Министерство образования Республики Беларусь

Учреждение образования   
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Отчет по лабораторным работам №1-3  
«Арифметические операции с целыми числами»

Выполнил:  
студент гр. 153504  
Климкович Н.В.

Проверила:

Калиновская А.А.

Минск 2023

# 1. Цель работы

Изучить арифметические операции над целыми числами с фиксированной точкой различными алгоритмами выполнения. Рассмотреть представление чисел в прямом и дополнительном кодах.

# 2. Постановка задачи

***Задание к лабораторной работе 1***

Написать программу эмулятора АЛУ, реализующего *операции сложения и вычитания с фиксированной точкой* над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

***Задание к лабораторной работе 2***

Написать программу эмулятора АЛУ, реализующего *операцию умножения*над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

***Задание к лабораторной работе 3***

Написать программу эмулятора АЛУ, реализующего *операцию деления с фиксированной точкой*над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

# 3. Теоретические сведения

### *Представление целых чисел.*

В двоичной системе счисления числа представляются с помощью комбинации единиц и нулей, знака "минус" и знака разделяющей точки между целой и дробной частью числа. Например, десятичное число -1.312510 в двоичном виде будет выглядеть как -1001.01012. Но в компьютере мы не можем хранить и обрабатывать символы знака и разделяющей точки — для "машинного" представления чисел могут использоваться только двоичные цифры (0 и 1). Если операции выполняются только с неотрицательными числами, то формат представления очевиден. В машинном слове из 8 бит можно представить числа в интервале от 0 до 255. Например:

|  |  |
| --- | --- |
| *00000000  =* | *0* |
| *00000000  =* | *0* |
| *00000001  =* | *1* |
| *00101001  =* | *41* |
| *10000000  =* | *128* |
| *11111111* | *255* |

В общем случае n-битовая последовательность двоичных цифр an-1an-2…a1a0интерпретирована как целое число А, значение которого равно

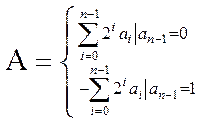


#### *Прямой код*

Существует несколько соглашений о едином формате представления как положительных, так и отрицательных чисел. Всех их объединяет то, что старший бит слова (с точки зрения европейца — самый левый, или бит, которому при представлении числа без знака должен быть приписан самый большой вес) является битом хранения знака или знаковым разрядом. Все последующие биты слова представляют значащие разряды числа, которые в каждом формате интерпретируются по-своему. Значение 1 в знаковом разряде интерпретируется как представление всем словом отрицательного числа.

|  |  |
| --- | --- |
| 00010010 = | +18 |
| 10010010 = | -18 |

Общее правило математически формулируется следующим образом:



Формат представления чисел в прямом коде неудобен для использования в вычислениях. Во-первых, сложение и вычитание положительных и отрицательных чисел выполняется по-разному, а потому требуется анализировать знаковые разряды операндов. Во-вторых, в прямом коде числу 0 соответствуют две кодовых комбинации:

|  |  |
| --- | --- |
| 00000000 = | +010 |
| 10000000 = | -010 |

Это также неудобно, поскольку усложняется анализ результата на равенство нулю, а такая операция в программах встречается очень часто.

Из-за этих недостатков прямой код практически не применяется при peaлизации в АЛУ арифметических операций над целыми числами. Вместо этого более широкое применение находит другой формат, получивший наименование дополнительного кода.

#### *Дополнительный код*

Как и в прямом, в дополнительном коде старший разряд в разрядной сетке отводится для представления знака числа. Остальные разряды интерпретируются не так, как в прямом коде. В табл. 1 перечислены основные свойства дополнительного кода и правила выполнения арифметических операций в дополнительном коде, которые мы рассмотрим в этом и следующем разделах.

Таблица 1. Свойства представления чисел в дополнительном коде

|  |  |
| --- | --- |
| *Диапазон представления на n-разрядной сетке* | от -2n-1 до 2n-1-1 |
| *Количество кодовых комбинаций, соответствующих числу 0* | Одна |
| *Отрицание* | Инвертировать значение в каждом разряде представления исходного числа (положительного или отрицательного), а затем сложить образовавшееся число с числом 0001 по правилам сложения чисел без знака |
| *Расширение разрядности* | Добавить дополнительные разряды слева и заполнить их представления значением, равным значению в знаковом разряде исходного представления |
| *Определение переполнения при сложении* | Если оба слагаемых имеют одинаковые знаки (оба положительны или оба отрицательны), то переполнение возникает в том и только в том случае, когда знак суммы оказывается отличным от знаков слагаемых |
| *Правило вычитания* | Для вычитания числа В из числа А инвертировать знак числа В, как описано выше, и сложить преобразованное число с А по правилам сложения в дополнительном коде |

Рассмотрим n-разрядное двоичное целое число А в дополнительном коде. Если А положительно, то значение его знакового разряда an-1равно 0. В значащих разрядах будет представлена абсолютная величина числа точно так же, как и в прямом коде:



Число 0 считается положительным и, следовательно, в знаковом разряде его представления будет записан код 0, а во всех значащих разрядах также коды 0. Очевидно, что диапазон представления положительных чисел n-разрядным дополнительным кодом простирается от числа 0 до числа 2n-1-1 (для этого числа значения во всех значащих разрядах равны 1). Для представления большего числа потребуется расширение разрядной сетки.

Теперь перейдем к отрицательным числам. Знаковый разряд an-1 дополнительного кода отрицательного числа А (А<0) равен 1. В *n-*1 значащих разрядах может содержаться произвольная комбинация нулей и единиц, а таких комбинаций может быть 2n-1. Следовательно, имеется потенциальная возможность представить отрицательные числа   
от -1 до -2n-1. Желательно таким образом установить соответствие между двоичными комбинациями и целыми отрицательными числами, чтобы арифметические операции над ним, выполнялись по тем же правилам, что и над числами без знака. В формате целых чисел без знака для вычисления значения числа по его двоичному представлению следует присвоить старшему разряду в разрядной сетке вес +2n-1. При представлении, включающем и знаковый разряд, это приводит к тому, что желаемые арифметические свойства сохраняются, если вес этого разряда будет равен -2n-1. Это соглашение используется при представлении чисел в дополнительном коде. Формально для отрицательного числа в дополнительном коде соблюдается соотношение



Знаковый разряд an-1, дополнительного кода положительного числа равен 0 и, следовательно, член -2n-1an-1=0. Таким образом, соотношение справедливо для дополнительного кода как положительных, так и отрицательных чисел.

