ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение
высшего профессионального образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Н. П. Вашкевич, Е. И. Калиниченко

ОСНОВЫ АРИФМЕТИКИ ЦИФРОВЫХ ПРОЦЕССОРОВ

Учебное пособие

ПЕНЗА 2010

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение
высшего профессионального образования
«Пензенский государственный университет» (ПГУ)

Н. П. Вашкевич, Е. И. Калиниченко

Основы арифметики цифровых процессоров

Допущено Учебно-методическим объединением вузов
по университетскому политехническому образованию
в качестве учебного пособия
для студентов высших учебных заведений, обучающихся по специальности
230101 «Вычислительные машины, комплексы, системы и сети»

Пенза
Издательство ПГУ
2010

УДК 681.3
В23

Р е ц е н з е н т ы :

кафедра «Вычислительные машины и системы»
ГОУ ВПО «Пензенская государственная технологическая академия»;

доктор технических наук, профессор, заведующий кафедрой
«Вычислительные системы и моделирование»
ГОУ ВПО «Пензенский государственный педагогический университет
им. В. Г. Белинского»
В. И. Горбаченко

Вашкевич, Н. П.

В23 Основы арифметики цифровых процессоров : учеб. пособие / Н. П. Вашкевич, Е. И. Калиниченко. – Пенза : Изд-во ПГУ, 2010. – 160 с.

ISBN 978-5-94170-290-9

Рассматриваются системы счисления, используемые при работе с цифровыми процессорами, форматы операндов цифровых процессоров (с фиксированной точкой, с плавающей точкой, BCD-коды), кодирование операндов, алгоритмы арифметических и логических операций во всех форматах и кодах, алгоритмы арифметики многократной точности, контроль выполнения арифметических операций, контроль передачи цифровой информации. Каждый раздел сопровождается большим количеством детальных примеров.

Учебное пособие подготовлено на кафедре «Вычислительная техника» и предназначено для студентов специальности 230101 при изучении дисциплины «Информатика» (I часть), выполнении курсового проектирования по дисциплинам, связанным с разработкой алгоритмов цифровой обработки сигналов; может быть использовано студентами других специальностей при изучении алгоритмов работы цифровых устройств.

УДК 681.3

ISBN 978-5-94170-290-9

ОГЛАВЛЕНИЕ

Предисловие

Специалистам в области информатики (computer science), занимающимся разработкой специализированных цифровых устройств на базе программируемых логических интегральных схем, программированием на языках низкого уровня, таких, как Ассемблер, C++, созданием драйверов цифровых устройств, нужно хорошо представлять, как работает цифровой процессор «изнутри», знать используемые форматы представления операндов, область использования каждого из них, способы кодирования операндов и алгоритмы перевода чисел в коды и обратно, алгоритмы базовых арифметических операций в разных форматах и кодах.

Такие знания позволят разработчику аппаратных и программных средств правильно выбирать решение на каждом этапе проектирования. В качестве примера покажем, к чему может привести выбор неправильного формата данных. В свое время одному из авторов данного пособия довелось принимать участие в разработке программного обеспечения для контроллера, управляющего промышленным объектом. Одна из задач при этом – получить сумму измерений на заданном временном интервале от датчика измерения напряжения и вычислить среднее значение. Изначально были выбраны формат целых чисел с фиксированной точкой и соответствующая ему арифметика. После того как программное обеспечение было разработано, при опытных испытаниях обнаружилось «неадекватное» поведение контроллера. Дальнейшие исследования показали, что при накоплении измерений от датчика напряжения часть измерений была равна нулю, что существенно искажало реальный результат. Оказалось, что в промышленных условиях из-за помех объект управления, к которому подключался датчик, давал такие «выбросы» напряжения, что результат измерения не помещался в разрядную сетку, т.е. старшие

разряды терялись, и получался результат измерения, близкий к нулю. Решением этой проблемы стало использование формата дробных чисел с фиксированной точкой и соответствующей ему арифметики. В этом случае при «выбросах» напряжения терялись младшие разряды результата измерения, что практически не искажало реальный результат. И если бы это решение было принято изначально, то не надо было бы переделывать программное обеспечение, что сократило бы время разработки контроллера.

В главе 1 рассматриваются системы счисления, используемые при работе с цифровыми процессорами, способы перевода чисел из одной системы в другую, двоичная система, так как она применяется для представления чисел внутри процессора. Также рассматриваются восьмеричная и шестнадцатеричная системы как широко используемые промежуточные системы между двоичной и десятичной. В главе 2 детально описываются форматы представления чисел в цифровых процессорах (с иллюстрацией многочисленными примерами) с фиксированной и плавающей точкой, а также BCD-формат. Глава 3 отведена под способы кодирования чисел в цифровых процессорах, а также сравнения их достоинств и недостатков. В главе 4 рассматриваются алгоритмы выполнения арифметических операций для всех форматов чисел и способов их кодирования, а также методы ускорения операции умножения и арифметика многократной точности. В главе 5 показываются способы контроля арифметических операций, а в главе 6 – способы контроля передачи информации в каналах связи.

Чтобы лучше усвоить материал пособия, авторы рекомендуют по каждому алгоритму выполнить вначале самостоятельно примеры, приведенные в пособии, а затем придумать и выполнить свои примеры с проверкой полученных результатов.

1. Системы счисления, используемые при работе с цифровыми процессорами

Большинство кодов, применяемых в цифровых процессорах, основано на системах счисления, использующих позиционный принцип образования числа, при котором значение каждой цифры зависит от ее положения в числе. Примером позиционной формы записи чисел является та, которой обычно пользуется человек (арабская форма записи чисел). Так, в числах 87 и 78 значения цифры 7, определяется ее положением в числе: в первом случае она обозначает семь единиц, а во втором – семь десятков. При арабской записи, для десятичного числа 625 его полное значение рассчитывается по формуле

$$6*10^2 + 2*10^1 + 5*10^0 = 625.$$

Здесь и в дальнейшем символ * означает операцию умножения.

В информатике используются в основном четыре системы счисления (все – позиционные): двоичная (представление информации в процессоре), восьмеричная, шестнадцатеричная (как промежуточные между процессором и человеком) и десятичная (используемая человеком). Двоичная система счисления используется для кодирования дискретного сигнала, который обрабатывает вычислительная техника. Это определяется тем, что двоичный сигнал просто представляется на аппаратном уровне (триггер установлен – "1", сброшен – "0"). Любая позиционная система счисления характеризуется основанием и базой. Основание системы счисления – это количество символов, используемых в каждой позиции (отсюда и берется название системы) для представления числа, а база – сами символы. Ниже в табл. 1.1 для всех четырех систем представлена используемая база (выделено серым цветом).

Используются следующие обозначения систем счисления, в которых представлены числа:

– для двоичных чисел – нижний индекс справа от числа в виде цифры 2 в скобках или буквы B (либо b) (binary – двоичный), либо знак B или b справа от числа. Например, $101000_{(2)} = 101000_b = 101000_B = 101000B = 101000b$;

–

Таблица 1.1

Системы счисления

Системы счисления			
Двоичная	Восьмеричная	Шестнадцатеричная	Десятичная
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

– для восьмеричных чисел – нижний индекс справа от числа в виде цифры 8 в скобках или букв O, o (octal – восьмеричный), либо знак O или o справа от числа. Например, $367_{(8)} = 367_O = 367_o = 367O = 367o$;

– для шестнадцатеричных чисел – нижний индекс справа от числа в виде числа 16 в скобках или букв H, h (hexadecimal – шестнадцатеричный), либо знак H или h справа от числа. Например, $3AB_{(16)} = 3AB_H = 3AB_h = 3ABH = 3ABh$.

Для перевода чисел из одной системы счисления в другую существуют определенные способы. Они различаются в зависимости от формата числа – целое или правильная дробь. Для вещественных чисел используется комбинация правил перевода для целого числа и правильной дроби. Рассмотрим различные способы перевода чисел из одной системы счисления в другую на конкретных примерах.

Перевод целых чисел. Надо перевести число 549 из десятичной в двоичную систему (рис. 1.1). Находим максимальную степень двойки, исходя из того, чтобы 2 в этой степени было меньше или равно исходному числу. Это будет 9, так как 2^9=512, а 2^{10}=1024, уже больше исходного числа. Тем самым, определяется и число разрядов результата в двоичной системе. Оно будет равно: 9+1=10 (будем нумеровать разряды двоичного числа, начиная с младшего разряда, номер которого равен 1). Результат будет иметь вид 1*********, где вместо * будут стоять двоичные цифры. Найдем девятый разряд двоичного числа. Вычисляем 2^9 и вычитаем из исходного числа: 549–2^9=37. Так как, 2^8=256, а остаток 37 меньше 256, то девятый разряд будет нулем, т.е. 10********. Получаем восьмой разряд. Так как 2^7=128>37, то он также будет нуль – 100*******. Седьмой разряд 2^6=64>37 также оказывается нулевым. Тогда двоичная запись числа будет: 1000******. 2^5=32>37, и шестой разряд равен 1 (10001*****). Следующий остаток 37–32=5, для которого выполняется неравенство 2^4=16<5, что означает равенство нулю пятого разряда. 2^3=8>5 и четвертый разряд равен 0 (1000100***). 2^2<5, третий разряд =1 (100010001**). Следующий остаток 5–4=1, для которого 2^1>1, и второй разряд равен 0 (1000100010*). 2^0=>1 и первый разряд равен 1 (1000100101). Таким образом получается разложение переводимого числа по степеням двойки:

549=1*2^9+0*2^8+0*2^7+0*2^6+1*2^5+0*2^4+0*2^3+1*2^2 +0*2^1+1*2^0.

Такой метод очень трудоемок, поэтому на практике используется другой способе перевода целых чисел, основанный на операции целочисленного деления. Рассмотрим то же самое число 549. Разделив его на 2, получим частное 274 и остаток 1. Выполним ту же самую операцию с числом 274. Получим частное 137, остаток 0. Опять делим полученное частное на 2, и так до тех пор, пока частное не станет меньше делителя. Теперь для того чтобы получить число в двоичной системе счисления, нужно записать в обратном порядке все полученные в процессе деления остатки (рисунок 1.1).

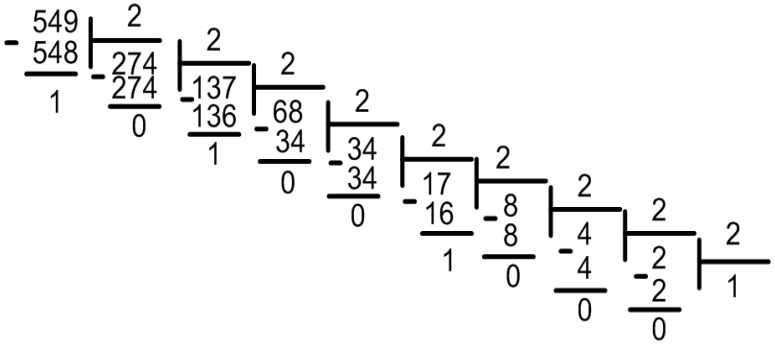

Рис. 1.1. Перевод чисел из десятичной системы в двоичную

Эти два способа применимы при переводе целых чисел из одной позиционной системы в другую. Действия вычитания, умножения выполняются в исходной системе. Рассмотрим перевод числа $500_{(10)}$ в систему счисления с основанием 16.

Произведем разложение данного числа по степеням основания шестнадцатеричной системы. Искомое число будет состоять из трех цифр, так как $16^2=256<500<16^3=4096$. Определим цифру старшего разряда. $1*16^2=256<500<2*16^2=512$, следовательно, число будет 1**, где вместо * будут стоять шестнадцатеричные цифры. Остаток равен: 500–256=244. Тогда 15*16=240 и это значит, что во втором разряде находится цифра F. Последний остаток равен 4 (244–240). Искомое шестнадцатеричное число равно 1F4.

При втором способе последовательно делим 500 на 16, и процесс деления заканчивается, когда частное становится меньше 16 (рис. 1.2).

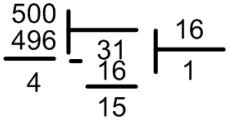

Рис. 1.2. Перевод числа из десятичной системы
в шестнадцатеричную

Операция перевода в десятичную систему проще, так как любое десятичное число можно представить в виде полинома $x=a_0*p^n+a_1*p^{n-1}+ ... +a_{n-1}*p^1+a_n*p^0$, где $a_0 ... a^n$ – цифры числа в системе счисления с

основанием р. Например, перевести число $1F4_{(16)}$ в десятичную систему. По определению, $1F4= 1*16^2+F*16^1+4*16^0$. Заменив F на 15, получим $1*16^2+15*16^1+4*16^0=500$.

Очень просто осуществляется перевод чисел из двоичной системы в системы с основанием, равным степеням двойки (4,8,16 и т.д.), и наоборот (поэтому они и используются в качестве промежуточных при работе с компьютером). Для того чтобы целое двоичное число перевести в систему счисления с основанием 2^n, нужно двоичное число разбить справа налево на группы по n цифр в каждой; если в последней левой группе окажется меньше n разрядов, то дополнить ее нулями до нужного числа разрядов; рассмотреть каждую группу как n-разрядное двоичное число и заменить ее соответствующей цифрой в системе счисления с основанием 2^n. Аналогично выполняется перевод из системы счисления с основанием 2^n в двоичную систему счисления. Рассмотрим примеры такого перевода.

Пример 1. Перевести $1010001010_{(2)}$ в восьмеричную систему счисления:

1	0	1	0	0	0	1	0	1	0
триада 4	триада 3			триада 2			триада 1		

триада $1=010_{(2)}=2_{(8)}$;

триада $2=001_{(2)}=1_{(8)}$;

триада $3=010_{(2)}=2_{(8)}$;

триада $4=1_{(2)}=1_{(8)}$.

Тогда, располагая восьмеричные цифры по весам разрядов, получаем:

$1010001010_{(2)}=1212_{(8)}$.

Пример 2. Перевести $4A3F_{(16)}$ в двоичную систему счисления:

$4_{(16)}=0100_{(2)}$ – тетрада;

$A_{(16)}=1010_{(2)}$ – тетрада;

$3_{(16)}=0011_{(2)}$ – тетрада;

$F_{(16)}=1111_{(2)}$ – тетрада.

Тогда, располагая каждую тетраду с учетом ее веса, получаем:

$4A3F_{(16)}=0100101000111111_{(2)}=100101000111111_{(2)}$.

Перевод правильных дробей. Правильная дробь имеет целую часть, равную 0.

Перевод дробей из десятичной системы счисления в двоичную, восьмеричную или шестнадцатеричную системы (аналогично осуществляется перевод в любую позиционную систему):

а) исходная дробь умножается на основание системы счисления, в которую переводится (например, на 2, 8 или 16);

б) в полученном произведении целая часть записывается в соответствии с табл. 1.1 цифрой системы счисления, в которую осуществляется перевод (она является старшей цифрой получаемой дроби в новой системе счисления);

в) полученная на предыдущем шаге целая часть отбрасывается, и оставшаяся дробная часть (это правильная дробь) вновь умножается на основание системы счисления с последующей обработкой полученного произведения в соответствии с шагами а) и б);

г) процедура умножения продолжается до тех пор, пока не будет получен нулевой результат в дробной части произведения (дробь переводится точно) или не будет достигнута требуемая точность, а именно необходимое число разрядов дробной части (дробь переводится приближенно);

д) формируется искомое число путем последовательной записи полученных в шаге б) цифр, т.е. в порядке уменьшения веса разрядов.

Рассмотрим пример такого перевода (рис. 1.3).

Пример 1. Выполнить перевод десятичного числа 0,641 в двоичную систему счисления. Перевод выполнить с точностью до двух значащих цифр после запятой. Более эффективным путем достижения цели будет перевод числа вначале в шестнадцатеричную/восьмеричную систему, а затем – в двоичную.

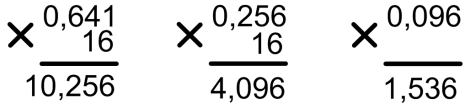

Рис. 1.3. Перевод дроби из десятичной системы в шестнадцатеричную

Таким образом, $0{,}641_{(10)} \approx 0{,}A4_{(16)} = 0{,}11011001_{(2)}$. Отметим, что третья получаемая шестнадцатеричная цифра после запятой равная 1, использована для округления предыдущего разряда по правилу «половина веса отбрасываемого разряда». Здесь «половина веса отбрасываемого разряда» равна 8.

Пример 2. Выполнить перевод числа $0{,}147_{(16)}$ в двоичную систему счисления:

$1_{(16)}=0001_{(2)}$;

$4_{(16)}=0100_{(2)}$;

$7_{(16)}=0111_{(2)}$.

Тогда, располагая тетрады с учетом их веса, получаем:

$0{,}147_{(16)}=0{,}000101000111_{(2)}$.

Перевод смешанных чисел. Смешанным числом в дальнейшем будем называть вещественное число, имеющее как целую, так и дробную части, отличные от нуля.

При переводе таких чисел отдельно переводится целая часть числа, отдельно – дробная. Результаты суммируются.

Пример 1. Выполнить перевод из десятичной системы счисления в шестнадцатеричную систему числа 19,847. Перевод выполнять с точностью до двух значащих цифр после запятой.

Представим исходное число как сумму целого числа и правильной дроби:

$19{,}847_{(10)}=19_{(10)}+0{,}847_{(10)}$;

$19_{(10)}=13_{(16)}=00010011_{(2)}=10011_{(2)}$;

$0{,}847_{(10)}\approx 0{,}D9_{(16)}=0{,}11011001_{(2)}$.

Тогда, располагая тетрады с учетом их веса, получаем:

$19{,}847_{(10)}=10011{,}11011001_{(2)}$.

Пример 2. Выполнить перевод числа $2BC{,}247_{(16)}$ в двоичную систему счисления:

$2_{(16)}=0010_{(2)}$;

$B_{(16)}=1011_{(2)}$;

$C_{(16)}=1100_{(2)}$;

$2=0010_{(2)}$;

$4=0100_{(2)}$;

$7=0111_{(2)}$.

Тогда, располагая тетрады с учетом их веса, получаем:

$2BC{,}247_{(16)}=001010111100{,}001001000111_{(2)}=1010111100{,}001001000111_{(2)}$.

2. Форматы представления чисел в цифровых процессорах

Цифровой процессор – это устройство, осуществляющее обработку информации, представленной в цифровом виде. Он может быть универсальным (центральный процессор) или специализированным. Одной из основных его характеристик является «разрядность процессора». Разрядность процессора – это количество битов в регистре процессора, которое за один такт работы используется для выполнения операций сложения, вычитания или сдвига. При выполнении **арифметических операций** цифровой процессор использует представление чисел (называемых операндами) в регистрах или оперативной памяти в одном из следующих форматов:

1) формат с фиксированной точкой;
2) формат с плавающей точкой;
3) формат двоично-десятичного кода.

Рассмотрим каждый из этих форматов.

2.1. Формат с фиксированной точкой

Формат с фиксированной точкой (ФТ), в зависимости от представляемых чисел, имеет три вида (табл. 2.1).

Таблица 2.1

Форматы чисел с ФТ

Вид ФТ	
Целые числа	без знака
	со знаком
Дробные числа	без знака
	со знаком
Смешанные числа	без знака
	со знаком

Рассмотрим на примерах представление чисел (операндов) в каждом из этих видов. Начнем с формата целых чисел (в дальнейшем будем использовать восьмиразрядный процессор, если не оговорено

иное). Целое число К, заданное в восьмеричной системе счисления, представим в формате с ФТ.

$К=51_{(8)}=101001_{(2)}$.

Без знака:

0	0	1	0	1	0	0	1

Со знаком:

знак	0	1	0	1	0	0	1

Дробное число М, заданное в восьмеричной системе счисления, представим в формате с ФТ.

$М=0,51_{(8)}=0,101001_{(2)}$.

Без знака:

1	0	1	0	0	1	0	0

Со знаком:

знак	1	0	1	0	0	1	0

Смешанное число L, заданное в восьмеричной системе счисления, представим в формате с ФТ в шестнадцатиразрядном процессоре, где левые 8 бит отведены под целую часть числа, включая знак, а остальные 8 – под дробную часть.

$L=51,15_{(8)}=101001,001101_{(2)}$.

Без знака:

0	0	1	0	1	0	0	1	0	0	1	1	0	1	0	0

Со знаком:

знак	0	1	0	1	0	0	1	0	0	1	1	0	1	0	0

2.2. Формат с плавающей точкой

Операнды цифрового процессора в формате с плавающей точкой представляют числа в экспоненциальной форме. Такой формат используется в основном в научно-технических расчетах, когда диапазон чисел в вычислениях может варьироваться от очень малых величин до очень больших. В отличие от формата с ФТ, в котором выполняются точные вычисления, операции в таком формате выполняются с приближением, определяемым разрядной сеткой процессора.

Число $2008_{(10)}$ в экспоненциальной форме можно представить как $2008_{(10)}=20{,}08*10^2=0{,}002008*10^6=0{,}2008*10^4$.

Такое представление числа состоит из двух частей: мантиссы и порядка. Для приведенного выше примера мантисса равна 20,08 или 0,002008, или 0,2008. Порядок равен соответственно 2 или 6, или 4. Любое число в экспоненциальной форме имеет множество представлений. Среди этих представлений есть так называемое нормализованное представление числа. Для каждого числа это представление – единственное. При нормализированном представлении числа в экспоненциальной форме мантисса (M) должна быть в интервале

$0{,}1_{(d)} \le M < 1_{(d)}$, где d – основание системы счисления.

Например, для числа $2008_{(10)}$ нормализованное представление равно $0{,}2008*10^4$. Для числа $0{,}0002ABB_{(16)}$ нормализованное представление равно $0{,}2ABB*10^{-3}$. Для числа $11010{,}0001_{(2)}$ нормализованное представление равно $0{,}110100001*10^5$. Следует обратить внимание, что 10 – это основание системы счисления, т.е. для десятичной оно равно $10_{(10)}$, для шестнадцатеричной равно $16_{(10)}$, для двоичной равно $2_{(10)}$,

В настоящее время общепринятым стандартом представления операндов в формате с плавающей точкой в цифровом процессоре является стандарт IEEE 754. В IEEE 754 число в экспоненциальной форме представляется операндом в двоичной системе счисления, состоящим из двух частей: мантиссы и порядка. Каждая из этих частей имеет знак. Мантисса представляется в прямом коде, а порядок «смещен» (увеличен) на константу. Смещение порядка на константу позволяет обойтись без явного бита знака порядка. Если значение смещенного порядка больше константы, он – положительный, если меньше – отрицательный. Есть три формата представления чисел с плавающей точкой в оперативной памяти процессора: «короткий вещественный (КВ)», «длинный вещественный (ДВ)», «внутренний вещественный (ВВ)». Они отличаются диапазоном представимых в них чисел. Первые два из них приведены на рис. 2.1.

На рис. 2.1 используются следующие обозначения: M – мантисса числа; S – знак мантиссы; P – порядок числа.

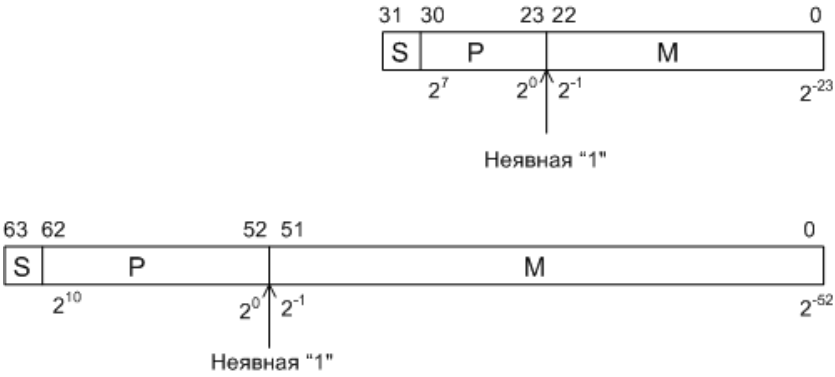

Рис. 2.1. Форматы с плавающей точкой КВ и ДВ

В формате КВ под мантиссу отводится 24 бит, а под порядок – 8 бит. Величина порядка операнда смещена на $127_{(10)}$, т.е. $P=P_x+127_{(10)}$.

В формате ДВ под мантиссу отводится 53 бит, а под порядок – 11 бит. Величина порядка операнда смещена на $1023_{(10)}$, т.е. $P=P_x+1023_{(10)}$.

В формате ВВ под мантиссу отводится 64 бит, а под порядок – 15 бит. Величина порядка операнда смещена на $16383_{(10)}$, т.е. $P=P_x+16383_{(10)}$.

Поскольку при нормализованном представлении операнда ($0,1_{(2)}\leq$ $\leq M<1_{(2)}$) в двоичной системе счисления первая цифра мантиссы после запятой всегда будет равна «1», это можно использовать для увеличения диапазона представимых чисел, для чего диапазон представления мантиссы нормализованного числа меняется на диапазон

$$1_{(2)}\leq M<2_{(2)}.$$

Причем единица целой части мантиссы учитывается неявно, т.е. под нее не отводится бит. В таком виде операнд хранится в памяти процессора. При выполнении арифметических операций над ним, при извлечении из памяти в операционный автомат этот скрытый бит восстанавливается, т.е. присутствует в явном виде.

В дальнейшем в пособии будет рассматриваться только формат КВ, поскольку представление в двух других форматах и алгоритмы выполнения арифметических операций полностью аналогичны.

Рассмотрим примеры представления операндов в формате КВ. Будем обозначать мантиссу через М, а порядок – через Р.

Пример 1. Представить число $16,AC_{(16)}$ в формате КВ.

$16,AC_{(16)}=10110,10101100_{(2)}=1,011010101100_{(2)}*10^{100}$.

$M=1,011010101100_{(2)}$;

$P=100_{(2)}+(127_{(10)}=1111111_{(2)})=10000011_{(2)}$.

Тогда формат КВ этого числа (серым цветом выделены биты порядка) будет:

0	1	0	0	0	0	0	1	1	0	1	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
байт								**байт**								**байт**								**байт**							

Пример 2. Представить число $-7,C8_{(16)}$ в формате КВ.

$-7,C8_{(16)}=-111,11001000_{(2)}=-1,1111001000_{(2)}*10^{10}$.

$M=-1,1111001000_{(2)}$;

$P=10_{(2)}+(127_{(10)}=1111111_{(2)})=10000001_{(2)}$.

Тогда формат КВ этого числа (серым цветом выделены биты порядка) будет:

1	1	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
байт								**байт**								**байт**								**байт**							

Пример 3. Представить число $-0,003A_{(16)}$ в формате КВ.

$-0,003A_{(16)}=-0,0000000000111010_{(2)}=-1,11010_{(2)}*10^{-1011}$.

