(Следовательно, системы со смешанным целочисленным основанием обладают этим свойством. Наиболее общими системами такого типа являются системы со смешанным основанием, у которых $\beta_1 = (c_0 + 1)\beta_0$, $\beta_2 = (c_1 + 1)(c_0 + 1)\beta_0$, $\ldots$, $\beta_{-1} = \beta_0/(c_{-1} + 1)$, $\ldots$.)

27. [*M21*] Покажите, что любое ненулевое число имеет единственное "знакопеременное двоичное представление"

$$2^{e_0} - 2^{e_1} + \cdots + (-1)^t 2^{e_t},$$

где $e_0 < e_1 < \cdots < e_t$.

▶ **28.** [*M24*] Покажите, что любое неотрицательное комплексное число вида $a + bi$, где a и b — целые числа, обладает единственным "периодическим двоичным представлением"

$$(1+i)^{e_0} + i(1+i)^{e_1} - (1+i)^{e_2} - i(1+i)^{e_3} + \cdots + i^t(1+i)^{e_t},$$

где $e_0 < e_1 < \cdots < e_t$ (ср. с упр. 27).

29. [*M35*] (Н. Г. де Брейн (N. G. de Bruijn).) Пусть S_0, S_1, S_2, $\ldots$ — множества неотрицательных целых чисел; говорят, что совокупность $\{S_0, S_1, S_2, \ldots\}$ обладает свойством B, если любое неотрицательное целое число n может быть единственным способом записано в виде

$$n = s_0 + s_1 + s_2 + \cdots, \qquad s_j \in S_j.$$

(Свойство B означает, что $0 \in S_j$ для всех j, поскольку $n = 0$ может быть представлено только как $0+0+0+\cdots$.) Любая система счисления со смешанным основанием $b_0, b_1, b_2, \ldots$ дает пример совокупности множеств, удовлетворяющих свойству B, если положить $S_j = \{0, B_j, \ldots, (b_j - 1)B_j\}$, где $B_j = b_0 b_1 \ldots b_{j-1}$. В таком случае представление $n = s_0 + s_1 + s_2 + \cdots$ очевидным образом соответствует представлению (9) этого числа по смешанному основанию. Далее, если совокупность $\{S_0, S_1, S_2, \ldots\}$ A_0, A_1, A_2, $\ldots$ обладает свойством B, то, каково бы ни было разбиение A_0, A_1, A_2, $\ldots$ неотрицательных целых чисел (т. е. $A_0 \cup A_1 \cup A_2 \cup \cdots = \{0, 1, 2, \ldots\}$ и $A_i \cap A_j = \emptyset$ при $i \neq j$, некоторые из множеств A_j могут быть пустыми), этим свойством обладает и полученная из нее путем "стягивания" совокупность $\{T_0, T_1, T_2, \ldots\}$, где множество T_j состоит из всех сумм вида $\sum_{i \in A_j} s_i$, взятых по всевозможным выборкам элементов $s_i \in S_i$.

Докажите, что *любая* последовательность $\{T_0, T_1, T_2, \ldots\}$, удовлетворяющая свойству B, может быть получена посредством "стягивания" некоторой совокупности $\{S_0, S_1, S_2, \ldots\}$, соответствующей системе счисления по смешанному основанию.

30. [*M39*] (Н. Г. де Брейн.) Пример системы счисления по основанию -2 показывает, что любое целое число (положительное, отрицательное или нуль) имеет единственное представление в виде

$$(-2)^{e_1} + (-2)^{e_2} + \cdots + (-2)^{e_t}, \qquad e_1 > e_2 > \cdots > e_t \geq 0, \qquad t \geq 0.$$

Назначение данного упражнения — несколько обобщить это свойство.

a) Пусть последовательность целых чисел b_0, b_1, b_2, $\ldots$ такова, что любое целое число n допускает единственное представление в виде

$$n = b_{e_1} + b_{e_2} + \cdots + b_{e_t}, \qquad e_1 > e_2 > \cdots > e_t \geq 0, \qquad t \geq 0.$$

(Данная последовательность $\langle b_n \rangle$ называется бинарным базисом.) Покажите, что найдется такое значение индекса j, что b_j нечетно, а для всех $k \neq j$ числа b_j четны.

b) Докажите, что бинарный базис $\langle b_n \rangle$ всегда может быть преобразован в последовательность вида d_0, $2d_1$, $4d_2$, $\ldots = \langle 2^n d_n \rangle$, где каждое из чисел d_k нечетно.

c) Докажите, что если каждое из чисел d_0, d_1, d_2, $\ldots$ из п. (b) равно ± 1, то последовательность $\langle b_n \rangle$ образует бинарный базис тогда и только тогда, когда существует бесконечно много d_j, равных $+1$, и бесконечно много d_j, равных -1.

d) Докажите, что последовательность $7, -13\cdot 2, 7\cdot 2^2, -13\cdot 2^3, \ldots, 7\cdot 2^{2k}, -13\cdot 2^{2k+1}, \ldots$ является бинарным базисом, и найдите представление числа $n = 1$.

▸ **31.** [*M35*] Одно обобщение представления чисел в обратном двоичном коде, известное как "2-адические числа", было предложено в работе K. Hensel, *Crelle* **127** (1904), 51–84. (В действительности К. Гензель предложил *p-адические числа* для любого простого числа p.) 2-адическое число можно рассматривать как двоичное число

$$u = (\ldots u_3 u_2 u_1 u_0 . u_{-1} \ldots u_{-n})_2,$$

представление которого бесконечно продолжается влево и лишь на конечное количество знаков вправо от разделяющей точки. Сложение, вычитание и умножение 2-адических чисел выполняются в соответствии с алгоритмом обычных арифметических операций, которые, в принципе, допускают возможность неограниченного продолжения влево. Например,

$$7 = (\ldots 000000000000111)_2 \qquad \tfrac{1}{7} = (\ldots 110110110110111)_2$$
$$-7 = (\ldots 111111111111001)_2 \qquad -\tfrac{1}{7} = (\ldots 001001001001001)_2$$
$$\tfrac{7}{4} = (\ldots 000000000000001.11)_2 \qquad \tfrac{1}{10} = (\ldots 110011001100110.1)_2$$
$$\sqrt{-7} = (\ldots 100000010110101)_2 \quad \text{или} \quad (\ldots 011111101001011)_2 .$$