В табл. 2 сравниваются представления 4-разрядных целых чисел в прямом и дополнительном кодах. Хотя с обычной точки зрения представление в дополнительном коде выглядит довольно экзотично, оно значительно упрощает правила выполнения арифметических операций сложения и вычитания. Поэтому такое представление используется для работы с целыми числами в подавляющем большинстве АЛУ современных процессоров.

Таблица 2. Варианты двоичного 4-разрядного представления целых чисел

|  |  |  |  |
| --- | --- | --- | --- |
| Десятичное  представление | Прямой код | Дополнительный код | Смещённое представление |
| +7 | *0111* | *0111* | *1111* |
| +6 | *0110* | *0110* | *1110* |
| +5 | *0101* | *0101* | *1101* |
| +4 | *0100* | *0100* | *1100* |
| +3 | *0011* | *0011* | *1011* |
| +2 | *0010* | *0010* | *1010* |
| +1 | *0001* | *0001* | *1001* |
| +0 | *0000* | *0000* | *1000* |
| -0 | *1000* | *-* | *0111* |
| -1 | *1001* | *1111* | *0110* |
| -2 | *1010* | *1110* | *0101* |
| *-3* | *1011* | *1101* | *0100* |
| *-4* | *1100* | *1100* | *0011* |
| *-5* | *1101* | *1011* | *0010* |
| *-6* | *1110* | *1010* | *0001* |
| *-7* | *1111* | *1001* | *0000* |
| *-8* | *-* | *1000* | *-* |

#### *Представление с фиксированной точкой*

И наконец, следует остановиться еще на одном нюансе. Описанные выше форматы объединяются часто одним термином — формат с фиксированной точкой. Суть его в том, что положение разделительной точки между целой и дробной частями числа неявно фиксируется на разрядной сетке. В настоящее время принято фиксировать точку справа от самого младшего значащего разряда. Программист может использовать аналогичное представление для работы с двоичными дробными числами, мысленно фиксируя точку перед старшим значащим разрядом и соответственно масштабируя результаты преобразований, выполняемых стандартными программными или аппаратными средствами.

### *Арифметические операций над целыми числами*

В этом разделе будут рассмотрены алгоритмы выполнения основных арифметических операций над целыми числами, представленными в дополнительном коде.

#### *Отрицание*

Операция отрицания числа, представленного в прямом коде, выполняется очень просто - нужно инвертировать значение знакового разряда. Если же число представлено в дополнительном коде, отрицание выполняется несколько сложнее. Правило выполнения этой операции формулируется следующим образом.

1.Следует инвертировать значение в каждом разряде представления исходного числа (положительного или отрицательного), включая и знаковый, т.е. установить значение 1 в тех разрядах, где ранее было значение 0, и значение 0 — в тех разрядах, где ранее было значение 1 (эту операцию иногда называют поразрядным дополнением — bitwise complement, а ее результат — инверсным кодом).

2.Нужно сложить образовавшееся число с числом 0. . .001 по правилам сложения чисел без знака.

Иногда эту операцию называют вычислением дополнения числа в дополнительном коде (twos complement operation). Например:

|  |  |  |
| --- | --- | --- |
| *+ 18=* | *00010010* | *(дополнительный код)* |
| поразрядное дополнение  *=* | *11101101* |  |
| *+* | *1* |  |
|  | *11101110* | *=-18* |

Теперь проверим, будет ли выполняться правило, гласящее, что отрицание отрицания равно исходному числу:

|  |  |  |
| --- | --- | --- |
| *- 18=* | *11101110* | *(дополнительный код)* |
| поразрядное дополнение  *=* | *00010001* |  |
| *+* | *1* |  |
|  | *00010010* | *=+18* |

Существуют два особых случая, на которых следует остановиться. Рассмотрим сначала случай А=0. Для восьмиразрядного представления такого числа получим

|  |  |  |
| --- | --- | --- |
| *0=* | *00000000* | *(дополнительный код)* |
| поразрядное дополнение  *=* | *11111111* |  |
| *+* | *1* |  |
|  | ***1****00000000* | *=0* |

Выделенный в нижней строке код **1** есть не что иное, как перенос из старшего разряда, который игнорируется. Таким образом, результатом отрицания числа 0 в дополнительном коде будет дополнительный код числа 0, как и следовало ожидать.

Следующий особый случай порождает определенные проблемы. Речь идет о числе, которое в дополнительном коде имеет вид 100. .00, т.е. имеет в знаковом разряде код 1, а в п-1 значащих — код 0. После выполнения отрицания по описанному выше правилу получим тот же самый код:

|  |  |  |
| --- | --- | --- |
| *-128=* | *10000000* | *(дополнительный код)* |
| поразрядное дополнение  *=* | *11111111* |  |
| *+* | *1* |  |
|  | *100000000* | *=-128* |

Подобные аномалии при работе с двоичными кодами неизбежны. В данном случае первопричина отклонения в следующем. Количество различных комбинаций п -разрядного двоичного кода равно 2п, т.е. является четным числом. С помощью этих комбинаций нам нужно представить положительные, отрицательные числа и число 0. Если представлять равное количество положительных и отрицательных чисел, то, как в прямом коде, получим две формы представления числа 0. Если же потребовать, чтобы числу 0 соответствовала только, единственная кодовая комбинация, то количества представляемых положительных и отрицательных чисел будут не равны. С помощью дополнительного кода можно представить число -2n, но нельзя представить число +2n.

#### *Сложение и вычитание в дополнительном коде*

Рассмотрим примеры:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *(а)* | *-7=* | *1001* |  | *(б)* | *-4=* | *1100* |  |
|  | *+5=* | *+0101* |  |  | *+4=* | *+0100* |  |
|  |  | *1110* | *=-2* |  |  | ***1****0000* | *=0* |
| *(в)* | *+3=* | *0011* |  | *(г)* | *-4=* | *1100* |  |
|  | *+4-* | *+0100* |  |  | *-1=* | *+1111* |  |
|  |  | *0111* | *=7* |  |  | *11011* | *=-5* |
| *(д)* | *+5=* | *0101* |  | *(е)* | *-7=* | *1001* |  |
|  | *+4=* | *+0100* |  |  | *-6=* | *+1010* |  |
|  |  | *1001* | *переполнение* |  |  | ***1****0011* | *переполнение* |

Первые четыре примера демонстрируют успешное выполнение операций. Если результат операции должен быть положительным, получается код положительного числа в дополнительном коде, а если отрицательным — код отрицательного числа в дополнительном коде. Обратите внимание на то, что в примере (г) формируется перенос из старшего (знакового) разряда, который игнорируется.