$M=-1,11010_{(2)}$;

$P=-1011_{(2)}+(127_{(10)}=1111111_{(2)})=01110100_{(2)}$.

Тогда формат КВ этого числа (серым цветом выделены биты порядка) будет:

1	0	1	1	1	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
байт								**байт**								**байт**								**байт**							

Пример 4. Перевести число из формата КВ в шестнадцатеричную систему счисления.

1	1	0	0	0	0	1	0	0	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
байт								**байт**								**байт**								**байт**							

$M=-1,000101011_{(2)}$;

$P=10000100_{(2)}-(127_{(10)}=1111111_{(2)})=101_{(2)}$.

$-1,000101011_{(2)}*10^{101}=-100010,1011_{(2)}=-22,B_{(16)}$.

В этих трех форматах, кроме допустимого множества значений операндов, предусмотрены специальные значения, представляющие «бесконечность», «минус бесконечность», «минус 0» и «не число» (обозначается NaN). Эти значения существуют для того, чтобы в случае возникновения ошибок, например арифметического переполнения, извлечения квадратного корня из отрицательного числа и деления на «0», можно было определенным двоичным кодом представить результат операции. Эти специальные значения сведены в табл. 2.2.

Таблица 2.2

Специальные значения формата с ПТ

Специальное число	Представление числа			Комментарий
	Знак М	Порядок	Мантисса	
+ NaN	0 0	11…11 11…11	11…11 00…01	Положительное «не число»
+ ∞	0	11…11	00…00	«Плюс бесконечность»
Положительное денормализованное число	0 0	00…00 00…00	11…11 00…00	Положительное денормализованное число
Нуль	0 1	00…00 00…00	00…00 00…00	«Плюс 0» «Минус 0»
Отрицательное денормализованное число	1 1	00…00 00…00	00…01 11…11	Отрицательное денормализованное число
– ∞	1	11…11	00…00	«Минус бесконечность»
–NaN	1 1	11…11 11…11	00…01 11…11	Отрицательное «не число»
Неопределенность	1	11…11	10…00	Неопределенное значение

2.3. Формат двоично-десятичного кода

Формат двоично-десятичного кода (BCD – binary coded decimal) используется в экономических расчетах. Так же как и в формате с ФТ, вычисления, выполняемые процессором в этом формате, точные.

Представление числа в BCD-коде – это такое представление, когда каждая десятичная цифра кодируется четырехбитовым кодом.

Четырехбитовый код десятичной цифры будем называть в дальнейшем тетрадой и обозначать

$$d=t(w1,w2,w3,w4),$$

где d – десятичная цифра; t – соответствующая ей тетрада; w_i – двоичный разряд, равный 0 или 1.

Можно предложить очень большое число таких кодов. Но для эффективного выполнения арифметических операций в цифровом процессоре BCD-код должен удовлетворять следующим четырем условиям:

1) большей десятичной цифре должен соответствовать больший двоичный код;

2) величина десятичной цифры должна очень просто определяться по значению тетрады, т.е. каждый двоичный разряд должен иметь определенный вес. Тогда значение десятичной цифры определяется как

$$d=t(w_1,w_2,w_3,w_4)=w_1*p_1+w_2*p_2+w_3*p_3+w_4*p_4,$$

где w_i – значение двоичного разряда; p_i – вес разряда;

3) желательно (не обязательно), чтобы имелось соответствие между четностью десятичной цифры и четностью соответствующего ей кода тетрады;

4) желательно (не обязательно), чтобы получение дополнительного и обратного кодов происходило по обычному правилу.

Наиболее широко в цифровых процессорах используются следующие три BCD-кода (табл. 2.3):

– 8421;
– 8421+3;
– 2421.

Таблица 2.3

Примеры BCD-кодов

Десятичная цифра	Код		
	8421	8421+3	2421
0	0000	0011	0000
1	0001	0100	0001
2	0010	0101	0010
3	0011	0110	0011
4	0100	0111	0100

Окончание табл. 2.3

Десятичная цифра	Код		
	8421	8421+3	2421
5	0101	1000	1011
6	0110	1001	1100
7	0111	1010	1101
8	1000	1011	1110
9	1001	1100	1111

Выполнение вышеперечисленных условий для BCD-кодов приводится в табл. 2.4.

Таблица 2.4

Выполнение условий для кодов

Коды	8421	8421+3	2421
Выполняются условия	1, 2, 3	1, 2, 4	1, 2, 3, 4

Несмотря на то что всем условиям удовлетворяет только код 2421, наибольшее применение получил BCD-код 8421.

Рассмотрим примеры представления в цифровых процессорах десятичных чисел в BCD-кодах.

Пример 1. Представить число $349_{(10)}$ в каждом из всех BCD-кодов в шестнадцатиразрядном процессоре, в котором одна тетрада отводится под знак, а остальные три – под числовые разряды (прямой код).

Код 8421:

знак	числовые разряды		
0000	0011	0100	1001

Код 8421+3:

знак	числовые разряды		
0000	0110	0111	1100

Код 2421:

знак	числовые разряды		
0000	0011	0100	1111

Пример 2. Представить число $-940_{(10)}$ в каждом из всех BCD-кодов в шестнадцатиразрядном процессоре, в котором одна тетрада отводится под знак, а остальные три – числовые разряды (прямой код).

Код 8421:

знак	числовые разряды		
111 1	100 1	010 0	000 0

Код 8421+3:

знак	числовые разряды		
1111	1100	0111	0011

Код 2421:

знак	числовые разряды		
1111	1111	0100	0000

В современных процессорах фирмы Intel и в большинстве других, в соответствии со стандартом IEEE 754 используется BCD-код 8421 в следующем формате. Под операнд отводится 10 байт, из них крайний левый байт – знаковый. В остальных числовых байтах записывается по две тетрады (табл. 2.5). Это позволяет обрабатывать восемнадцатиразрядные десятичные числа, что является достаточным для любых экономических расчетов.

Таблица 2.5

Формат BCD-кодирования по стандарту IEEE 754

байт 9	байт 8	байт 7	байт 6	байт 5	байт 4	байт 3	байт 2	байт 1	байт 0
знак	$d_{17}d_{16}$	$d_{15}d_{14}$	$d_{13}d_{12}$	$d_{11}d_{10}$	d_9d_8	d_7d_6	d_5d_4	d_3d_2	d_1d_0

3. Кодирование чисел в цифровых процессорах

В цифровых процессорах в каждом формате операнды представляются в одном из следующих кодов:

– прямом;

– дополнительном;

– обратном.

Введем следующую систему обозначений:

$[X]_1$ – представление операнда X в прямом коде;

$[X]_2$ – представление операнда X в дополнительном коде;

$[X]_3$ – представление операнда X в обратном коде.

Рассмотрим представление операндов в каждом из этих кодов в формате с фиксированной точкой.

3.1. Представление операндов в прямом коде

Представление операнда в прямом коде состоит из двух частей:

– знака числа, под него отводится крайний левый бит (старший);

– модуля числа.

Если число положительное – знак кодируется 0, а если отрицательное, то 1.

Рассмотрим примеры представления целых операндов в этом коде:

$K_1=51_{(8)}=101001_{(2)}$.

$[K]_1=$

0	0	1	0	1	0	0	1

$K_2=-51_{(8)}=-101001_{(2)}$.

$[K]_2=$

1	0	1	0	1	0	0	1

Рассмотрим примеры представления дробных операндов в этом коде:

$M_1=0{,}15_{(8)}=0{,}001101_{(2)}$.

$[M]_1=$

0	0	0	1	1	0	1	0

$M_2=-0{,}15_{(8)}=-0{,}001101_{(2)}$.

$[M]_2=$

1	0	0	1	1	0	1	0

Рассмотрим примеры представления смешанных операндов в этом коде для шестнадцатиразрядного процессора (по 8 бит под целую и дробную части числа):

$L_1=51,15_{(8)}=101001,001101_{(2)}$.

$[M]_1=$

0	0	1	0	1	0	0	1	0	0	1	1	0	1	0	0

$L_2= -51,15_{(8)}= -101001,001101_{(2)}$.

$[M]_2=$

1	0	1	0	1	0	0	1	0	0	1	1	0	1	0	0

Достоинство прямого кода состоит в том, что в нем более просто реализуются алгоритмы выполнения «длинных» операций – умножения и деления.

Недостатками прямого кода являются:

1) двойное представление нуля:

0	0	0	0	0	0	0	0

$= +0$

1	0	0	0	0	0	0	0

$= -0$,

которое должно быть учтено либо аппаратными, либо программными средствами;

2) представление операнда в виде двух частей (знака и модуля) приводит к достаточно сложным алгоритмам операций сложения/вычитания в цифровом процессоре.

3.2. Представление операндов в дополнительном коде

Дополнительный код положительного числа совпадает с самим числом, представленным в заданном формате, а знаковый разряд кодируется 0.

Дополнительный код отрицательного числа получается:

1) для целых чисел – дополнением до модуля, по которому работает процессор (отсюда и название кода);

2) для дробных чисел – дополнением до 1 (так как модуль, по которому работает процессор, равен 1);

3) для смешанных чисел – комбинацией 1) и 2).

Знак кодируется 1.

Рассмотрим примеры представления целых операндов в этом коде:

$K_1=51_{(8)}=101001_{(2)}$.

$[K_1]_2=$

0	0	1	0	1	0	0	1

$K_2=-51_{(8)}=-101001_{(2)}$.

Так как восьмиразрядный процессор (1 бит под знак) работает по модулю $128_{(10)}=200_{(8)}$, то дополнением $51_{(8)}$ до $200_{(8)}$ будет:

	2	0	$0_{(8)}$
–		5	$1_{(8)}$
=	1	2	$7_{(8)}$

Таким образом:

$[K_2]_2=$

1	1	0	1	0	1	1	1

Рассмотрим примеры представления дробных операндов в этом коде:

$M_1=0{,}15_{(8)}=0{,}001101_{(2)}$.

$[M_1]_2=$

0	0	0	1	1	0	1	0

$M_2=-0{,}15_{(8)}=-0{,}001101_{(2)}$

Так как восьмиразрядный процессор (1 бит под знак) работает по модулю $1_{(10)}=1_{(8)}$, то дополнением $0{,}15_{(8)}$ до $1_{(8)}$ будет:

	1	,	0	$0_{(8)}$
–	0	,	1	$5_{(8)}$
=	0	,	6	$3_{(8)}$

Таким образом:

$[M_2]_2=$

1	1	1	0	0	1	1	0

Этот способ получения дополнительного кода следует из теории чисел. На самом деле, дополнительный код числа в цифровом процессоре (с учетом формата представления) получается по следующему алгоритму:

1) если число положительное, то в знаковый бит записывается 0, а само число – в остальную разрядную сетку;

2) если число отрицательное, то в знаковый бит записывается 1, а остальные биты получаются инверсией разрядов числа с добавлением 1 в младший разряд.

Например, число $51{,}15_{(8)}=101001{,}001101_{(2)}$ в шестнадцатиразрядном процессоре будет представлено в дополнительном коде как

0	0	1	0	1	0	0	1	0	0	1	1	0	1	0	0
знак	целая часть							дробная часть							

А число $-51{,}15_{(8)}=-101001{,}001101_{(2)}$ в шестнадцатиразрядном процессоре будет представлено в дополнительном коде как

1) кодирование знака числа и инверсия разрядов;

1	1	0	1	0	1	1	0	1	1	0	0	1	0	1	1
знак	целая часть							дробная часть							

2) прибавление 1 к младшему биту;

1	1	0	1	0	1	1	0	1	1	0	0	1	1	0	0
знак	целая часть							дробная часть							

Перевод числа из дополнительного кода в двоичную систему счисления выполняется (с учетом формата представления) по тому же алгоритму, что и в дополнительный:

1) если знаковый бит равен 0 (число положительное), то знак числа – «плюс», а само число берется из остальных битов;

2) если знаковый бит равен 1 (число отрицательное), то знак числа – «минус», а остальные разряды числа получаются инверсией битов с добавлением 1 в младший бит.

Например, если в шестнадцатиразрядном процессоре (в смешанном формате с ФТ) операнд X в дополнительном коде представлен как

1	1	0	1	0	1	1	0	1	1	0	0	1	1	0	0
знак	целая часть							дробная часть							

Тогда представление числа X в двоичной системе счисления будет со знаком «минус», и разряды числа, полученные инверсией битов с добавлением 1 в младший, будут равняться:

–	0	1	0	1	0	0	1	0	0	1	1	0	0	1	1
	целая часть							дробная часть + 1							
–	0	1	0	1	0	0	1	0	0	1	1	0	1	0	0

$X=-101001{,}001101_{(2)}$.

Достоинство дополнительного кода в том, что знаковый и числовые биты операнда составляют единый код, а не две части, как в прямом. Как следствие этого, алгоритмы сложения и вычитания реализуются более просто.

Недостатками дополнительного кода являются:

1) несимметричный диапазон представления положительных и отрицательных чисел в цифровом процессоре (диапазон отрицательных больше на одно число, чем положительных);

2) более сложные по сравнению с прямым кодом алгоритмы операций умножения и деления.

Первый недостаток добавляет в алгоритмы перевода чисел в дополнительный и из дополнительного кода учет этого обстоятельства. Для примера, возьмем пятиразрядный процессор, крайний левый разряд которого знаковый. Диапазон представления целых чисел в формате с ФТ в таком процессоре от -2^4 до $2^4 - 1$, т.е. от -16 до 15. Для представления числа, равного ($-16_{(10)}$), описанный выше алгоритм перевода не подходит. Согласно ему (так как $-16_{(10)} = -10000_{(2)}$) знак «минус» даст 1 в знаковый бит, инверсия остальных разрядов с добавлением 1 дает 10000. Итого получается 6 бит, что превышает разрядную сетку процессора. Поэтому алгоритм представления максимального (по модулю) отрицательного числа иной, а именно: в знаковый бит записывается 1, все остальные биты равны 0. Аналогично, при переводе в двоичную систему счисления максимального отрицательного числа знак будет «минус», само число равно 100…0, где количество нулей будет (n – 1). Здесь n – разрядность процессора.

3.3. Представление операндов в обратном коде

Обратный код положительного числа совпадает с самим числом, а знаковый разряд кодируется 0.

Обратный код отрицательного числа получается по следующему алгоритму: в знаковый бит записывается 1, а остальные биты получаются инверсией разрядов исходного числа. Например:

$K_1 = 51_{(8)} = 101001_{(2)}$.

$[K_1]_3 =$

0	0	1	0	1	0	0	1

$K_2 = -51_{(8)} = -101001_{(2)}$.

Таким образом:

$[K_2]_3$= | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

Достоинствами обратного кода являются:

1) знаковый и числовые биты операнда составляют единый код, поэтому просто реализуются алгоритмы сложения и вычитания (но следует отметить, что они немного сложнее, чем в дополнительном коде);

2) симметричный диапазон представления положительных и отрицательных чисел в цифровом процессоре.

Недостатками обратного кода являются:

1) двойное представление нуля:

0	0	0	0	0	0	0	0	= +0

1	1	1	1	1	1	1	1	= – 0,

которое должно быть учтено либо аппаратными, либо программными средствами;

2) более сложные по сравнению с прямым кодом алгоритмы операций умножения и деления.

4. Алгоритмы базовых арифметических операций в цифровом процессоре

Базовыми арифметическими операциями в цифровом процессоре являются: сложение (вычитание), умножение, деление, при выполнении которых используются поразрядные логические операции, а также операции сдвигов.

4.1. Алгоритмы операции сложения

Существуют два основных способа сложения в цифровых процессорах – последовательный и параллельный.

При последовательном способе сложение операндов, находящихся в регистрах процессора, выполняется по тактам, начиная с младшего бита, и количество тактов равно разрядности процессора. Рассмотрим этот способ на примере сложения чисел, представленных в десятичной системе счисления. Пусть надо получить сумму $S=K_1+K_2$, где $K_1=265_{(10)}$; $K_2=346_{(10)}$.

Тогда такое сложение будет выполнено за три такта:

Такты	1-е слагаемое	2-е слагаемое	Перенос из предыдущего разряда	Сумма	Перенос в следующий разряд
1	5	6	0	1	1
2	6	4	1	1	1
3	2	3	1	6	0

Сложение в одном разряде двоичной системы счисления можно представить табл. 4.1 (где X_i, Y_i – значения слагаемых в i-м разряде; C_{i-1} – перенос из предыдущего разряда; S_i – сумма в i-м разряде; C_i – перенос в следующий разряд).

Таблица 4.1

Сложение двух двоичных разрядов

X_i	Y_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Рассмотрим на примере последовательный способ сложения чисел X и Y в пятиразрядном цифровом процессоре (какой код используется, не имеет значения, поскольку числа положительные).

$X=7_{(10)}=111_{(2)}$;

$Y=2_{(10)}=10_{(2)}$.

X=

0	0	1	1	1

Y=

0	0	0	1	0

Такты	X_i	Y_i	C_{i-1}	S_i	C_i
1	1	0	0	1	0
2	1	1	0	0	1
3	1	0	1	0	1
4	0	0	1	1	0
5	0	0	0	0	0

В результате сложения получена сумма, равная

S=

0	1	0	0	1

Проверим результат: $7_{(10)}+2_{(10)}=9_{(10)}=1001_{(2)}$.

Далее в пособии при изучении алгоритмов арифметических операций в цифровых процессорах будут рассматриваться и соответствующие им структуры операционных автоматов (ОА). Операционный автомат – это часть цифрового устройства, в которой выполняются действия над операндами, находящимися в регистрах. Вначале рассмотрим общую структуру цифрового устройства (ЦУ), составной частью которого является ОА (рис. 4.1).

На рисунке приняты следующие сокращения:

КОП – код операции, которую выполняет ЦУ по сигналу «старт»;

ОП – оперативная память;

ОА – операционный автомат;

УА – управляющий автомат, обеспечивающий выполнение цифровым устройством алгоритма заданного КОП;

{Y} – управляющие сигналы от УА, соответствующие возможным микрооперациям в ОА;

{X} – осведомительные сигналы (флаги) от ОА, передаваемые в УА, для управления ходом алгоритма, значение которых определяется результатом выполненной микрооперации.

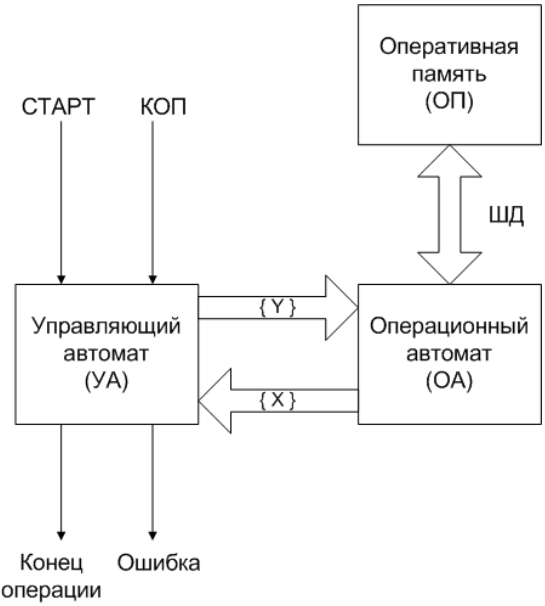

Рис. 4.1. Общая структурная схема цифрового устройства

В состав ОА входят регистры, триггеры, комбинационные схемы, связанные между собой и УА таким образом, чтобы можно было выполнить определенный алгоритм (алгоритмы). УА и ОА вместе образуют простой цифровой процессор. Если этот цифровой процессор универсальный и предназначен для выполнения операций сложения, вычитания, умножения, деления, то КОП определяет, какую именно операцию будет выполнять процессор при подаче сигнала «старт».

Структура операционного аппарата для выполнения алгоритма сложения последовательным способом приведена на рис. 4.2.

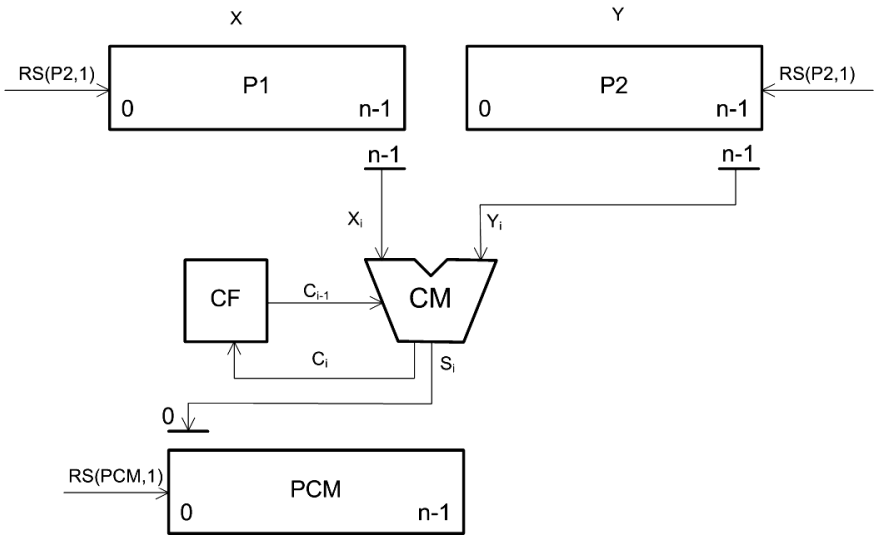

Рис. 4.2. Функциональная схема ОА для операции сложения последовательным способом

На этой функциональной схеме (и всех последующих) прямоугольниками обозначены регистры (устройства для хранения информации), трапецией – комбинационные схемы (устройства для преобразования информации без ее запоминания). Назначение регистров и комбинационных схем на рис. 4.2 таково:

P1 – регистр для хранения операнда X;

P2 – регистр для хранения операнда Y;

PCM – регистр для хранения суммы;

CM – одноразрядный сумматор (работает в соответствии с табл. 4.1);

CF – флаг (триггер) переноса;

RS – сигнал сдвига вправо регистра на один разряд;

0 … (n–1) – разрядность регистра (процессора).

При параллельном способе сложение операндов выполняется за один такт работы процессора, так как используется многоразрядный сумматор (число разрядов сумматора равно разрядности процессора). Схема ОА для алгоритма сложения параллельным способом представлена на рис. 4.3.

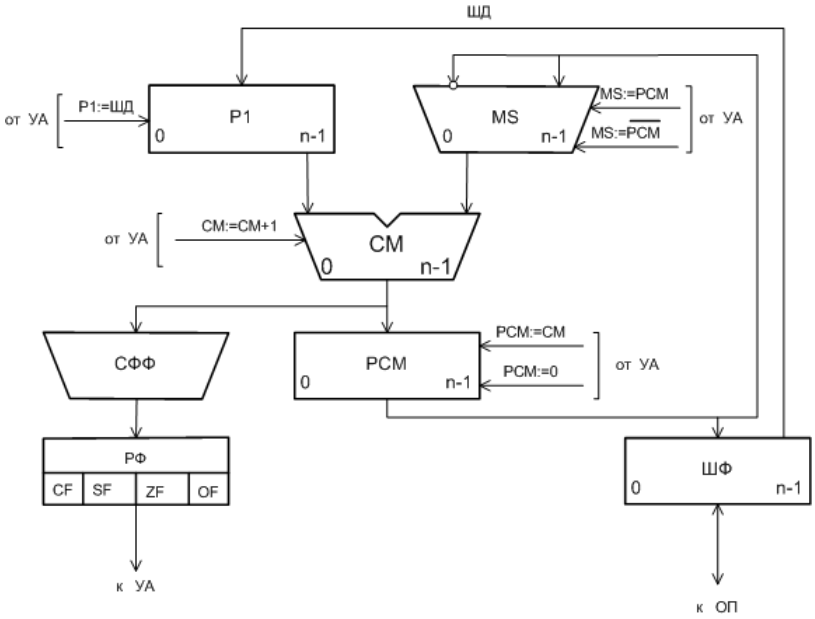

Рис. 4.3. Функциональная схема ОА для операции сложения параллельным способом

P1 – регистр для хранения операнда X;

PCM – регистр для хранения операнда Y, а затем для хранения полученной суммы S;

0…(n–1) – разрядность регистра (процессора);

MS – мультиплексор (это схема, имеющая управляющие входы, несколько информационных входов и только один информационный выход. В зависимости от значения управляющих входов в каждый момент времени только один информационный вход подключается к выходу);

CM – n-разрядный сумматор, который может работать или в дополнительном, или в обратном коде;

СФФ – схема формирования флагов;

РФ – четырехбитный регистр флагов (содержит 4 бит);

CF – флаг переноса (carry flag);

ZF – флаг равенства 0 результата операции (zero flag). Если результат операции равен 0, то ZF = 1, в противном случае – 0;

SF – флаг знака (sign flag). В него записывается значение знакового разряда результата;

OF – знак переполнения (overflow flag). Если полученный результат не помещается в разрядной сетке процессора, OF=1, в противном случае – 0;

ШФ – шинный формирователь позволяет управлять шиной данных либо на прием, либо на передачу.

Горизонтальная стрелка, подходящая к узлу, и надпись над ней, обозначают микрооперацию, которая может быть выполнена в этом узле.

Так как в этом конкретном ОА для ввода операнда предусмотрена только одна шина данных, то для того чтобы сложить операнды X и Y, нужно последовательно, друг за другом, выполнить два следующих алгоритма: «посылка X» и «сложение Y».

Алгоритмы этих операций представлены на рис. 4.4 и 4.5.

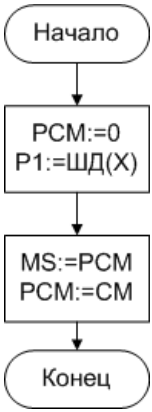

Рис. 4.4. Алгоритм операции «посылка X»

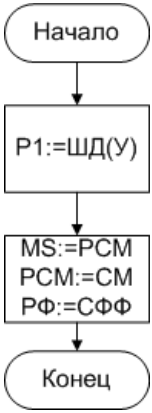

Рис. 4.5. Алгоритм операции «сложение Y»

4.1.1. Алгоритм сложения операндов в дополнительном коде

Так как в данном коде знаковый и числовой разряды рассматриваются как единое целое, то сумматор дополнительного кода выполняет над ними одинаковые действия.

Покажем примеры сложения S=X+Y в разных форматах для восьмиразрядного процессора. Рассмотрим сложение для формата целых чисел с ФТ.

Пример 1. $X=53_{(8)}=101011_{(2)}$; $Y=35_{(8)}=11101_{(2)}$.

$[X]_2$		0	0	1	0	1	0	1	1
$[Y]_2$	+	0	0	0	1	1	1	0	1
$[S]_2$	=	0	1	0	0	1	0	0	0

SF=0; CF=0; ZF=0; OF=0.

