Информатика

Филиппов Константин Сергеевич
Инженер и преподаватель ГОУ «Многопрофильный технический лицей №1501».

Научили железку считать...

В предыдущем номере журнала (статья «Логика “железная”, а с арифметикой что?») рассматривалась работа основных логических элементов, составляющих конструкцию компьютера и то, как машина выполняет операцию арифметического сложения чисел в двоичной системе. Сегодня, как и было обещано, речь пойдёт о «машинном» вычитании.

Операция арифметического вычитания

Казалось бы: чего уж проще! Ещё в начальной школе нас учили складывать и вычитать «в столбик». Правда, делалось это в привычной десятичной системе. Но, если приноровиться, и в двоичной системе задача вычитания не покажется сложной. Рассмотрим основные правила:

$$0_2 - 0_2 = 0_2;$$

$$1_2 - 1_2 = 0_2;$$

$$1_2 - 0_2 = 1_2;$$

$$10_2 - 1_2 = 1_2.$$

Вычтем:

	5	4	3	2	1	0	– номер разряда
	●	●					– занимаем единицу
	1	1	0	0	0	1	– уменьшаемое
–		1	0	1	1	1	– вычитаемое
		1	1	0	1	0	– разность

Обратите внимание: когда мы вычитаем «0 – 1» в 1-ом разряде, в результате запишем «1», но нам потребуется занять единицу в старшем разряде. Ближайший такой разряд – 4-й (т. к. во 2-ом и 3-ем нули). Как только мы это сделали, в 4-ом разряде уменьшаемого появится ноль, а во 2-ом и 3-ем – единицы.

Теперь давайте представим себе ситуацию, когда из меньшего числа вычитается большее. Должно получиться отрицательное число. Следовательно, надо придумать, как реализовать эту ситуацию технически. Самое простое решение, которое напрашивается, – представить результат в виде двух сигналов: значения разности и знака. При этом признаком положительного результата будет являться «0», а отрицательного – «1» (см. последнюю колонку таблицы на рис. 1).

A	B	A–B *(математически)*	A–B *(техническая реализация)*
0	0	0 – 0 = 1	0 – 0 = 00
0	1	0 – 1 = – 1	0 – 1 = 11
1	0	1 – 0 = 1	1 – 0 = 01
1	1	1 – 1 = 0	1 – 1 = 00

Рис. 1

Вы наверняка заметили, что при выполнении вычитания значение результата (правая цифра) в точности повторяет результат сложения на тех же наборах исходных данных (рис. 2):

A	B	A – B		A + B	
		PO	S	PO	S
0	0	0	0	0	0
0	1	1	1	0	1
1	0	0	1	0	1
1	1	0	0	1	0

Рис. 2

Если мы захотим построить логическую схему вычитания, нам потребуется четвертьсумматор и дополнительный элемент, отслеживающий ситуацию вычитания «0 – 1». Эта схема может выглядеть так (рис. 3):

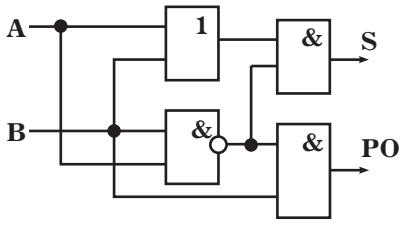
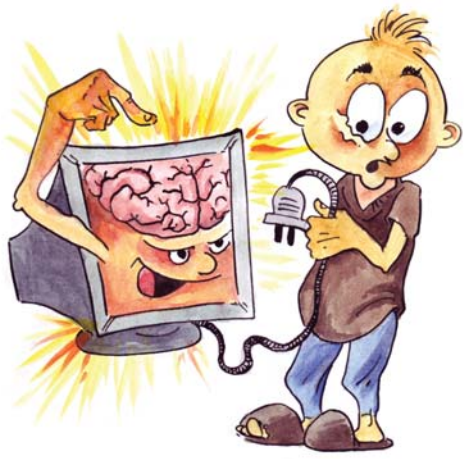


Рис. 3

Представление целых отрицательных чисел

Как видите, эта схема лишь незначительно отличается от полусумматора, разработанного нами ранее для выполнения операции сложения. И было бы большой ошибкой строить для сложения и вычитания отдельные схемы. Более рациональным подходом будет разработка способа, при котором операция вычитания превратилась бы в сложение. Что означает выполнить вычитание «A – B»? То же самое, что сложить «A + (–B)». Следовательно, требуется разработать особый способ представления отрицательных чисел. И именно в таком виде отрицательные числа должны храниться в памяти.