При выполнении сложения чисел с одинаковыми знаками результат может оказаться таким, что не вмещается в используемую разрядную сетку, т.е. получается число, которое выходит за диапазон представления. Появление такого результата расценивается как переполнение (overflow), и на схему АЛУ возлагается функция выявить переполнение и выработать сигнал, который должен воспрепятствовать использованию в дальнейшем полученного ошибочного результата. Существует следующее правило обнаружения переполнения:

Если знаки слагаемых совпадают, то переполнение возникает в том и только в том случае, когда знак суммы, полученной по правилам сложения в дополнительном коде, отличается от знака слагаемых.

Примеры (д) и (е) иллюстрируют появление переполнения при сложении положительных и отрицательных чисел. Обратите внимание на то, что переполнение может появиться и в том случае, когда возникает перенос из знакового разряда и когда перенос не возникает.

Операция вычитания выполняется по следующему правилу:   
Для вычитания одного числа (вычитаемого) из другого (уменьшаемого) необходимо предварительно выполнить операцию отрицания над вычитаемым, а затем сложить результат с уменьшаемым по правилам сложен дополнительном коде.

Примеры выполнения с различными знаками.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *(а)   М= 2 =* | *0010* | | | | *(б)* *М= 5 =               0101* | | | |
| *S= 7 =* | *0111* | | | | *S= 2 =               0010* | | | |
| *-S =* | *1001* | | | | *-S =               1110* | | | |
|  | *0010* | | | | *0101* | | | |
|  | *+ 1001* | | | | *+  1110* | | | |
|  | *1011* | | | *=-5* | *3=* | | ***1****0011* | |
| *(в)   М=-5 =* | *1011* | | | | *(г)* *М= 5 =               0101* | | | |
| *S=2 =* | *0010* | | | | *S=-2 =               1110* | | | |
| -S = | *1110* | | | | *-S =                  0010* | | | |
|  | *1011* | | | | *0101* | | | |
|  | *+ 1110* | | | | *+  0010* | | | |
|  | ***1****1001* | | *= -7* | | *7=* | | | *0111* |
| *(д)   М= 7 =* | *0111* | | | | *(е)*  *М=-6 =              1010* | | | |
| *S=-7 =* | *1001* | | | | *S= 4 =              0100* | | | |
| *-S =* | *0111* | | | | *-S=                 1100* | | | |
|  | *0111* | | | | *1010* | | | |
|  | *+ 0111* | | | | *+  1100* | | | |
|  | *1110* | *переполнение* | | | *переполнение* | ***1*** | | |

#### *Умножение*

Алгоритмы выполнения умножения значительно сложнее, чем сложения или вычитания, причем в современных вычислительных системах можно встретить как аппаратную его реализацию, так и программную. Существует много вариантов этих алгоритмов, причем многие из них имеют не только теоретический, но и практический интерес, и выбор одного из многих может быть произведен только с учетом специфики применения конкретной cистемы. В данном разделе мы ставили перед собой задачу дать читателю общее представление о подходе, на основе которого такие алгоритмы npoeктируются. Начнем с простой задачи перемножения двух чисел без знака (т.е. неотрицательных чисел), а затем рассмотрим один из наиболее широко известных алгоритмов умножения целых чисел со знаком, представленных в двоичном коде.

Умножение чисел без знака

Ниже показана схема выполнения умножения двоичных чисел без знака, знакомая всем в десятичной интерпретации ещё со школьной скамьи, которую ещё называют умножением в столбик.

|  |  |
| --- | --- |
| *1011* | *Множимое(11)* |
| *х 1101* | *Множитель(13)* |
| *1011* |  |
| *0000* | *Частичные произведения* |
| *1011* |  |
| *1011* |  |
| *1000111* | *Произведение(143)* |

Анализируя этот пример, отметим следующее.

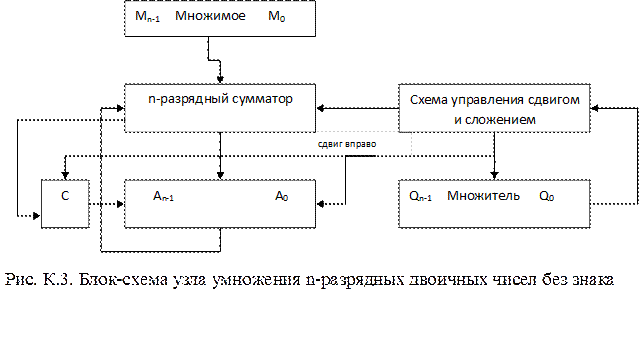
1.При выполнении умножения необходимо формировать частичные произведения, по одному на каждый разряд множителя. Эти частичные произведения затем суммируются, а их сумма и есть результат умножения — полное произведение.

2.Сформировать частичные произведения в двоичном коде довольно легко. Если соответствующий разряд множителя равен 0, частичное произведение  
также равно 00..00. Если соответствующий разряд множителя равен 1  
частичное произведение равно множимому.

3.Полное произведение вычисляется суммированием частичных произведений, причем каждое очередное частичное произведение в этой сумме сдвигается на одну позицию влево относительно предыдущего.

4.Результатом перемножения двух n-разрядных целых чисел будет 2n–разрядное число.

Реализация алгоритма умножения техническими или программными средствами позволяет несколько повысить его эффективность по сравнению с тем вариантом, который мы традиционно используем при вычислении в столбик вручную. Во-первых, суммирование очередного частичного произведения можно выполнять немедленно после того, как оно будет сформировано, не дожидаясь остальных. Таким образом, отпадает необходимость в средствах для временного хранения частичных произведений, т.е. для аппаратной реализации потребуется меньше регистров. Во-вторых, можно сберечь время, необходимое для формирования частичных произведений. Для каждого разряда в коде множителя, равного 1, необходимо выполнить сдвиг и сложение кода множимого, а для разряда, равного 0, — только сдвиг.