Переведем полученный результат в восьмеричную систему счисления. Получаем $S=110_{(8)}$. Выполним проверку сложения через восьмеричную систему:

X			5	3
Y		+	3	5
S	=	1	1	0

Результат верен.

Пример 2. $X=53_{(8)}=101011_{(2)}$; $Y=-35_{(8)}=-11101_{(2)}$

$[X]_2$		0	0	1	0	1	0	1	1
$[Y]_2$	+	1	1	1	0	0	0	1	1
$[S]_2$	=	0	0	0	0	1	1	1	0

SF=0; CF=1; ZF=0; OF=0.

Переведем полученный результат в восьмеричную систему счисления. Получаем $S=16_{(8)}$. Выполним проверку сложения через восьмеричную систему:

X		5	3
Y	–	3	5
S	=	1	6

Результат верен.

Пример 3. $X = -53_{(8)} = -101011_{(2)}$; $Y = 35_{(8)} = 11101_{(2)}$.

$[X]_2$		1	1	0	1	0	1	0	1
$[Y]_2$	+	0	0	0	1	1	1	0	1
$[S]_2$	=	1	1	1	1	0	0	1	0

SF=1; CF=0; ZF=0; OF=0.

Переведем полученный результат в восьмеричную систему счисления. Так как результат сложения имеет отрицательный знак, то вначале нужно перевести его из дополнительного кода в двоичную систему счисления. Получаем $S = -0001110_{(2)} = -1110_{(2)} = -16_{(8)}$. Выполним проверку сложения через восьмеричную систему:

X			5	3
		–		
Y		+	3	5
S	=		1	6
		–		

Результат верен.

Рассмотрим сложение для формата дробных чисел с ФТ.

Пример 4. $X = -0{,}53_{(8)} = -0{,}101011_{(2)}$; $Y = 0{,}35_{(8)} = 0{,}11101_{(2)}$.

$[X]_2$		1	0	1	0	1	0	1	0
$[Y]_2$	+	0	0	1	1	1	0	1	0
$[S]_2$	=	1	1	1	0	0	1	0	0

SF=1; CF=0; ZF=0; OF=0.

Переведем полученный результат в восьмеричную систему счисления. Так как результат сложения имеет отрицательный знак, то вначале нужно перевести его из дополнительного кода в двоичную систему счисления. Получим $S = -0{,}0011100_{(2)} = -0{,}001110_{(2)} = -0{,}16_{(8)}$. Выполним проверку сложения через восьмеричную систему:

X		–	0,	5	3
Y		+	0,	3	5
S	=	–	0	1	6

Результат верен.

Пример 5. $X=0,53_{(8)}=0,101011_{(2)}$; $Y=0,35_{(8)}=0,11101_{(2)}$.

$[X]_2$	0	1	0	1	0	1	1	0
$[Y]_2$ +	0	0	1	1	1	0	1	0
$[S]_2$ =	1	0	0	1	0	0	0	0

SF=1; CF=0; ZF=0; OF=1.

Результат сложения двух положительных чисел имеет отрицательный знак. Это произошло потому, что сумма не поместилась в 7 бит и заняла еще один (знаковый) бит. Такой случай называется «исключительным». Поскольку при выполнении арифметических операций могут быть и другие «исключительные» случаи, например деление на 0, то каждый такой случай получил свое название. Этот называется «переполнение». В случае переполнения разрядной сетки регистр РСМ содержит неправильный результат сложения. **Результатом сложения будет значение OF=1**. Более подробно о переполнении будет сказано далее.

Из рассмотренных примеров видно, что алгоритм сложения дробных чисел в формате с ФТ ничем не отличается от алгоритма сложения целых чисел в формате с ФТ.

Рассмотрим сложение для формата смешанных чисел с ФТ в шестнадцатиразрядном процессоре (8 бит – целая часть и знак, 8 бит – дробная).

Пример 6. $X=-53,15_{(8)}=-101011,001101_{(2)}$; $Y=35,11_{(8)}=11101,001001_{(2)}$.

$[X]_2$	1	1	0	1	0	1	0	1	1	1	0	0	1	1	0	0
$[Y]_2$ +	0	0	0	1	1	1	0	1	0	0	1	0	0	1	0	0
$[S]_2$ =	1	1	1	1	0	0	1	0	1	1	1	1	0	0	0	0

SF=1; CF=0; ZF=0; OF=0.

Переведем полученный результат в восьмеричную систему счисления. Так как результат сложения имеет отрицательный знак, то вначале нужно перевести его из дополнительного кода в двоичную систему счисления. Получаем $S=-0001110,00010000_{(2)}=-1110,000100_{(2)}=-16,04_{(8)}$. Выполним проверку сложения через восьмеричную систему:

X		–	5	3	,	1	5
Y		+	3	5	,	1	1
S	=	–	1	6	,	0	4

Результат верен.

Таким образом, видно, что сложение чисел в любом формате с ФТ происходит по одному и тому же алгоритму.

4.1.2. Алгоритм сложения операндов в обратном коде

Отличие алгоритма сложения в обратном коде от алгоритма сложения в дополнительном коде состоит в том, что сложение выполняется в два этапа. На первом этапе складываются операнды, а на втором к полученной сумме прибавляется значение флага CF, полученного при сложении операндов.

Так же как и в дополнительном коде, алгоритм сложения в обратном коде одинаков для всех форматов с ФТ. Рассмотрим пример сложения целых чисел в формате с ФТ на восьмиразрядном процессоре.

Пример. $X=53_{(8)}=101011_{(2)}$; $Y=-35_{(8)}=-11101_{(2)}$.

1-й этап:

$[X]_3$		0	0	1	0	1	0	1	1
$[Y]_3$	+	1	1	1	0	0	0	1	0
$[S]_3$	=	0	0	0	0	1	1	0	1

CF=1;

2-й этап:

$[S]_3$		0	0	0	0	1	1	0	1
CF	+								1
		0	0	0	0	1	1	1	0

SF=0; CF=0; ZF=0; OF=0.

Переведем полученный результат в восьмеричную систему счисления. Получаем $S=0001110_{(2)}=1110_{(2)}=16_{(8)}$. Выполним проверку сложения через восьмеричную систему:

X		5	3
Y	–	3	5
S	=	1	6

Результат верен.

4.1.3. «Исключительные» случаи при выполнении операции сложения (вычитания)

В цифровом процессоре при сложении или вычитании (алгоритмы вычитания будут рассмотрены далее) возможны случаи, когда полученный результат не помещается в разрядную сетку процессора. В одном из ранее рассмотренных примеров при сложении возникло переполнение разрядной сетки. Такой случай называется «исключительным», так как алгоритм сложения (вычитания) не может быть завершен с получением правильного результата. В алгоритме сложения или вычитания предусматривается ветвь для обработки этой ситуации. «Исключительный» случай переполнения разрядной сетки получил название – «переполнение». Переполнение может быть обнаружено или аппаратными, или программными средствами по одному из следующих способов.

Первый способ основан на том, что если знаки исходных слагаемых одинаковы, а знак суммы противоположен знаку слагаемых – это переполнение. Рассмотрим пример для восьмиразрядного процессора, работающего в дополнительном коде с целыми числами в формате с ФТ.

$X=53_{(8)}=101011_{(2)}$; $Y=161_{(8)}=1110001_{(2)}$.

$[X]_2$		0	0	1	0	1	0	1	1
$[Y]_2$	+	0	1	1	1	0	0	0	1
$[S]_2$	=	1	0	0	0	1	1	0	0

SF=1; CF=0; ZF=0; OF=1.

Флаг OF=1, так как знаки исходных слагаемых положительные, а знак суммы – отрицательный. В этом случае результатом операции сложения будет значение флага OF=1, а не значение регистра РСМ (там неверный результат).

Недостатком этого способа является необходимость хранения знаков слагаемых операндов до момента получения суммы, что требует дополнительных аппаратных или программных затрат.

Второй способ основан на использовании модифицированного кода. Модифицированным называется такой код, когда под знак отводится 2 бит. Переполнение обнаруживается, если после сложения сумма име-

ет неодинаковые значения знаковых битов. Причем по их значению можно определить, в какую область произошло переполнение:

00 – нет переполнения, сумма положительная;

01 – переполнение в область положительных чисел;

10 – переполнение в область отрицательных чисел;

11 – нет переполнения, сумма отрицательная.

Рассмотрим пример для восьмиразрядного процессора, работающего в дополнительном коде с целыми числами в формате с ФТ.

$X=53_{(8)}=101011_{(2)}$; $Y=61_{(8)}=110001_{(2)}$.

$[X]_2$		0	0	1	0	1	0	1	1
$[Y]_2$	+	0	0	1	1	0	0	0	1
$[S]_2$	=	0	1	0	0	1	1	0	0

SF=0; CF=0; ZF=0; OF=1.

Флаг OF=1, так как знаковые биты суммы равны 01 (переполнение в область положительных чисел). В этом случае результатом операции сложения будет значение флага OF=1, а не значение регистра РСМ (там неверный результат).

Недостатком этого способа является сокращение в два раза диапазона представления чисел в процессоре. Например, операнд $Y=161_{(8)}$ уже не поместится в разрядную сетку этого процессора, если используется модифицированный код.

Третий способ основан на сравнении значений переносов из старшего числового бита в знаковый и из знакового бита – во флаг CF. Если значение переносов равны – переполнения нет, если не равны – есть переполнение. Также по их значению можно определить, в какую область произошло переполнение:

00 – нет переполнения, сумма положительная;

01 – переполнение в область положительных чисел;

10 – переполнение в область отрицательных чисел;

11 – нет переполнения, сумма отрицательная.

Рассмотрим пример для восьмиразрядного процессора, работающего в дополнительном коде с целыми числами в формате с ФТ (строка «С» таблицы показывает значение переносов).

$X=53_{(8)}=101011_{(2)}$; $Y=161_{(8)}=1110001_{(2)}$.

$[X]_2$		0	0	1	0	1	0	1	1
$[Y]_2$	+	0	1	1	1	0	0	0	1
C		0	1	1	0	0	0	1	1
$[S]_2$	=	1	0	0	1	1	1	0	0

SF=1; CF=0; ZF=0; OF=1.

Флаг OF=1, так как значение переноса из старшего числового бита равно 1, а из знакового бита равно 0 (переполнение в область положительных чисел). В этом случае результатом операции сложения будет значение флага OF=1, а не значение регистра РСМ (там неверный результат).

Недостаток этого способа в том, что при его аппаратной реализации необходимо обеспечить доступ к цепям переноса в сумматоре (это не всегда возможно), а при программной – получить значения этих переносов возможно только с очень большими затратами.

4.1.4. Алгоритм вычитания операндов

В настоящее время в цифровых процессорах используется как сумматор, так и вычитатель. Вычитатель – это комбинационная схема на два входа и один выход, во многом аналогичная сумматору. На один вход подается уменьшаемое, на другой – вычитаемое, а на выходе получается разность. Но в простых цифровых процессорах нет вычитателя, и для операции вычитания используется сумматор.

В этом случае операция вычитания X – Y выполняется на сумматоре как X+(–Y). Поэтому в ОА для сложения параллельным способом, который представлен ранее на рис. 4.3, есть две микрооперации: инверсия РСМ и прибавление 1 в младший разряд сумматора:

РСМ:= РСМ;

СМ:= СМ+1.

Вместе эти две микрооперации позволяют получить на входе сумматора СМ значение операнда, находящегося в РСМ в дополнительном коде со знаком «минус», а первая из них – значение операнда, находящегося в РСМ в обратном коде, со знаком «минус».

Рассмотрим пример вычитания X – Y для восьмиразрядного процессора, работающего в дополнительном коде, с целыми числами в формате с ФТ, не имеющего вычитателя. Это вычисление будет выполнено как X+(–Y).

$X=53_{(8)}=101011_{(2)}$; $Y=35_{(8)}=11101_{(2)}$; $-Y=-35_{(8)}=-11101_{(2)}$.

$[X]_2$		0	0	1	0	1	0	1	1
$[-Y]_2$	+	1	1	1	0	0	0	1	1
$[S]_2$	=	0	0	0	0	1	1	1	0

SF=0; CF=1; ZF=0; OF=0.

Переведем полученный результат в восьмеричную систему счисления. Получаем $S=16_{(8)}$. Выполним проверку вычитания через восьмеричную систему:

X		5	3
Y	–	3	5
S	=	1	6

Результат верен.

В дальнейшем в пособии будут рассматриваться только цифровые процессоры, не имеющие вычитателя.

4.1.5. Алгоритмы сложения и вычитания целых чисел без знака (беззнаковая или модульная арифметика)

Такая арифметика используется в цифровом процессоре для целых чисел в формате с ФТ, если все биты регистра рассматриваются как числовые (нет знакового бита). Она используется не для проведения арифметических расчетов, а для специальных арифметических вычислений. Например, вычисления смещения в командах условной и безусловной передачи управления на языках Assembler или C++. Название «модульная» она получила потому, что сложение и вычитание выполняются по модулю, определяемому разрядной сеткой процессора. В модульной арифметике не используется обнаружение переполнения.

Рассмотрим примеры для восьмиразрядного процессора, работающего в дополнительном коде с целыми числами без знака в формате с ФТ. Модуль, по которому работает восьмиразрядный процессор, равен $400_{(8)}$.

Пример 1. X+Y,

где $X=62_{(8)}=110010_{(2)}$; $Y=225_{(8)}=10010101_{(2)}$.

$[X]_2$		0	0	1	1	0	0	1	0

$[Y]_2$	+	1	0	0	1	0	1	0	1
$[S]_2$	=	1	1	0	0	0	1	1	1

SF=*; CF=0; ZF=0; OF=0.

Переведем полученный результат в восьмеричную систему счисления. Получаем $S=307_{(8)}$. Выполним проверку сложения через восьмеричную систему:

X			6	2
Y	+	2	2	5
S	=	3	0	7

$Mod_{400}(S)=Mod_{400}(307)=307$.

Результат верен.

Пример 2. X+Y,

где $X=262_{(8)}=10110010_{(2)}$; $Y=225_{(8)}=10010101_{(2)}$.

$[X]_2$		1	0	1	1	0	0	1	0
$[Y]_2$	+	1	0	0	1	0	1	0	1
$[S]_2$	=	0	1	0	0	0	1	1	1

SF=*; CF=1; ZF=0; OF=0.

Переведем полученный результат в восьмеричную систему счисления. Получаем $S=107_{(8)}$. Выполним проверку сложения через восьмеричную систему:

X		2	6	2
Y	+	2	2	5
S	=	5	0	7

$Mod_{400}(S)=Mod_{400}(507)=107$.

Результат верен.

Пример 3. X – Y,

где $X=62_{(8)}=110010_{(2)}$; $Y=225_{(8)}=10010101_{(2)}$; $-Y=-225_{(8)}=01101011_{(2)}$.

$[X]_2$		0	0	1	1	0	0	1	0
$[-Y]_2$	+	0	1	1	0	1	0	1	1
$[S]_2$	=	1	0	0	1	1	1	0	1

SF=*; CF=0; ZF=0; OF=0.

Переведем полученный результат в восьмеричную систему счисления. Зная, что значение разности отрицательно и в дополнительном коде, вначале переведем его в двоичную систему счисления. Получим $01100011_{(2)}=1100011_{(2)}$, $S=143_{(8)}$. Выполним проверку вычитания через восьмеричную систему:

X		2	2	5
Y	–		6	2
S	=	1	4	3

$Mod_{400}(S)=Mod_{400}(143)=143$.

Результат верен.

4.2. Алгоритмы операций сдвига в цифровых процессорах

Операция сдвига – это одновременное перемещение значений битов операнда в регистре процессора на фиксированное количество разрядов влево или вправо.

Различают три типа сдвига:

- логический;
- циклический;
- арифметический.

Рассмотрим алгоритмы выполнения каждого из них.

4.2.1. Логический сдвиг

При логическом сдвиге все биты регистра перемещаются влево или вправо на константу сдвига с заполнением освобождающихся битов нулями. Примем следующие обозначения: LS – левый сдвиг (left shift); RS – правый сдвиг (right shift).

Операция логического сдвига будет обозначаться:

<логический сдвиг>(<регистр>,<константа сдвига>).

Например, логический сдвиг влево на 3 бит регистра PCM будет записан как LS(PCM,3). Рассмотрим примеры выполнения арифметического сдвига. Пусть регистр P восьмиразрядного процессора хранит операнд 01110111.

Пример 1.

Регистр P до сдвига	0	1	1	1	0	1	1	1
Регистр P после сдвига RS(P,2)	0	0	0	1	1	1	0	1

CF=; SF=0; ZF=0; OF=0.

Пример 2.

Регистр P до сдвига	0	1	1	1	0	1	1	1
Регистр P после сдвига LS(P,1)	1	1	1	0	1	1	1	0

CF=0; SF=1; ZF=0; OF=0.

Алгоритм выполнения логического сдвига не зависит от кода, в котором работает процессор.

4.2.2. Циклический сдвиг

Циклический сдвиг выполняется таким образом, что значения выдвигаемых битов записываются на место освобождающихся (как бы по циклу). В эту цепь сдвига может включаться флаг переноса, а может и не включаться (зависит от процессора). Примем следующие обозначения: LC – левый циклический сдвиг (left cycle); RC – правый циклический сдвиг (right cycle).

Операция циклического сдвига будет обозначаться:

<циклический сдвиг>(<регистр>,<константа сдвига>).

Например, циклический сдвиг влево на 2 бит регистра PCM будет записан как LC(PCM,2). Рассмотрим примеры выполнения циклического сдвига. Пусть регистр P восьмиразрядного процессора хранит операнд 01110111.

Пример 1 (в цепь сдвига не включается флаг CF).

Регистр P до сдвига	0	1	1	1	0	1	1	1
Регистр P после сдвига LC(P,2)	1	1	0	1	1	1	0	1

CF=<без изменения>; SF=1; ZF=0; OF=0.

Пример 2 (в цепь сдвига не включается флаг CF).

Регистр P до сдвига	0	1	1	1	0	1	1	1
Регистр P после сдвига RC(P,1)	1	0	1	1	1	0	1	1

CF=<без изменения>; SF=1; ZF=0; OF=0.

Пример 3 (в цепь сдвига включается флаг CF, значение которого до сдвига равно 1).

Регистр P до сдвига	0	1	1	1	0	1	1	1
Регистр P после сдвига LC(P,1)	1	1	1	0	1	1	1	1

CF=0; SF=1; ZF=0; OF=0.

Пример 4 (в цепь сдвига включается флаг CF, значение которого до сдвига равно 0).

Регистр P до сдвига	0	1	1	1	0	1	1	1
Регистр P после сдвига RC(P,1)	0	0	1	1	1	0	1	1

CF=1; SF=0; ZF=0; OF=0.

Алгоритм выполнения циклического сдвига не зависит от кода, в котором работает процессор.

4.2.3. Арифметический сдвиг

Арифметический сдвиг выполняется с учетом знака операнда, поэтому алгоритм его выполнения зависит от кода, в котором работает процессор. Арифметический сдвиг операнда влево на 1 бит эквивалентен увеличению его в два раза, а арифметический сдвиг вправо на 1 бит эквивалентен делению операнда на 2. Примем следующие обозначения: AL – левый арифметический сдвиг (arithmetical left); AR – правый арифметический сдвиг (arithmetical right).

Операция арифметического сдвига будет обозначаться:

<арифметический сдвиг>(<регистр>,<константа сдвига>).

4.2.3.1. Алгоритм арифметического сдвига в прямом коде

В прямом коде сдвигается только числовая часть операнда, знак остается на месте. При этом освобождающиеся биты слева или справа (в зависимости от направления сдвига) заполняются нулями. При сдвиге влево делается анализ на переполнение разрядной сетки процессора. Переполнение возникает, если значение старшего сдвигаемого числового бита равно 1.

Рассмотрим примеры выполнения арифметического сдвига в прямом коде.

Пример 1. Пусть регистр P восьмиразрядного процессора хранит операнд 01110111.

Регистр P до сдвига	0	1	1	1	0	1	1	1

Регистр P после сдвига AL(P,1)	0	1	1	0	1	1	1	0

CF=<без изменения>; SF=0; ZF=0; **OF=1**, т.е. возникло переполнение (старший сдвигаемый числовой бит равен 1).

Пример 2. Пусть регистр P восьмиразрядного процессора хранит операнд 00000111.

Регистр P до сдвига	0	0	0	0	0	1	1	1
Регистр P после сдвига AR(P,1)	0	0	0	0	0	0	1	1

CF=1; SF=0; ZF=0; OF=0.

До сдвига $P=7_{(8)}$, а после сдвига $P=3_{(8)}$ (результат деления $7_{(8)}$ нацело на 2).

Пример 3. Пусть регистр P восьмиразрядного процессора хранит операнд 10000111.

Регистр P до сдвига	1	0	0	0	0	1	1	1
Регистр P после сдвига AL(P,1)	1	0	0	0	1	1	1	0

CF=1; SF=1; ZF=0; OF=0.

До сдвига $P= -7_{(8)}$, а после сдвига $P= -16_{(8)}$ (результат умножения $7_{(8)}$ на 2).

4.2.3.2. Алгоритм арифметического сдвига в дополнительном коде

При арифметическом сдвиге в дополнительном коде сдвигаются все биты, включая знаковый бит.

При сдвиге влево освобождающиеся биты заполняются нулями. Возможно возникновение переполнения, которое определяется процессором, одним из тех способов, что и при сложении (вычитании).

При сдвиге вправо знаковый бит остается на месте, и его значение передается в старший числовой бит. Рассмотрим примеры выполнения арифметического сдвига в дополнительном коде.

Пример 1. Пусть регистр P восьмиразрядного процессора хранит операнд 01110111.

Регистр P до сдвига	0	1	1	1	0	1	1	1
Регистр P после сдвига AL(P,1)	1	1	1	0	1	1	1	0

CF=0; SF=1; ZF=0; **OF=1**, т.е. возникло переполнение (есть перенос из старшего числового бита, а из знакового нет).

Пример 2. Пусть регистр P восьмиразрядного процессора хранит операнд 00000111.

Регистр P до сдвига	0	0	0	0	0	1	1	1
Регистр P после сдвига AR(P,1)	0	0	0	0	0	0	1	1

CF=1; SF=0; ZF=0; OF=0.

До сдвига $P=7_{(8)}$, а после сдвига $P=3_{(8)}$ (результат деления нацело $7_{(8)}$ на 2).

Пример 3. Пусть регистр P восьмиразрядного процессора хранит операнд 11100111.

Регистр P до сдвига	1	1	1	0	0	1	1	1
Регистр P после сдвига AL(P,1)	1	1	0	0	1	1	1	0

CF=1; SF=1; ZF=0; OF=0.

До сдвига $P= -31_{(8)}$, а после сдвига $P= -62_{(8)}$ (результат умножения $(-31_{(8)})$ на 2).

Пример 4. Пусть регистр P восьмиразрядного процессора хранит операнд 11100111.

До сдвига	1	1	1	0	0	1	1	1
После сдвига AR(P,1)	1	1	1	1	0	0	1	1

CF=1; SF=1; ZF=0; OF=0.

До сдвига $P= -31_{(8)}$, а после сдвига $P= -15_{(8)}$ (результат деления $(-31_{(8)})$ нацело на 2).

4.2.3.3. Алгоритм арифметического сдвига в обратном коде

Алгоритм арифметического сдвига в обратном коде выполняется так же, как и в дополнительном коде, но при сдвиге влево значение выдвигаемого бита записывается в младший бит регистра.

Пример. Пусть регистр P восьмиразрядного процессора хранит операнд 11100111.

Регистр P до сдвига	1	1	1	0	0	1	1	1
Регистр P после сдвига AL(P,1)	1	1	0	0	1	1	1	1

CF=<без изменения>; SF=1; ZF=0; OF=0.

До сдвига P= – $30_{(8)}$, а после сдвига P= – $60_{(8)}$ (результат умножения (– $30_{(8)}$) на 2).

4.3. Алгоритм операции сложения (вычитания) в формате с плавающей точкой

Рассмотрим общие положения по сложению (вычитанию) операндов с плавающей точкой. Исходя из рассмотренного ранее формата представления чисел с плавающей точкой операнды X и Y представляются как:

$X=M_x*2^{Px}$, где $0,5<=M_x<1$ и $P_{min}<=P_x<=P_{max}$;

$Y=M_y*2^{Py}$, где $0,5<=M_y<1$ и $P_{min}<=P_y<=P_{max}$.

Напомним, что только при записи чисел в память в формате с плавающей точкой мантиссы хранятся в диапазоне $1<=M<2$, а при извлечении из памяти в ОА скрытый бит восстанавливается и $0,5<=M<1$.

Полученная сумма Z=X+Y также должна быть представлена как

$Z=M_z*2^{Pz}$, где $0,5<=M_z<1$ и $P_{min}<=P_z<=P_{max}$.

Для стандарта «короткое вещественное»:

$P_{max}=127_{(10)}$;

$P_{min}= – 126_{(10)}$;

$P=128_{(10)}$ – «превышение порядка» (больше максимально допустимого);

P= – $127_{(10)}$ – «потеря порядка» (меньше минимально допустимого).

Арифметические операции над числами, представленными в экспоненциальной форме, выполняются раздельно над мантиссами и порядками.

Так как можно складывать только разряды чисел, имеющих одинаковый вес, то для сложения (вычитания) мантисс нужно, чтобы их порядки были одинаковы. Это достигается тем, что процессор перед сложением (вычитанием) выравнивает порядки. Порядки обязательно выравниваются до большего порядка. Если выравнивать порядки до меньшего, то может возникнуть переполнение, т.е. потеря значимых разрядов. При выравнивании порядков процессор действует по следующему алгоритму:

1) из порядка первого операнда вычитается порядок второго;

2) если разность равна 0, то порядки равны и выполняется сложение мантисс;

3) если разность > 0, то нужно выравнять порядок второго операнда до порядка первого, а если < 0, то порядок первого операнда до порядка второго.

Выравнивание порядков производится следующим образом. В счетчик заносится разность порядков. Мантисса соответствующего операнда арифметически сдвигается вправо на один разряд и из счетчика вычитается единица. Это повторяется в цикле до тех пор, пока счетчик не станет равным 0.