В памяти компьютера под хранение каждого числа отводится строго определённое количество ячеек, независимо от величины числа. Выполнение процессором арифметических действий над числами в двоичной системе возможно, только если они имеют одинаковую длину. В реальности это так и есть: перед выполнением арифметической операции числа передаются из основной памяти в *регистры процессора*, которые имеют одинаковый размер (разрядность).



На рисунке 4 изображён четырёхразрядный параллельный регистр.

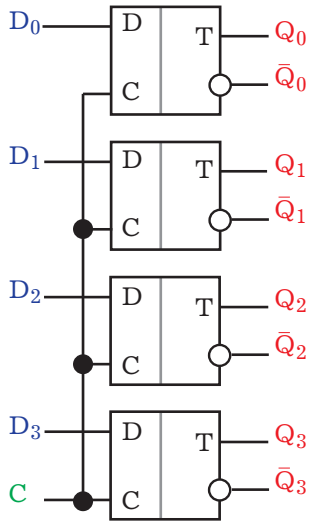


Рис. 4

Он построен на уже знакомых вам по прошлой статье четырёх синхронных *D*-триггерах. Причём заметьте, на выходах каждого триггера присутствуют как прямой, так и инверсный сигналы. Это важное свойство очень пригодится в дальнейшем.

Итак, вычитание – это сложение с отрицательным числом. Давайте подумаем, как может выглядеть отрицательное число в памяти компьютера.

Идея первая лежит, как говорится, на поверхности и следует из работы логической схемы вычитания, рассмотренной ранее. Пусть старший бит отвечает за знак числа (биты нумеруются с нуля справа налево). Если в нём «0», то число положительное, а если «1» – отрицательное. Этот бит имеет специальное название – *«знаковый разряд»*. Память компьютера имеет байтовую структуру (состоит из ячеек объёмом 1 байт = 8 бит). Следовательно, минимальный объём памяти, который может быть выделен под хранение целого числа, – 1 байт. Тогда, например, запись десятичного числа 53 будет иметь вид (рис. 5):

Номер разряда:	7	6	5	4	3	2	1	0	
значение:	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	$=53_{10}$

Рис. 5

Равное ему по модулю отрицательное число запишется так (рис. 6):

Номер разряда:	7	6	5	4	3	2	1	0	
значение:	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	$=-53_{10}$

Рис. 6

При такой форме записи наибольшим числом будет $127_{10} = 01111111_2$, а наименьшим $-127_{10} = 11111111_2$.

Попробуем теперь вычислить значение выражения «109 – 53», представив его как сумму «109 + (–53)». Должно получиться 56. Проверим это сложением «в столбик». (Рассмотренная ранее цепочка полных сумматоров будет выполнять сложение похожим образом.)

номер разряда:	**PO**	7	6	5	4	3	2	1	0	
первое число:		**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	$=109_{10}$
второе число:		**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	$=-53_{10}$
сумма =	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	$=34_{10}$

Рис. 7

При вычислении суммы произошло переполнение. То есть получившееся число $100100010_2 = 290_{10}$ не уместилось в отведённые 8 бит. Старший бит был потерян, и в ячейке осталось число $00100010_2 = 34_{10}$, что не соответствует правильному ответу. Следовательно, предложенный способ хранения отрицательного числа не упрощает задачу вычитания. Кроме того, при таком варианте представления существует неоднозначность в записи числа ноль (рис. 8):

номер разряда:	7	6	5	4	3	2	1	0	
значение:	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**= 0_{10}**
	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**= -0_{10}**

Рис. 8

Что это? И чем эти нули отличаются?..

Увы, придётся признать, что наше предположение оказалось ошибочным.