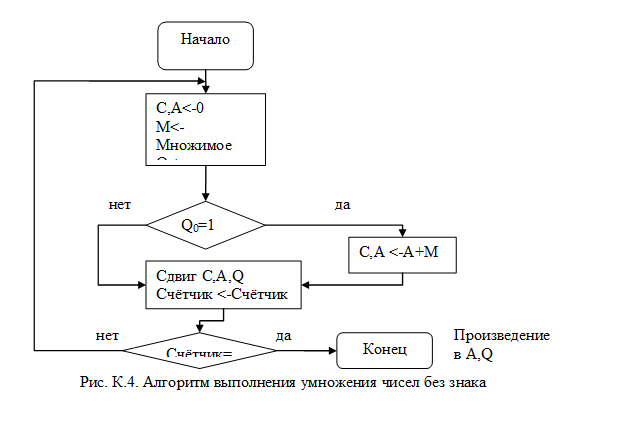


Возможный вариант реализации схемы умножения с учетом отмеченных усовершенствований алгоритма представлен на рис. 3. Множитель загружается в регистр Q, а множимое — в регистр М. В схеме имеется еще и третий регистр, регистр А, который в исходном состоянии сбрасывается в нуль. Одноразрядный регистр С, в который перед началом операции также записывается код 0, используется для хранения потенциального бита переноса, который может возникнуть в процессе суммирования.

Выполнение операции умножения в этой схеме происходит следующим образом. Схема управления сдвигом и сложением анализирует разряды множителе по одному, начиная с младших. Если Q0 равен 1, множимое складывается с содержимым регистра А и результат сохраняется в этом же регистре, причем регистр С используется для фиксации переполнения. Затем все разряды регистров С, А и Q сдвигаются на одну позицию вправо: содержимое С записывается в Аn-1, А0 переписывается в Qn-1 , а значение Q0 теряется. Если Q0 равен 0, сложение не выполняется, а только сдвигается содержимое регистров С, А и Q. Этот процесс выполняется для всех разрядов множителя. В результате в регистрах А и формируется 2n -разрядное произведение.

Ниже представлен пример выполнения умножения по такой схеме — изменение состояний регистров после каждого такта.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Исходные* *значения* | | *С 0* | *А 0000* | *Q*  *1101* | *М*  *1011* |
| *1-й такт* | *Сложение*  *Сдвиг* | *0 0* | *1101 0101* | *1101 1110* | *1011 1011* |
| *2-й такт* | *Сдвиг* | *0* | *0010* | *1111* | *1011* |
| *3-й такт* | *Сложение*  *Сдвиг* | *0 0* | *1101*  *0110* | *1111*  *1110* | *1011 1011* |
| *4-й такт* | *Сложение Сдвиг* | *0 0* | *0001 1000* | *1111*  *1111* | *1011* |



Умножение чисел в дополнительном коде

Мы уже отмечали, что при выполнении сложения и вычитания чисел в дополнительном коде они интерпретируются как числа без знака. Рассмотрим следующий пример:

|  |
| --- |
| *1101* |
| *+ 0011* |
| *1100* |

Если рассматривать операнды как числа без знака, то

*9(1001) + 3(0011) = 12(1100)*

А если интерпретировать их как числа в дополнительном коде*:*

*-7(1001) + 3(0011) = -4(1100)*

К сожалению, эта простая вычислительная схема оказывается неработоспособной при выполнении умножения. Чтобы убедиться в этом, вновь рассмотрим приведенный ранее пример. Мы умножали число 11 (1011) на 13 (1101) и в результате получили 143 (10001111). Но если интерпретировать сомножители как числа в дополнительном коде, то выходит, что умножение -5 (1011) на -3 (1101) дает -113 (10001111). Этот пример показывает, что такая простая схема неприменима, если оба сомножителя отрицательны. Фактически она не дает правильного результата в случае, если отрицателен хотя бы один из сомножителей. Рассмотрим, как интерпретируется алгоритм умножения в столбик, если выразить его в терминах взвешенной суммы степеней числа 2. Тогда:

*1001 = 1х23+ 1х22+ 0x21 + 1x20= 23+22+20…*

Умножение двоичного числа на 2n реализуется сдвигом влево на п разрядов. Приняв это во внимание, получим следующую схему умножения рассматриваемых чисел:

|  |  |
| --- | --- |
| *1011*  *Х 1101* |  |
| *00001011* | *1011Х1Х20* |
| *00000000* | *1011Х0Х21* |
| *00101100* | *1011Х1Х22* |
| *01011000* | *1011Х1Х23* |
| *10001111* |  |

Единственное отличие этой схемы от представленной выше в том, что частичные произведения трактуются как 2n-разрядные числа, сформированные п-разрядного множимого.

Рассматривая исходное 4-разрядное множимое 1011 как число без знака, получим после расширения до восьми разрядов код 00001011. Любое частичное произведение, соответствующее умножению этого числа на некоторый разряд множителя, отличный от 0-го (т.е. с весом 20), формируется сдвигом расширенного кода множимого влево на соответствующее число разрядов, причем освободившиеся справа разряды заполняются кодом 0. Например, в результате сдвига кода 00001011 на два разряда влево образуется код 00101100.

Теперь мы готовы показать, почему эта простая схема не работает в случае, когда множимое отрицательно. Дело в том, что каждое частичное произведение, образованное от отрицательного множимого, также является 2n-разрядным отрицательным числом и, следовательно, при его формировании знаковый разряд исходного числа должен быть расширен. В приведённом ниже примере показаны два варианта умножения двоичных кодов 1001 и 0011, причем в первом случае код 1001 интерпретируется как число без знака (его значение 9), а во втором — как число -7 в дополнительном коде. В первом случае получается результат 00011011, который соответствует в десятичном представлении числу 27 (и действительно, 9x3 = 27). Во втором же случае каждое частичное произведение формируется из расширенного (8-разрядного) кода 11111001 отрицательного множимого. Обратите внимание, что расширенный код можно сформировать, расширив влево значение знакового разряда исходного кода множимого.

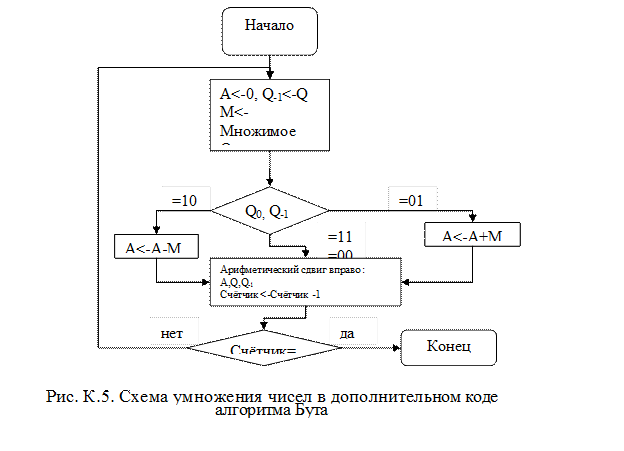
|  |  |  |  |
| --- | --- | --- | --- |
| *Числа без знака* | | *Числа в дополнительном коде* | |
| *1001* | *(9)* | *1011* | *(-7)* |
| *х 0011* | *(3)* | *х 0011* | *(3)* |
| *00001001* | *1011х20* | *11111001* | *(-7)х20=(-7)* |
| *00010010* | *1011х21* | *11110010* | *(-7)х21=(-14)* |
| *00011011* | *(27)* | *11101011* | *(-21)* |