Здесь число 7 — обычное число "семь" в двоичном представлении, а -7 — обратный код, неограниченно продолженный влево. Легко проверить, что обычная процедура сложения двоичных чисел дает $-7 + 7 = (\ldots 00000)_2 = 0$, если ее выполнение продолжать неограниченно долго. Значения $\frac{1}{7}$ и $-\frac{1}{7}$ представляют собой единственные 2-адические числа, которые после формального умножения на 7 дают соответственно 1 и -1. Значения $\frac{7}{4}$ и $\frac{1}{10}$ есть примеры 2-адических чисел, не являющихся 2-адическими "целыми", так как они имеют ненулевые биты справа от разделяющей точки. Приведенные два значения $\sqrt{-7}$, получающиеся одно из другого в результате перемены знака, являются единственными 2-адическими числами, которые после формального возведения в квадрат дают $(\ldots 111111111111001)_2$.

a) Докажите, что любое 2-адическое число u можно разделить на произвольное ненулевое 2-адическое число v, чтобы вычислить 2-адическое число w, удовлетворяющее равенству $u = vw$. (Следовательно, множество 2-адических чисел образует поле; см. раздел 4.6.1.)

b) Докажите, что 2-адическое представление рационального числа $-1/(2n+1)$, где n — положительное целое число, можно получить следующим образом. Сначала находим обычное двоичное разложение числа $+1/(2n+1)$, которое имеет вид периодической дроби $(0.\alpha\alpha\alpha\ldots)_2$ (α — некоторая строка из нулей и единиц). Тогда $(\ldots\alpha\alpha\alpha)_2$ будет 2-адическим представлением числа $-1/(2n+1)$.

c) Докажите, что 2-адическое представление числа u периодично (т. е. $u_{N+\lambda} = u_N$ для всех больших N при некотором $\lambda \geq 1$) тогда и только тогда, когда u рационально (т. е. $u = m/n$ для некоторых целых чисел m и n).

d) Докажите, что если n — целое число, то $\sqrt{n}$ является 2-адическим числом только в том случае, если для некоторого неотрицательного целого числа n оно удовлетворяет условию $n \bmod 2^{2k+3} = 2^{2k}$. (Таким образом, либо $n \bmod 8 = 1$, либо $n \bmod 32 = 4$, и т. д.)

32. [*M40*] (И. З. Руца (I. Z. Ruzsa).) Сформируйте бесконечно много целых чисел, в троичных представлениях которых используются только нули и единицы, а в четверичном представлении — только нули, единицы и двойки.

33. [*M40*] (Д. А. Кларнер (D. A. Klarner).) Пусть множество D — произвольное множество целых чисел, b — любое положительное целое число, а k_n — количество различных целых чисел, которые могут быть записаны как n-разрядные числа $(a_{n-1}\ldots a_1a_0)_b$ по основанию b с цифрами a_i в D. Докажите, что последовательность $\langle k_n\rangle$ удовлетворяет линейному рекуррентному соотношению, и поясните, как вычислить производящую функцию $\sum_n k_n z^n$. Проиллюстрируйте разработанный алгоритм, показав, что в случае, когда $b = 3$ и $D = \{-1, 0, 3\}$, число k_n есть число Фибоначчи.

▸ **34.** [*22*] (Г. В. Райтвайзнер (G. W. Reitwiesner), 1960.) Поясните, как представить заданное целое число n в виде $(\ldots a_2a_1a_0)_2$, где каждое из a_j есть -1, 0 либо 1, используя наименьшую ненулевую цифру.

4.2. АРИФМЕТИКА ЧИСЕЛ С ПЛАВАЮЩЕЙ ТОЧКОЙ

В ЭТОМ РАЗДЕЛЕ рассмотрены основные принципы выполнения арифметических операций над числами с "плавающей точкой" и проанализирован внутренний механизм таких вычислений. Вероятно, у многих читателей данная тема не вызовет слишком большого интереса либо потому, что в вычислительных машинах, на которых они работают, имеются встроенные команды операций над числами с плавающей точкой, либо потому, что нужные подпрограммы содержатся в операционной системе. Но не следует считать, что материал этого раздела относится исключительно к компетенции инженеров — конструкторов ЭВМ или узкого круга лиц, которые пишут системные подпрограммы для новых машин. *Каждый* грамотный программист должен иметь представление о том, что происходит при выполнении элементарных шагов арифметических операций над числами с плавающей точкой. Предмет этот совсем не так тривиален, как принято считать; в нем удивительно много интересного.

4.2.1. Вычисления с однократной точностью

А. Обозначение чисел с плавающей точкой. В разделе 4.1 были рассмотрены различные способы обозначения чисел с фиксированной точкой. При таком способе обозначения программист знает, где положено находиться разделяющей точке в числах, с которыми выполняются те или иные операции. В некоторых ситуациях при выполнении программы значительно удобнее сделать положение разделяющей точки динамически изменяющимся, иными словами, сделать точку "плавающей" и связать с каждым числом информацию о ее положении. Эта идея уже давно использовалась в научных расчетах, в особенности для представления очень больших чисел наподобие числа Авогадро $N = 6.02214 \times 10^{23}$ или таких очень малых чисел, как постоянная Планка $h = 6.6261 \times 10^{-27}$ эрг·с.