После сложения мантисс процессор проверяет результат на нарушение нормализации:

1) $0{,}5<M_z<1$ – нарушения нормализации нет;

2) $M_z \geq 1$ – нарушение нормализации влево, т.е. мантисса не поместилась в разрядную сетку (способы обнаружения нарушения нормализации влево такие же, как и способы обнаружения переполнения при сложении чисел с ФТ). В этом случае выполняется операция нормализации вправо – мантисса суммы арифметически сдвигается вправо на один разряд, а порядок увеличивается на 1. При выполнении операции нормализации вправо процессор после сдвига проверяет, не превышает ли порядок суммы максимально допустимый. Если такая ситуация возникла, процессор фиксирует переполнение при сложении чисел с плавающей точкой;

3) $M_z<0{,}5$ – нарушение нормализации вправо, т.е. старший разряд мантиссы равен 0. В этом случае процессор в цикле применяет операцию нормализации влево, а именно: мантисса суммы арифметически сдвигается влево, порядок уменьшается на 1, до тех пор, пока не выполнится условие $0{,}5 \leq M_z<1$. При этом процессор проверяет ситуацию, не оказался ли $P_z<P_{min}$. Если получается, что $P_z<P_{min}$, то возникает исключительная ситуация – «потеря значимости». В этом случае в некоторых процессорах результат принимается равным 0, а в других возникает исключительный случай «потеря значимости».

Рассмотрим детально алгоритм сложения (вычитания) с плавающей точкой в формате КВ, входными данными для которого являются операнды X и Y, хранящиеся в оперативной памяти в формате КВ.

1. Извлекаются операнды из ОП и размещаются в процессоре с плавающей точкой (из-за сложной структурной схемы такого процессора она здесь не приводится – будет рассмотрена в других дисциплинах). Порядок и мантисса каждого операнда записываются в отдельные регистры (регистр порядка, регистр мантисс). При извлечении операнда из ОП восстанавливается скрытый бит мантиссы, поэтому разрядность регистра мантиссы составляет 25 бит (а не 24 бит).

2. С использованием арифметики целых чисел с ФТ определяется разность порядков первого и второго операндов ($P_x - P_y$).

3. С учетом разности порядков выполняется денормализация одного из операндов так, чтобы порядки чисел были равны (выравнивание порядков).

4. Переводятся мантиссы из прямого кода в дополнительный код. С использованием арифметики дробных чисел с ФТ выполняется сложение мантисс.

5. Определяется нарушение нормализации влево или вправо. Если нарушение есть – выполняется соответствующая операция нормализации. Проверяется превышение максимально допустимого порядка, и если оно есть, то фиксируется переполнение. Проверяется достижение порядком величины меньше минимально допустимой, и если оно есть, то фиксируется потеря значимости. Мантисса суммы переводится в прямой код.

6. Полученный результат записывается в ОП в формате КВ (со скрытием бита мантиссы и объединением содержимого регистра порядка и регистра мантиссы).

Рассмотрим примеры выполнения такого алгоритма. Выполним операции сложения и вычитания над операндами X и Y.

Пример 1. X+Y, где $X=24_{(8)}$, $Y=-65_{(8)}$. Представим эти операнды в формате КВ, в том виде, в котором они хранятся в ОП.

$X=24_{(8)}=10100_{(2)}=1{,}0100_{(2)}*2^{100}$. Тогда с учетом скрытия старшего бита мантиссы получаем:

X=

0	1	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
байт								**байт**								**байт**								**байт**							

$Y=-65_{(8)}=-110101_{(2)}=-1{,}10101_{(2)}*2^{101}$. Тогда с учетом скрытия старшего бита мантиссы получаем:

Y=	1	1	0	0	0	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	байт								**байт**								**байт**								**байт**							

Рассмотрим выполнение алгоритма сложения.

Шаг 1. Операнды извлекаются из ОП (с восстановлением скрытого бита) и помещаются в ОА в регистры мантисс и порядков:

РП$_x$=

1	**0**	**0**	**0**	**0**	**1**	**0**	**0**

РМ$_x$=

0	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

РП$_y$=

1	**0**	**0**	**0**	**0**	**1**	**0**	**1**

РМ$_y$=

1	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Шаг 2. Определяется разность порядков $[РП_x+(-РП_y)]_2$:

$РП_x$=	1	0	0	0	0	1	0	0
$-РП_y$=	0	1	1	1	1	0	1	1
$РП_x+(-РП_y)$=	1	1	1	1	1	1	1	1

Так как разность отрицательная, нужно выравнять порядок первого операнда до порядка второго. Разность записывается в счетчик разности порядков:

СЧ=

1	1	1	1	1	1	1	1

Шаг 3. Денормализация первого операнда.

AR(РМ$_x$,1)=

0	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

После сдвига значение счетчика разности порядков увеличивается на 1 (так как разность отрицательная, в противном случае его значение уменьшалось бы на 1).

СЧ	1	1	1	1	1	1	1	1
+1	0	0	0	0	0	0	0	1
СЧ=	0	0	0	0	0	0	0	0

Так как счетчик равен 0, порядки операндов выравнены.

Шаг 4. Перевод мантисс в дополнительный код и их сложение:

$[PM_x]_2=$	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$[PM_y]_2=$	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$[PM_z]_2=$	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SF=1; OF=0

Шаг 5. Так как старший числовой разряд мантиссы значащий – нет нарушения нормализации вправо. Перевод мантиссы суммы M_z в прямой код:

$[PM_z]_1=$	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Шаг 6. Порядок результата $РП_z=РП_y$ (так как до него происходило выравнивание). Далее объединяются порядок и мантисса суммы (со скрытием старшего числового бита) в формат КВ и записываются в ОП.

Z=	1	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	байт								**байт**								**байт**								**байт**					

Переводим результат из формата КВ в восьмеричную систему счисления. Получаем:

$Z= – 1{,}00001*2^5= – 100001_{(2)}= – 41_{(8)}$.

Проверяем результат:

Y			6	5
		–		
X	+		2	4
Z	=	–	4	1

Результат верен.

Пример 2. X – Y с теми же операндами. Так как при выполнении этой операции шаги с первого по третий будут теми же, что и при сложении, покажем операцию вычитания, начиная с шага 4.

Шаг 4. Перевод мантисс в дополнительный код и вычитание (через сложение):

$[PM_x]_2=$	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$[– PM_y]_2=$	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$[PM_z]_2$=

1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CF=0; SF=1; OF=1.

Шаг 5. Так как возникло нарушение нормализации влево, выполняется операция нормализации вправо (при OF=1 арифметический сдвиг вправо в дополнительном коде выполняется с вдвиганием в знаковый разряд флага CF).

AR(PM_z,1)=

0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Порядок суммы $РП_z$=$РП_y$ увеличивается на 1.

$РП_z$=	1	0	0	0	0	1	0	1
+	0	0	0	0	0	0	0	1
$РП_z$=	1	0	0	0	0	1	1	0

Так как старший числовой разряд мантиссы стал равен 1, нарушение нормализации устранено. Далее следует перевод мантиссы суммы M_z в прямой код:

$[PM_z]_1$=

0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Шаг 6. Объединяются порядок и мантисса разности (со скрытием старшего числового бита) в формат КВ и записываются в ОП.

Z=

0	1	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
байт								**байт**								**байт**								**байт**							

Переведем результат из формата КВ в восьмеричную систему счисления. Получаем:

$Z=1{,}001001*2^6=1001001_{(2)}=111_{(8)}$.

Проверяем результат:

X	+		2	4
Y			6	5
Z	=	1	1	1

Результат верен.

4.4. Поразрядные логические операции в цифровых процессорах

Прежде чем изучать алгоритмы так называемых «длинных» операций – умножения и деления – рассмотрим четыре «поразрядные логические операции», которые являются составной частью этих алгоритмов. Одноместная (унарная) операция – «**инверсия**». В такой операции над каждым битом операнда выполняется булева функция отрицания.

x	y= ¬x
0	1
1	0

Рассмотрим пример. Пусть содержимое восьмиразрядного регистра P=$37_{(8)}$. Тогда поразрядная операция отрицания P дает следующий результат:

P=	0	0	0	1	1	1	1	1
¬P=	1	1	1	0	0	0	0	0

Эта операция используется для получения значения операнда со знаком «минус». Например, для операнда X:

– X= ¬X+1 в дополнительном коде;

– X= ¬X в обратном коде.

Двухместная (бинарная) операция – **поразрядное сложение «по модулю 2»** или **«исключающее или»**. В такой операции над каждыми битами двух операндов, имеющих одинаковый вес, выполняется булева функция «**исключающее или»**.

x	y	y=x^y
0	0	0
0	1	1
1	0	1
1	1	0

Эта операция используется для сравнения операндов на равенство. Рассмотрим примеры.

Пример 1. Пусть в восьмиразрядном регистре P1 находится операнд $32_{(8)}$, а в регистре P2 – операнд $65_{(8)}$. Тогда поразрядная опера-

ция сложения «по модулю 2» содержимого регистров P1 и P2 дает следующий результат:

P1=	0	0	0	1	1	0	1	0
P2=	0	0	1	1	0	1	0	1
P1^P2	0	0	1	0	1	1	1	1

SF=0; ZF=0. Так как флаг ZF=0, операнды в регистрах не равны.

Пример 2. Пусть в восьмиразрядном регистре P1 находится операнд $65_{(8)}$, а в регистре P2 – операнд $65_{(8)}$. Тогда поразрядная операция сложения «по модулю 2» содержимого регистров P1 и P2 дает следующий результат:

P1=	0	0	1	1	0	1	0	1
P2=	0	0	1	1	0	1	0	1
P1^P2	0	0	0	0	0	0	0	0

SF=0; ZF=1. Так как флаг ZF=1, операнды в регистрах равны.

Двухместная (бинарная) операция – **поразрядная конъюнкция**. В такой операции над каждыми битами двух операндов, имеющих одинаковый вес, выполняется булева функция – **конъюнкция**.

x	y	y=x&y
0	0	0
0	1	0
1	0	0
1	1	1

Эта операция используется для выделения (сохранения) заданных битов операнда, а остальные биты устанавливаются в 0. Рассмотрим пример.

Пусть в восьмиразрядном регистре P1 находится операнд $65_{(8)}$. Нужно выделить левые два и правые три бита операнда. Тогда формируется так называемая маска (M), которая содержит 1 в сохраняемых битах и 0 – в остальных. Поразрядная конъюнкция P1 и M дает следующий результат:

P1=	0	0	1	1	0	1	0	1
M=	1	1	0	0	0	1	1	1

P1&M	0	0	0	0	0	1	0	1

Из полученного результата видно, что сохранены значения выделяемых битов, а остальные установлены в 0.

Двухместная (бинарная) операция – **поразрядная дизъюнкция**. В такой операции над каждыми битами двух операндов, имеющих одинаковый вес, выполняется булева функция **дизъюнкция** («объединяющее или»).

x	y	y=xvy
0	0	0
0	1	1
1	0	1
1	1	1

Эта операция используется для установки определенных битов операнда в 1. Рассмотрим пример.

Пусть в восьмиразрядном регистре P1 находится операнд $65_{(8)}$. Нужно установить в 1 левые два бита операнда. Тогда формируется так называемая маска (M), которая содержит 1 в устанавливаемых разрядах и 0 – в остальных. Поразрядная дизъюнкция P1 и M дает следующий результат:

P1=	0	0	1	1	0	1	0	1
M=	1	1	0	0	0	0	0	0
P1vM	1	1	1	1	0	1	0	1

Из полученного результата видно, что установлены в 1 значения двух левых битов, а остальные биты оставлены без изменения.

4.5. Алгоритмы умножения в цифровых процессорах

Напомним алгоритмы умножения в десятичной системе счисления. Вначале рассмотрим алгоритм умножения с младших разрядов множителя. Пусть надо умножить $121_{(10)}$ на $131_{(10)}$.

	1	2	1	множимое
*	1	3	1	множитель
	—	—	—	
	1	2	1	умножение на младший разряд

	+	3	6	3		умножение на средний разряд
		—	—	—	—	суммирование
		3	7	5	1	частичное произведение
	+ 1	2	1			умножение на старший разряд
	—	—	—	—	—	суммирование
	1	5	8	5	1	произведение

Теперь рассмотрим алгоритм умножения, начиная со старших разрядов множителя. Пусть надо умножить $121_{(10)}$ на $131_{(10)}$.

	1	2	1			множимое
*	1	3	1			множитель
	—	—	—			
	1	2	1			умножение на старший разряд
+		3	6	3		умножение на средний разряд
	—	—	—	—		суммирование
	1	5	7	3		частичное произведение
+			1	2	1	умножение на младший разряд
	—	—	—	—	—	суммирование
	1	5	8	5	1	произведение

Алгоритмы умножения в цифровых процессорах в двоичной системе счисления базируются на тех же принципах, что и в десятичной системе счисления. Рассмотрим пример. Пусть надо умножить $1001_{(2)}$ на $101_{(2)}$.

			1	0	0	1	множимое
			*	1	0	1	множитель
			—	—	—	—	
			1	0	0	1	умножение на младший разряд
	+	0	0	0	0		умножение на средний разряд
			—	—	—	—	суммирование
			1	0	0	1	частичное произведение
+	1	0	0	1			умножение на старший разряд
	—	—	—	—	—	—	суммирование
	1	0	1	1	0	1	произведение

Поскольку таблица умножения в двоичной системе тривиальна, то, как видно из примера, если очередная цифра множителя равна 1, к частичному произведению добавляется множимое, в противном случае – нули (т.е. частичное произведение не меняется).

Существует следующая классификация алгоритмов умножения (табл. 4.2).

Таблица 4.2

Классификация алгоритмов умножения

№ п/п	Умножение начинается	Направление сдвига		
		множимого	множителя	частичного произведения
1	с младших разрядов множителя	неподвижно	вправо	вправо
2		влево	вправо	неподвижно
3	со старших разрядов множителя	неподвижно	влево	влево
4		вправо	влево	неподвижно

Наибольшее применение в цифровых процессорах получил первый способ. Следующий по степени использования – четвертый способ. Он используется при умножении дробных чисел в формате с ФТ, когда на n-разрядном процессоре нужно получить не 2*n разрядное произведение, а только его старшие n разрядов.

Функциональная схема ОА для умножения по первому алгоритму приведена на рис. 4.6.

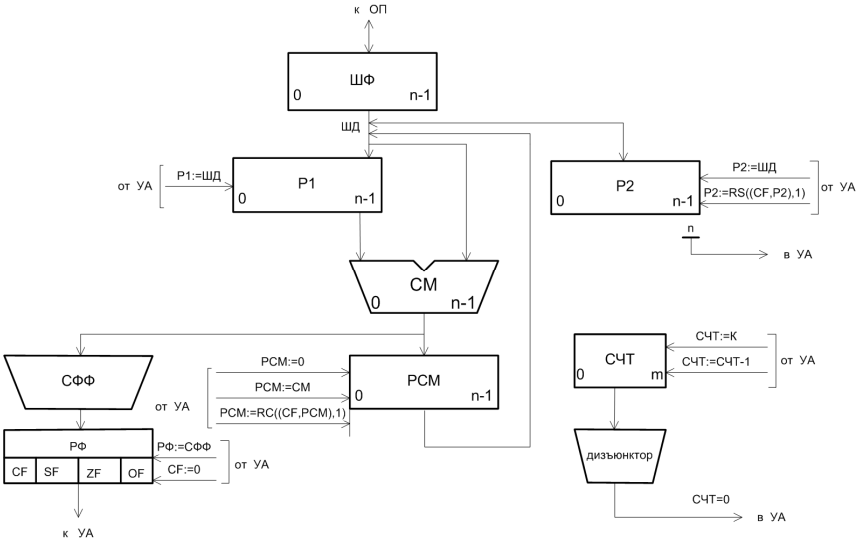

Рис. 4.6. Функциональная схема ОА для умножения по первому алгоритму

В схеме использованы следующие обозначения:

ШФ – шинный формирователь управляет направлением передачи данных (или из ОП, или в ОП);

ШД – шина данных ОА;

Р1 – регистр, в который записывается множимое;

Р2 – регистр, в который записывается множитель, а затем в него по ходу умножения пересылаются младшие разряды произведения (по окончании умножения в нем находятся младшие разряды произведения);

СМ – сумматор, работающий в дополнительном коде;

РСМ – регистр для получения частичного произведения (по окончании умножения в нем находится старшая часть произведения);

СФФ – схема формирования флагов;

РФ – регистр флагов;

СЧТ – счетчик тактов работает на вычитание и служит для подсчета количества тактов умножения;

(CF, P2) – объединение в единое CF и P2 при сдвиге;

(CF, PCM) – объединение в единое CF и PCM при сдвиге.

В этом операционном автомате из-за того, что есть только одна шина для ввода данных, операция умножения X*Y распадается на две операции: «посылка X» и «умножение на Y». Причем они выполняются последовательно друг за другом. На рис. 4.7 и 4.8 приведены граф-схемы алгоритмов этих операций.

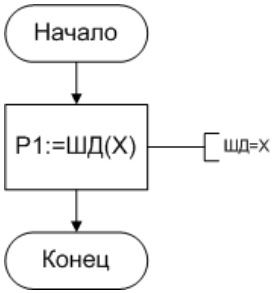

Рис. 4.7. Алгоритм операции «посылка X»

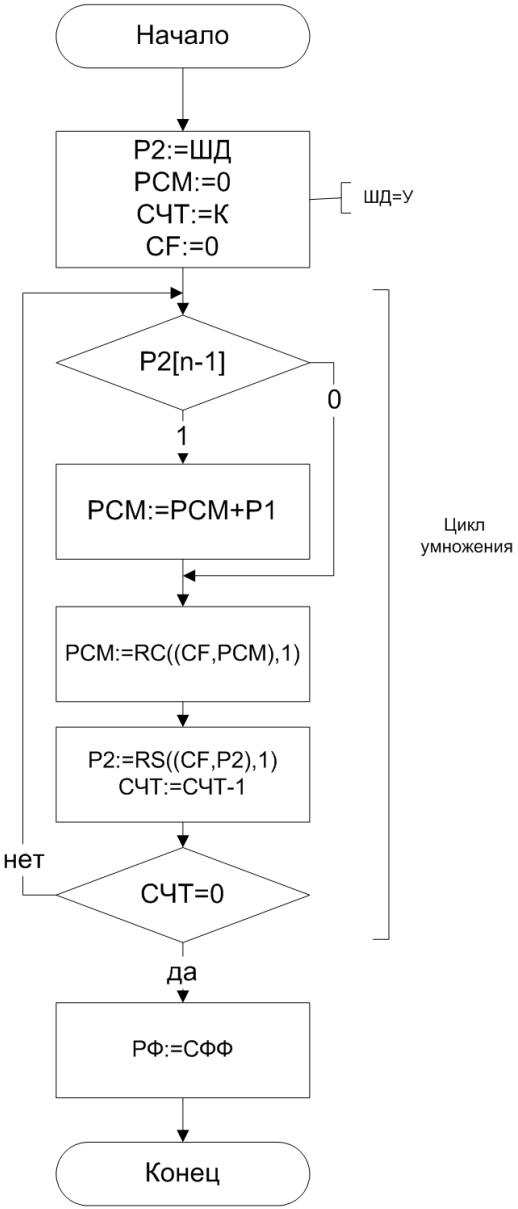

Рис. 4.8. Алгоритм операции «умножение на Y»

4.5.1. Алгоритм умножения целых беззнаковых чисел в формате с ФТ

Рассмотрим алгоритм умножения целых беззнаковых чисел в формате с ФТ.

Шаг 1. Множимое записывается в регистр Р1 (операция «посылка X»).

Шаг 2. Множитель записывается в Р2 (операция «умножение на Y»).

Шаг 3. Регистр сумматора и флаг CF обнуляются. В счетчик тактов записывается значение, равное количеству разрядов регистра множителя.

Шаг 4. Проверяется значение младшего бита регистра Р2. Если он равен 1, то к регистру сумматора (РСМ) прибавляется содержимое Р1.

Шаг 5. Содержимое флага CF и РСМ циклически сдвигается вправо на один разряд (младший разряд регистра сумматора заносится во флаг CF).

Шаг 6. Содержимое флага CF и регистра Р2 сдвигается логически вправо на один разряд. Счетчик тактов уменьшается на 1.

Шаг 7. Значение счетчика тактов сравнивается с 0. Если значение счетчика тактов не равно 0, то переход к шагу 4, иначе к шагу 8.

Шаг 8. Устанавливается значение регистра флагов. Значение РСМ и Р2 последовательно переписываются в ОП.

Рассмотрим пример выполнения этого алгоритма.

Выполнить умножение операндов X и Y на восьмиразрядном процессоре. $X=270_{(8)}$, $Y=71_{(8)}$ (табл. 4.3). Обратите внимание, что при рассмотрении примера двойной рамкой выделяется значение младшего бита Р2, который анализируется на шаге 4.

Когда СЧТ=0, в регистрах РСМ и Р2 содержится произведение: в РСМ – старшая часть, а в Р2 – младшая часть. Выполним проверку:

$X*Y=0010100011111000_{(2)}=10100011111000_{(2)}=24370_{(8)}$.

		2	7	0
	x		7	1
—	—	—	—	—
		2	7	0
2	4	1	0	
—	—	—	—	—
2	4	3	7	0

Результат верен.

Таблица 4.3

Умножение беззнаковых операндов X*Y, где $X = 270_{(8)}$, $Y = 71_{(8)}$

CF	PCM								P2								СЧТ				Комментарий
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	**Исходное состояние**
0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	**CF:=0; PCM:=0; P2:=Y; СЧТ=8**
0	1	0	1	1	1	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	**+P1**
0	1	0	1	1	1	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	**PCM:=PCM+P1**
0	0	1	0	1	1	1	0	0	0	0	1	1	1	0	0	1	1	0	0	0	**(CF,PCM):=RC((CF,PCM),1)**
0	0	1	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	1	1	1	**(CF,P2):=RS((CF,P2),1) СЧТ:=СЧТ – 1**
0	0	0	1	0	1	1	1	0	0	0	0	1	1	1	0	0	0	1	1	1	**(CF,PCM):=RC((CF,PCM),1)**
0	0	0	1	0	1	1	1	0	0	0	0	0	1	1	1	0	0	1	1	0	**(CF,P2):=RS((CF,P2),1) СЧТ:=СЧТ – 1**
0	0	0	0	1	0	1	1	1	0	0	0	0	1	1	1	0	0	1	1	0	**(CF,PCM):=RC((CF,PCM),1)**
0	0	0	0	1	0	1	1	1	0	0	0	0	0	1	1	1	0	1	0	1	**(CF,P2):=RS((CF,P2),1) СЧТ:=СЧТ – 1**
0	1	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	**+P1**
0	1	1	0	0	1	1	1	1	0	0	0	0	0	1	1	1	0	1	0	1	**PCM:=PCM+P1**
1	0	1	1	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	0	1	**(CF,PCM):=RC((CF,PCM),1)**
0	0	1	1	0	0	1	1	1	1	0	0	0	0	0	1	1	0	1	0	0	**(CF,P2):=RS((CF,P2),1) СЧТ:=СЧТ – 1**

Окончание табл. 4.3

0	1	0	1	1	1	0	0	0	1	0	0	0	0	0	1	1	0	1	0	0	**+P1**
1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	0	1	0	0	**PCM:=PCM+P1**
1	1	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	0	1	0	0	**(CF,PCM):=RC((CF,PCM),1)**
0	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	1	1	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**
0	1	0	1	1	1	0	0	0	1	1	0	0	0	0	0	1	0	0	1	1	**+P1**
1	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	0	1	1	**PCM:=PCM+P1**
1	1	0	1	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0	1	1	**(CF,PCM):=RC((CF,PCM),1)**
0	1	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	0	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**
1	0	1	0	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1	0	**(CF,PCM):=RC((CF,PCM),1)**
0	0	1	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**
1	0	0	1	0	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1	**(CF,PCM):=RC((CF,PCM),1)**
0	0	0	1	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**

CF=0; SF=0; ZF=0; OF=0.

4.5.2. Алгоритм умножения целых чисел в формате с ФТ в прямом коде

Алгоритм умножения целых чисел в формате с ФТ в прямом коде отличается от ранее рассмотренного алгоритма умножения целых беззнаковых чисел в формате с ФТ следующим:

1. Умножаются модули операндов.

2. Знак произведения определяется операцией сложения по модулю 2 знаковых разрядов операндов (сомножителей):

Знак множимого	Знак множителя	Сложение по модулю 2	Знак произведения
0	0	0^0=0	0
0	1	0^1=1	1
1	0	1^0=1	1
1	1	1^1=0	0

3. В счетчик тактов (СЧТ) устанавливается значение на единицу меньше количества битов регистра множителя (так как на знаковый бит умножение не производится).

4. Чтобы избавиться от знакового разряда множителя, который при значении СЧТ=0 находится в младшем разряде Р2, выполняется так называемый «n сдвиг». Это делается логическим сдвигом вправо на один бит регистра сумматора РСМ и регистра Р2 (с переносом выдвигаемого разряда из РСМ в старший разряд Р2).

4.5.3. Алгоритм умножения целых чисел в дополнительном коде

Алгоритм умножения целых чисел в дополнительном коде отличается от алгоритма умножения целых чисел в формате с ФТ в прямом коде следующим:

1. Умножается дополнительный код множимого на дополнительный код множителя.

2. В РСМ выполняется арифметический сдвиг.

3. При умножении на множитель, имеющий отрицательный знак, требуется коррекция результата. Коррекция состоит в том, что когда

СЧТ становится равным 0, из РСМ вычитается множимое (прибавляется множимое со знаком «минус») и только после этого выполняется «n сдвиг».

Необходимость коррекции объясняется следующим. Дополнительный код отрицательного числа Y получается как:

$[Y]_2=2^n + Y$, где n – разрядность процессора.

Откуда $Y=[Y]_2 – 2^n$.

Тогда

$[X]_2*[Y]_2 =[X]_2*([Y]_2 –2^n)=[X]_2*[Y]_2–([X]_2*2^n)=[X]_2*[Y]_2 +(–[X]_2*2^n)$, где второе слагаемое – $([X]_2*2^n)$ – это и есть коррекция в РСМ.

Пример. Выполнить умножение операндов X и Y на восьмиразрядном процессоре. $X= – 160_{(8)}$, $Y=67_{(8)}$ (табл. 4.4). Обратите внимание, что при рассмотрении примера двойной рамкой выделяется значение младшего бита Р2, который анализируется на шаге 4.

$X= – 160_{(8)}= – 1110000_{(2)}$; $[X]_2=10010000$;

$Y=67_{(8)}=110111_{(2)}$; $[Y]_2=110111$;

Р1=10010000;

Р2=00110111.

В табл. 4.4 через ARC обозначается комбинация арифметического и логического сдвигов. В этом случае выдвигаемый разряд записывается во флаг CF. Если при сложении возникает переполнение, то это учитывается при арифметическом сдвиге (значение CF записывается в знаковый бит, а значение знакового бита – в старший числовой разряд). После «n сдвига» в регистрах РСМ и Р2 содержится произведение: в РСМ – старшая часть, а в Р2 – младшая часть. Выполним проверку:

$[X*Y]_2=[1110011111110000]_2= – 001100000010000_{(2)}= – 14020_{(8)}$.

