\alpha}|$

если $\boldsymbol{\alpha} < 0$, то $|A_C| = 2^{n-1} - |\boldsymbol{\alpha}|$.

Во-первых, смещённый код удобен для сравнения чисел, если $\boldsymbol{\alpha} < \boldsymbol{\beta}$, то $|A_C| < |B_C|$. Во-вторых, если $C=2^{n-1}$, то смещённый код отличается от дополнительного кода того же числа только старшим (знаковым) разрядом. Это означает, что смещённый код столь же «арифметичен», как и дополнительный код. Складывая смещённые коды, результат получаем в дополнительном коде. Индикатор переполнения проще – два одинаковых бита в соседних разрядах знака и переноса. Изменение знака аналогично изменению знака дополнительного кода:

$$(-A)_C = \overline{A_C} + 1$$

2.6. Умножение

Умножение выполняется как итерационный процесс, когда текущее значение частичного произведения G_{i+1} получено из предыдущего значения G_i с учётом множимого A и очередного разряда множителя: $G_{i+1}=G_i+A\bullet b_i\bullet 2^i$, $i=0\div(n-1)$, $G_0=0$, G_n – результат (см. п.2.1.2.). Изменяющийся множитель 2^i делает эту итерацию не удобной для машинной реализации. Для более удобной реализации преобразуем это выражение: $G_{i+1}/2^i=G_i/2^i +A\bullet b_i$, пусть $H_i=G_i/2^i$, тогда $2\bullet H_{i+1}=H_i+A\bullet b_i$, или иначе

$$(H_{i+1},q_i)=(H_i+A\bullet b_i)/2. \qquad \text{(xx)}$$

Справа от равенства в скобках сумма n-разрядных слагаемых с (n+1)-разрядным результатом. Делением на 2 (сдвигом вправо) получаем n старших разрядов частичного произведения и остаток – один бит (q_i), который является i–битом результата (младших разрядов результата) с весом 2^i. $H_0=0$, H_n – старшие разряды результата, $Q=(q_{n-1} \ldots q_1\, q_0)$ – младшие разряды результата.

Машинная реализация этой (xx) итерационной идеи может быть самой разнообразной. Например:

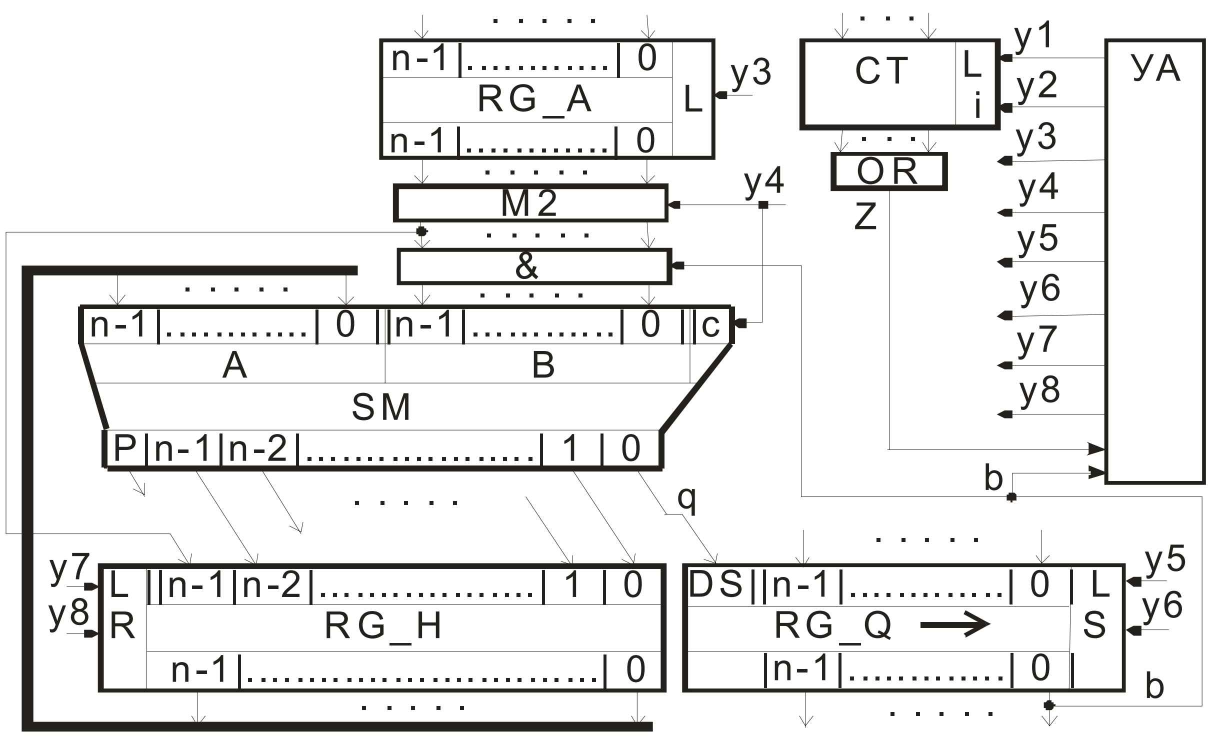

Рис.2. Схема умножения

Процедурное описание умножения целых чисел со знаком в реализации рис.2 будет следующим:

RG_A:=A -- множимое

RG_Q:=B -- множитель (младшие разряды произведения)

RG_H:=0 -- старшие разряды произведения

CT:=n-2

-- В дальнейшем вместо RG_X будем писать X

```
loop:   (H: , q)=(A[n-1],(H + A•b)/2)  -- b = Q[0]
        (Q: , x)=SR(q , Q)
        CT:=CT–1
        z =(CT≠0) -- значение CT как в правой части операции
                  -- присваивания
        If (z ≠0) Then Go_To loop
```

If (b==1) Then {(H: , q)=($\overline{A[n-1]}$, (H – A)/2)
(Q: , x)=SR(q , Q)}
Else {(H: , q)=(A[n-1] , (H + 0)/2)
(Q: , x)=SR(q , Q)}

Основной цикл вычисления (в рамке) выполняется минимум за один такт. Операция «/2»-деление на 2 обозначает косую передачу вправо. Операция «SR» означает сдвиг вправо (Shift Right).

Числа со знаком и числа без знака умножаются по-разному. Для чисел со знаком при сдвиге (косой передаче) частичного произведения в старший разряд записывается значение знакового разряда множимого. Для чисел без знака при сдвиге (косой передаче) частичного произведения в старший разряд записывается значение выходного переноса сумматора. При умножении без знака окончание процедуры тривиально в сравнении с умножением со знаком.

A	B	- числа со знаком числа без знака -		A	B
1011	1011			1011	1011
H	Q.B		P	H	Q.B
0000	.1011			0000	.1011
1011				1011	
1011		сумма	0	1011	
1101	1.101	косая передача и сдвиг мн-теля		0101	1.101
1011				1011	
1000		сумма	1	0000	
1100	01.10	косая передача и сдвиг мн-теля		1000	01.10
0000				0000	
1100		сумма	0	1000	
1110	001.1	косая передача и сдвиг мн-теля		0100	001.1
0101				1011	
0011		сумма	0	1111	
0001	1001.	косая передача и сдвиг мл. разряд.		0111	1001.

Умножение дробей, особенности см. п.п. 2.1.5.

2.7. Деление

Деление целых без знака. Рассмотрим алгоритм деления близкий (похожий) на школьный алгоритм деления уголком. Начнём с примера 13/3.

		делимое	делитель	
	0000	1101	0011	
			1101	Отрицательный делитель без знакового разряда
частное	остаток			
	0001	101x	сдвиг остатка (делимого)	
	1101		вычитание делителя	
0	1110		отрицательный остаток	
	0001	101x	восстановление положител. остатка	
	0011	01xx	сдвиг остатка	
	1101		вычитание делителя	
1	0000		положительный остаток	
	0000	1xxx	сдвиг остатка	
	1101		вычитание делителя	
0	1101		отрицательный остаток	
	0000	1xxx	восстановление положител. остатка	
	0001	xxxx	сдвиг остатка	
	1101		вычитание делителя	
0	1110		отрицательный остаток	
	0001	восстановленный положительный остаток (результат)		

1101b/0011b = 0100b – частное, 0001b – остаток.

13/3=4 – частное, 1 – остаток, при этом 13=3•4+1

Всегда при целочисленном делении A=B•E+R,

где A – делимое, B – делитель, E – частное, R – остаток.

Полное название алгоритма: ***деление с восстановлением и сдвигом остатка***.

Цифры частного получаются последовательно, начиная со старшего разряда: на первом шаге путём вычитания делителя из делимого удвоенной длины, а затем делителя из полученного остатка. Если получен неотрицательный остаток, то цифра частного равна единице, если остаток отрицательный, то цифра частного равна нулю, при этом восстанавливается предыдущий неотрицательный остаток.