В этом разделе речь пойдет о *p-разрядных числах с плавающей точкой по основанию b с избытком q.* Такое число представляется парой величин (e, f), которой отвечает значение

$$(e, f) = f \times b^{e-q}. \tag{1}$$

Здесь e — целое число, изменяющееся в соответствующем интервале значений, а f — дробное число со знаком. Условимся, что

$$|f| < 1,$$

иными словами, разделяющая точка в позиционном представлении f находится в крайней слева позиции. Точнее говоря, соглашение о том, что мы имеем дело с p-разрядными числами, означает, что $b^p f$ — целое число и

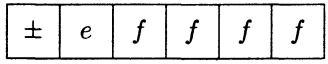
$$-b^p < b^p f < b^p. \tag{2}$$

Термин "двоичное число с плавающей точкой", как всегда, будет означать, что $b = 2$, термин "десятичное число с плавающей точкой" — что $b = 10$ и т. д. Используя 8-разрядные десятичные числа с плавающей точкой с избытком 50, можно, например, написать

$$\begin{aligned} &\text{число Авогадро} && N = (74, +.60221400); \\ &\text{постоянная Планка} && h = (24, +.66261000). \end{aligned} \tag{3}$$

Две компоненты, e и f, числа с плавающей точкой называются его *порядком* и *дробной частью* соответственно. (Иногда используются и другие названия, особенно "характеристика" и "мантисса"; однако слово "мантисса" для обозначения дробной части приводит к путанице в терминологии, так как этот термин употребляется совсем в другом смысле в теории логарифмов и, кроме того, английское слово "mantissa" означает "мало дающее добавление".)

В компьютере MIX числа с плавающей точкой имеют вид



(4)

Это представление с плавающей точкой по основанию b с избытком q, с четырьмя значащими "цифрами", где b есть размер байта (т. е. $b = 64$ или $b = 100$) и q равняется $\lfloor \frac{1}{2}b \rfloor$. Дробная часть равна $\pm\, f\, f\, f\, f$, а порядок и e находится в интервале $0 \le e < b$. Такое внутреннее представление — типичный пример соглашений, которые приняты в большинстве существующих компьютеров, хотя основание b здесь гораздо больше, чем обычно используемое.

B. Нормализованные вычисления. Число с плавающей точкой (e, f) является нормализованным, либо если наиболее значимая цифра в представлении f отлична от нуля, так что

$$1/b \le |f| < 1, \tag{5}$$

либо если $f = 0$, а e принимает наименьшее возможное значение. Чтобы установить, какое из двух нормализованных чисел с плавающей точкой имеет бóльшую величину, достаточно сравнить их порядки; только если порядки равны, нужно анализировать и дробные части.

Большинство ныне применяемых стандартных подпрограмм работает почти исключительно с нормализованными числами: предполагается, что входные значения для подпрограмм нормализованы, а результаты всегда нормализуются. При реализации этих соглашений в системных библиотеках мы теряем возможность представлять некоторые числа очень малой величины (например, значение $(0, .00000001)$ не может быть нормализовано без формирования отрицательного порядка), но мы выигрываем в скорости, единообразии и получаем возможность сравнительно легко ограничить относительную ошибку вычислений. (Арифметика ненормализованных чисел с плавающей точкой будет рассмотрена в разделе 4.2.2.)

Рассмотрим теперь арифметические операции над нормализованными числами с плавающей точкой подробнее. Попутно затронем и структуру подпрограмм, реализующих эти операции (предполагая, что в нашем распоряжении имеется компьютер без аппаратной реализации этих арифметических операций).

В стандартных подпрограммах для выполнения арифметических действий над числами с плавающей точкой, написанных на машинном языке, в очень большой степени используются крайне специфические особенности конкретной модели компьютера. Именно поэтому так мало сходства между двумя подпрограммами, скажем, сложения чисел с плавающей точкой, написанными для разных машин. Все же тщательный анализ большого числа подпрограмм как для двоичных, так и для десятичных компьютеров показывает, что в действительности данные программы имеют много общего, и обсуждение этой темы, вполне возможно, не зависит от конкретной машины.

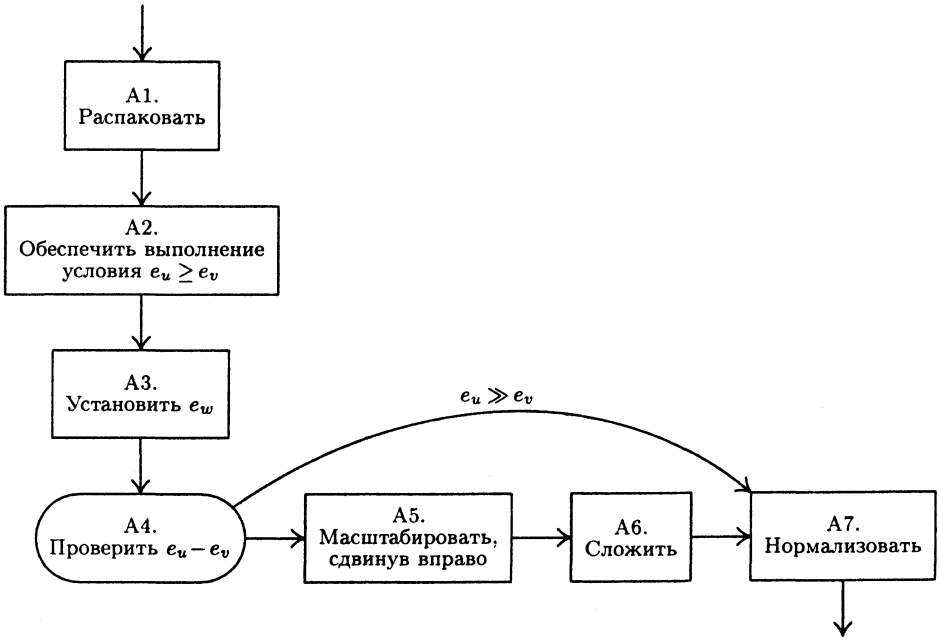


Рис. 2. Сложение чисел с плавающей точкой.

Первый (и наиболее трудный!) из алгоритмов, обсуждаемых в этом разделе, — это процедура сложения чисел с плавающей точкой:

$$(e_u, f_u) \oplus (e_v, f_v) = (e_w, f_w). \tag{6}$$

Ввиду того что арифметические действия над числами с плавающей точкой являются по самой своей сути приближенными, а не точными, для обозначения операций сложения, вычитания, умножения и деления с плавающей точкой здесь будут использоваться "округленные" символы

$$\oplus, \quad \ominus, \quad \otimes, \quad \oslash,$$

чтобы отличать приближенные операции от точных.