		1	6	0
	x		6	7
—	—	—	—	—
	1	4	2	0
1	2	4	0	
—	—	—	—	—
1	4	0	2	0

Результат верен.

Таблица 4.4

Умножение в дополнительном коде X*Y, где X =– $160_{(8)}$, Y= $67_{(8)}$

CF	PCM								P2								СЧТ				Комментарий
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	**Исходное состояние**
0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	1	1	1	**CF:=0; PCM:=0; P2:=$[Y]_2$; СЧТ=7**
0	1	0	0	1	0	0	0	0	0	0	1	1	0	1	1	1	0	1	1	1	**+P1**
0	1	0	0	1	0	0	0	0	0	0	1	1	0	1	1	1	0	1	1	1	**PCM:=PCM+P1**
0	1	1	0	0	1	0	0	0	0	0	1	1	0	1	1	1	0	1	1	1	**(CF,PCM):=ARC((CF,PCM),1)**
0	1	1	0	0	1	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**
0	1	0	0	1	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	**+P1**
1	0	1	0	1	1	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	**PCM:=PCM+P1;** переполнение при сложении!
0	1	0	1	0	1	1	0	0	0	0	0	1	1	0	1	1	0	1	1	0	**(CF,PCM):=ARC((CF,PCM),1)**
0	1	0	1	0	1	1	0	0	0	0	0	0	1	1	0	1	0	1	0	1	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**
0	1	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	**+P1**
1	0	0	1	1	1	1	0	0	0	0	0	0	1	1	0	1	0	1	0	1	**PCM:=PCM+P1** переполнение при сложении!
0	1	0	0	1	1	1	1	0	0	0	0	0	1	1	0	1	0	1	0	1	**(CF,PCM):=ARC((CF,PCM),1)**
0	1	0	0	1	1	1	1	0	0	0	0	0	0	1	1	0	0	1	0	0	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**

Окончание табл. 4.4

0	1	1	0	0	1	1	1	1	0	0	0	0	0	1	1	0	0	1	0	0	**(CF,PCM):=ARC((CF,PCM),1)**
0	1	1	0	0	1	1	1	1	0	0	0	0	0	0	1	1	0	0	1	1	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**
0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	**+P1**
1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	1	1	0	0	1	1	**PCM:=PCM+P1** переполнение при сложении!
1	1	0	1	0	1	1	1	1	0	0	0	0	0	0	1	1	0	0	1	1	**(CF,PCM):=ARC((CF,PCM),1)**
0	1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**
0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	**+P1**
1	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	**PCM:=PCM+P1** переполнение при сложении!
1	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	**(CF,PCM):=ARC((CF,PCM),1)**
0	1	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**
1	1	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	**(CF,PCM):=ARC((CF,PCM),1)**
0	1	1	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**
1	1	1	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	**"n"-й сдвиг;** **(CF,PCM):=ARC((CF,PCM),1)**
0	1	1	1	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	**(CF,P2):=RS((CF,P2),1)**

CF=0; SF=1; ZF=0; OF=0.

4.5.4. Алгоритм умножения двоичных дробных чисел в формате с ФТ

Алгоритм умножения двоичных дробных чисел в формате с ФТ имеет следующие отличия по отношению к раннее рассмотренным. Нельзя избавиться от знакового бита множителя, после достижения счетчиком тактов значения, равного 0, выполнив дополнительный «n сдвиг» (произведение было бы уменьшено в два раза). Нужно обнулить младший бит регистра P2, например, выполнив операцию поразрядной конъюнкции с маской, содержащей 1 во всех битах, кроме младшего. Рассмотрим пример выполнения такого алгоритма.

Выполнить операцию умножения операндов X и Y на восьмиразрядном процессоре в прямом коде (табл. 4.5). Обратите внимание, что при рассмотрении примера двойной рамкой выделяется значение младшего бита P2, который анализируется на шаге 4.

$X=0{,}24_{(8)}$, $Y=-0{,}65_{(8)}$;

Z=X*Y;

$X=0{,}24_{(8)}=0{,}0101000_{(2)}$; $[X]_1=00101000$;

$Y=-0{,}65_{(8)}=-0{,}1101010_{(2)}$; $[Y]_1=11101010$;

$P1=[|X|]_1=00101000$;

$P2=[|Y|]_1=01101010$.

В этой таблице через ARC обозначается комбинация арифметического и логического сдвигов. В этом случае выдвигаемый разряд записывается во флаг CF. Если при сложении возникает переполнение, то это учитывается при арифметическом сдвиге (значение CF записывается в знаковый бит, а значение знакового бита – в старший числовой разряд). После поразрядной конъюнкции P2 с маской $11111110_{(2)}$ в регистрах PCM и P2 содержится произведение: в PCM старшая часть, а в P2 – младшая часть. Выполним проверку:

$[X*Y]_1=[1010000100100000]_1=-0{,}01000010010000_{(2)}=-0{,}20440_{(8)}$.

```
          0,  6  5
      x   0,  2  4
  —   —   —   —  —
          3   2  4
      1   5   2
  —   —   —   —  —
  0,  2   0   4  4
```

Результат верен.

Таблица 4.5

Умножение в прямом коде X*Y, где X = $0{,}24_{(8)}$, Y= $-0{,}65_{(8)}$

CF	PCM								P2								СЧТ				Комментарий
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	**Исходное состояние**
0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	1	1	1	**CF:=0; PCM:=0; P2:= $[Y]_1$; СЧТ=7**
0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	1	1	1	**(CF,PCM):=ARC((CF,PCM),1)**
0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	1	0	**(CF,P2):=RS((CF,P2),1) СЧТ:=СЧТ – 1**
	0	0	1	0	1	0	0	0	0	0	1	1	0	1	0	1	0	1	1	0	**+P1**
0	0	0	1	0	1	0	0	0	0	0	1	1	0	1	0	1	0	1	1	0	**PCM:=PCM+P1;**
0	0	0	0	1	0	1	0	0	0	0	1	1	0	1	0	1	0	1	1	0	**(CF,PCM):=ARC((CF,PCM),1)**
0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	1	0	0	1	0	1	**(CF,P2):=RS((CF,P2),1) СЧТ:=СЧТ – 1**
0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	1	0	0	1	0	1	**(CF,PCM):=ARC((CF,PCM),1)**
0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	1	0	1	0	0	**(CF,P2):=RS((CF,P2),1) СЧТ:=СЧТ – 1**
	0	0	1	0	1	0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	**+P1**
0	0	0	1	1	0	1	1	0	0	0	0	0	1	1	0	1	0	1	0	0	**PCM:=PCM+P1;**
0	0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1	0	1	0	0	**(CF,PCM):=ARC((CF,PCM),1)**
0	0	0	0	1	1	0	1	1	0	0	0	0	0	1	1	0	0	0	1	1	**(CF,P2):=RS((CF,P2),1) СЧТ:=СЧТ – 1**
1	0	0	0	0	1	1	0	1	0	0	0	0	0	1	1	0	0	0	1	1	**(CF,PCM):=ARC((CF,PCM),1)**

Окончание табл. 4.5

0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	1	1	0	0	1	0	**(CF,P2):=RS((CF,P2),1) СЧТ:=СЧТ – 1**
	0	0	1	0	1	0	0	0	1	0	0	0	0	0	1	1	0	0	1	0	**+P1**
0	0	0	1	1	0	1	0	1	1	0	0	0	0	0	1	1	0	0	1	0	**PCM:=PCM+P1**
1	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	1	0	0	1	0	**(CF,PCM):=ARC((CF,PCM),1)**
0	0	0	0	1	1	0	1	0	1	1	0	0	0	0	0	1	0	0	0	1	**(CF,P2):=RS((CF,P2),1) СЧТ:=СЧТ – 1**
	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	**+P1**
0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	**PCM:=PCM+P1**
0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	1	**(CF,PCM):=ARC((CF,PCM),1)**
0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	**(CF,P2):=RS((CF,P2),1) СЧТ:=СЧТ – 1**
0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	**P2&11111110; удаление знака множителя**

Знак Z=(знак X)^(знак У)=0^1=1

CF=0; SF=1; ZF=0; OF=0.

4.5.5. Алгоритм умножения чисел в формате с плавающей точкой

Так же как и при сложении, операция умножения операндов, представленных в формате с плавающей точкой, выполняется раздельно над мантиссами и порядками. Операция Z=X*Y, где $X=M_x*2^{Px}$, $Y=M_y*2^{Py}$, выполняется как

$M_z=M_x*M_y$; $P_z=P_x+P_y$.

Алгоритм состоит из следующих шагов:

1. Операнды извлекаются из ОП с разделением на мантиссу (восстанавливается скрытый бит) и порядок.

2. Содержимое регистров порядков складывается с вычитанием константы $127_{(10)}$ (смещение). Проверяется превышение максимально и минимально допустимого порядков, и если такой случай есть, то устанавливается соответствующий флаг.

3. Содержимое регистра мантиссы X умножается на содержимое регистра мантиссы Y в прямом коде по алгоритму умножения дробных чисел в формате с ФТ (выравнивание мантисс не нужно).

4. Проверяется нарушение нормализации вправо мантиссы произведения. Если нарушение есть, производится нормализация влево. При этом проверяется, чтобы порядок произведения P_z не превысил максимально допустимый (если превысил, то фиксируется переполнение).

5. Полученный результат записывается в ОП в формате КВ (со скрытием бита мантиссы и объединением содержимого регистра порядка и регистра мантиссы).

Рассмотрим пример выполнения такого алгоритма. Выполним операцию умножения операндов Z=X*Y.

$X=24_{(8)}$, $Y=-65_{(8)}$. Представим эти операнды в формате КВ, в том виде, в котором они хранятся в ОП, т.е. со скрытым битом.

$X=24_{(8)}=10100_{(2)}=0{,}10100_{(2)}*2^{100}$. Тогда с учетом скрытия старшего бита получаем:

X=

0	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
байт								**байт**								**байт**								**байт**							

$Y = -65_{(8)} = -110101_{(2)} = -0{,}110101_{(2)} * 2^{101}$. Тогда с учетом скрытия старшего бита получаем:

Y=	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	байт								**байт**								**байт**								**байт**							

Рассмотрим выполнение алгоритма умножения.

Шаг 1. Операнды извлекаются из ОП (с восстановлением скрытого бита) и помещаются в ОА в регистры мантисс и порядков:

РП$_x$=	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**

РМ$_x$=	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

РП$_y$=	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**

РМ$_y$=	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Шаг 2. Определяется сумма порядков $[РП_x+(-127_{(10)})+РП_y]_2$:

$РП_x$	1	0	0	0	0	1	0	0
$[-127_{(10)}]_2$	1	0	0	0	0	0	0	1
$РП_x+[-127_{(10)}]_2=$	0	0	0	0	0	1	0	1
$РП_y$	1	0	0	0	0	1	0	1
$РП_x+(-127_{(10)})+РП_y$	1	0	0	0	1	0	1	0

Шаг 3. Умножение мантисс (табл. 4.6). Так как значащие разряды мантисс занимают 8 бит, в таблице показываются только по 8 бит в регистрах РСМ и Р2.

Таблица 4.6

Умножение мантисс в прямом коде

CF	РСМ								Р2								СЧТ				Комментарий
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	**Исходное состояние**
0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0	1	1	1	**CF:=0; РСМ:=0; Р2:= $[My]_1$; СЧТ=7**
0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0	1	1	1	**(CF,РСМ):=ARC((CF,РСМ),1)**
0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	1	1	0	**(CF,Р2):=RS((CF,Р2),1) СЧТ:=СЧТ – 1**
	0	1	0	1	0	0	0	0	0	1	1	1	0	1	0	1	0	1	1	0	**+ РМх**
0	0	1	0	1	0	0	0	0	0	1	1	1	0	1	0	1	0	1	1	0	**РСМ:=РСМ+Р1;**
0	0	0	1	0	1	0	0	0	0	1	1	1	0	1	0	1	0	1	1	0	**(CF,РСМ):=ARC((CF,РСМ),1)**
0	0	0	1	0	1	0	0	0	0	0	1	1	1	0	1	0	0	1	0	1	**(CF,Р2):=RS((CF,Р2),1) СЧТ:=СЧТ – 1**
0	0	0	0	1	0	1	0	0	0	0	1	1	1	0	1	0	0	1	0	1	**(CF,РСМ):=ARC((CF,РСМ),1)**
0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	0	1	0	1	0	0	**(CF,Р2):=RS((CF,Р2),1) СЧТ:=СЧТ – 1**
	0	1	0	1	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0	**+ РМх**
0	0	1	1	0	0	1	0	0	0	0	0	1	1	1	0	1	0	1	0	0	**РСМ:=РСМ+Р1;**
0	0	0	1	1	0	0	1	0	0	0	0	1	1	1	0	1	0	1	0	0	**(CF,РСМ):=ARC((CF,РСМ),1)**
0	0	0	1	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	1	**(CF,Р2):=RS((CF,Р2),1) СЧТ:=СЧТ – 1**

Окончание табл. 4.6

0	0	0	0	1	1	0	0	1	0	0	0	0	1	1	1	0	0	0	1	1	**(CF,PCM):=ARC((CF,PCM),1)**
0	0	0	0	1	1	0	0	1	0	0	0	0	0	1	1	1	0	0	1	0	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**
	0	1	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0	**+ PMx**
0	0	1	1	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	1	0	**PCM:=PCM+P1;**
1	0	0	1	1	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	**(CF,PCM):=ARC((CF,PCM),1)**
0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0	1	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**
	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	1	**+ PMx**
0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0	1	**PCM:=PCM+P1;**
0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	1	**(CF,PCM):=ARC((CF,PCM),1)**
0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	**(CF,P2):=RS((CF,P2),1)** **СЧТ:=СЧТ – 1**

Шаг 4. У мантиссы произведения нет нарушения нормализации вправо.

Шаг 5. Запись в ОП результата.

Знак Z=знак X^знак У=0^1=1.

$[M_z]_1$=11000010010000000000000000;

P_z=10001010.

После скрытия старшего бита мантиссы получаем:

Z=	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	байт								**байт**								**байт**								**байт**							

Если перевести операнд из формата КВ в двоичную систему счисления, а затем – в восьмеричную, то получим:

$Z= -1{,}00001010*2^{1001}= -10000100100_{(2)}=2044_{(8)}$.

Результат верен.

4.6. Алгоритмы деления в цифровых процессорах

В цифровом процессоре операция деления сводится к последовательности выполнения операций вычитания и сдвига. Количество получаемых битов частного определяется разрядностью процессора. При делении в формате целых чисел результатом операции являются частное и остаток от деления. При делении в формате дробных чисел и в формате с плавающей точкой результатом является только частное, а остаток не сохраняется. Существуют два алгоритма деления:

1) с восстановлением остатка;

2) без восстановления остатка.

Алгоритм деления с восстановлением остатка. Цикл алгоритма деления с восстановлением остатка состоит в следующем. Из старших разрядов делимого (а затем из остатка) вычитается делитель. Если разность положительная или равна 0, то очередная цифра частного равна 1. Если же отрицательная, то цифра частного равна 0, и остаток восстанавливается до предыдущего значения прибавлением делителя. Затем для следующего цикла деления остаток увеличивается присоединением очередной цифры делимого.

Рассмотрим алгоритм деления с восстановлением остатка «столбиком» в двоичной системе счисления на примере деления целых чисел.

Пример. X/Y=Z(R), где Z – частное, а R – остаток.

$X=251_{(8)}=10101001_{(2)}$;

$Y=5_{(8)}=101_{(2)}$.

Операция	Делимое/Остаток								Частное						Комментарий
	1	0	1	0	1	0	0	1							
–	1	0	1												вычитание
	0	0	0	0	1	0	0	1	1						остаток положительный
–		1	0	1											вычитание
	–	1	0	1					1	0					остаток отрицательный
+		1	0	1											восстановление остатка
	0	0	0	0	1	0	0	1							остаток восстановлен
–			1	0	1										вычитание
		–	1	0	0				1	0	0				остаток отрицательный
+			1	0	1										восстановление остатка
					1	0	0	1							остаток восстановлен
–				1	0	1									вычитание
				–	1	1			1	0	0	0			остаток отрицательный
+				1	0	1									восстановление остатка
					1	0	0	1							остаток восстановлен
–					1	0	1								вычитание
						–	1		1	0	0	0	0		остаток отрицательный
+					1	0	1								восстановление остатка

Операция	Делимое/Остаток								Частное						Комментарий
					1	0	0	1							остаток восстановлен
–						1	0	1							вычитание
						1	0	0	1	0	0	0	0	1	остаток положительный
Так как последнее вычитание делителя было с учетом младшего разряда делимого, то деление завершено															

Результатом деления будут частное, равное $41_{(8)}$, остаток, равный $4_{(8)}$. Таким образом, алгоритм деления с восстановлением остатка включает следующие основные действия:

1. Биты частного получаются в цикле деления, начиная со старшего разряда.

2. Каждый цикл деления включает операцию вычитания делителя из остатка делимого (на первом шаге из старших разрядов делимого).

3. Если при вычитании получен отрицательный остаток, то очередная цифра частного равна 0, а если положительный или равный 0, то очередная цифра частного равна 1.

4. Если полученный остаток отрицательный, то он восстанавливается до предыдущего значения добавлением делителя.

5. Остаток увеличивается присоединением очередной цифры делимого.

6. Число тактов деления определяется разрядностью процессора.

Алгоритм деления без восстановления остатка. Цикл алгоритма деления без восстановления остатка состоит в следующем. Из старших разрядов делимого (а затем из остатка) вычитается делитель. Если разность положительная или равна 0, то очередная цифра частного равна 1. Если же отрицательная, то цифра частного равна 0 и предыдущий остаток не восстанавливается. Затем остаток увеличивается присоединением очередной цифры делимого. В следующем цикле к увеличенному остатку прибавляется делитель (на самом деле, поскольку знаки слагаемых разные, происходит вычитание).

Рассмотрим алгоритм деления без восстановления остатка «столбиком» в двоичной системе счисления на примере деления целых чисел.

Пример. X/Y=Z(R), где Z – частное, а R – остаток.

$X=251_{(8)}=10101001_{(2)}$;

$Y=5_{(8)}=101_{(2)}$.

Обратите внимание, что при увеличении отрицательного остатка (делимое – положительное) присоединением следующего разряда делимого, равного 1, происходит сложение отрицательного и положительного чисел.

Операция	Делимое/Остаток								Частное						Комментарий
	1	0	1	0	1	0	0	1							
–	1	0	0												вычитание
	0	0	0	0	1	0	0	1	1						остаток положительный
–		1	0	1											вычитание
	–	1	0	1					1	0					остаток отрицательный
	–	1	0	0	1										добавляется следующий разряд делимого (– 1010+ +1=1001)
+			1	0	1										прибавление делителя
		–	1	0	0				1	0	0				остаток отрицательный
		–	1	0	0	0									добавляется следующий разряд делимого (– 1000+ +0=1000)
+				1	0	1									прибавление делителя
				–	1	1			1	0	0	0			остаток отрицательный
				–	1	1	0								добавляется следующий разряд делимого (–110+ +0=110)

Операция	Делимое/Остаток								Частное						Комментарий
+					1	0	1								прибавление делителя
						–	1		1	0	0	0	0		остаток отрицательный
							–	1							добавляется следующий разряд делимого (–10+ +1= –1)
+						1	0	1							прибавление делителя
						1	0	0	1	0	0	0	0	1	остаток положительный
Так как при делении использованы все разряды делимого, то деление завершено.															

Результатом деления будут частное, равное $41_{(8)}$, остаток, равный $4_{(8)}$. Алгоритм деления без восстановления остатка включает следующие основные действия:

1. Биты частного получаются в цикле деления, начиная со старшего разряда.

2. Каждый цикл деления включает либо операцию вычитания делителя из остатка делимого (на первом шаге из делимого), либо операцию сложения делителя с остатком делимого.

3. Если на предыдущем шаге получен отрицательный остаток, то очередная цифра частного равна 0, а если положительный или равный 0, то очередная цифра частного равна 1.

4. Остаток увеличивается присоединением очередной цифры делимого.

5. Число тактов деления определяется разрядностью процессора.

6. Если знак последнего остатка не совпадает со знаком делимого, то остаток не истин и производится его восстановление добавлением к нему делителя.

Алгоритм деления без восстановления остатка при получении n цифр частного содержит только n операций сложения/вычитания, а в алгоритме с восстановлением остатка (в наихудшем случае) таких операций может быть 2*n. Поэтому, несмотря на то что алгоритм

деления с восстановлением остатка более простой, в большинстве современных цифровых процессоров используется алгоритм деления без восстановления остатка. Поэтому в дальнейшем, если не оговорено специально, рассматриваем алгоритмы деления без восстановления остатка.

4.6.1. Алгоритмы деления целых чисел в формате с ФТ

При делении операндов в формате целых чисел с ФТ делимое имеет удвоенную длину по отношению к делителю, так в n-разрядном процессоре делимое содержит 2*n битов, а делитель – n битов. Результатом деления являются частное и остаток, каждый из которых занимает по n битов. Функциональная схема операционного автомата для операции деления приведена на рис. 4.9, где использованы следующие обозначения:

Р1 – регистр, в который записывается делитель;

РСМ – регистр сумматора, в который перед началом деления записываются старшие n разрядов делимого, а затем (в цикле деления) в нем находится остаток от делимого;

Р2 – регистр, в который перед началом деления записываются младшие разряды делимого, а затем (в цикле деления) при сдвигах влево в него последовательно заносятся цифры частного;

СМ – n-разрядный сумматор;

MS – n-разрядный мультиплексор, который используется или для прибавления делителя (MS:=P1), или для вычитания делителя (MS:=¬P1 и СМ:=СМ +1);

СЧТ используется для подсчета количества циклов деления (работает на вычитание);

ШФ – шинный формирователь, который подключает ШД на передачу или прием к ОП;

дизъюнктор 1 определяет равенство 0 делителя;

дизъюнктор 2 определяет равенство 0 СЧТ;

СФФ – схема формирования флагов;

РФ – регистр флагов.

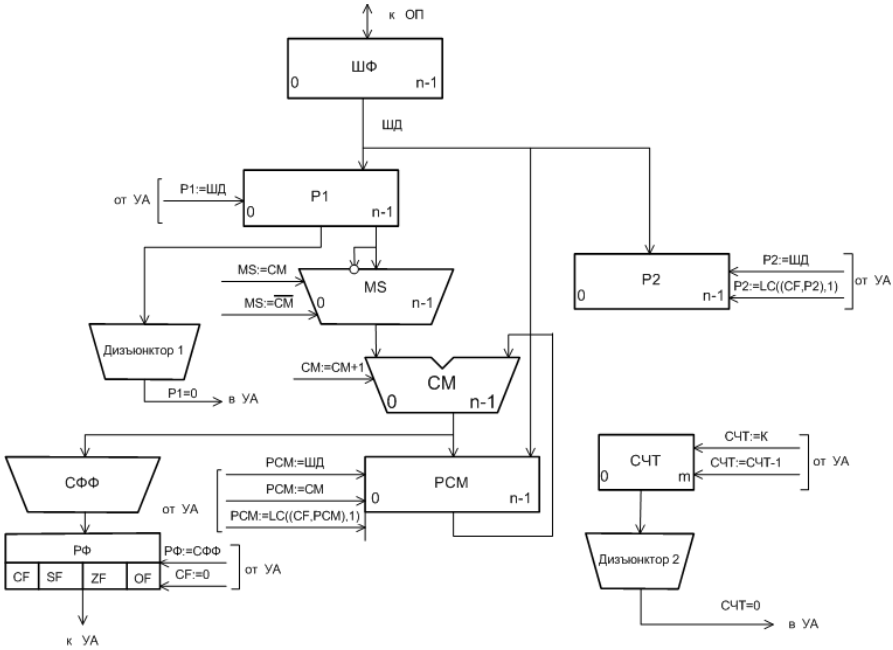

Рис. 4.9. Функциональная схема операционного автомата для операции деления

Рассмотрим общие положения по делению беззнаковых чисел в формате с ФТ. Перед началом операции деления делимое записывается в регистры РСМ и Р2 так, чтобы младший бит делимого был записан в младший разряд Р2. Делитель записывается в Р1. В счетчик тактов заносится значение n. По окончании операции деления в Р2 будет частное, а в РСМ – остаток от деления.

Сумматор работает в дополнительном коде. При выполнении сдвига влево выдвигаемый из Р2 бит делимого передается в младший разряд регистра РСМ, а на место освободившегося младшего бита Р2 заносится очередная цифра частного. Значение цифры частного зависит от знака остатка. Для определения знака остатка при делении беззнаковых чисел используется флаг CF. Если после сложения CF=0, то остаток отрицательный, а если CF=1 – остаток положительный.

При делении чисел возможно переполнение разрядной сетки процессора. Это происходит в том случае, если частное не помещается в регистр частного. Определение возможности переполнения при делении производится при первом вычитании. Если при первом («пробном») вычитании знак остатка положительный, то частное не поместится в регистр частного и фиксируется переполнение, в противном случае переполнения не будет и деление состоится. Это вытекает из следующего. Для того чтобы частное поместилось в регистр частного, должно выполниться условие:

$X/Y<2^n$,

где n – разрядность процессора.

Откуда получаем:

$X<Y*2^n$;

$X - Y*2^n<0$.

$Y*2^n$ – это и есть то, что вычитается на первом шаге деления из старших разрядов делимого.

Деление X/Y на данном ОА, из-за того что шина обмена с ОП имеет n разрядов, распадается на три операции: «посылка старших разрядов X», «посылка младших разрядов X», «деление на Y».

На рис. 4.10, 4.11, 4.12 приведены граф-схемы этих алгоритмов.