Формальное описание итерационного процесса:

$$(e_{n-(i+1)}, A_{i+1}) = 2A^{+}_{i} - G,$$

где $G=B\bullet 2^{n}$, $i=0\div(n-1)$, $A^{+}_{0}=A$ – 2n-разрядное делимое, e_k – бит частного с весом 2^k, A^{+}_{n} – остаток-результат, A^{+}_{i} – неотрицательный восстановленный остаток

$A^{+}_{i} = A_i$, если $A_i \geq 0$

$A^{+}_{i} = A_i + G$, если $A_i < 0$,

последняя операция восстанавливает неотрицательный остаток. Как вариант можно не запоминать отрицательный остаток, а только положительный.

Деление без восстановления остатка. Итак, в выше рассмотренном алгоритме, если на очередной итерации получен остаток $A_i<0$, то $A_{i+1} = 2A^{+}_{i} - G = 2(A_i + G) - G = 2A_i + G$, т.е. вместо восстановления неотрицательного остатка на этом шаге надо прибавить делитель вместо вычитания.

$$(e_{n-(i+1)}, A_{i+1}) = 2A_i \begin{matrix} -\,G, \text{ если } A_i \geq 0 \\ +\,G, \text{ если } A_i < 0 \end{matrix}$$

Приведём пример возможной аппаратной реализации:

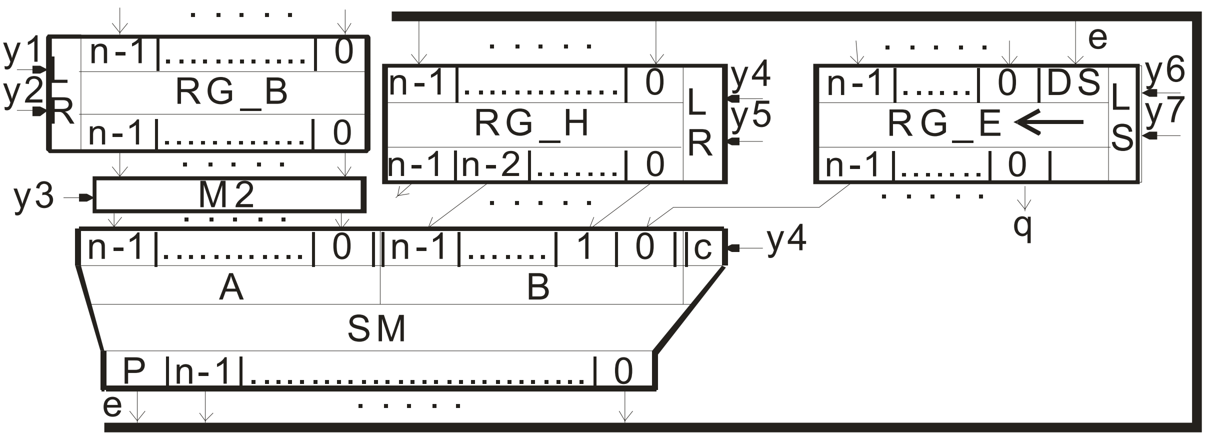

Рис.3. Схема деления

(Счетчик циклов и управляющая часть на схеме не показаны, см. схему умножения.)

Процедурное описание деления в реализации рис.3 будет следующим:

RG_E:=A -- делимое (частное)

RG_B:=B -- делитель

RG_H:=0 -- старшие разряды делимого

```
-- В дальнейшем вместо RG_X  будем писать  X
-- Деление с восстановлением остатка
        CT:= n
loop:   (e , S) = (2•H , E[n-1]) – B
        CT:=CT–1

        If (e ==1) Then {H:= S
                         (x , E:)=SL(E , e)}
                   Else (x , E:)=SL(E , e)
        z =(CT≠0)  -- значение CT как в левой части операции
                   -- присваивания
        If (z ≠0) Then Go_To loop

-- Деление без восстановления остатка
        CT:= n–1
        Go_To mm
          -- q = E[0]
loop:   If (q ==1)  Then
 mm:                     {(e , H:) = (2•H , E[n-1]) – B
                          (x , E:)=SL(E , e)}
                   Else {(e , H:) = (2•H , E[n-1]) + B
                          (x , E:)=SL(E , e)}
        CT:=CT–1
        z =(CT≠0)  -- значение CT как в правой части операции
                   -- присваивания
        If (z ≠0) Then Go_To loop
```

Операция «2•»- умножение на 2 обозначает косую передачу влево. Операция «SL» означает сдвиг влево (Shift Left).