Идея, лежащая в основе сложения с плавающей точкой, довольно проста. Полагая, что $e_u \geq e_v$, формируем результат по принципу $e_w = e_u$, $f_w = f_u + f_v/b^{e_u - e_v}$ (таким образом, выравнивается положение разделяющих точек и соответственно положение разрядов слагаемых), а затем нормализуем результат. Может возникнуть несколько ситуаций, которые делают выполнение этого процесса нетривиальным; более точное описание метода дается в следующем алгоритме.

Алгоритм А (*Сложение чисел с плавающей точкой*). Для заданных p-разрядных нормализованных чисел с плавающей точкой $u = (e_u, f_u)$ и $v = (e_v, f_v)$ по основанию b с избытком q строится сумма $w = u \oplus v$. Данный алгоритм (рис. 2) можно использовать и для вычитания чисел с плавающей точкой, если v заменить на $-v$.

А1. [Распаковать.] Выделить порядок и дробную часть в представлениях для u и v.

A2. [Обеспечить выполнение условия $e_u \geq e_v$.] Если $e_u < e_v$, поменять местами u и v. (Во многих случаях удобнее совместить шаг A2 с шагом A1 или с каким-нибудь из последующих шагов.)

A3. [Установить e_w.] Установить $e_w \leftarrow e_u$.

A4. [Проверить $e_u - e_v$.] Если $e_u - e_v \geq p + 2$ (большая разница в порядках), установить $f_w \leftarrow f_u$ и перейти к шагу A7. (Так как предполагается, что u нормализовано, на этом выполнение алгоритма можно было бы и закончить, но часто полезно использовать операцию сложения с нулем для гарантированной нормализации любого, возможно, и ненормализованного, числа.)

A5. [Масштабировать, сдвинув вправо.] Сдвинуть f_v вправо на $e_u - e_v$ позиций, т. е. разделить f_v на $b^{e_u - e_v}$. [*Замечание.* Величина сдвига может достигать $p + 1$ разрядов, вследствие чего для выполнения следующего шага (сложения дробной части f_u с f_v) потребуется аккумулятор, способный хранить $2p+1$ цифр по основанию b справа от позиционной точки. Если такого вместительного аккумулятора нет, можно сократить сдвиг до $p + 2$ или $p + 3$ разрядов, но с соответствующими предосторожностями; подробности обсуждаются в упр. 5.]

A6. [Сложить.] Установить $f_w \leftarrow f_u + f_v$.

A7. [Нормализовать.] (В этот момент (e_w, f_w) представляет сумму u и v, но $|f_w|$ может содержать более p цифр и может быть больше единицы или меньше $1/b$.) Выполнить описываемый ниже алгоритм N, который нормализует и округлит (e_w, f_w), а также сформирует окончательный результат. ▮

Алгоритм N (*Нормализация*). "Грубый порядок" e и "сырая дробная часть" f приводятся к нормализованному виду с округлением при необходимости до p разрядов. В этом алгоритме (рис. 3) предполагается, что $|f| < b$.

N1. [Проверить f.] Если $|f| \geq 1$ (переполнение дробной части), перейти к шагу N4. Если $f = 0$, установить e равным его наименьшему значению и перейти к шагу N7.

N2. [f нормализовано?] Если $|f| \geq 1/b$, перейти к шагу N5.

N3. [Масштабировать, сдвинув влево.] Сдвинуть f на один разряд влево (т. е. умножить на b) и уменьшить e на 1. Возвратиться к шагу N2.

N4. [Масштабировать, сдвинув вправо.] Сдвинуть f вправо на один разряд (т. е. разделить на b) и увеличить e на 1.

N5. [Округлить.] Округлить f до p разрядов. (Это означает, что f изменяется до ближайшего кратного b^{-p}. Возможно, что $(b^p f) \bmod 1 = \frac{1}{2}$, т. е. имеется *два* ближайших кратных. Если b четно, то заменяем f ближайшим кратным b^{-p}, таким, что $b^p f' + \frac{1}{2}b$ нечетно (обозначим результат округления в таком случае через f'). Более подробное обсуждение аспектов округления приводится в разделе 4.2.2.) Важно отметить, что операция округления может привести к равенству $|f| = 1$ (переполнение при округлении); в такой ситуации следует вернуться к шагу N4.

N6. [Проверить e.] Если порядок e слишком велик, т. е. больше допустимой границы, это воспринимается, как выполнение условия *переполнения порядка*. Если e слишком мал, это воспринимается, как выполнение условия *исчезновения*

Рис. 3. Нормализация (e, f).

порядка. (Дополнительная информация по этому вопросу приводится ниже; эти ситуации интерпретируются обычно, как сигнал об ошибке, в том смысле, что результат не может быть представлен в виде нормализованного числа с плавающей точкой из требуемого интервала значений.)

N7. [Упаковать.] Объединить порядок e и дробную часть f для выдачи искомого результата. ▮

Несколько простых примеров сложения чисел с плавающей точкой рассматривается в упр. 4.

Приведенные ниже подпрограммы для сложения и вычитания на компьютере MIX чисел, имеющих форму (4), служат примером программной реализации алгоритмов A и N. Эти подпрограммы извлекают одно входное значение u по символическому адресу ACC, другое входное значение v извлекается из регистра A при входе в подпрограмму. Результат w одновременно появляется в регистре A и в поле ACC. Таким образом, последовательность команд

```
LDA A;  ADD B;  SUB C;  STA D,                    (7)
```

работающих с числами с фиксированной точкой, соответствовала бы такой последовательности команд, работающих с числами с плавающей точкой:

```
LDA A, STA ACC;  LDA B, JMP FADD;  LDA C, JMP FSUB;  STA D.    (8)
```

Программа A (*Сложение, вычитание и нормализация*). Следующая программа представляет собой подпрограмму, реализующую алгоритм A, причем она построена таким образом, что нормализующий фрагмент может использоваться другими