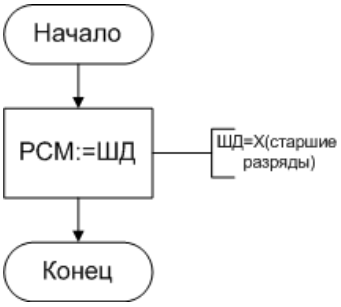

Рис. 4.10. Алгоритм операции «посылка старших разрядов X»

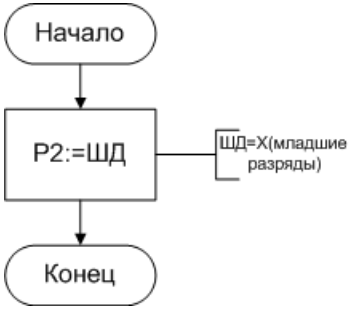

Рис. 4.11. Алгоритм операции «посылка младших разрядов X»

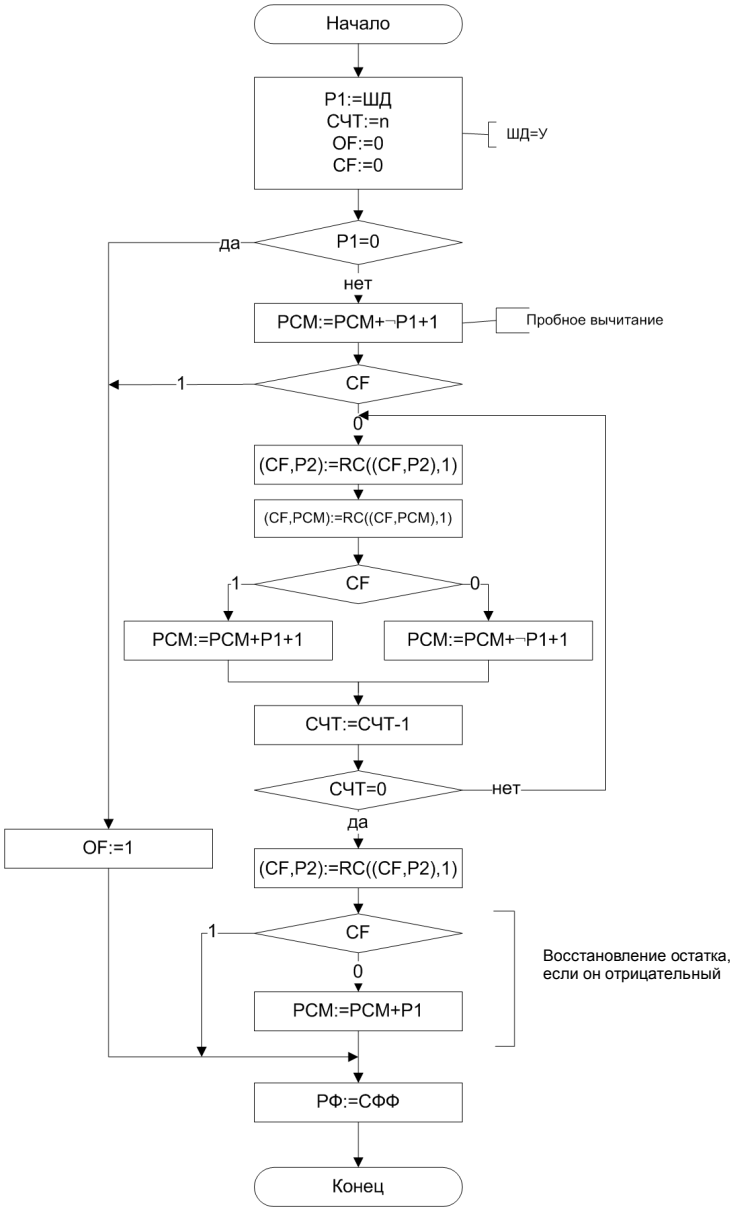

Рис. 4.12. Алгоритм операции «деление на Y»

Алгоритм деления целых беззнаковых чисел в формате с ФТ (без восстановления остатка) состоит из следующих шагов.

Шаг 1. Запись делимого в регистры РСМ и Р2. Младшая половина битов делителя записывается в регистр Р2, а старшая – в регистр РСМ. Запись делителя в регистр Р1. Сброс флагов CF и OF.

Шаг 2. Проверка Р1 на равенство нулю. Если Р1 равен 0, установка OF в 1 и завершение деления (так как деление на 0 не определено).

Шаг 3. Пробное вычитание – из старших разрядов делимого (т.е. регистра РСМ) вычитается делитель (Р1), при этом:

а) если остаток ≥ 0 (CF=1), то будет переполнение при делении, флаг OF устанавливается в единицу, деление завершается.

б) если остаток < 0 (CF=0), то деление состоится, переход к шагу 4.

Шаг 4. Циклический сдвиг влево Р2 и флага CF на 1 бит.

Шаг 5. Циклический сдвиг влево РСМ и флага CF на 1 бит.

Шаг 6. Если CF=1 (полученный остаток отрицательный), то к содержимому РСМ прибавляется содержимое Р1, иначе из РСМ вычитается Р1 (прибавляется –Р1+1).

Шаг 7. Содержимое СЧТ уменьшается на 1. Если СЧТ $\neq 0$, переход к шагу 4, иначе к шагу 8.

Шаг 8. Выполняется циклический сдвиг влево регистра Р2 и флага CF на 1 бит.

Шаг 9. Проверяется знак последнего остатка от деления (поскольку числа беззнаковые, остаток должен быть положительным). Если CF=0, то производится восстановление остатка путем добавления Р1 к РСМ.

Рассмотрим пример деления X/Y по этому алгоритму в восьмиразрядном процессоре (табл. 4.7).

$X=251_{(8)}$; $Y=5_{(8)}$.

$X=10101001_{(2)}$; $Y=101_{(2)}$.

Для восьмиразрядного процессора получаем:

$X=10101001_{(2)}$; $Y=00000101_{(2)}$; $[-Y]_2=11111011$.

Проверим результат. Частное находится в регистре $P2=00100001_{(2)}=41_{(8)}$, а остаток в $РСМ=00000100_{(2)}=4_{(8)}$. Результат верен.

Таблица 4.7

Деление беззнаковых операндов в прямом коде X/Y, где $X = 251_{(8)}$, $Y = 5_{(8)}$

CF	РСМ								Р2								СЧТ				Комментарий
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	**Исходное состояние**
0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	(РСМ,Р2):=X; СЧТ:=8, CF:=0
0	1	1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	1	0	0	0	+(–Р1+1)
0	1	1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	1	0	0	0	Результат пробного вычитания
1	1	1	1	1	1	0	1	1	0	1	0	1	0	0	1	0	1	0	0	0	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	1	0	0	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	1	0	0	0	+Р1
0	1	1	1	1	1	1	0	0	0	1	0	1	0	0	1	0	1	0	0	0	Результат сложения
0	1	1	1	1	1	1	0	0	0	1	0	1	0	0	1	0	0	1	1	1	**СЧТ:=СЧТ – 1**
0	1	1	1	1	1	1	0	0	1	0	1	0	0	1	0	0	0	1	1	1	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	1	0	0	0	1	0	1	0	0	1	0	0	0	1	1	1	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	1	0	1	0	0	1	0	0	0	1	1	1	+Р1
0	1	1	1	1	1	1	0	1	1	0	1	0	0	1	0	0	0	1	1	1	Результат сложения
0	1	1	1	1	1	1	0	1	1	0	1	0	0	1	0	0	0	1	1	0	**СЧТ:=СЧТ – 1**
1	1	1	1	1	1	1	0	1	0	1	0	0	1	0	0	0	0	1	1	0	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0	1	1	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	0	1	0	0	1	0	0	0	0	1	1	0	+Р1
1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	Результат сложения
1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	1	**СЧТ:=СЧТ - 1**
0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	1	**(CF,P2):=LC((CF,P2),1)**
0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	1	**(CF,PCM):=LC((CF,PCM),1)**
0	1	1	1	1	1	0	1	1	1	0	0	1	0	0	0	1	0	1	0	1	+(–Р1+1)
0	1	1	1	1	1	0	1	1	1	0	0	1	0	0	0	1	0	1	0	1	Результат сложения

Окончание табл. 4.7

0	1	1	1	1	1	0	1	1	1	0	0	1	0	0	0	1	0	1	0	0	**СЧТ:=СЧТ – 1**
1	1	1	1	1	1	0	1	1	0	0	1	0	0	0	1	0	0	1	0	0	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	0	1	1	1	0	0	1	0	0	0	1	0	0	1	0	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	1	0	0	1	0	0	+P1
0	1	1	1	1	1	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	Результат сложения
0	1	1	1	1	1	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	**СЧТ:=СЧТ – 1**
0	1	1	1	1	1	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1	+P1
0	1	1	1	1	1	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1	Результат сложения
0	1	1	1	1	1	1	0	1	0	1	0	0	0	1	0	0	0	0	1	0	**СЧТ:=СЧТ – 1**
0	1	1	1	1	1	1	0	1	1	0	0	0	1	0	0	0	0	0	1	0	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	1	0	1	0	1	0	0	0	1	0	0	0	0	0	1	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	0	0	0	1	0	+P1
0	1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	1	0	Результат сложения
0	1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	1	**СЧТ:=СЧТ – 1**
1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	1	+P1
1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	Результат сложения
1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	**СЧТ:=СЧТ – 1**
0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	**(CF,P2):=LC((CF,P2),1)**
Так как после последнего сложения CF=1 (остаток положительный), то восстановления остатка не требуется.																					

4.6.2. Деление целых двоичных чисел в формате с ФТ в прямом коде

Операция деления в прямом коде выполняется над модулями операндов, а знак частного получается сложением по «модулю 2» знаковых разрядов делимого и делителя. Деление модуля делимого на модуль делителя выполняется по алгоритму деления беззнаковых чисел. Знак остатка должен совпадать со знаком делимого, так как остаток остается от делимого. Поэтому при СЧТ=0 сравнивается знак остатка со знаком делимого, и если они не совпадают, то к РСМ прибавляется делитель со знаком, противоположным знаку РСМ (называется «восстановление остатка»).

Рассмотрим пример деления X/Y по этому алгоритму в восьмиразрядном процессоре (табл. 4.8).

$X=251_{(8)}$; $Y=-5_{(8)}$.

$X=10101001_{(2)}$; $Y=-101_{(2)}$.

В формате байта получаем:

$X=10101001_{(2)}$; $Y=-00000101_{(2)}$; $[|Y|]_2=00000101$; $[-|Y|]_2=11111011$.

$[|X|]_2=0000000010101001$.

Проверим результат. Модуль частного находится в регистре $P2=00100001_{(2)}=41_{(8)}$, а остаток – в $РСМ=00000100_{(2)}=4_{(8)}$. Определяем знак частного:

знак Z= (знак X)^(знак У)=0^1=1.

$[Z]_1=10100001$; $Z=-0100001_{(2)}=-41_{(8)}$.

Результат верен.

Таблица 4.8

Деление в прямом коде операндов со знаком X/Y, где $X = 251_{(8)}$, $Y = -5_{(8)}$

CF	РСМ								Р2								СЧТ				Комментарий
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	**Исходное состояние**
0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	0	1	1	1	(РСМ,Р2):=$[\|X\|]_2$; СЧТ:=7, CF:=0
0	1	1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	1	1	1	+(–Р1+1)
0	1	1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	1	1	1	Результат пробного вычитания
1	1	1	1	1	1	0	1	1	0	1	0	1	0	0	1	0	0	1	1	1	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	0	1	1	1	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	0	1	1	1	+Р1
0	1	1	1	1	1	1	0	0	0	1	0	1	0	0	1	0	0	1	1	1	Результат сложения
0	1	1	1	1	1	1	0	0	0	1	0	1	0	0	1	0	0	1	1	0	**СЧТ:=СЧТ – 1**
0	1	1	1	1	1	1	0	0	1	0	1	0	0	1	0	0	0	1	1	0	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	1	0	0	0	1	0	1	0	0	1	0	0	0	1	1	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	1	0	1	0	0	1	0	0	0	1	1	0	+Р1
0	1	1	1	1	1	1	0	1	1	0	1	0	0	1	0	0	0	1	1	0	Результат сложения
0	1	1	1	1	1	1	0	1	1	0	1	0	0	1	0	0	0	1	0	1	**СЧТ:=СЧТ – 1**
1	1	1	1	1	1	1	0	1	0	1	0	0	1	0	0	0	0	1	0	1	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0	1	0	1	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	0	1	0	0	1	0	0	0	0	1	0	1	+Р1
1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	1	Результат сложения
1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	**СЧТ:=СЧТ – 1**
0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	**(CF,P2):=LC((CF,P2),1)**
0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	**(CF,PCM):=LC((CF,PCM),1)**
0	1	1	1	1	1	0	1	1	1	0	0	1	0	0	0	1	0	1	0	0	+(–Р1+1)

Окончание табл. 4.8

0	1	1	1	1	1	0	1	1	1	0	0	1	0	0	0	1	0	1	0	0	Результат сложения
0	1	1	1	1	1	0	1	1	1	0	0	1	0	0	0	1	0	0	1	1	**СЧТ:=СЧТ – 1**
1	1	1	1	1	1	0	1	1	0	0	1	0	0	0	1	0	0	0	1	1	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	0	1	1	1	0	0	1	0	0	0	1	0	0	0	1	1	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	1	0	0	0	1	1	+P1
0	1	1	1	1	1	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	Результат сложения
0	1	1	1	1	1	1	0	0	0	0	1	0	0	0	1	0	0	0	1	0	**СЧТ:=СЧТ – 1**
0	1	1	1	1	1	1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	0	+P1
0	1	1	1	1	1	1	0	1	0	1	0	0	0	1	0	0	0	0	1	0	Результат сложения
0	1	1	1	1	1	1	0	1	0	1	0	0	0	1	0	0	0	0	0	1	**СЧТ:=СЧТ – 1**
0	1	1	1	1	1	1	0	1	1	0	0	0	1	0	0	0	0	0	0	1	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	1	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	0	0	0	0	1	+P1
0	1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	1	Результат сложения
0	1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	**СЧТ:=СЧТ – 1**
1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	1	+P1
1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	Результат сложения
1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	**СЧТ:=СЧТ – 1**
0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	**(CF,P2):=LC((CF,P2),1)**
Так как после последнего сложения знак остатка совпадает со знаком делимого, то восстановления остатка не требуется.																					

4.6.3. Деление целых чисел в формате с ФТ в дополнительном коде

Этот алгоритм отличается от алгоритма деления целых чисел в формате с ФТ в прямом коде следующим:

1. Дополнительный код делимого делится на дополнительный код делителя. Частное и остаток получаются в дополнительном коде.

2. Проверка на переполнение при делении осуществляется сравнением знаков делимого и знака остатка при пробном вычитании. Если знаки не совпадают, то переполнения нет, в противном случае – переполнение.

3. Знак частного получается при пробном вычитании по правилу, по которому будет определяться и очередной бит частного.

4. Определение микрооперации в сумматоре в очередном цикле деления (сложение или вычитание) осуществляется в соответствии с табл. 4.9.

Таблица 4.9

Определение микрооперации в сумматоре

Знак РСМ	Знак Р1	Операция в сумматоре
0	0	РСМ+(¬Р1+1)
0	1	РСМ+Р1
1	0	РСМ+Р1
1	1	РСМ+(¬Р1+1)

5. Значение очередного бита частного Z_i определяется по логическому выражению: $Z_i=CF_i$^(Знак Р1).

6. По окончании цикла деления (СЧТ=0) может потребоваться коррекция частного. Коррекция частного (добавление единицы к младшему биту частного) требуется в случаях, определяемых табл. 4.10.

Таблица 4.10

Коррекция частного

Знак X	Знак Y	Коррекция
0	0	нет
0	1	есть
1	0	есть
1	1	нет

Рассмотрим пример деления X/Y по этому алгоритму в восьмиразрядном процессоре (табл. 4.11).

$X=2557_{(8)}$; $Y=-41_{(8)}$.

$X=10101101111_{(2)}$; $Y=-100001_{(2)}$.

Представляя операнды в формате процессора, получаем:

$X=0000010101101111_{(2)}$; $Y=-00100001_{(2)}$;

(РСМ, Р2)=$[X]_2$=0000010101101111.

Р1=$[Y]_2$=11011111.

(–Р1+1)=$[-Y]_2$=00100001.

Выполним проверку полученного результата.

$[Z]_2=11010110=-00101010_{(2)}=-52_{(8)}$;

[R]2=00000101=101(2)=$5_{(8)}$.

$X/Y=2557_{(8)}/(-41)_{(8)}=-52_{(8)}(5)_{(8)}$.

Результат верен.

Таблица 4.11

Деление в дополнительном коде X/Y, где $X = 2557_{(8)}$, $Y = -41_{(8)}$

8,	РСМ								Р2								СЧТ				Комментарий
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	**Исходное состояние**
0	0	0	0	0	0	1	0	1	0	1	1	0	1	1	1	1	1	0	0	0	(РСМ,Р2):=$[X]_2$; СЧТ:=8, CF:=0
0	1	1	0	1	1	1	1	1	0	1	1	0	1	1	1	1	1	0	0	0	+Р1
0	1	1	1	0	0	1	0	0	0	1	1	0	1	1	1	1	1	0	0	0	Результат пробного вычитания: знак остатка≠знак делимого, переполнения нет
0	1	1	1	0	0	1	0	0	1	1	0	1	1	1	1	1	1	0	0	0	**(CF,P2):=LS((CF,P2),1)∨(CF_i^(P1[0]))**
1	1	1	0	0	1	0	0	0	1	1	0	1	1	1	1	1	1	0	0	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	1	0	0	0	0	1	1	1	0	1	1	1	1	1	1	0	0	0	+(–Р1+1)
0	1	1	1	0	1	0	0	1	1	1	0	1	1	1	1	1	1	0	0	0	Результат сложения
0	1	1	1	0	1	0	0	1	1	1	0	1	1	1	1	1	0	1	1	1	**СЧТ:=СЧТ – 1**
1	1	1	1	0	1	0	0	1	1	0	1	1	1	1	1	1	0	1	1	1	**(CF,P2):=LS((CF,P2),1)∨(CF_i^(P1[0]))**
1	1	1	0	1	0	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	1	0	0	0	0	1	1	0	1	1	1	1	1	1	0	1	1	1	+(–Р1+1)
0	1	1	1	1	0	1	0	0	1	0	1	1	1	1	1	1	0	1	1	1	Результат сложения
0	1	1	1	1	0	1	0	0	1	0	1	1	1	1	1	1	0	1	1	0	**СЧТ:=СЧТ – 1**
1	1	1	1	1	0	1	0	0	0	1	1	1	1	1	1	1	0	1	1	0	**(CF,P2):=LS((CF,P2),1)∨(CF_i^(P1[0]))**
1	1	1	1	0	1	0	0	1	0	1	1	1	1	1	1	1	0	1	1	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	1	0	0	0	0	1	0	1	1	1	1	1	1	1	0	1	1	0	+(–Р1+1)
1	0	0	0	0	1	0	1	0	0	1	1	1	1	1	1	1	0	1	1	0	Результат сложения
1	0	0	0	0	1	0	1	0	0	1	1	1	1	1	1	1	0	1	0	1	**СЧТ:=СЧТ – 1**
0	0	0	0	0	1	0	1	0	1	1	1	1	1	1	1	0	0	1	0	1	**(CF,P2):=LS((CF,P2),1)∨(CF_i^(P1[0]))**
0	0	0	0	1	0	1	0	0	1	1	1	1	1	1	1	0	0	1	0	1	**(CF,PCM):=LC((CF,PCM),1)**
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	+Р1
0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	1	0	1	Результат сложения
0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	1	0	0	**СЧТ:=СЧТ – 1**
1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	0	1	0	1	0	0	**(CF,P2):=LS((CF,P2),1)∨(CF_i^(P1[0]))**

Окончание табл. 4.11

1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	1	0	1	0	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	1	0	0	0	0	1	1	1	1	1	1	1	0	1	0	1	0	0	+(–P1+1)
1	0	0	0	0	1	0	0	0	1	1	1	1	1	1	0	1	0	1	0	0	Результат сложения
1	0	0	0	0	1	0	0	0	1	1	1	1	1	1	0	1	0	0	1	1	**СЧТ:=СЧТ – 1**
1	0	0	0	0	1	0	0	0	1	1	1	1	1	0	1	0	0	0	1	1	**(CF,P2):=LS((CF,P2),1)∨(CF$_i$^(P1[0]))**
0	0	0	0	1	0	0	0	1	1	1	1	1	1	0	1	0	0	0	1	1	**(CF,PCM):=LC((CF,PCM),1)**
0	1	1	0	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	1	1	+P1
0	1	1	1	1	0	0	0	0	1	1	1	1	1	0	1	0	0	0	1	1	Результат сложения
0	1	1	1	1	0	0	0	0	1	1	1	1	1	0	1	0	0	0	1	0	**СЧТ:=СЧТ – 1**
1	1	1	1	1	0	0	0	0	1	1	1	1	0	1	0	1	0	0	1	0	**(CF,P2):=LS((CF,P2),1)∨(CF$_i$^(P1[0]))**
1	1	1	1	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	1	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	0	1	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	1	0	+(¬P1+1)
1	0	0	0	0	0	0	1	0	1	1	1	1	0	1	0	1	0	0	1	0	Результат сложения
1	0	0	0	0	0	0	1	0	1	1	1	1	0	1	0	1	0	0	0	1	**СЧТ:=СЧТ – 1**
1	0	0	0	0	0	0	1	0	1	1	1	0	1	0	1	0	0	0	0	1	**(CF,P2):=LS((CF,P2),1)∨(CF$_i$^(P1[0]))**
0	0	0	0	0	0	1	0	1	1	1	1	0	1	0	1	0	0	0	0	1	**(CF,PCM):=LC((CF,PCM),1)**
0	1	1	0	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	0	1	+P1
0	1	1	1	0	0	1	0	0	1	1	1	0	1	0	1	0	0	0	0	1	Результат сложения
0	1	1	1	0	0	1	0	0	1	1	1	0	1	0	1	0	0	0	0	0	**СЧТ:=СЧТ – 1**
1	1	1	1	0	0	1	0	0	1	1	0	1	0	1	0	1	0	0	0	0	**(CF,P2):=LS((CF,P2),1)∨(CF$_i$^(P1[0]))**
1	1	1	1	0	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	0	**коррекция частного (P2+1)**
1	0	0	1	0	0	0	0	1	1	1	0	1	0	1	1	0	0	0	0	0	**так как знак остатка не совпадает со знаком делимого, восстановление остатка (PCM+(–P1+1))**
0	0	0	0	0	0	1	0	1	1	1	0	1	0	1	1	0	0	0	0	0	Результат сложения
Конечное состояние регистров процессора после завершения алгоритма деления.																					

4.6.4. Деление дробных чисел в формате с ФТ

Алгоритм деления дробных чисел отличается от алгоритма деления целых чисел следующим:

1. При делении дробных чисел делимое не обязательно имеет удвоенную длину по отношению к делителю (может иметь такую же разрядность, как и делитель). Делимое записывается в регистры РСМ и Р2, начиная со старшего бита РСМ. Если делимое имеет одинарную длину, то оно записывается только в РСМ.

2. В общем случае деление дробных чисел представляет собой бесконечную операцию деления. В процессоре количество циклов деления определяется требуемой разрядностью частного (разрядностью цифрового процессора).

3. При делении дробных чисел процессор получает только частное, остаток не фиксируется.

4. Значение СЧТ устанавливается на 1 меньше, чем разрядность регистра частного.

Рассмотрим пример деления X/Y в прямом коде в пятиразрядном процессоре.

$X=0{,}264_{(8)}$; $Y=0{,}5_{(8)}$.

$X=0{,}010110100_{(2)}$; $Y=0{,}101_{(2)}$.

Представляя операнды в формате процессора, получаем:

$X=0010110100_{(2)}$; $Y=0{,}1010_{(2)}$;

$(PCM, P2)=[|X|]_2=0010110100$;

$P1=[|Y|]_2=01010$;

$[-|Y|]_2=10110$.

CF	PCM					P2					СЧТ			Комментарий
*	*	*	*	*	*	*	*	*	*	*	*	*	*	**Исходное состояние**
0	0	0	1	0	1	1	0	1	0	0	1	0	0	(PCM,P2):=$[X]_2$; СЧТ:=8, CF:=0
0	1	0	1	1	0	1	0	1	0	0	1	0	0	+(–P1+1)
0	1	1	0	1	1	1	0	1	0	0	1	0	0	Результат пробного вычитания
1	1	1	0	1	1	0	1	0	0	0	1	0	0	**(CF,P2):=LC((CF,P2),1)**

1	1	0	1	1	1	0	1	0	0	0	1	0	0	**(CF,PCM):= =LC((CF,PCM),1)**
1	0	1	0	1	0	0	1	0	0	0	1	0	0	+P1
1	0	0	0	0	1	0	1	0	0	0	1	0	0	Результат сложения
1	0	0	0	0	1	0	1	0	0	0	0	1	1	**СЧТ:=СЧТ – 1**
0	0	0	0	0	1	1	0	0	0	1	0	1	1	**(CF,P2):=LC((CF,P2),1)**
0	0	0	0	1	0	1	0	0	0	1	0	1	1	**(CF,PCM):= =LC((CF,PCM),1)**
0	1	0	1	1	0	1	0	0	0	1	0	1	1	+(–P1+1)
0	1	1	0	0	0	1	0	0	0	1	0	1	1	Результат сложения
0	1	1	0	0	0	1	0	0	0	1	0	1	0	**СЧТ:=СЧТ – 1**
1	1	1	0	0	0	0	0	0	1	0	0	1	0	**(CF,P2):=LC((CF,P2),1)**
1	1	0	0	0	1	0	0	0	1	0	0	1	0	**(CF,PCM):= =LC((CF,PCM),1)**
1	0	1	0	1	0	0	0	0	1	0	0	1	0	+P1
0	1	1	0	1	1	0	0	0	1	0	0	1	0	Результат сложения
0	1	1	0	1	1	0	0	0	1	0	0	0	1	**СЧТ:=СЧТ – 1**
0	1	1	0	1	1	0	0	1	0	0	0	0	1	**(CF,P2):=LC((CF,P2),1)**
1	1	0	1	1	0	0	0	1	0	0	0	0	1	**(CF,PCM):= =LC((CF,PCM),1)**
1	0	1	0	1	0	0	0	1	0	0	0	0	1	+P1
1	0	0	0	0	0	0	0	1	0	0	0	0	1	Результат сложения
1	0	0	0	0	0	0	0	1	0	0	0	0	0	**СЧТ:=СЧТ – 1**
0	0	0	0	0	0	0	1	0	0	1	0	0	0	**(CF,P2):=LC((CF,P2),1)**
Так как СЧТ=0, деление завершено														

Частное находится в P2, т.е. $Z=0{,}1001_{(2)}=0{,}44_{(8)}$.

В рассмотренном примере остаток от деления равен 0, поэтому частное получено точно, а в общем случае оно получается приближенно.

Проверим полученный результат:

$0{,}264_{(8)}/0{,}5_{(8)}=0{,}44_{(8)}(0)$.

4.6.5. Алгоритм деления чисел в формате с ПТ (формат КВ)

Так же, как и при сложении, умножении, операция деления выполняется раздельно над мантиссами и порядками. Операция Z=X/Y, где $X=M_x*2^{P_x}$, $Y=M_y*2^{P_y}$, выполняется как: $M_z=M_x/M_y$; $P_z=P_x–P_y$.