Деление более «капризная» операция, чем умножение. Если используется алгоритм «деление без восстановления остатка», то надо добавить старший разряд к обоим операндам (можно считать его знаковым), иначе операции с делителем большим 2^{n-1} будут выполняться неверно. (См. также деление чисел со знаком в дополнительном коде.)

Сравним алгоритмы с восстановлением и без восстановления остатка. Основной критерий, по которому следует их сравнивать это – суммарное время выполнения деления. Основной вклад в это время вносит цикл, выполняемый n раз. В алгоритме «с восстановлением» он выполняется минимум за два такта, в алгоритме «без восстановления» – минимум за один такт.

Можно выполнять деление чисел со знаками непосредственно в дополнительных кодах.

Начнём с примера –13/3.

		делимое	делитель
	11111	10011	00011
частное	остаток		
	11111	0011x	сдвиг остатка (делимого)
	00011		сложение с делителем
1	00010		положительный остаток
	11111	0011x	восстановление отрицательн. остатка
	11110	011xx	сдвиг остатка
	00011		сложение с делителем
1	00001		положительный остаток
	11110	011xx	восстановление отрицательн. остатка
	11100	11xxx	сдвиг остатка
	00011		сложение с делителем
0	11111		отрицательный остаток
	11111	1xxxx	сдвиг остатка
	00011		сложение с делителем
1	00010		положительный остаток
	11111	1xxxx	восстановление отрицательн. остатка
	11111	xxxxx	сдвиг остатка
	00011		сложение с делителем
1	00010		положительный остаток
	11111	восстановленный отрицательный остаток (результат)	

Частное получилось равное –5. Всегда, если остаток не нулевой, то при целочисленном делении отрицательное частное на единицу меньше правильного результата. (Сравните с делением на 2 отрицательного числа в дополнительном коде сдвигом вправо.) Знак остатка должен быть равен знаку делимого. Разумеется, можно использовать алгоритм деления без восстановления остатка.

Рассмотрим различные варианты сочетания знаков операндов:

$A \geq 0$, $B > 0$, частное и остаток – положительные числа.

$A \geq 0$, $B < 0$ частное должно быть отрицательными, а получится положительным, придётся изменять знак частного, остаток должен быть положительным.

$A < 0$, $B > 0$ частное должно быть отрицательными и получится отрицательным. Ответ правильный, если остаток ноль (остаток равный $-|B|$ эквивалентен нулевому), если остаток не нулевой, то для получения правильного результата к отрицательному частному надо прибавить единицу. Остаток должен быть отрицательными.

$A < 0$, $B < 0$ при этом сочетании знаков лучше перед делением изменить знак делимого, тогда частное должно быть положительным и получится положительным. Остаток должен быть отрицательными.

При делении дробей должно выполняться $A < B$. Разрядность делимого не надо увеличивать в два раза. При делении дробей без знака надо добавлять знаковый разряд, иначе деление на число более ½ может быть неверным. При делении дробей со знаком в дополнительном коде, если делимое отрицательное и остаток не нулевой, то полученное отрицательное частное на единицу меньше младшего разряда дроби, чем правильный результат. В некоторых приложениях ошибка в младшем разряде может быть не существенной и тогда её не надо исправлять. Выше описанные алгоритмы деления дробей называют «необратимыми», в том смысле, что полученный в результате работы алгоритма не нулевой остаток неверен. Но при делении дробей чаще всего остаток игнорируется.

Сравним деление модулей чисел со знаком с делением в дополнительном коде. При делении модулей наихудшим надо считать случай $A < 0$, $B > 0$ – придется изменять знак у четырёх чисел. При делении в дополнительном коде надо менять знак только одного числа (не считая остатка), либо скорректировать отрицательное частное на единицу.

2.8. Числа в формате с плавающей точкой

Числа в этом формате содержат три поля:

– поле (бит) знака числа S (sign bit).