Теперь понятно, почему обычная схема умножения не будет работать и тогда, когда множитель является отрицательным числом. Причина в том, что разряды отрицательного множителя в дополнительном коде не будут при этом соответствовать частичным произведениям, сформированным сдвигом кода множимого. Например, 4-разрядный двоичный дополнительный код числа -3 имеет вид 1101. Если воспользоваться рассмотренной ранее схемой, то должна получиться последовательность сложения частичных произведений

*-(1х23+ 1х22+ 0x21+ 1x20) = -(23+22+20).*

А правильный результат должен иметь вид -(21+20). Следовательно, для управления суммированием частичных произведений нельзя напрямую исполь­зовать значения разрядов кода множителя так, как мы делали это ранее.

Решить эту дилемму можно по-разному. Один из способов состоит в том, чтобы преобразовать оба сомножителя в положительные числа, перемножить их по правилам умножения чисел без знака, а затем, если знаки сомножите­лей были разными, выполнить операцию отрицания результата по правилам, принятым для чисел в дополнительном коде. Однако конструкторы АЛУ предпочитают способ, который не требует выполнения дополнительного преобразования после завершения умножения. Одним из таких способов является алгоритм Бута (Booth). Этот алгоритм также позволяет ускорить выполнение операции. Схема алгоритма Бута приведена на рис. 5. Как и в ранее рассмотренном алгоритме, сомножители размещаются в регистрах Q (множитель) и М (множимое). Кроме них имеется одноразрядный регистр Q-1 который связан с младшим разрядом (Q0) регистра Q. Его назначение будет описано чуть ниже. Произведение формируется в регистрах А и Q. В исходном состоянии в регистрах А и Q , записаны нули. Как и ранее, схема управления анализирует разряды множителя, но на сей раз анализируется пара соседних разрядов — основной и тот, который находится справа от него. Если оба разряда имеют одинаковые значения (11 или 00), все разряды регистров A, Q и Q-1 сдвигаются на 1 разряд вправо. Если соседние разряды имеют отличающиеся коды, то выполняется сложение или вычитание кода множимого из содержимого регистра А. Сложение выполняется при комбинации кодов в соседних разрядах 01, а вычитание — при комбинации 10. Вслед за сложением или вычитанием выполняется сдвиг кодов в регистрах на один разряд вправо. Сдвиг выполняется таким образом, что старший разряд (крайний левый) регистра А (разряд Аn-1) сохраняется, хотя и переписывается в разряд Аn-2 . Это необходимо для сохранения знака кода в регистрах А и Q. Такую операцию принято называть сдвигом с сохранением знака или арифметическим сдвигом.



Ниже представлен пример выполнения умножения чисел 7 и 3 по такому алгоритму — изменение состояний регистров после каждого такта.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Исходные значения* | | *А*  *0000* | *Q*  *0011* | *Q-1*  *0* | *М*  *0111* |
| *1-й такт* | *А<-А-М*  *Сдвиг* | *1001*  *1100* | *0011*  *1001* | *0*  *1* | *0111*  *0111* |
| *2-й такт* | *Сдвиг* | *1110* | *0100* | *1* | *0111* |
| *3-й такт* | *А<-А+М*  *Сдвиг* | *0101*  *0010* | *0100*  *1010* | *1*  *0* | *0111*  *0111* |
| *4-й такт* | *Сдвиг* | *0001* | *0101* | *0* | *0111* |

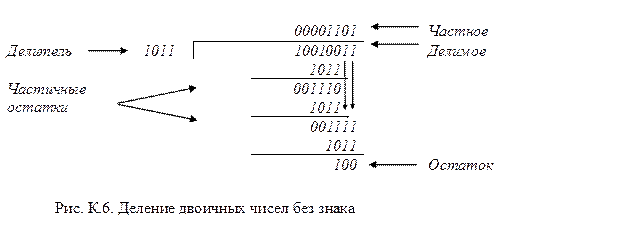
Очевидно, что алгоритм Бута позволяет обойтись при выполнении умножения чисел в дополнительном коде меньшим количеством операций сложения и вычитания, чем простейший алгоритм.

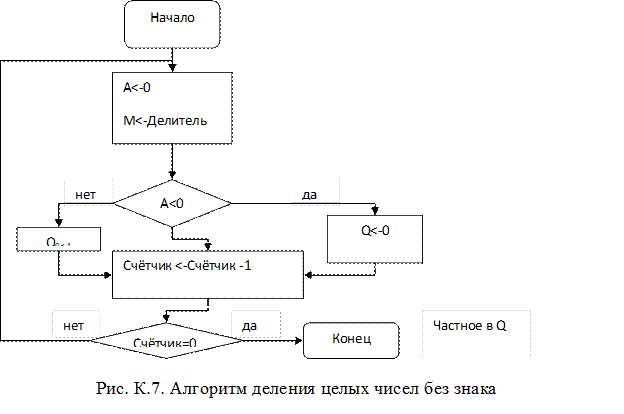
#### *Деление*

По сравнению с умножением операция деления выполняется несколько сложнее, хотя соответствующие алгоритмы основываются на тех же принципах поразрядного анализа операндов. Исходный алгоритм, как и при умножении, — тот, который используется при вычислении вручную, карандашом на бумаге. Алгоритм состоит из повторяющейся последовательности шагов элементарных сдвигов и сложений или вычитаний.

На рис. К.6 показан пример "длинного" деления целых чисел без знака. Чрезвычайно полезно детально рассмотреть последовательность шагов этого алгоритма. Сначала разряды делимого анализируются слева направо до тех пор, пока последовательность разрядов не будет представлять число, большее, чем делитель, или равное ему. При делении этого числа на делитель получается частное, которое больше или равно 1. До тех пор, пока такое число не будет обнаружено при просмотре делимого (событие превышения), в частное заносятся коды 0, которые последовательно сдвигаются слева направо. Когда возникает событие превышения, в частное заносится 1, а делитель вычитается из частичного делимого. Результат этой операции называется частичным остатком. Далее выполняется циклическая процедура. В каждом цикле в частичный остаток добавляется очередной разряд делимого, и так до тех пор, пока при анализе частичного остатка не наступит событие превышения. В этом случае делитель вычитается из частичного остатка и формируется новый частичный остаток, а в очередной разряд частного заносится код 1. Процесс продолжается до тех пор, пока не будут проанализированы все разряды делимого.