Идея вторая. Давайте подойдём к проблеме с другой стороны. Что такое «–B»? Такое число можно получить, если вычесть из нуля B. Выполним вычитание «в столбик» по правилам двоичной системы: если из меньшей цифры вычитается большая, то нужно «занять» единицу в старшем разряде. В качестве примера возьмём всё то же число 53_{10}.

номер разряда:	*	7	6	5	4	3	2	1	0	
первое число:	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**= 0_{10}**
второе число:		**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**= 53_{10}**
разность =		**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**= $?_{10}$**

Рис. 9

*) Предположим, что у нас существует логическая схема вычитания. При вычитании большего числа из меньшего занимаем единицу за пределами области хранения числа – своего рода «мнимую единицу». Технически это возможно реализовать. Как только мы вычтем из нуля единицу в младшем разряде, в оставшихся разрядах появятся единицы:

номер разряда:	*	7	6	5	4	3	2	1	0
первое число:	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	
второе число:		**0**	**0**	**1**	**1**	**0**	**1**	**0**	
разность=		**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**

Рис. 10

В результате мы получили некое число 11001011_2, которое непонятно как следует интерпретировать: то ли это положительное число 203_{10}, то ли отрицательное -75_{10} (если вспомнить нашу первую идею и изначальный посыл, что у отрицательных чисел старший бит равен 1). В любом случае на 53 в привычном представлении никак не похоже. Но может быть всё-таки в этой «странной» последовательности нулей и единиц и зашифровано отрицательное число 53? Давайте это проверим! Возьмём то же выражение «109 – 53»

и выполним сложение «109 + (–53)», где в качестве двоичной записи числа «–53» возьмём только что полученную последовательность 11001011 (рис. 11).

номер разряда:	**РО**	7	6	5	4	3	2	1	0	
первое число:		**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**= 109_{10}**
второе число:		**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**= -53_{10}**
сумма =	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**= 56_{10}**

Рис. 11

При вычислении суммы произошло переполнение. Получившееся число $100111000_2 = 312_{10}$ не уместилось в отведённые 8 бит. Старший бит был потерян, и в ячейке осталось число $00111000_2 = 56_{10}$, что как раз соответствует правильному ответу!

Итак, наш эксперимент увенчался успехом. Мы нашли такую запись отрицательного числа, в которой старший бит равен «1» (и является признаком отрицательного числа), а сложение данного числа с другим числом даёт верный результат. И мы располагаем техническим средством для суммирования многоразрядных двоичных чисел.

«Превращения чисел»

Следующая проблема состоит в том, как из положительного числа сделать соответствующее ему отрицательное? Рассмотренный выше способ не подходит, так как нашей изначальной целью было избавиться от отдельной технической реализации операции вычитания.

Давайте сравним двоичную запись чисел 53 и –53 (рис. 12):

номер разряда:	7	6	5	4	3	2	1	0
53_{10} =	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**
-53_{10} =	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**

Рис. 12

Чтобы уловить закономерность, возьмём ещё одно число (рис. 13):

номер разряда:	7	6	5	4	3	2	1	0
92_{10} =	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**
-92_{10} =	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**

Рис. 13

Итак, для того чтобы получить двоичную запись отрицательного числа, надо:

1) записать соответствующее положительное число в двоичной системе, отводя под запись такое количество разрядов, чтобы в старшем разряде не было единицы. Такая запись носит название *«прямой код числа»* (ПК);

2) применить к прямому коду числа операцию инверсии. То есть заменить единицы нулями, а нули единицами. Такая запись будет называться *«обратным кодом числа»* (ОК);

3) к обратному коду числа *арифметически* прибавить единицу. Получим *«дополнительный код числа»* (ДК).

Пример показан на рис. 14:

номер разряда:	7	6	5	4	3	2	1	0	
1) $\mathbf{92_{10}} =$	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	*– прямой код*
2)	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	*– обратный код*
3) +								**1**	
$\mathbf{-92_{10}} =$	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	*– дополнительный код*

Рис. 14

Задача такого преобразования легко решается технически. Если вспомнить, что обрабатываемые числа находятся в регистрах процессора, то для получения обратного кода нужно снимать сигналы не с прямых, а с инверсных выходов. А далее с помощью сумматора прибавить единицу. Например, для *четырёхразрядных* чисел схема может выглядеть так (рис. 15).