– поле порядка (характеристики) Pc (binary biased exponent (characteristic)), основанием порядка будем считать 2 (самый распространённый вариант, хотя возможно и 2^k), сам порядок кодируется смещенным кодом. Истинный порядок числа P=(|Pc|–C), где C – смещение.
– поле мантиссы M (mantissa);

Истинное значение числа X определяется выражением:

$$X = (-1)^S \bullet M \bullet 2^{|Pc|-C}$$

Будем считать, что мантисса положительная нормализованная двоичная дробь. Понятие *нормализованная* означает, что старший разряд дроби с весом 2^{-1} содержит 1, т.е. $1 > M \geq ½$. (Используются и другие форматы мантиссы, например, с одним битом в целой части числа, тогда нормализованная мантисса $2 > M \geq 1$.)

CISC процессоры фирмы Intel, применяемые в персональных компьютерах, работают на аппаратном уровне с числами в следующих форматах с плавающей точкой:
32 – разряда, (8 – разрядов порядок, 23 – разряда мантисса),
64 – разряда, (11 – разрядов порядок, 52 – разряда мантисса),
80 – разрядов, (15 – разрядов порядок, 64 – разряда мантисса).
Основание порядка равно 2.

2.8.1. Сложение и вычитание.

Пусть $A = m_a \bullet 2^{Pa}$, $B = m_b \bullet 2^{Pb}$, $A \pm B = m \bullet 2^P$,
Где m_a , m_b , m – нормализованные мантиссы ($1 > m \geq ½$) в прямом коде со знаком.

Если $P_a \neq P_b$, то надо выравнивать порядки. Это означает, что меньший порядок надо увеличить на величину $\Delta P=|P_a - P_b|$, что означает сдвиг мантиссы числа с меньшим порядком вправо на количество разрядов равное ΔP. Если $\Delta P \geq n$, где n разрядность мантиссы, то результат равен числу с большим порядком с соответствующим знаком.

Порядки выровнены, т.е. $P = Max(P_a , P_b)$. Вычисление мантиссы см. п.2.4.(сложение и вычитание в прямом коде), обозначим это действие как $m_a \pm m_b$.

1) Если $|m_a \pm m_b| \geq 1$ (разумеется $|m_a \pm m_b|<2$), то мантисса результата $m = (m_a \pm m_b)/2$, $P= Max(P_a , P_b) + 1$. При выполнении этой операции может произойти переполнение порядка в положительную сторону.

2) Если $1 > |m_a \pm m_b| \geq ½$, то $P = Max(P_a , P_b)$, $m = m_a \pm m_b$.
3) Если $|m_a \pm m_b| < ½$, т.е. мантисса не нормализована. Нормализуя мантиссу, т.е. сдвигая влево надо уменьшать порядок, при этом может произойти переполнение порядка в отрицательную сторону. Реакция, предусмотренная на такое событие, может быть различной, обычно в этом случае формируется код «машинного» нуля.

2.8.2. Умножение.

Пусть $A=m_a \bullet 2^{Pa}$, $B=m_b \bullet 2^{Pb}$, $A \bullet B = m \bullet 2^P$,
Где m_a , m_b , m – нормализованные мантиссы ($1 > m \geq ½$)

$1 > m_a \bullet m_b \geq ¼$

1) Если $m_a \bullet m_b \geq ½$, то $m = m_a \bullet m_b$, $P = P_a+P_b$.
2) Если $½ > m_a \bullet m_b \geq ¼$, то $m = 2 \bullet m_a \bullet m_b$, $P = P_a+P_b-1$.

Особые случаи. При выполнении операций с порядками возможны ситуации переполнения:
1) положительное переполнение,
2) отрицательное переполнение – в этом случае формируется код «машинного» нуля.

2.8.3. Деление.

Пусть $A=m_a \bullet 2^{Pa}$, $B=m_b \bullet 2^{Pb}$, $A/B = m \bullet 2^P$,
Где m_a , m_b , m – нормализованные мантиссы ($1 > m \geq ½$)

$$max/min > m_a/m_b > min/max$$
$$2 > m_a/m_b > ½,$$

1) Если m_a=min= ½, m_b=max=$(1-2^{-n})$, то m=½ , что не противоречит приведённому выше неравенству, т. к. строго говоря, $m_a/m_b=m+R$.
2) Если $m_a/m_b < 1$, то $m = m_a/m_b$, $P = P_a-P_b$.
3) Если $m_a/m_b \geq 1$, то $m = (½) \bullet m_a/m_b$, $P = P_a-P_b+1$.

Особые случаи. При выполнении операций с порядками возможны ситуации переполнения:
1) положительное переполнение,
2) отрицательное переполнение – в этом случае формируется код «машинного» нуля.