подпрограммами, которые будут рассмотрены ниже. Как в этой программе, так и во многих других программах данной главы, идентификатор OFLO именует подпрограмму, которая печатает сообщение о том, что индикатор переполнения машины MIX внезапно перешел в состояние "включено". Предполагается, что размер байта b кратен 4. В подпрограмме нормализации NORM предполагается, что rI2 $= e$ и rAX $= f$, где rA $= 0$ влечет за собой rX $= 0$ и rI2 $< b$.

```
00  BYTE  EQU  1(4:4)        Размер байта b.
01  EXP   EQU  1:1           Определение поля порядка.
02  FSUB  STA  TEMP          Подпрограмма вычитания чисел с плавающей точкой.
03        LDAN TEMP          Изменить знак операнда.
04  FADD  STJ  EXITF         Подпрограмма сложения чисел с плавающей точкой.
05        JOV  OFLO          Снятие блокировки переполнения.
06        STA  TEMP          TEMP ← v.
07        LDX  ACC           rX ← u.
08        CMPA ACC(EXP)      Шаги A1–A3 скомбинированы ниже.
09        JGE  1F            Переход, если e_v ≥ e_u.
10        STX  FU(0:4)       FU ← ± f f f f 0.
11        LD2  ACC(EXP)      rI2 ← e_w.
12        STA  FV(0:4)
13        LD1N TEMP(EXP)     rI1 ← −e_v.
14        JMP  4F
15  1H    STA  FU(0:4)       FU ← ± f f f f 0 (u, v меняются ролями).
16        LD2  TEMP(EXP)     rI2 ← e_w.
17        STX  FV(0:4)
18        LD1N ACC(EXP)      rI1 ← −e_v.
19  4H    INC1 0,2           rI1 ← e_u − e_v. (Шаг A4 не является необходимым.)
20  5H    LDA  FV            A5. Масштабировать, сдвинув вправо.
21        ENTX 0             Очистить rX.
22        SRAX 0,1           Сдвиг вправо на e_u − e_v позиций.
23  6H    ADD  FU            A6. Сложение.
24        JOV  N4            A7. Нормализация. Переход, если произошло
                               переполнение.
25        JXZ  NORM          Простой случай?
26        CMPA =0=(1:1)      f нормализовано?
27        JNE  N5            Если нормализовано, округлить его.
28        SRC  5             |rX| ↔ |rA|.
29        DECX 1             (rX положительно.)
30        STA  TEMP          (Операнды имеют противоположные знаки;
31        STA  HALF(0:0)       нужно уточнить состояние регистров
32        LDAN TEMP            перед округлением и нормализацией.)
33        ADD  HALF
34        ADD  HALF          Дополнить наименьшей значащей величиной.
35        SRC  4             Переход к подпрограмме нормализации.
36        JMP  N3A
37  HALF  CON  1//2          Половина размера слова (знак может изменяться).
38  FU    CON  0             Дробная часть f_u.
39  FV    CON  0             Дробная часть f_v.
```

```
40  NORM  JAZ  ZRO          N1. Проверить f.
41  N2    CMPA =0=(1:1)     N2. Нормализовано ли f?
42        JNE  N5           Перейти к шагу N5, если ведущий байт
                              отличен от нуля.
43  N3    SLAX 1            N3. Масштабировать, сдвинув влево.
44  N3A   DEC2 1            Уменьшить e на 1.
45        JMP  N2           Вернуться к шагу N2.
46  N4    ENTX 1            N4. Масштабировать, сдвинув вправо.
47        SRC  1            Выполнить сдвиг вправо и вставить "1"
                              с соответствующим знаком.
48        INC2 1            Увеличить e на 1.
49  N5    CMPA =BYTE/2=(5:5) N5. Округление.
50        JL   N6           |остаток| < (1/2)b?
51        JG   5F
52        JXNZ 5F           |остаток| > (1/2)b?
53        STA  TEMP         |остаток| = (1/2)b; округлить до нечетного.
54        LDX  TEMP(4:4)
55        JXO  N6           Перейти к шагу N6, если rX нечетно.
56  5H    STA  *+1(0:0)     Сохранить знак rA.
57        INCA BYTE         Добавить b^{-4} к |f| (знак может изменяться).
58        JOV  N4           Проверить переполнение из-за округления.
59  N6    J2N  EXPUN        N6. Проверка e. Исчезновение порядка, если e < 0.
60  N7    ENTX 0,2          N7. Упаковать. rX ← e.
61        SRC  1
62  ZRO   DEC2 BYTE         rI2 ← e − b.
63  8H    STA  ACC
64  EXITF J2N  *            Выход, если только не e ≥ b.
65  EXPOV HLT  2            Обнаружено переполнение порядка.
66  EXPUN HLT  1            Обнаружено исчезновение порядка.
67  ACC   CON  0            Аккумулятор для операций с плавающей точкой.
```

Довольно большой фрагмент кода программы (строки 25–37) включен в программу по той причине, что в MIX имеется 5-байтовый аккумулятор для сложения чисел со знаком, в то время как алгоритм A в общем случае требует для него $2p + 1 = 9$ разрядов. В действительности всю программу можно сократить почти вдвое, если пожертвовать хотя бы небольшой долей точности. Однако, как будет показано в следующем разделе, всегда лучше получать наибольшую возможную точность. В строке 55 используется нестандартная команда MIX, описанная в разделе 4.5.2. Время выполнения сложения и вычитания чисел с плавающей точкой зависит от нескольких факторов, анализируемых ниже, в разделе 4.2.4.

Рассмотрим теперь операции умножения и деления, которые выполняются проще, чем сложение, и довольно похожи одна на другую.

Алгоритм M (*Умножение или деление чисел с плавающей точкой*). По данным p-разрядным нормализованным числам с плавающей точкой $u = (e_u, f_u)$ и $v = (e_v, f_v)$ по основанию b с избытком q строится произведение $w = u \otimes v$ или частное $w = u \oslash v$.

M1. [Распаковать.] Выделить порядки и дробные части в представлениях u и v.

(Иногда удобно, хотя и необязательно, проверить в ходе выполнения этого шага, не равны ли операнды нулю.)