На рис. К.7 представлена схема машинного алгоритма "длинного" деления. Перед началом выполнения операции делитель помещается в регистр М, делимое — в регистр Q, а регистр А очищается. На каждом шаге содержимое регистров А и Q сдвигается на 1 разряд влево. Содержимое регистра М вычитается из содержимого А, определяется, делится ли А на М (т.е. результат вычитания больше или меньше нуля). Если результат положительный, в младший разряд частного, Q0, заносится код 1. В противном случае в младший разряд частного вносится код 0, а значение частичного остатка (содержимое регистра А) восстанавливается, для чего к нему прибавляется содержимое регистра М. После этого уменьшается значение в счетчике циклов, в который перед началом выполнения операции записывается число n — количество элементарных циклов. После завершения n циклов частное будет находиться в регистре Q, а остаток — в регистре А.





Этот же алгоритм с небольшими изменениями можно использовать и для деления отрицательных чисел. Ниже мы опишем один из вариантов алгоритма деления чисел, представленных в дополнительном коде.

1.     Загрузить делитель в регистр М, а делимое — в регистры А и Q. Делимое должно иметь формат 2n -разрядного дополнительного кода. Например, 4-разрядное число 0111 должно быть представлено в формате 00000111, а число 1001 должно быть преобразовано в 11111001.

2.     Сдвинуть содержимое регистров А и Q на один разряд влево.

3.     Если коды в регистрах М и А имеют одинаковые знаки, вычесть из содержимого А содержимое М и оставить результат в А. В противном случае добавить к содержимому А код из М.

4.     Предыдущая операция считается успешной, если знак кода в регистре А не изменился в результате ее выполнения.

·        Если операция была успешной или содержимое регистров А и Q равно нулю, то установить в младшем разряде частного (разряде Q0) код 1.

·        Если операция не увенчалась успехом и содержимое одного из регистров А и Q (или обоих) отлично от нуля, то установить в младшем разряде частного, Q0, код 0 и восстановить прежнее значение в регистре А.

5.     Повторять операции, указанные в пунктах 2 и 4, столько раз, сколько разрядов в регистре Q.

6.     После завершения операции прочесть значение остатка в регистре А. Если знаки делимого и делителя одинаковы, значение частного извлечь из регистра Q; в противном случае значение частного равно содержимому регистра Q с обратным знаком (операция отрицания должна быть выполнена по правилам для дополнительного кода).

Ниже приведены примеры выполнения деления по этому алгоритму при разных сочетаниях знаков операндов.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *(а) 7/3* | | | *(б)  7/(-3)* | | |  |  |
| *А* | *Q* | *М=0011* | *А* | *Q* | *M=1101* |  |  |
| *0000* | *0111* | *Исходные значения* | *0000* | *0111* | *Исходные значения* |  |  |
| *0000* | *1110* | *Сдвиг* | *0000* | *1110* | *Сдвиг* |  |  |
| *1101* |  | *Вычитание* | *1101* |  | *Сложение* |  |  |
| *0000* | *1110* | *Восстановление* | *0000* | *1110* | *Восстановление* |  |  |
| *0001* | *1100* | *Сдвиг* | *0001* | *1100* | *Сдвиг* |  |  |
| *1110* |  | *Вычитание* | *1110* |  | *Сложение* |  |  |
| *0001* | *1100* | *Восстановление* | *0001* | *1100* | *Восстановление* |  |  |
| *ООН* | *1000* | *Сдвиг* | *0011* | *1000* | *Сдвиг* |  |  |
| *0000* |  | *Вычитание* | *0000* |  | *Сложение* |  |  |
| *0000* | *1001* | *Q0=l* | *0000* | *1001* | *Q0=l* |  |  |
| *0001* | *0010* | *Сдвиг* | *0001* | *0010* | *Сдвиг* |  |  |
| *1110* |  | *Вычитание* | *1110* |  | *Сложение* |  |  |
| *0001* | *0010* | *Восстановление* | *0001* | *0010* | *Восстановление* |  |  |

Остаток определяется по формуле:

D = QxV + R,

где D — делимое, Q — частное, V — делитель, a R остаток.

# 4. Программная реализация

В качестве языка для реализации эмулятора АЛУ был выбран язык Python.

Для работы с операциями сложения, вычитания, умножения и деления предусмотрено хранение двоичных чисел в 16-битном и сдвоенном 32-битном регистрах. Предусмотрена возможность обработки ситуаций переполнения с выводом информации о проблеме.

На рисунках представлен результат работы программы. Выполняется сложение и вычитание, умножения и деления с остатком двух чисел с фиксированной точкой. Показана обработка переполнений при различных операций.



Рис.8 Пример 1. Сумма, вычитание и умножение чисел: 23478 и -754.



Рис.9 Пример 1. Деление чисел: 23478 и -754.



Рис.10 Пример 2. Сумма, вычитание, умножение и деление чисел: 31943 и 0.