Алгоритм включает следующие шаги:

1. Операнды извлекаются из ОП с разделением на мантиссу (восстанавливается скрытый бит) и порядок.

2. Порядок частного получается вычитанием из содержимого регистра порядка делимого содержимого регистра порядка делителя и добавлением константы $127_{(10)}$. Если порядок частного P_z больше максимально допустимого порядка, то фиксируется переполнение при делении. Если порядок P_z меньше минимально допустимого порядка, то в зависимости от реализации процессора либо частное принимается равным 0, либо фиксируется «потеря значимости».

3. Мантисса делимого делится на мантиссу делителя в прямом коде по алгоритму деления дробных чисел в формате с ФТ (выравнивание порядков мантисс не требуется).

4. При выполнении деления мантисс на шаге пробного вычитания может оказаться, что деление не состоится из-за возникновения переполнения. В этом случае выполняется масштабирование мантиссы делимого, для чего в РСМ восстанавливается исходное значение мантиссы делимого добавлением к РСМ делителя. Затем мантисса делимого уменьшается в два раза путем сдвига вправо на один разряд, при этом порядок делимого увеличивается на 1. Затем выполняется деление мантисс. Если при масштабировании мантиссы делимого и увеличении порядка на единицу оказывается, что $P_x>P_{max}$, то фиксируется переполнение.

5. После завершения деления мантисс производится нормализация частного.

6. При делении чисел с плавающей точкой в процессоре фиксируется только частное.

7. Полученные мантисса частного (со скрытием старшего бита) и порядок частного объединяются в формат КВ и записываются в ОП.

Рассмотрим пример деления X/Y по данному алгоритму. В этом примере операнды имеют небольшие значения, их мантиссы помещаются в 8 бит. Тогда 25-битовые регистры мантисс условно будут сокращены до 8 бит.

Пример. $X/Y = -6406_{(8)}/54_{(8)}$.

$X = -110100000110_{(2)} = -1{,}10100000110*2^{11}$;

$Y = 101100_{(2)} = 1{,}01100*2^{5}$;

В формате КВ:

X=	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	байт								**байт**								**байт**								**байт**							
Y=	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	байт								**байт**								**байт**								**байт**							

Шаг 1. Операнды извлекаются из ОП (с восстановлением скрытого бита) и помещаются в ОА в регистры мантисс и порядков:

$РП_x$=	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**

$РМ_x$=	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

$РП_y$=	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**

$РМ_y$=	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Шаг 2. Определяется разность порядков $[РП_x+(-РП_y)]_2$:

$РП_x$=	1	0	0	0	1	0	1	1
$[-РП_y]_2$=	0	1	1	1	1	0	1	1
$РП_z=РП_x+(-РП_y)$=	0	0	0	0	0	1	1	0
$+127_{(10)}$ смещение	0	1	1	1	1	1	1	1
=	1	0	0	0	0	1	0	1

Так как порядок частного $P_{min}<=P_z<=P_{max}$, выполняется деление содержимого регистров мантисс.

Шаг 3. Деление мантисс (табл. 4.12).

(РСМ, Р2)=$[|M|]_x$=0110100000110000;

Р1=$[|M|]_y$=01011000;

–Р1+1=$[-|M|]_y$=10101000.

Р2=$|M_z|$=01001011.

Таблица 4.12

Деление мантисс в прямом коде

CF	PCM								P2								СЧТ				Комментарий
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	**Исходное состояние**
0	0	1	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	(PCM,P2):=[|M_x|]; СЧТ:=7, CF:=0
0	1	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	+(–P1+1)
1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	Результат пробного вычитания. Есть переполнение при делении мантисс
1	0	1	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	**Восстановление мантиссы (+P1)**
0	0	1	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	Результат сложения
0	0	0	1	1	0	1	0	0	0	0	1	1	0	0	0	0	0	1	1	1	**(PCM,P2):=AR((PCM,P2),1); P_z:=P_z+1**
0	1	0	1	0	1	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	+(–P1+1)
0	1	1	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	1	1	1	Результат пробного вычитания. Переполнения нет
0	1	1	0	1	1	1	0	0	0	0	1	1	0	0	0	0	0	1	1	1	**(CF,P2):=LC((CF,P2),1)**
1	1	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	**(CF,PCM):=LC((CF,PCM),1)**
1	0	1	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	+P1
1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	Результат сложения
1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	**СЧТ:=СЧТ – 1**
0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0	1	1	0	**(CF,P2):=LC((CF,P2),1)**
0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	1	0	**(CF,PCM):=LC((CF,PCM),1)**
0	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	1	0	1	1	0	+(–P1+1)
0	1	1	0	0	1	0	0	0	0	1	1	0	0	0	0	1	0	1	1	0	Результат сложения
0	1	1	0	0	1	0	0	0	0	1	1	0	0	0	0	1	0	1	0	1	**СЧТ:=СЧТ – 1**
0	1	1	0	0	1	0	0	0	1	1	0	0	0	0	1	0	0	1	0	1	**(CF,P2):=LC((CF,P2),1)**

Окончание табл. 4.12

1	1	0	0	1	0	0	0	0	1	1	0	0	0	0	1	0	0	1	0	1	**(CF,PCM):=LC((CF,PCM),1)**
1	0	1	0	1	1	0	0	0	1	1	0	0	0	0	1	0	0	1	0	1	+P1
0	1	1	1	0	1	0	0	0	1	1	0	0	0	0	1	0	0	1	0	1	Результат сложения
0	1	1	1	0	1	0	0	0	1	1	0	0	0	0	1	0	0	1	0	0	**СЧТ:=СЧТ – 1**
1	1	1	1	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	**(CF,P2):=LC((CF,P2),1)**
1	1	1	0	1	0	0	0	1	1	0	0	0	0	1	0	0	0	1	0	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	1	0	1	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	+P1
1	0	0	1	0	1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	0	Результат сложения
0	0	0	1	0	1	0	0	1	1	0	0	0	0	1	0	0	0	0	1	1	**СЧТ:=СЧТ – 1**
1	0	0	1	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	1	**(CF,P2):=LC((CF,P2),1)**
0	0	1	0	1	0	0	1	1	0	0	0	0	1	0	0	1	0	0	1	1	**(CF,PCM):=LC((CF,PCM),1)**
0	1	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	+(¬P1+1)
0	1	1	1	1	1	0	1	1	0	0	0	0	1	0	0	1	0	0	1	1	Результат сложения
0	1	1	1	1	1	0	1	1	0	0	0	0	1	0	0	1	0	0	1	0	**СЧТ:=СЧТ – 1**
0	1	1	1	1	1	0	1	1	0	0	0	1	0	0	1	0	0	0	1	0	**(CF,P2):=LC((CF,P2),1)**
1	1	1	1	1	0	1	1	0	0	0	0	1	0	0	1	0	0	0	1	0	**(CF,PCM):=LC((CF,PCM),1)**
1	0	1	0	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	+P1
1	0	1	0	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	1	0	Результат сложения
1	0	1	0	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	1	**СЧТ:=СЧТ – 1**
0	0	1	0	0	1	1	1	0	0	0	1	0	0	1	0	1	0	0	0	1	**(CF,P2):=LC((CF,P2),1)**
0	1	0	0	1	1	1	0	0	0	0	1	0	0	1	0	1	0	0	0	1	**(CF,PCM):=LC((CF,PCM),1)**
0	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	+(–P1+1)
1	0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	1	Результат сложения
1	0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	**СЧТ:=СЧТ – 1**
0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	1	1					**(CF,P2):=LC((CF,P2),1)**

Шаг 4. Определение знака мантиссы частного:

$(Зн\ M_x)^(Зн\ M_y)=1^0=1$.

Шаг 5. Частное равно:

$[PM_z]_1=11001011$;

$РП_z=10000110$ (был увеличен на 1 при делении мантисс на шаге пробного вычитания).

Так как старший числовой разряд мантиссы равен 1– нет нарушения нормализации.

Шаг 6. Далее объединяются порядок и мантисса результата (со скрытием старшего бита) в формат КВ и записываются в ОП.

Z=	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	байт								**байт**								**байт**								**байт**							

Переводим результат из формата КВ в восьмеричную систему счисления. Получаем:

$Z= – 1{,}001011*2^6= – 1001011_{(2)}= – 113_{(8)}$.

Проверяем результат:

$X/Y = – 113_{(8)}$.

Результат верен.

4.7. Ускорение операции умножения в цифровых процессорах

Операция умножения относится к так называемым длинным операциям, т.е. время ее выполнения намного больше времени выполнения коротких операций (сложения, вычитания). В то же время она часто используется при вычислениях. Поэтому разработаны различные способы сокращения времени выполнения операции умножения. Рассмотрим эти способы.

Первый способ. Время выполнения операции умножения можно сократить за счет операций сдвига. Для этого в состав арифметического устройства включают дополнительное оборудование, которое позволяет производить сдвиги на произвольное количество битов за один такт, что сокращает время, если в множителе встречаются следующие двоичные комбинации:

а) если встречается комбинация 0000…0 (количество 0 равно m), то дополнительное оборудование позволяет сдвинуть частичное произведение и множитель на m битов за один такт (вместо m тактов);

б) если встречается комбинация 01111…1 (количество 1 равно m), тогда она преобразуется в 1000…0 (количество 0 равно m), и в этом случае из частичного произведения вычитается множимое, а затем производится сдвиг на m битов (тратится два такта вместо m).

Второй способ. Использование аппаратной реализации матричного метода умножения. В приведенном ниже примере показано умножение четырехбитовых операндов $Z=X*У$, где $X=x_1x_2x_3x_4$ и $У=y_1y_2y_3y_4$

			x_1	x_2	x_3	x_4
		*	y_1	y_2	y_3	y_4
			x_1*y_4	x_2*y_4	x_3*y_4	x_4*y_4
		x_1*y_3	x_2*y_3	x_3*y_3	x_4*y_3	
	x_1*y_2	x_2*y_2	x_3*y_2	x_4*y_2		
x_1*y_1	x_2*y_1	x_3*y_1	x_4*y_1			
z_7	z_6	z_5	z_4	z_3	z_2	z_1

Поскольку произведение x_i*y_i равно конъюнкции этих битов, то можно построить комбинационную схему, реализующую умножение (рис. 4.13).

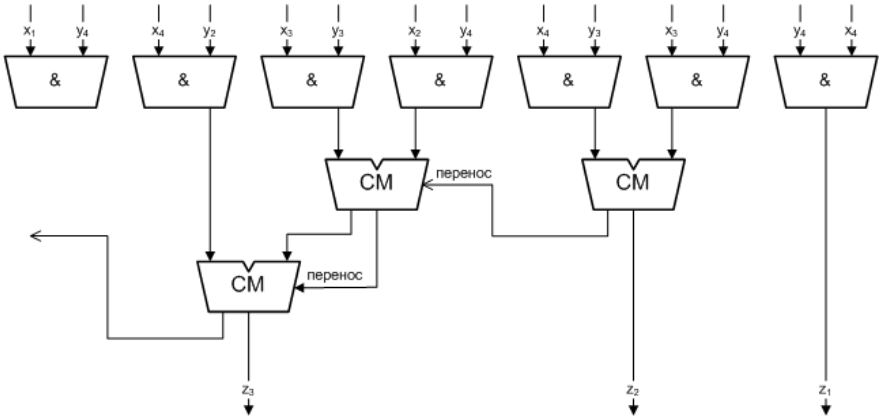

Рис. 4.13. Аппаратная реализация матричного метода умножения

Третий способ состоит в одновременном умножении на два разряда множителя. Алгоритм этого умножения следующий:

1. В СЧТ устанавливается значение, равное половине количества битов множителя, на которые производится умножение.

2. Если очередные два бита множителя, на которые производится умножение, равны 0 (комбинация 00), то производится сдвиг на два разряда вправо частичного произведения и множителя.

3. Если очередные два бита множителя, на которые производится умножение, равны 01, то к частичному произведению прибавляется множимое, а затем производится сдвиг на два разряда вправо частичного произведения и множителя.

4. Если очередные два бита множителя, на которые производится умножение, равны 10, то к регистру РСМ прибавляется удвоенное множимое, а затем производится сдвиг на два разряда вправо частичного произведения и множителя.

5. Если очередные два бита множителя, на которые производится умножение, равны 11, то к регистру РСМ прибавляется отрицательное множимое, а затем производится сдвиг на два разряда вправо частичного произведения и множителя, т.е. $11_{(2)}=3_{(10)}$ преобразуется в $4_{(10)}=100_{(2)}$. При этом следующая комбинация двух битов множителя учитывает добавляемую 1.

Рассмотрим пример выполнения такого алгоритма.

Выполнить в дополнительном коде умножение X*У в девятиразрядном процессоре (крайний левый разряд – знаковый) ускоренным способом на два разряда множителя.

$X=112_{(8)}=1001010_{(2)}$;

$Y=307_{(8)}=11000111_{(2)}$;

$[X]_2=001001010$;

$[Y]_2=011000111$;

$[-X]_2=110110110$;

$[2*X]_2=010010100$.

CF	РСМ										Р2									СЧТ	Комментарий
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	**Исходное состояние**
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	100	**CF:=0; РСМ:=0; Р2:=$[Y]_2$; СЧТ=4**
	1	1	0	1	1	0	1	1	0	0	0	1	1	0	0	0	1	1	1	100	**+(– Р1)**
0	1	1	0	1	1	0	1	1	0	0	0	1	1	0	0	0	1	1	1	100	**РСМ:=РСМ+(– Р1)**
0	1	1	1	1	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	100	**(CF,РСМ,Р2):=AR((CF,РСМ,Р2),2)**
0	1	1	1	1	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	011	**СЧТ:=СЧТ – 1**
0	0	1	0	0	1	0	1	0	0	1	1	0	0	1	1	0	0	0	1	011	**+(2*Р1), так как 01+1=10**
1	0	1	0	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	1	011	**РСМ:=РСМ+(2*Р1)**
0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	011	**(CF,РСМ,Р2):=AR((CF,РСМ,Р2),2)**
0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	010	**СЧТ:=СЧТ – 1**
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	1	1	010	**(CF,РСМ,Р2):=AR((CF,РСМ,Р2),2)**
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	1	1	001	**СЧТ:=СЧТ – 1**
0	1	1	0	1	1	0	1	1	0	0	0	0	0	1	1	0	0	1	1	001	**+(– Р1)**
0	1	1	0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	1	1	001	**РСМ:=РСМ+(– Р1)**
0	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	001	**(CF,РСМ,Р2):=AR((CF,РСМ,Р2),2)**
0	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	000	**СЧТ:=СЧТ – 1**
0	0	0	1	0	0	1	0	1	0	1	0	0	0	0	0	1	1	0	0	000	**+Р1**
1	0	0	0	1	1	1	0	0	1	1	0	0	0	0	0	1	1	0	0	000	**РСМ:=РСМ+Р1**
1	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0	1	1	0	0	0	**n+1 сдвиг**
Результат= $0000111001100001 10_{(2)}$=$1110011000011 0_{(2)}$=$34606_{(8)}$ верен																					

Следует отметить (такой случай есть в вышеприведенном примере), что если при значении СЧТ=0 в Р2 нужно учесть дополнительную 1 от предыдущей комбинации 11, то к РСМ прибавляется множимое без последующего сдвига; затем делается «n сдвиг», если умножаются целые операнды, а если дробные – обнуляется последний бит Р2.

4.8. Выполнение арифметических операций в цифровых процессорах с многократной точностью

Во всех ранее рассмотренных алгоритмах арифметических и связанных с ними логических операций и операций сдвига цифровой процессор обрабатывал операнд одиночной длины, т.е. длины, равной разрядности процессора. Операция в этом случае выполняется одной командой процессора за один такт его работы (сложение, вычитание, сдвиги, логические операции) или за несколько тактов (умножение, деление). Но можно выполнять обработку операндов, разрядность которых кратна разрядности (двух, трех, и т.д.) процессора. Например, сложение операндов двойной длины будет выполнено не одной командой, а двумя, и затрачено будет на это два такта. Такую арифметику будем называть арифметикой многократной точности. Для поддержки такой арифметики в процессоре есть специальные команды. Например, для сложения в процессорах фирмы Intel есть команды ADD <регистр источник>,<регистр приемник> и ADC <регистр источник >,<регистр приемник>. Первая команда складывает регистр источника с регистром приемником. Результат сложения запоминается в регистре приемнике, а значение переноса из старшего разряда суммы запоминается во флаге CF. Вторая команда складывает регистр источника с регистром приемником и с флагом переноса. Результат сложения запоминается в регистре приемнике, а значение переноса из старшего разряда суммы запоминается во флаге CF. Используя эти две команды друг за другом можно по «цепочке» складывать операнды любой длины кратной разрядности процессора.

Рассмотрим примеры сложения и вычитания в арифметике с многократной точностью. Пусть процессор, работающий в дополнительном коде, имеет разрядность четыре и крайний левый разряд знаковый. Выполнить операции X+Y и X –Y над восьмиразрядными операндами, у которых крайний левый разряд знаковый. Пусть операнды X и Y хранятся соответственно в следующих регистрах процессора:

PX1 – младшие 4 бита;

PX2 – старшие 4 бита;

PY1 – младшие 4 бита;

PY2 – старшие 4 бита.

Результат сложения записывается в PY.

Пример 1. X+Y,

где $X=73_{(8)}=00111011_{(2)}$;

$Y=52_{(8)}=00101010_{(2)}$;

$[X]_2=00111011_{(2)}$;

$[Y]_2=00101010_{(2)}$.

Тогда PX1=1011;

PX2=0011;

PY1=1010;

PY2=0010.

Такты	CF	Слагаемые операнды				Комментарий
1 (ADD)	*	1	0	1	1	PX1
	*	1	0	1	0	PY1
	1	0	1	0	1	результат сложения в PY1
2 (ADC)	1	0	0	1	1	PX2
	1	0	0	1	0	PY2
	1				1	CF
	0	0	1	1	0	результат сложения в PY2

SF=0; CF=0; ZF=0; OF=0.

Сумма находится в $[(PY2,PY1)]_2=01100101=01100101_{(2)}=145_{(8)}$. Результат верен.

Пример 2. X – Y=X+(– Y),

где $X=73_{(8)}=00111011_{(2)}$;

$Y=52_{(8)}=00101010_{(2)}$;

$[X]_2=00111011_{(2)}$;

$[– Y]_2=11010110_{(2)}$;

Тогда PX1=1011;

PX2=0011;

PY1=0110;

PY2=1101.

Такты	CF	Слагаемые операнды				Комментарий
1	*	1	0	1	1	PX1
	*	0	1	1	0	PY1
	1	0	0	0	1	результат сложения в PY1
2	1	0	0	1	1	PX2
	1	1	1	0	1	PY2
	1				1	CF
	1	0	0	0	1	результат сложения в PY2

SF=0; CF=1; ZF=0; OF=0.

Разность находится в $[(PY2,PY1)]_2=00010001=00010001_{(2)}=21_{(8)}$. Результат верен.

Пример 3. X+Y,

где $X=73_{(8)}=00111011_{(2)}$;

$Y=121_{(8)}=01010001_{(2)}$;

$[X]_2=00111011_{(2)}$;

$[Y]_2=01010001_{(2)}$.

Тогда PX1=1011;

PX2=0011;

PY1=0001;

PY2=0101.

Такты	CF	Слагаемые операнды				Комментарий
1	*	1	0	1	1	PX1
	*	0	0	0	1	PY1
	0	1	1	0	0	результат сложения в PY1
2	1	0	0	1	1	PX2
	1	0	1	0	1	PY2
	1				0	CF
	0	1	0	0	0	результат сложения в PY2

SF=1; CF=0; ZF=0; OF=1.

Возникло переполнение, которое обнаружено по значению цепей переноса из старшего числового разряда в знаковый и из знакового при сложении РХ2 и РУ2.

Рассмотрим алгоритм умножения в арифметике с многократной точностью. Пусть надо умножить восьмиразрядные операнды X и Y на четырехразрядном процессоре.

Обозначим младшие четыре разряда операндов соответственно через X_1 и Y_1, а старшие – через X_2 и Y_2 :

X=

x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
X_2				X_1			

Y=

y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
Y_2				Y_1			

Представим X и Y следующим образом:

$X=X_1+2^4*X_2$;

$Y=Y_1+2^4*Y_2$.

Тогда умножение X на Y можно представить как

$$X*Y=(X_1+2^4*X_2)*(Y_1+2^4*Y_2)=X_1*Y_1+X_1*Y_2*2^4+X_2*Y_1*2^4+X_2*Y_2*2^8.$$

Алгоритм умножения по этому выражению будет следующий:

1. Получить произведения: X_1*Y_1, X_1*Y_2, X_2*Y_1, X_2*Y_2. Каждое из них – восьмибитовое, будет храниться в двух регистрах (или ячейках памяти) процессора.

2. Просуммировать их значения в соответствии с вышеприведенным выражением. Схематично это будет выглядеть следующим образом:

Такты	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Комментарий
1													x	x	x	x	$+X_1*Y_1$ (мл. часть)
2									x	x	x	x					$+X_1*Y_1$ (ст. часть)
3									x	x	x	x					$+X_1*Y_2$ (мл. часть)
4									x	x	x	x					$+X_2*Y_1$ (мл. часть)
5					x	x	x	x									$+X_1*Y_2$

																	(ст. часть)
6					x	x	x	x									$+X_2*Y_1$ (ст. часть)
7					x	x	x	x									$+X_2*Y_2$ (мл. часть)
8	x	x	x	x													$+X_2*Y_2$ (ст. часть)

Рассмотрим конкретный пример.

$X=00100100_{(2)}$, $Y=01010010_{(2)}$.

$X_1=0100_{(2)}$;

$Y_1=0010_{(2)}$;

$X_2=0010_{(2)}$;

$Y_2=0101_{(2)}$;

1. Получаем произведения: X_1*Y_1, X_1*Y_2, X_2*Y_1, X_2*Y_2:

$X_1*Y_1=00001000_{(2)}$;

$X_1*Y_2=00010100_{(2)}$;

$X_2*Y_1=00000100_{(2)}$;

$X_2*Y_2=00001010_{(2)}$.

2. Просуммируем эти значения в соответствии с вышеприведенным выражением.

Такты	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Комментарий
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Исходное состояние
1													1	0	0	0	$+X_1*Y_1$ (мл. часть)
	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	результат суммирования
2									0	0	0	0					$+X_1*Y_1$ (ст. часть)
	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	результат суммирования
3									0	1	0	0					$+X_1*Y_2$ (мл. часть)

	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	результат суммирования
4									0	1	0	0					$+X_2*Y_1$ (мл. часть)
	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	результат суммирования
5					0	0	0	1									$+X_1*Y_2$ (ст. часть)
	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	результат суммирования
6					0	0	0	0									$+X_2*Y_1$ (ст. часть)
	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	результат суммирования
7					1	0	1	0									$+X_2*Y_2$ (мл. часть)
	0	0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	результат суммирования
8	0	0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	$+X_2*Y_2$ (ст. часть)
	0	0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	результат суммирования

Результат верен.

Следует отметить, что для алгоритма деления в арифметике с многократной точностью нет аналогичного умножению способа сократить количество операций вычитания и сдвигов. Поэтому деление с многократной точностью выполняется с более значительными аппаратными или программными затратами.

4.9. Операции сложения и вычитания в BCD-кодах

Сложение в BCD-кодах выполняется суммированием тетрад слагаемых, имеющих одинаковый десятичный вес в двоичной системе счисления. При этом учитывается (прибавляется) перенос, возникающий при сложении тетрад слагаемых предыдущего десятичного разряда. После сложения должна корректироваться тетрада суммы

(коррекция тетрады). Величина коррекции определяется из следующих положений:

1. Тетрада суммы должна получаться по тем же правилам, что и тетрады исходных слагаемых.

2. Тетрада суммы должна удовлетворять двум условиям (X_i – обозначает i-й десятичный разряд):

а) $t(X_i)+t(Y_i)+CF_{i-1}=t(X_i+Y_i+CF_{i-1})$, если $X_i+Y_i+CF_{i-1}\leq 9_{(10)}$;

б) $t(X_i)+t(Y_i)+CF_{i-1}=t(X_i+Y_i+CF_{i-1}-10_{(10)})+16_{(10)}$, если $X_i+Y_i+CF_{i-1}>9_{(10)}$.

Исходя из этих условий рассмотрим, какая коррекция должна быть **в коде 8421.**

Из условия а) для кода 8421 следует, что:

$$t(X_i)+t(Y_i)+CF_{i-1}=t(X_i+Y_i+CF_{i-1})+K,$$

где К – величина коррекции.

Чтобы выполнялось условие а), К должно равняться 0 (т.е. тетрада суммы остается без изменения). Рассмотрим пример.

$4_{(10)}+5_{(10)}=9_{(10)}$. Так как операнды одноразрядные, то $CF_{i-1}=0$.

Перенос в (i+1)-тетраду	(i)-тетрада				Комментарий
	0	1	0	0	1-е слагаемого – $4_{(10)}$
	0	1	0	1	2-е слагаемого – $5_{(10)}$
	0	0	0	0	CF_{i-1} перенос из предыдущей тетрады равен 0
0	1	0	0	1	сумма – $9_{(10)}$ и перенос 0 в следующую тетраду
	1	0	0	1	поскольку выполняется условие а), сумма $(X_i+Y_i+CF_{i-1})^{8421}\leq(1001_{(2)}=9_{(10)})$, коррекция тетрады не нужна

Из условия б) для кода 8421 следует, что:

$$t(X_i)+t(Y_i)+CF_{i-1}=t(X_i+Y_i+CF_{i-1}-10)+10+K,$$

где К – величина коррекции.

Чтобы выполнялось условие б), К+10 должно равняться 16. К+10=16, откуда получаем, что К=6 (т.е. значение тетрады увеличивается на $110_{(2)}$).