M2. [Выполнить операцию.] Установить

$$\begin{aligned} e_w &\leftarrow e_u + e_v - q, \qquad & f_w &\leftarrow f_u f_v && \text{для умножения;} \\ e_w &\leftarrow e_u - e_v + q + 1, & f_w &\leftarrow (b^{-1} f_u)/f_v && \text{для деления.} \end{aligned} \tag{9}$$

(Поскольку предполагается, что вводимые числа нормализованы, в результате получим, что либо $f_w = 0$, либо $1/b^2 \le |f_w| < 1$, либо возникнет ошибка "деление на нуль".) Здесь, если в этом есть необходимость, можно урезать f_w до $p+2$ или $p+3$ разрядов, как в упр. 5.

M3. [Нормализовать.] Применить к (e_w, f_w) алгоритм N с тем, чтобы нормализовать, округлить и упаковать результат. (*Замечание.* В этом случае нормализация выполняется проще ввиду того, что масштабирование посредством сдвига влево происходит не более одного раза и после деления переполнение не может возникнуть вследствие округления.) ▮

В подпрограммах для MIX, тексты которых приведены ниже, используются те же соглашения, что и в программе A. Эти подпрограммы служат примером машинной реализации алгоритма M.

Программа M (*Умножение и деление чисел с плавающей точкой*).

```
01  Q      EQU  BYTE/2       q есть половина размера байта.
02  FMUL   STJ  EXITF        Подпрограмма умножения чисел с плавающей точкой.
03         JOV  OFLO         Снятие блокировки переполнения.
04         STA  TEMP         TEMP ← v.
05         LDX  ACC          rX ← u.
06         STX  FU(0:4)      FU ← ± f f f f 0.
07         LD1  TEMP(EXP)
08         LD2  ACC(EXP)
09         INC2 -Q,1         rI2 ← e_u + e_v − q.
10         SLA  1
11         MUL  FU           Умножить f_u на f_v.
12         JMP  NORM         Нормализовать, округлить и выйти из подпрограммы.

13  FDIV   STJ  EXITF        Подпрограмма деления чисел с плавающей точкой.
14         JOV  OFLO         Снятие блокировки переполнения.
15         STA  TEMP         TEMP ← v.
16         STA  FV(0:4)      FV ← ± f f f f 0.
17         LD1  TEMP(EXP)
18         LD2  ACC(EXP)
19         DEC2 -Q,1         rI2 ← e_u − e_v + q.
20         ENTX 0
21         LDA  ACC
22         SLA  1            rA ← f_u.
23         CMPA FV(1:5)
24         JL   *+3          Переход, если |f_u| < |f_v|.
25         SRA  1            Иначе — масштабировать f_u вправо
26         INC2 1                  и увеличить rI2 на 1.
```

```
27          DIV  FV          Разделить.
28          JNOV NORM        Нормализовать, округлить и выйти из подпрограммы.
29  DVZRO HLT  3             Ненормализовано или делитель равен нулю.  ▮
```

Наиболее интересная особенность этой программы — подготовка к выполнению деления, осуществляемая командами в строках 23–26. Эти операции проводятся для того, чтобы обеспечить достаточную точность при округлении ответа. При $|f_u| < |f_v|$, непосредственно применив алгоритм M, можно сохранить результат в форме "$\pm 0 f f f f$" в регистре A, что сделает невозможным чистое округление без специального анализа остатка (он хранится в регистре X). Поэтому в такой ситуации программа вычисляет $f_w \leftarrow f_u/f_v$, гарантируя, что f_w либо равно нулю, либо во всех случаях нормализовано. Процедура округления может оперировать пятью значащими байтами, возможно, проверяя, не равен ли остаток нулю.

Иногда может потребоваться перевод из представления с фиксированной точкой в представление с плавающей точкой и обратно. При помощи описанного выше алгоритма нормализации легко получается программа перевода "из фиксированной в плавающую". Например, целое число переводится в форму с плавающей точкой с помощью следующей подпрограммы на языке MIX.

```
01  FLOT STJ  EXITF  Предполагаем, что rA = u есть целое число.
02        JOV  OFLO  Снятие блокировки переполнения.
03        ENT2 Q+5   Установить "грубый" порядок.                    (10)
04        ENTX 0
05        JMP  NORM  Нормализовать, округлить и выйти из подпрограммы.  ▮
```

Подпрограмма перевода "из плавающей в фиксированную" служит предметом упр. 14.

Отладка подпрограмм выполнения арифметических операций над числами с плавающей точкой — обычно довольно сложная задача из-за обилия различных случаев, которые нужно предусмотреть. Ниже перечислены распространенные ловушки, подстерегающие программиста, который занимается программами "плавающей арифметики".

1) *Потеря знака.* Во многих машинах (к MIX это не относится) команды сдвига регистров воздействуют на знаковый разряд, поэтому следует подробно анализировать операции сдвига, используемые при нормализации и изменении масштаба дробной части числа. Часто знак теряется и при появлении "минус нуля". (Например, операторы 30–34 программы A ответственны за установку знакового разряда регистра A. См. также упр. 6.)

2) *Невозможность правильного определения, что произошло — исчезновение или переполнение порядка.* Не следует проверять величину e_w *до окончания* операций округления и нормализации, так как предварительные проверки могут привести к ошибочным результатам. Исчезновение и переполнение порядка могут происходить и при выполнении сложения и вычитания, а не только при умножении и делении, и, несмотря на то что это событие довольно редкое, проверку необходимо проводить для каждого конкретного случая. Для того чтобы выполнить необходимые корректирующие операции после обнаружения исчезновения или переполнения, нужно заранее позаботиться о сохранении достаточной для этого информации.