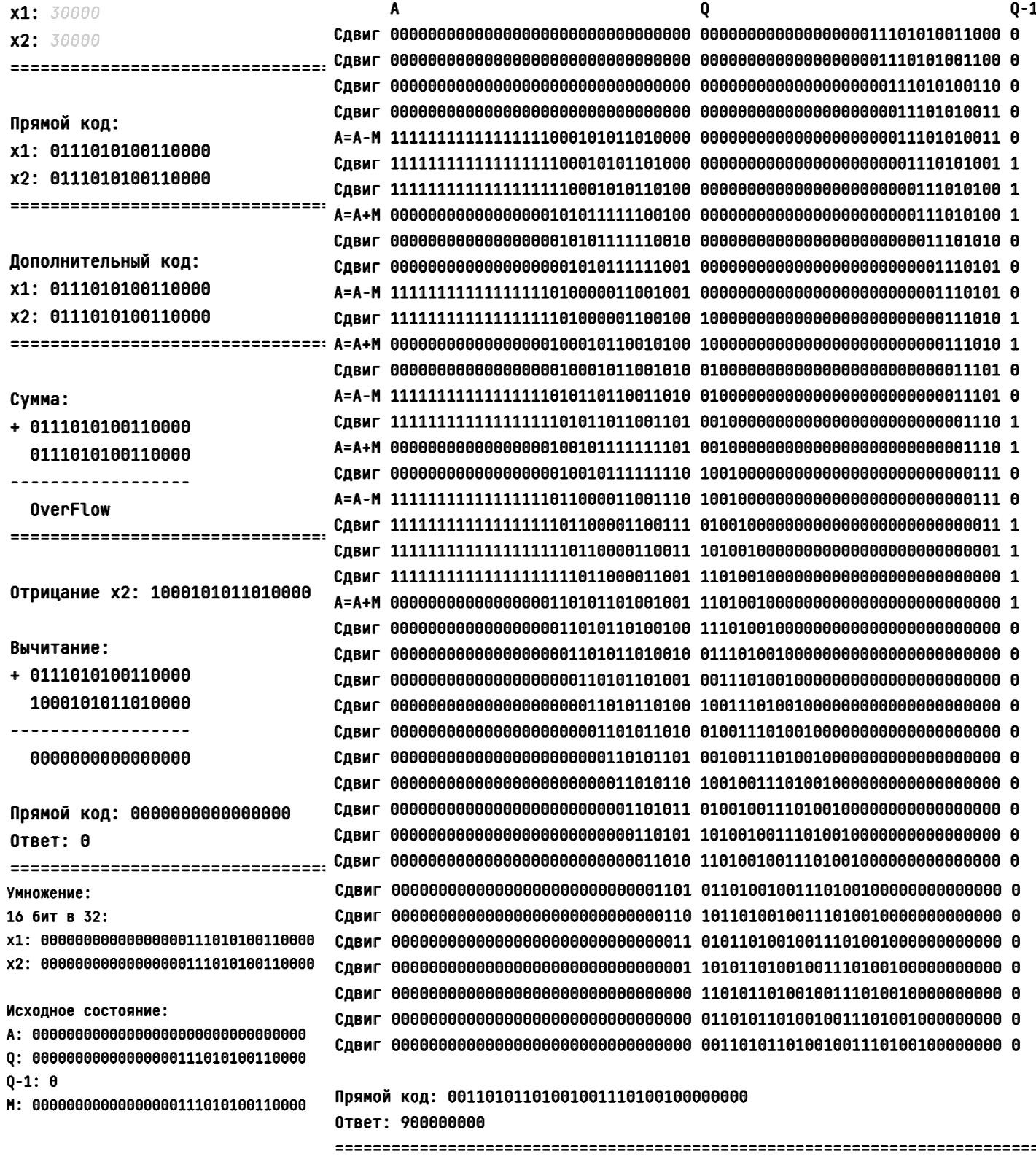


Рис.11 Пример 3. Сумма, вычитание и умножение чисел: 30000 и 30000.

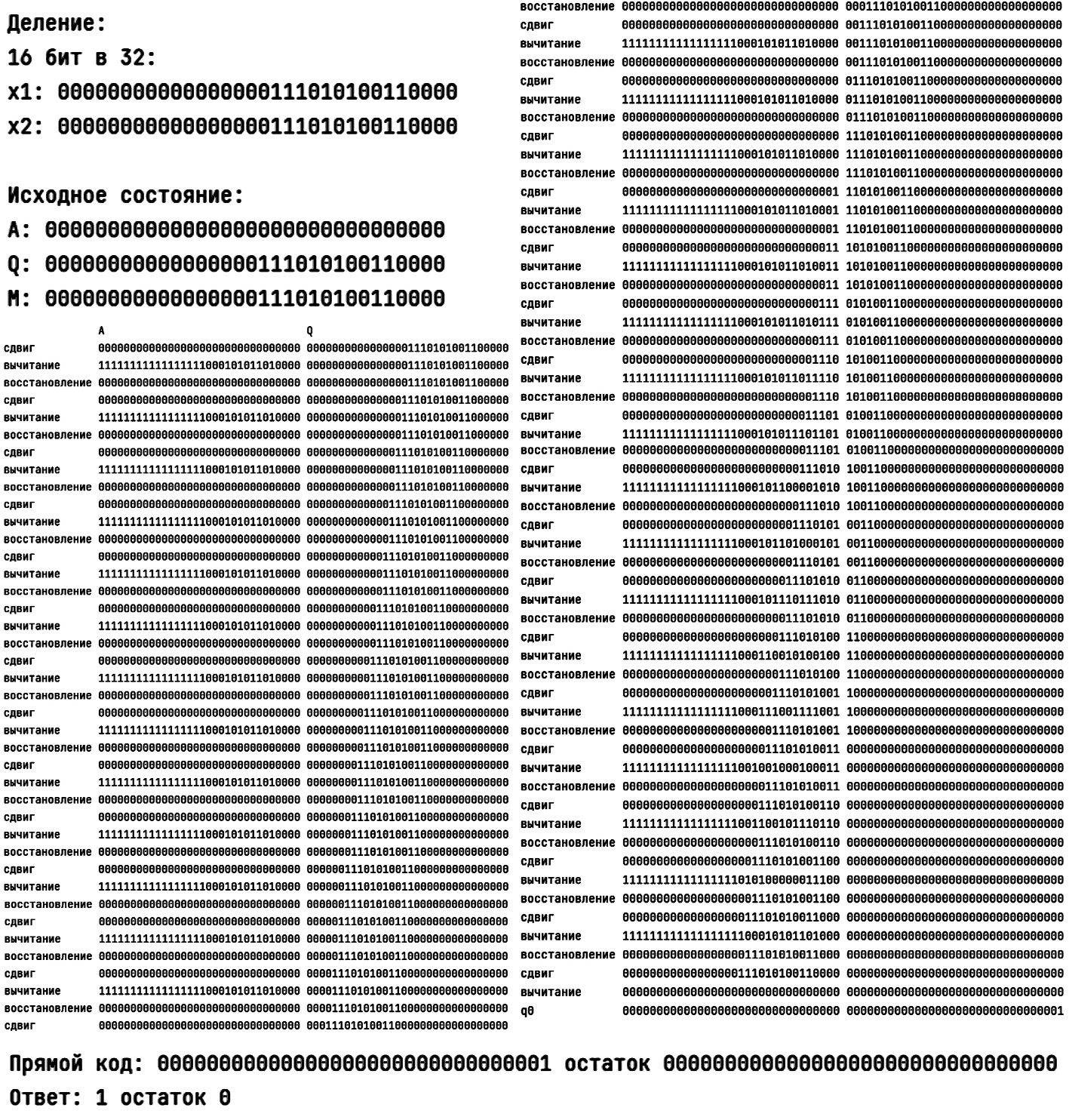


Рис.12 Пример 3. Деление чисел: 30000 и 30000.