По такому же принципу можно построить схему преобразования n-разрядных чисел.

Выше было введено понятие дополнительного кода. Рассмотрим его подробнее.

Дополнительным кодом некоторого двоичного числа, хранящегося в n-разрядной ячейке памяти, называется такое двоичное число,

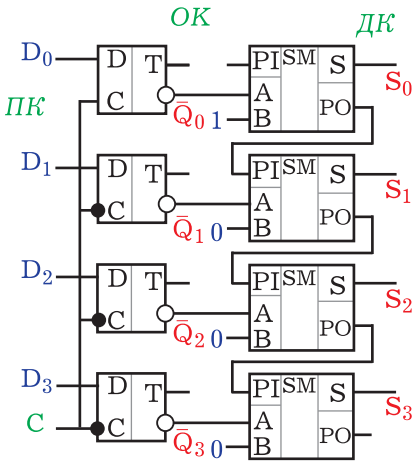


Рис. 15

которое *дополняет* исходное число до значения 2^n. Проиллюстрируем это определение на примере (рис. 16):

номер разряда:	**PO**	7	6	5	4	3	2	1	0	
первое число:		**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	$\mathbf{= 53_{10}}$
второе число:		**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	$= \mathbf{-53_{10}}$
сумма =	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	$\mathbf{= 0_{10}}$

Рис. 16

При сложении этих восьмибитовых чисел получилось число $100000000_2 = 256_{10} = 2^8$. Однако старшая единица была потеряна из-за переполнения. В результате в ячейке памяти оказалось значение 0 (ноль). Что, впрочем, соответствует законам математики: 53 + (–53) = 0. Кстати, если не принимать во внимание знаковую запись числа и перевести двоичное число 11001011 в десятичную систему, то получится 203. Сложив 53 и 203, получим $256 = 2^8$.

Рассмотрим обратную задачу – преобразование дополнительного кода в прямой. Или, если угодно, получение из отрицательного числа соответствующего ему положительного. Для этого нужно из дополнительного кода арифметически вычесть единицу, а затем

к тому, что получилось, применить операцию инверсии. Вычитать будем опять-таки путём сложения. Вначале найдём восьмибитовый дополнительный код числа 1 (рис. 17):

номер разряда:	7	6	5	4	3	2	1	0	
1) 1_{10} =	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	*– прямой код*
2)	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	*– обратный код*
3) +								**1**	
-1_{10} =	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	*– дополнительный код*

Рис. 17

Теперь выполним сложение с исходным отрицательным числом (например, с числом –53) (рис. 18):

номер разряда:	**РО**	7	6	5	4	3	2	1	0	
первое число:		**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	$= -53_{10}$
второе число:		**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	$= -1_{10}$
сумма =	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	$= -54_{10}$

Рис. 18

При вычислении суммы произошло переполнение. Получившееся число 111001010_2 не уместилось в отведённые 8 бит. Старший бит был потерян, и в ячейке осталось число 11001010_2. Инвертировав его, получим 00110101_2, что соответствует десятичному числу 53. Кстати, обратите внимание, что если не инвертировать результат, то он будет соответствовать отрицательному числу «–54» (записанному в дополнительном коде), что соответствует значению выражения «–53 – 1».

Техническая реализация этого преобразования похожа на реализацию получения дополнительного кода. Вы наверняка без труда сами составите схему.

Подведём итоги

Рассмотренный метод вычислений универсален. Он позволяет получить правильный результат и в случае, когда из меньшего числа вычитается большее. Например, при вычитании 100 из 80 получится –20, и это число будет представлено в дополнительном коде (рис. 19):

номер разряда:	**РО**	7	6	5	4	3	2	1	0	
первое число:		**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	$= 80_{10}$
второе число:		**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	$= -100_{10}$
сумма =	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	$= -20_{10}$

Рис. 19

А что будет, если попробовать вычислить значение выражения –80 – 100 (в восьмибитовом представлении)? Математически должно получиться –180. Проверим:

номер разряда:	**РО**	7	6	5	4	3	2	1	0	
первое число:		**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	$= -80_{10}$
второе число:		**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	$= -100_{10}$
сумма =	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	$= 76_{10}$

Рис. 20

Получившееся число 101001100_2 не уместилось в отведённые 8 бит. Старший бит был потерян, и в ячейке осталось число 01001100_2. Но это неправильный ответ! Почему же в предыдущих примерах эффект арифметического переполнения приводил к верному ответу, а здесь – нет? Всё дело в том, что в данном примере ожидаемый результат выходит за допустимые границы диапазона целых знаковых чисел, которые могут быть записаны восемью битами. Каковы же эти границы?