К сожалению, некоторые программисты во многих случаях пренебрегают исчезновением порядка. Они просто полагают, что исчезающе малые результаты равны нулю без индикации ошибки. Часто это приводит к серьезной потере точности (а на самом деле к потере *всех* значащих цифр), что нарушает соглашения, принятые в операциях над числами с плавающей точкой. Таким образом, системная подпрограмма действительно должна информировать прикладного программиста об исчезновении порядка. Приравнивание результата к нулю допустимо только в отдельных случаях, когда результат должен складываться со значительно большей величиной. Если исчезновение порядка не фиксируется, то возникают таинственные ситуации, когда $(u \otimes v) \otimes w$ равно нулю, а $u \otimes (v \otimes w)$ нет, поскольку, скажем, умножение $u \otimes v$ приводит к исчезновению порядка, а $u \otimes (v \otimes w)$ вычисляется и без выхода порядков за пределы допустимого интервала. Подобным же образом можно найти пять таких положительных чисел a, b, c, d и y, что

$$\begin{aligned} (a \otimes y \oplus b) \oslash (c \otimes y \oplus d) &\approx \tfrac{2}{3}, \\ (a \oplus b \oslash y) \oslash (c \oplus d \oslash y) &= 1, \end{aligned} \tag{11}$$

если исчезновение порядка не фиксируется (см. упр. 9). Даже с учетом того, что подпрограммы арифметики с плавающей точкой не идеально точны, такие несуразные результаты, как (11), совсем уж неожиданны для случая, когда все числа a, b, c, d и y *положительны*! Исчезновение порядка обычно не предугадывается программистом, так что ему следует об этом сообщать*.

3) *Попадание "мусора".* При выполнении сдвига влево необходимо проследить, чтобы в освобождающиеся разряды справа не было введено чего-либо, отличного от нулей. Например, обратите внимание на команду `ENTX 0` в строке 21 программы A и "слишком легко забываемую" команду `ENTX 0` в строке 04 подпрограммы `FLOT` в (10). (Но было бы ошибкой очищать регистр X после строки 27 в подпрограмме деления.)

4) *Непредусмотренное переполнение при округлении.* Когда число наподобие .999999997 округляется до 8 цифр, происходит перенос влево от десятичной точки и результат сдвигается вправо. Многие ошибочно считали, что в ходе выполнения

* С другой стороны, нужно отметить, что современные языки программирования высокого уровня предоставляют программисту (или вовсе не предоставляют) весьма ограниченные возможности использования информации, содержащейся в стандартных программах арифметики с плавающей точкой. Подпрограммы для `MIX`, которые представлены в этом разделе, просто останавливают работу, обнаружив такую ситуацию, а это отнюдь не выход. Существует множество приложений, для которых исчезновение порядка относительно безвредно, и потому желательно найти способ, пользуясь которым программист смог бы просто и безопасно для приложения справиться с возникшей проблемой. Практика подстановки "втихомолку" нуля вместо результата с исчезнувшим порядком себя полностью дискредитировала, но существует альтернатива, которая в последнее время завоевывает все большую популярность. Ее суть в том, чтобы модифицировать самое определение формата представления чисел с плавающей точкой, допуская существование ненормализованной дробной части в случае, если порядок имеет минимальное допустимое значение. Эта идея "постепенной потери значимости" впервые была реализована в компьютере Electrologica X8; она лишь незначительно усложнила алгоритмы выполнения операций, но совершенно исключила возможность исчезновения порядка при сложении и вычитании. Рассмотрение этой идеи выходит за рамки данной книги, поэтому в простых формулах анализа относительной ошибки вычислений в разделе 4.2.2 появление постепенной потери значимости не учитывается. Тем не менее, используя формулы, подобные $\text{round}(x) = x(1-\delta)+\epsilon$, где $|\delta| < b^{1-p}/2$ и $|\epsilon| < b^{-p-q}/2$, можно показать, что формат с постепенной потерей значимости успешно срабатывает во многих важных случаях. (См. W. M. Kahan and J. Palmer, *ACM SIGNUM Newsletter* (October, 1979), 13–21.)

умножения переполнение при округлении невозможно, так как максимальное значение $|f_u f_v|$ равно $1-2b^{-p}+b^{-2p}$, а это число не может округлиться до 1. Ошибочность такого рассуждения продемонстрирована в упр. 11. Любопытно, что переполнение при округлении действительно *невозможно* при делении чисел с плавающей точкой (см. упр. 12).

Существует направление, представители которого утверждают, что можно безболезненно "округлять" .999999997 до .99999999, а не до 1.0000000, поскольку последний результат представляет все числа из интервала

$$[1.0000000 - 5 \times 10^{-8} \,..\, 1.0000000 + 5 \times 10^{-8}],$$

в то время как .99999999 представляет все числа из гораздо меньшего интервала

$$(.99999999 - 5 \times 10^{-9} \,..\, .99999999 + 5 \times 10^{-9}).$$

Хотя второй интервал и не содержит исходного числа .999999997, каждое число из второго интервала содержится в первом, так что последующие вычисления со вторым интервалом не менее точны, чем с первым. Но этот довод несовместим с математической идеологией арифметики с плавающей точкой, представленной ниже, в разделе 4.2.2.

5) *Округление до нормализации.* Неточность результата порождается и преждевременным округлением в неверных цифровых разрядах. Эта ошибка очевидна, когда округление производится слева от соответствующего разряда. Она также опасна в менее очевидном случае, когда округление сначала выполняется намного правее, а затем — в истинном разряде. По этой причине ошибочно осуществлять округление в ходе операции "сдвиг вправо" на шаге A5; исключением является случай, рассмотренный в упр. 5. (Однако специальный случай округления на шаге N5, а затем повторного округления уже после переполнения при округлении безобиден, потому что переполнение при округлении всегда приводит к значению ± 1.0000000, которое не меняется в результате последующей процедуры округления.)

6) *Невозможность сохранения достаточной точности в промежуточных вычислениях.* Детальный анализ точности арифметических операций с плавающей точкой, проводимый в следующем разделе, показывает, что нормализующие программы арифметики с плавающей точкой должны всегда обеспечивать максимальную точность подходящим образом округленного результата. Не должно быть никаких отступлений от этого принципа, даже в тех случаях, появление которых предельно маловероятно. Надлежащее число значащих цифр следует сохранять в ходе всех промежуточных вычислений, как реализовано в алгоритмах A и M.

C. Аппаратная реализация арифметических действий над числами с плавающей точкой. В арсенале почти каждой большой ЭВМ, предназначенной для научных расчетов, содержатся встроенные операции и команды арифметических операций над числами с плавающей точкой. К несчастью, в аппаратных реализациях таких команд обычно присутствуют некоторые дефекты, приводящие при определенных обстоятельствах к удручающе скверному поведению машины, и, надо надеяться, в будущем создатели вычислительной техники будут уделять больше внимания данному вопросу. Затраты на это очень малы, и соображения, представленные в следующем разделе, показывают, какой выигрыш может быть достигнут. Из того, что сегодня известно, следует, что для современных компьютеров не подходят вчерашние компромиссные решения.