# 5. Выводы

В ходе лабораторной работы были изучены принципы работы арифметических операций в АЛУ. Рассмотрены виды представлений чисел в прямом и дополнительном кодах, были обнаружены достоинства того или иного представления в определенных операциях, рассмотрены случаи поведения программы: деление на ноль, переполнение разрядной сетки.

# Список литературы

1. Волорова Н. А. Лабораторный практикум по курсу «Архитектура вычислительных систем» для студентов специальности «Информатика» / 93-444-487-2- Мн.: БГУИР, 2003. – 32с.:ил.

# Приложение 1. Текст программы

import re

# двоичное в десятичное

def decimal(a):

buf = 0

minus = 1

if a[0] == '1':

minus = -1

a = '0' + a[1:]

for i in range(len(a)):

buf += int(a[len(a) - 1 - i]) \* int(pow(2, i))

return buf \* minus

# ввод

def input\_num(num):

while True:

a = str(input('x' + str(num) + ': '))

try:

if (-32768 < int(a) < 32768) and re.search("^[0-9,-]+$", a):

break

else:

print('Error')

except:

print('Error')

if int(a) >= 0:

code = bin(int(a))[2:]

code = '0' \* (16 - (len(code))) + code

else:

code = bin(int(a))[3:]

code = '1' + '0' \* (15 - (len(code))) + code

return code

# сумма

def sum(a, b, of):

buf = 0

rez = ""

for i in range(len(a) - 1, -1, -1):

if int(a[i]) + int(b[i]) + buf > 1:

rez += str((int(a[i]) + int(b[i]) + buf) % 2)

buf = 1

else:

rez += str(int(a[i]) + int(b[i]) + buf)

buf = 0

rez = rez[::-1]

if (a[0] == b[0]) and (a[0] != rez[0]):

if of:

return 'OverFlow'

return rez

# Отрицание

def negation(x):

buf = ""

for i in range(len(x)):

if x[i] == '1':

buf += '0'

else:

buf += '1'

return sum(buf, '0' \* (len(buf) - 1) + '1', True)

# дополнительный код

def additional\_code(x):

if x[0] == '0':

return x

buf = "1"

for i in range(1, len(x)):

if x[i] == '1':

buf += '0'

else:

buf += '1'

return sum(buf, '0' \* (len(buf) - 1) + '1', True)

# отрицание

def subtraction(a, b, of):

b = negation(b)

if of:

print('Отрицание x2:', b, '\n')

print('Вычитание:')

print('+', a)

print(' ', b)

print('-' \* 18)

return sum(a, b, of)

# перевод из 16 в 32 разряда доп код

def \_16\_to\_32\_additional\_code(x):

if x[0] == '1':

return '1' \* 16 + x

return '0' \* 16 + x

# умножение

def multiplication(x1\_, x2\_):

print('16 бит в 32:')

m = \_16\_to\_32\_additional\_code(x1\_)

a = '0' \* 32

q = \_16\_to\_32\_additional\_code(x2\_)

q\_1 = '0'

print('x1:', m)

print('x2:', q, '\n')

print('Исходное состояние:')

print('A:', a)

print('Q:', q)

print('Q-1:', q\_1)

print('M:', m, '\n')

print(' A Q Q-1')

for i in range(32):

if q[-1] + q\_1 == '01':

a = sum(a, m, False)

print('А=А+М', a, q, q\_1)

elif q[-1] + q\_1 == '10':

a = subtraction(a, m, False)

print('А=А-М', a, q, q\_1)

q\_1 = q[-1]

q = a[-1] + q[:-1]

a = a[0] + a[:-1]

print('Сдвиг', a, q, q\_1)

return q

# деление

def division(x1\_, x2\_):

m = \_16\_to\_32\_additional\_code(x2\_)

q = \_16\_to\_32\_additional\_code(x1\_)

if q[0] == '1':

a = '1' \* 32

else:

a = '0' \* 32

print('16 бит в 32:')

print('x1:', q)

print('x2:', m, '\n')

print('Исходное состояние:')

print('A:', a)

print('Q:', q)

print('M:', m, '\n')

print(' A Q')

for i in range(32):

a = a[1:] + q[0]

q = q[1:] + '0'

print('сдвиг ', a, q)

buf\_a = a

if m[0] == a[0]:

a = subtraction(a, m, False)

print('вычитание ', a, q)

else:

a = sum(a, m, False)

print('сложение ', a, q)

if (a[0] == buf\_a[0]) or (a == '0' \* 32 == q):

q = q[:-1] + '1'

print('q0 ', a, q)

else:

a = buf\_a

q = q[:-1] + '0'

print('восстановление', a, q)

if x1\_[0] == x2\_[0]:

return q, negation(a) if x1\_[0] == '1' else a

return negation(q), negation(a) if x1\_[0] == '1' else a

x1 = input\_num(1)

x2 = input\_num(2)

print('=' \* 85, '\n')

print('Прямой код:')

print('x1:', x1)

print('x2:', x2)

x1\_additional = additional\_code(x1)

x2\_additional = additional\_code(x2)

print('=' \* 85, '\n')

print('Дополнительный код:')

print('x1:', x1\_additional)

print('x2:', x2\_additional)

# Сложение

summa = sum(x1\_additional, x2\_additional, True)

print('=' \* 85, '\n')

print('Сумма:')

print('+', x1\_additional)

print(' ', x2\_additional)

print('-' \* 18)

print(' ', summa)

if summa != 'OverFlow':

print('\nПрямой код:', additional\_code(summa))

print('Ответ:', decimal(additional\_code(summa)))

# вычитание

print('=' \* 85, '\n')

sub = subtraction(x1\_additional, x2\_additional, True)

print(' ', sub)

if sub != 'OverFlow':

print('\nПрямой код:', additional\_code(sub))

print('Ответ:', decimal(additional\_code(sub)))

# Умножение

print('=' \* 85, '\n')

print('Умножение:')

if int(x1) == 0 or int(x2) == 0:

print('Ответ: 0')

else:

mult = multiplication(x1\_additional, x2\_additional)

print('\nПрямой код:', additional\_code(mult))

print('Ответ:', decimal(additional\_code(mult)))

# Деление

print('=' \* 85, '\n')

print('Деление:')

if int(x2) == 0:

print('Ответ: OverFlow')

else:

div, remainder = division(x1\_additional, x2\_additional)

print('\nПрямой код:', additional\_code(div), 'остаток', additional\_code(remainder))

print('Ответ:', decimal(additional\_code(div)), 'остаток', decimal(additional\_code(remainder)))