При восьмибитовом знаковом представлении целого числа наибольшим положительным числом будет $01111111_2 = 127_{10}$, а наименьшим отрицательным – $10000000_2 = -128_{10}$. Единица в старшем бите определяет, что число отрицательное и записано в дополнительном коде. Вот несколько примеров (рис. 21).

номер разряда:	7	6	5	4	3	2	1	0
$\mathbf{127_{10} =}$	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
$\mathbf{53_{10} =}$	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**
$\mathbf{1_{10} =}$	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
$0 =$	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
$\mathbf{-1_{10} =}$	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
$\mathbf{-53_{10} =}$	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**
$\mathbf{-126_{10} =}$	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
$\mathbf{-127_{10} =}$	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
$\mathbf{-128_{10} =}$	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Рис. 21

Если бы в ячейке памяти объёмом в 1 байт хранилось целое число в беззнаковом представлении, то в такую ячейку можно было бы записать число из диапазона [0; 255] – всего 256 различных чисел (или 2^8).

При знаковом представлении общее количество различных чисел также равно 256. Однако на долю положительных и отрицательных придётся лишь половина. Вот и получается диапазон [–128; +127]. Ноль будет отнесён к положительным числам, что уравняет количества положительных и отрицательных чисел. А это, в свою очередь, неизбежно приведёт к тому, что у числа –128 не окажется пары – равного ему по модулю положительного числа. Проведём эксперимент: найдём дополнительный код числа 128 (рис. 22).

номер разряда:	7	6	5	4	3	2	1	0	
1) $\mathbf{128_{10} =}$	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	*– прямой код*
2)	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	*– обратный код*
3) **+**								**1**	
$\mathbf{-128_{10} =}$	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	*– дополнительный код*

Рис. 22

Получается, что в восьмибитовом представлении прямой и дополнительный коды числа 128 выглядят одинаково. Этот факт и то, что единица в старшем бите определяет отрицательное число, объясняет не-

возможность восьмибитовой записи числа +128 в знаковом представлении.

При сложении чисел ситуация аналогична:

номер разряда:	**РО**	7	6	5	4	3	2	1	0	
первое число:		**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	$=80_{10}$
второе число:		**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	$=100_{10}$
сумма =	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	$=-76_{10}$

Рис. 23

Переполнения не возникло, но при знаковом представлении мы получили в результате отрицательное число. А это ошибка!

Очень важно понимать природу таких ситуаций. Ибо ни процессор, ни операционная система никак их не контролируют. Следить за возможностью арифметического переполнения должен программист. При написании программы он сам определяет, какой объём памяти необходимо выделить под хранение значения той или иной переменной.

В этой статье удалось осветить лишь маленькую часть вопроса о «машинной арифметике». Следует отметить, что представленные здесь логические схемы, непосредственно реализующие арифметические операции и получение дополнительного кода, – упрощённые и призваны лишь *проиллюстрировать принципы* выполнения этих действий. Многие схемотехнические тонкости «остались за кадром». И пусть для неподготовленного человека операции в двоичной системе могут вызвать трудности, но это с непривычки. Компьютер намного глупее людей! Именно мы «научили его быть умным». Да, много чего компьютер способен сделать быстрее нас, но «муки творчества» присущи лишь Человеку.

Юмор Юмор Юмор Юмор Юмор Юмор

Преимущество компьютера

– Неудобно всё же было пользоваться пишущей машинкой, – заметила сотрудница офиса, когда появились компьютеры.

– Почему? – поинтересовалась другая.

– Когда не работаешь – сразу слышно.