Компьютер MIX, используемый в этой серии книг как пример "типичной" вычислительной машины, оснащается средством расширения для работы с числами в формате с плавающей точкой — арифметическим расширителем. Он доступен за небольшую дополнительную сумму и обеспечивает выполнение следующих шести команд на аппаратном уровне.

• FADD, FSUB, FMUL, FDIV, FLOT, FCMP (C = 1, 2, 3, 4, 5, 56 соответственно; F = 6). Содержимое регистра rA после выполнения команды FADD V такое же, как и содержимое rA после выполнения команд

STA ACC; LDA V; JMP FADD.

Здесь FADD — подпрограмма, которая уже появлялась выше в этом разделе, но оба операнда автоматически нормализуются непосредственно перед входом в подпрограмму, если они еще не были нормализованы. (Если во время предварительной нормализации, но не во время нормализации результата, возникает исчезновение порядка, вызывающая программа о нем не извещается.) Аналогичные замечания относятся к операциям FSUB, FMUL и FDIV. Содержимое регистра rA после выполнения операции FLOT совпадает с его же содержимым после выполнения команды JMP FLOT в подпрограмме (10). Содержимое rA не искажается командой FCMP V. Эта команда устанавливает индикатор сравнения в состояние LESS, EQUAL или GREATER в зависимости от того, будет ли содержимое rA "заметно меньше, чем", "примерно равно" или "заметно больше, чем" V, как обсуждается в следующем разделе. Работа этой команды в точности моделируется подпрограммой FCMP из упр. 4.2.2–17 с EPSILON в ячейке 0.

Ни одна из команд арифметики с плавающей точкой не воздействует ни на какой другой регистр, помимо rA. Если происходит переполнение или исчезновение порядка, то включается индикатор переполнения и указывается порядок результата по модулю размера байта. Попытка деления на нуль оставляет в регистре rA "мусор" (произвольное значение). Времена выполнения: $4u$, $4u$, $9u$, $11u$, $3u$ и $4u$ соответственно.

• FIX (C = 5; F = 7). Содержимое rA заменяется целым числом "round(rA)", округленным до ближайшего целого, как на шаге N5 алгоритма N. Однако, если этот результат слишком велик и не вмещается в разрядную сетку регистра, устанавливается индикатор переполнения и результат должен трактоваться как неопределенный. Время выполнения: $3u$.

Иногда полезно использовать операторы арифметики с плавающей точкой нестандартным образом. Например, если бы операция FLOT не была реализована как часть арифметического расширителя компьютера MIX, можно было бы легко обеспечить ее выполнение для 4-байтовых чисел, написав маленькую подпрограмму

```
FLOT STJ  9F
     SLA  1
     ENTX Q+4
     SRC  1                    (12)
     FADD =0=
9H   JMP  *
```

Эта программа не эквивалентна команде FLOT, так как в ней предполагается, что 1:1 байт регистра rA равен нулю; кроме того, она портит содержимое регистра rX.

В более общих ситуациях приходится прибегать ко всяким хитростям, потому что переполнение при округлении может происходить даже во время выполнения команды FLOT.

Аналогично предположим, что MIX имеет команду FADD, но не имеет FIX. Чтобы округлить число u, записанное в формате с плавающей точкой, до ближайшего целого числа с фиксированной точкой (причем известно, что число неотрицательно и займет не более трех байтов), в программе можно записать

```
FADD  FUDGE,
```

где в ячейке FUDGE содержится константа

+	Q+4	1	0	0	0

;

результат в rA будет иметь вид

+	Q+4	1	Округленное(u)

. (13)

D. История и библиография. Истоки арифметики чисел с плавающей точкой прослеживаются вплоть до вавилонян (около 1800 г. до н. э. или ранее), которые применяли арифметические операции над числами с плавающей точкой по основанию 60, но не имели обозначения для порядка. Нужный порядок всегда некоторым образом "подразумевался" тем, кто производил вычисления. По крайней мере в одном случае обнаружено, что у вавилонян получился ошибочный ответ, потому что сложение осуществлялось при неверно выполненном выравнивании операндов, однако это случалось весьма редко (см. O. Neugebauer, *The Exact Sciences in Antiquity* (Princeton, N. J.: Princeton University Press, 1952), 26–27). Другой пример раннего обращения к формату с плавающей точкой связан с именем греческого математика Аполлония (3 в. до н. э.), который, по-видимому, был первым, кто объяснил, как можно упростить умножение, по крайней мере в простых случаях, собирая степени 10 отдельно от их коэффициентов. Метод Аполлония обсуждается в труде Паппа Александрийского "Математическое собрание" (4 в. н. э.). После гибели вавилонской цивилизации представление с плавающей точкой существенным образом использовалось для формирования произведений и частных лишь много веков спустя, когда были открыты логарифмы (1600 г.) и вскоре после этого Отред (Oughtred) изобрел логарифмическую линейку (1630 г.). Примерно в тот же период было введено современное обозначение "x^n" для порядков; отдельные символы для квадрата x, куба x и т. д. использовались и раньше.

Арифметика с плавающей точкой была включена в конструкцию некоторых из самых ранних вычислительных машин. Это было независимо предложено Леонардо Торресом-и-Овьедо (Leonardo Torres y Quevedo) в Мадриде (1914 г.), Конрадом Цузе (Konrad Zuse) в Берлине (1936 г.) и Джорджем Стибицем (George Stibitz) в Нью-Джерси (1939 г.). В машине Цузе использовалось двоичное представление с плавающей точкой, которое он назвал полулогарифмической нотацией; он также реализовал соглашение относительно операций с некоторыми особыми величинами, подобными "∞" и "не определено". Первой американской вычислительной машиной, в которой появились средства для выполнения операций в формате с плавающей точкой, была Модель V (Bell Laboratories), за которой последовала гарвардская машина Марк II. Обе эти машины, спроектированные в 1944 году,