Пример 1. $7_{(10)}+6_{(10)}=13_{(10)}$

Перенос в (i+1)-тетраду	(i)-тетрада				Комментарий
	0	1	1	1	1-е слагаемого – $7_{(10)}$
	0	1	1	0	2-е слагаемого – $6_{(10)}$
	0	0	0	0	CF_{i-1} перенос из предыдущей тетрады равен 0
0	1	1	0	1	сумма и перенос 1 в следующую тетраду
	0	1	1	0	поскольку выполняется условие б), сумма $(X_i+Y_i+CF_{i-1})^{8421}>(1001_{(2)}=9_{(10)})$, коррекция тетрады равна $110_{(2)}$
1	0	0	1	1	значение тетрады и переноса после коррекции, т.е. сумма в этом разряде равна 3 и перенос, равный 1, в следующую тетраду

Пример 2. $7_{(10)}+9_{(10)}=16_{(10)}$

Перенос в (i+1)-тетраду	(i)-тетрада				Комментарий
	0	1	1	1	1-е слагаемого – $7_{(10)}$
	1	0	0	1	2-е слагаемого – $9_{(10)}$
	0	0	0	0	CF_{i-1} перенос из предыдущей тетрады равен 0
1	0	0	0	0	сумма и перенос 1 в следующую тетраду
	0	1	1	0	поскольку выполняется условие б), сумма с учетом переноса $(X_i+Y_i+CF_{i-1})^{8421}>(1001_{(2)}=9_{(10)})$, коррекция тетрады равна $110_{(2)}$
1	0	1	1	0	значение тетрады и переноса после коррекции, т.е. сумма в этом разряде равна 6 и перенос, равный 1, в следующую тетраду

Таким образом, цифровой процессор, работающий в BCD-коде 8421, определяет необходимость коррекции ($+110_{(2)}$) в тетраде по одному из двух признаков:

- значение тетрады суммы больше $1001_{(2)}$;
- значение переноса из тетрады суммы равно 1.

Теперь рассмотрим, какая коррекция должна быть **в коде 8421+3.**

Из условия а) для кода 8421+3 следует, что (рассматриваем тетрады как двоичный код):

$t(X_i)+t(Y_i)+CF_{i-1}=t(X_i+3)+t(Y_i+3)+CF_{i-1}=t(X_i+Y_i+CF_{i-1}+3)+3+K$,

где К – величина коррекции.

Чтобы выполнялось условие а), К+3 должно равняться 0, откуда получаем, что К= – 3 (т.е. значение тетрады уменьшается на $11_{(2)}$).

Пример. $4_{(10)}+5_{(10)}=9_{(10)}$

Перенос в (i+1)-тетраду	(i)-тетрада				Комментарий
	0	1	1	1	1-е слагаемого – $4_{(10)}$
	1	0	0	0	2-е слагаемого – $5_{(10)}$
	0	0	0	0	CF_{i-1} перенос из предыдущей тетрады равен 0
0	1	1	1	1	сумма и перенос в следующую тетраду
	1	1	1	1	поскольку для кода 8421+3 выполняется условие а), сумма $(X_i+Y_i+CF_{i-1})^{8421+3} \leq (1100_{(2)}= =9_{(10)})$, то коррекция тетрады равна $11_{(2)}$
	1	1	0	1	коррекция: прибавление $-0011_{(2)}=[1101]_2$. Обратите внимание, что при замене вычитания $0011_{(2)}$ на сложение с $[1101]_2$ перенос в следующую тетраду блокируется
	1	1	0	0	результат равен $9_{(10)}$

Из условия б) для кода 8421+3 следует, что:

$t(X_i)+t(Y_i)+CF_{i-1}=t(X_i+3)+t(Y_i+3)+CF_{i-1}=t(X_i+Y_i+3+CF_{i-1}-10)+3+10+K$,

где К – величина коррекции.

Чтобы выполнялось условие б), К+10+3 должно равняться 16. К+10+3=16, откуда получаем, что К=3 (т.е. значение тетрады увеличивается на $11_{(2)}$).

Цифровой процессор, работающий в BCD-коде 8421+3, определяет необходимость коррекции ($-11_{(2)}$ или $+11_{(2)}$) в тетраде суммы соответственно по одному из двух признаков:

– из тетрады суммы перенос равен 0;

– из тетрады суммы перенос равен 1.

Пример. $7_{(10)}+6_{(10)}=13_{(10)}$

Перенос в (i+1)-тет-	(i)-тетрада	Комментарий

раду					
	1	0	1	0	1-е слагаемого – $7_{(10)}$
	1	0	0	1	2-е слагаемого – $6_{(10)}$
	0	0	0	0	CF_{i-1} перенос из предыдущей тетрады равен 0
1	0	0	1	1	сумма и перенос 1 в следующую тетраду
	0	0	1	1	поскольку выполняется условие (б), сумма $(X_i+Y_i+CF_{i-1}))^{8421+3}>(1100_{(2)}=9_{(10)})$, коррекция значения тетрады равна $11_{(2)}$
1	0	1	1	0	значение тетрады и переноса после коррекции, т.е. сумма в этом разряде равна 3 и перенос, равный 1, в следующую тетраду

4.9.1. Получение дополнительного кода двоично-десятичных чисел

Выполнение арифметических операций сложения и вычитания в BCD-кодах будем рассматривать только в дополнительном коде, а умножения и деления – только в прямом. Прямой код BCD-операнда получается так же, как и для двоичных чисел, т.е. состоит из знака и модуля операнда.

Получение дополнительного кода операндов рассмотрим подробно для кодов 8421 и 8421+3.

Дополнительный код операнда в коде 8421 получается по следующему алгоритму.

1. Если операнд положительный, его код совпадает с самим операндом.

2. Если операнд отрицательный, то поскольку код 8421 не обладает свойством дополнения, получение дополнительного кода в цифровом процессоре происходит следующим образом:

– выполняется инверсия каждого бита каждой тетрады;

– из каждой тетрады вычитается $110_{(2)}$;

– в младший бит младшей тетрады добавляется 1.

Рассмотрим примеры получения дополнительного кода для шестнадцатиразрядного процессора, работающего в формате целых чисел в коде 8421, в котором под знак отводится одна тетрада, а под операнд – три тетрады.

$X=264_{(10)}$.

$[X^{8421}]_2=$

Знак	Тетрады											
0000	0	0	1	0	0	1	1	0	0	1	0	0

$Y= - 264$.

$[Y^{8421}]_2=$

Знак	Тетрады												Комментарий
1111	0	0	1	0	0	1	1	0	0	1	0	0	$[Y^{8421}]_1$
1111	1	1	0	1	1	0	0	1	1	0	1	1	инверсия тетрад
	1	0	1	0	1	0	1	0	1	0	1	0	вычитание $110_{(2)}$, прибавлением $1010=$ $=[-0110]_{(2)}$, перенос блокируется
1111	0	1	1	1	0	0	1	1	0	1	0	1	результат вычитания
												1	прибавление 1 в младшую тетраду
1111	0	1	1	1	0	0	1	1	0	1	1	0	$[Y^{8421}]_2$

Выполним проверку полученного результата. Так как процессор работает по модулю $1000_{(10)}$, то дополнением $264_{(10)}$ до $1000_{(10)}$ будет $736_{(10)}$. Результат верен.

Перевод отрицательного операнда из дополнительного кода в десятичную систему счисления выполняется по тому же правилу. Рассмотрим пример перевода операнда Y из дополнительного кода в десятичную систему счисления. Процессор работает в формате целых чисел (под знак отводится одна тетрада, а под разряды – три тетрады).

Пример.$[Y^{8421}]_2=$

1	1	1	1	0	1	1	1	0	0	1	1	0	1	1	0

Перевод:

Знак	Тетрады												Комментарий
1111	0	1	1	1	0	0	1	1	0	1	1	0	
1111	1	0	0	0	1	1	0	0	1	0	0	1	инверсия тетрад

	1	0	1	0	1	0	1	0	1	0	1	0	вычитание $110_{(2)}$, прибавлением $1010=[-0110]_{(2)}$, перенос блокируется
1111	0	0	1	0	0	1	1	0	0	0	1	1	результат вычитания
												1	прибавление 1 в младшую тетраду
–	0	0	1	0	0	1	1	0	0	1	0	0	$-264_{(10)}$

$Y= -264_{(10)}$.

Дополнение операнда в коде 8421+3 получается по следующему алгоритму. Так как этот код обладает свойством дополнения, то получение дополнительного кода отрицательного числа выполняется по общему правилу: инверсия всех числовых битов операнда с добавлением 1 в младший бит младшей тетрады.

Рассмотрим примеры получения дополнительного кода для процессора, работающего в формате целых чисел, в котором под знак отводится одна тетрада, а под операнд – три тетрады.

$X=264_{(10)}$.

$[X^{8421+3}]_2=$

Знак	Тетрады											
0000	0	1	0	1	1	0	0	1	0	1	1	1

$Y= -264$.

$[Y^{8421+3}]_2=$

Знак	Тетрады												Комментарий
1111	0	1	0	1	1	0	0	1	0	1	1	1	$[Y^{8421+3}]_1$
1111	1	0	1	0	0	1	1	0	1	0	0	0	инверсия
												1	добавление единицы
1111	1	0	1	0	0	1	1	0	1	0	0	1	$[Y^{8421+3}]_2$

Есть особый случай в алгоритме перевода в дополнительный код отрицательного операнда, оканчивающегося на один или несколько нулей в BCD-коде 8421+3. В этом случае после инверсии всех разрядов числовых тетрад к тетрадам добавляется $00\ldots1_{(10)}= (0011)(0011)\ldots ...(0100)_{(2)} =[00\ldots1]^{8421+3}$.

Пример. $Y= -260$ (операнд имеет 0 в младшем разряде).

$[Y^{8421+3}]_2$=

Знак	Тетрады												Комментарий
1111	0	1	0	1	1	0	0	1	0	0	1	1	$[Y^{8421+3}]_1$
1111	1	0	1	0	0	1	1	0	1	1	0	0	инверсия
	0	0	1	1	0	0	1	1	0	1	0	0	добавление $(001)_{10}^{8421+3}$
								1	0	0	0	0	результат сложения, перенос не блокируется
									0	0	1	1	так как перенос из тетрады равен 1, коррекция +0011
				0	1	1	1	0	0	0	1	1	сложение в средней тетраде
					1	1	0	1					так как перенос из тетрады равен 0, коррекция (–0011), перенос блокируется
					0	1	1	1					результат в средней тетраде
	1	1	0	1									сложение в старшей тетраде
	1	1	0	1									так как перенос из тетрады равен 0, коррекция (–0011), перенос блокируется
	1	0	1	0									результат в старшей тетраде
1111	1	0	1	0	0	1	1	1	0	0	1	1	$[Y^{8421+3}]_2$

Алгоритм перевода отрицательного операнда из дополнительного кода в десятичную систему счисления такой же: инверсия всех числовых битов операнда с добавлением 1 в младший бит младшей тетрады (с учетом особого случая, когда операнд заканчивается нулями).

4.9.2. Алгоритм сложения (вычитания) целых беззнаковых чисел в BCD-кодах

Сложение в BCD-кодах выполняется последовательным способом, т.е. начиная с тетрады младшего разряда. После сложения тет-

рад одинакового веса в тетраде суммы выполняется коррекция, величина которой зависит от используемого кода. Как и ранее, будем рассматривать процессор, не имеющий вычитателя, в котором операция вычитания заменяется сложением с операндом, взятым с противоположным знаком.

Рассмотрим примеры сложения в дополнительном коде операндов в BCD-кодах.

Пример 1. Сложить X+Y в процессоре, работающем в коде 8421 в формате целых беззнаковых чисел, имеющем три тетрады.

$X=645_{(10)}$; $[X^{8421}]_2=011001000101$;

$Y=275_{(10)}$; $[Y^{8421}]_2=001001110101$.

CF	Тетрада 3				Тетрада 2				Тетрада 1				Комментарий
*	0	1	1	0	0	1	0	0	0	1	0	1	$[X^{8421}]_2$
*	0	0	1	0	0	1	1	1	0	1	0	1	$[Y^{8421}]_2$
*									1	0	1	0	сложение младших тетрад $X_1+Y_1+CF_0$
*									0	1	1	0	так как $(X_1+Y_1+CF_0)^{8421}>$ $>(1001_{(2)}=9_{(10)})$, коррекция $+0110_{(2)}$
								1	0	0	0	0	результат коррекции
*				0	1	1	0	0	0	0	0	0	сложение средних тетрад $X_2+Y_2+CF_1$
*					0	1	1	0	0	0	0	0	так как $(X_2+Y_2+CF_1)^{8421}>$ $>(1001_{(2)}=9_{(10)})$, коррекция $+0110_{(2)}$
*				1	0	0	1	0	0	0	0	0	результат коррекции
0	1	0	0	1	0	0	1	0	0	0	0	0	сложение старших тетрад $X_3+Y_3+CF_2$
0	1	0	0	1	0	0	1	0	0	0	0	0	так как $(X_2+Y_2+CF_1)^{8421}<$ $<(1001_{(2)}=9_{(10)})$, то коррекция не нужна, результат, равный $920_{(10)}$, верен

Пример 2. Выполнить вычитание X–Y в процессоре, работающем в коде 8421 в формате целых беззнаковых чисел, имеющем три тетрады.

$X=645_{(10)}$; $[X^{8421}]_2=011001000101$;

$Y=275_{(10)}$; $[Y^{8421}]_2$=001001110101.

CF	Тетрада 3				Тетрада 2				Тетрада 1				Комментарий
*	0	1	1	0	0	1	0	0	0	1	0	1	$[X^{8421}]_2$
*	0	1	1	1	0	0	1	0	0	1	0	1	$[-Y^{8421}]_2$
*								0	1	0	1	0	сложение младших тетрад $X_1+Y_1+CF_0$
*									0	1	1	0	так как $(X_1+Y_1+CF_0)^{8421}>$ $>(1001_{(2)}=9_{(10)})$, коррекция $+0110_{(2)}$
*								1	0	0	0	0	результат коррекции
*				0	0	1	1	1	0	0	0	0	сложение средних тетрад $X_2+Y_2+CF_1$
*				0	0	1	1	1	0	0	0	0	так как $(X_2+Y_2+CF_1)^{8421}<$ $<(1001_{(2)}=9_{(10)})$, коррекция не нужна
0	1	1	0	1	0	1	1	1	0	0	0	0	сложение старших тетрад $X_3+Y_3+CF_2$
	0	1	1	0	0	1	1	1	0	0	0	0	так как $(X_1+Y_1+CF_0)^{8421}>$ $>(1001_{(2)}=9_{(10)})$, коррекция $+0110_{(2)}$
1	0	0	1	1	0	1	1	1	0	0	0	0	результат коррекции
1	0	0	1	1	0	1	1	1	0	0	0	0	результат, равный $370_{(10)}$, верен

Пример 3. Выполнить сложение X+Y в процессоре, работающем в коде 8421+3 в формате целых беззнаковых чисел, имеющем три тетрады.

$X=645_{(10)}$; $[X^{8421+3}]_2$=100101111000;

$Y=275_{(10)}$; $[Y^{8421+3}]_2$=010110101000.

CF	Тетрада 3				Тетрада 2				Тетрада 1				Комментарий
*	1	0	0	1	0	1	1	1	1	0	0	0	$[X^{8421+3}]_2$
*	0	1	0	1	1	0	1	0	1	0	0	0	$[Y^{8421+3}]_2$
*								1	0	0	0	0	сложение младших тетрад $X_1+Y_1+CF_0$
*									0	0	1	1	так как перенос равен 1, коррекция $+0011_{(2)}$
									0	0	1	1	результат коррекции
*				1	0	0	1	0	0	0	1	1	сложение средних тетрад $X_2+Y_2+CF_1$
*					0	0	1	1	0	0	1	1	так как перенос равен 1, коррекция $+0011_{(2)}$
*					0	1	0	1	0	0	1	1	результат коррекции

0	1	1	1	1	0	1	0	1	0	0	1	1	сложение старших тетрад $X_3+Y_3+CF_2$
0	1	1	0	1	0	1	0	1	0	0	1	1	так как перенос равен 0, коррекция $[-0011_{(2)}]=1101$
0	1	1	0	0	0	1	0	1	0	0	1	1	результат коррекции, перенос блокируется
0	1	1	0	0	0	1	0	1	0	0	1	1	результат, равный $920_{(10)}$, верен

Пример 4. Выполнить вычитание X–Y в процессоре, работающем в коде 8421+3 в формате целых беззнаковых чисел, имеющем три тетрады.

$X=645_{(10)}$; $[X^{8421+3}]_2=100101111000$;

$Y=275_{(10)}$; $[Y^{8421+3}]_2=100101011000$.

CF	Тетрада 3				Тетрада 2				Тетрада 1				Комментарий
*	1	0	0	1	0	1	1	1	1	0	0	0	$[X^{8421+3}]_2$
*	1	0	0	1	0	1	0	1	1	0	0	0	$[-Y^{8421+3}]_2$
*								1	0	0	0	0	сложение младших тетрад $X_1+Y_1+CF_0$
*									0	0	1	1	так как перенос равен 1, коррекция $+0011_{(2)}$
*								1	0	0	1	1	результат коррекции
*				0	1	1	0	1	0	0	1	1	сложение средних тетрад $X_2+Y_2+CF_1$
*					1	1	0	1	0	0	1	1	так как перенос равен 0, коррекция $[-0011_{(2)}]=1101$
					1	0	1	0	0	0	1	1	результат коррекции, перенос блокируется
1	0	0	1	0	1	0	1	0	0	0	1	1	сложение старших тетрад $X_3+Y_3+CF_2$
	0	0	1	1	1	0	1	0	0	0	1	1	так как перенос равен 1, коррекция $+0011_{(2)}$
1	0	1	0	1	1	0	1	0	0	0	1	1	результат коррекции
1	0	1	0	1	1	0	1	0	0	0	1	1	результат, равный $370_{(10)}$, верен

4.9.3. Алгоритм сложения (вычитания) целых чисел со знаком в BCD-кодах

При сложении таких чисел в BCD-кодах также возможно переполнение разрядной сетки процессора. Способы обнаружения переполнения такие же, как и при сложении в двоичной системе счисле-

ния. Рассмотрим примеры сложения (дополнительный код) операндов в BCD-кодах.

Пример 1. Сложить X+Y в процессоре, работающем в коде 8421 в формате целых чисел со знаком (под знак отводится одна тетрада, а под числовые разряды – три тетрады).

$X=645_{(10)}$; $[X^{8421}]_2=011001000101$;

$Y=475_{(10)}$; $[Y^{8421}]_2=010001110101$.

CF	Знак	Тетрада 3				Тетрада 2				Тетрада 1				Комментарий
*		0	1	1	0	0	1	0	0	0	1	0	1	$[X^{8421}]_2$
*	0000	0	1	0	0	0	1	1	1	0	1	0	1	$[Y^{8421}]_2$
*	0000									1	0	1	0	сложение младших тетрад $X_1+Y_1+CF_0$
*	0000									0	1	1	0	так как $(X_1+Y_1+CF_0)^{8421}>$ $>(1001_{(2)}=9_{(10)})$, коррекция $+0110_{(2)}$
	0000								1	0	0	0	0	результат коррекции
*	0000				0	1	1	0	0	0	0	0	0	сложение средних тетрад $X_2+Y_2+CF_1$
*	0000					0	1	1	0	0	0	0	0	так как $(X_2+Y_2+CF_1)^{8421}>$ $>(1001_{(2)}=9_{(10)})$, коррекция $+0110_{(2)}$
*	0000				1	0	0	1	0	0	0	0	0	результат коррекции
0	0000	1	0	1	1	0	0	1	0	0	0	0	0	сложение старших тетрад $X_3+Y_3+CF_2$
0	0000	0	1	1	0	0	0	1	0	0	0	0	0	так как $(X_2+Y_2+CF_1)^{8421}>$ $>(1001_{(2)}=9_{(10)})$, коррекция $+0110_{(2)}$
0	1111	0	0	0	1	0	0	1	0	0	0	0	0	результат коррекции
0	1111	0	0	0	1	0	0	1	0	0	0	0	0	переполнение (перенос из старшей тетрады в знак равен 1, а из знака в CF равен 0) OF=1

Пример 2. Сложить X+Y в процессоре, работающем в коде 8421+3 в формате целых чисел со знаком (под знак отводится одна тетрада, а под числовые разряды – три тетрады).

$X= -645_{(10)}$; $[X^{8421+3}]_2=011010001000$;

$Y=275_{(10)}$; $[Y^{8421+3}]_2=010110101000$.

CF	Знак	Тетрада 3				Тетрада 2				Тетрада 1				Комментарий
*	1	0	1	1	0	1	0	0	0	1	0	0	0	$[X^{8421+3}]_2$
*	0	0	1	0	1	1	0	1	0	1	0	0	0	$[Y^{8421+3}]_2$
*									1	0	0	0	0	сложение младших тетрад $X_1+Y_1+CF_0$
*										0	0	1	1	так как перенос равен 1, коррекция $+0011_{(2)}$
										0	0	1	1	результат коррекции
*					1	0	0	1	1	0	0	1	1	сложение средних тетрад $X_2+Y_2+CF_1$
*						0	0	1	1	0	0	1	1	так как перенос равен 1, коррекция $+0011_{(2)}$
*						0	1	1	0	0	0	1	1	результат коррекции
0	1	1	1	0	0	0	1	1	0	0	0	1	1	сложение старших тетрад $X_3+Y_3+CF_2$
0	1	1	1	0	1	0	1	1	0	0	0	1	1	так как перенос равен 0, коррекция $[-0011_{(2)}]=1101$
1	1	1	0	0	1	0	1	1	0	0	0	1	1	результат коррекции, перенос блокируется
Выполним проверку результата. Так как результат оканчивается на 0, то:														
1	1	0	1	1	0	1	0	0	1	1	1	0	0	перевод $[X+Y^{8421+3}]_2$ в прямой код, инверсия
		0	0	1	1	0	0	1	1	0	1	0	0	добавление $001_{10}{}^{8421+3}=0100_{(2)}$
									1	1	1	0	0	результат сложения в младшей тетраде
1						1	1	1	0	0	0	0	0	прибавление переноса в среднюю тетраду
						1	1	0	1	0	0	1	1	так как нет переноса, коррекция (–0011), перенос блокируется
1						1	0	1	0	0	0	1	1	результат коррекции в средней тетраде
		1	0	0	1	1	0	1	0	0	0	1	1	прибавление переноса в старшую тетраду
		1	1	0	1									так как нет переноса, коррекция (–0011), перенос блокируется
		0	1	1	0									результат коррекции в

														старшей тетраде
1	1	0	1	1	0	1	0	1	0	0	0	1	1	результат, равный $-370_{(10)}$, верен

4.10. Выполнение операций сдвига в BCD-кодах на один двоичный разряд

Операции сдвига используются в алгоритмах умножения и деления. Поскольку в дальнейшем алгоритмы умножения и деления будут рассматриваться только в BCD-коде 8421, то и операции сдвига на один двоичный разряд будем рассматривать только в этом коде.

Операция сдвига влево.

Сдвиг операнда в BCD-коде влево на один двоичный разряд увеличивает его значение в два раза. Рассмотрим вначале операцию сдвига влево на конкретных примерах.

Пример 1. Сдвинуть влево тетраду, содержащую $2_{(10)}$. В результате должно получиться $4_{(10)}$.

CF	Тетрада				Комментарий
	0	0	1	0	5^{8421}
	0	1	0	0	AL(5^{8421},1)
	0	1	0	0	результат сдвига

Пример 2. Сдвинуть влево тетраду, содержащую $5_{(10)}$. В результате должно получиться $10_{(10)}$ (т.е. 0 в этой тетраде и 1 переноса в следующую тетраду).

CF	Тетрада				Комментарий
	0	1	0	1	5^{8421}
	1	0	1	0	AL(5^{8421},1)
	0	1	1	0	коррекция $+110_{(2)}$
1	0	0	0	0	результат сдвига

Пример 3. Сдвинуть влево тетраду, содержащую $8_{(10)}$. В результате должно получиться $16_{(10)}$ (т.е. 8 в этой тетраде и 1 переноса в следующую тетраду).

CF	Тетрада				Комментарий
	1	0	0	0	8^{8421}
1	0	0	0	0	AL(5^{8421},1)

	0	1	1	0	коррекция $+110_{(2)}$
1	0	1	1	0	результат сдвига

Таким образом, при выполнении сдвига влево операнда в коде 8421 на один двоичный разряд требуется выполнить коррекцию сдвигаемой тетрады в двух случаях:

1) если значение тетрады после сдвига больше ($1001_{(2)}=9_{(10)}$);

2) если при сдвиге перенос из этой тетрады равен 1.

Операция сдвига вправо.

Сдвиг операнда в BCD-коде вправо на один двоичный разряд уменьшает его значение в два раза. Рассмотрим вначале операцию сдвига вправо на конкретных примерах.

Пример 1. Сдвинуть вправо операнд в коде 8421, расположенный в двух тетрадах и равный $46_{(10)}$. В результате должно получиться $23_{(10)}$.

CF	Тетрада				Тетрада				Комментарий
0	0	1	0	0	0	1	1	0	46^{8421}
0	0	0	1	0	0	0	1	1	$AR(46^{8421},1)$
0	0	0	1	0	0	0	1	1	результат сдвига, равный 23^{8421}

Пример 2. Сдвинуть вправо операнд в коде 8421, расположенный в двух тетрадах и равный $30_{(10)}$. В результате должно получиться $15_{(10)}$.

CF	Тетрада				Тетрада				Комментарий
0	0	0	1	1	0	0	0	0	30^{8421}
0	0	0	0	1	1	0	0	0	$AR(30^{8421},1)$
0	0	0	0	0	1	1	0	1	коррекция $[-11_{(2)}]=1101$
0	0	0	0	1	0	1	0	1	результат коррекции, перенос блокируется
0	0	0	0	1	0	1	0	1	результат сдвига, равный 15^{8421}

Таким образом, при выполнении сдвига вправо операнда в коде 8421 требуется выполнить коррекцию тетрады на величину $-11_{(2)}$ в том случае, если в нее был перенос, равный 1, из предыдущей тетрады.

4.10.1. Алгоритмы умножения в BCD-коде 8421

Рассмотрим два алгоритма умножения в коде 8421. Первый основан на алгоритме умножения с младших разрядов множителя и сдвиге час-

тичного произведения и множителя вправо, который рассматривался при умножении операндов в двоичной системе счисления. Этот алгоритм будет иметь свои особенности при умножении в BCD-коде. Поскольку в каждом разряде множителя может находиться значение в интервале от 0 до 9, при умножении на цифру множителя множимое прибавляется к содержимому сумматора столько раз, чему равна цифра множителя. Для этого в процессоре используется еще один счетчик СЧТ1. В этот счетчик заносится значение очередной цифры множителя и при каждом прибавлении множимого он уменьшается на 1. Достижение счетчиком СЧТ1 значения 0 означает завершение умножения на цифру множителя. Все остальные шаги идентичны алгоритму умножения в двоичной системе счисления. Рассмотрим пример выполнения такого алгоритма.

Умножить X на Y в процессоре, работающем в формате целых чисел со знаком в коде 8421 (под знак отводится одна тетрада, а под разряды – две тетрады).

$X=24_{(10)}$; $[X^{8421}]_1$=000100100;

P1=000000100100;

$Y=25_{(10)}$; $[Y^{8421}]_1$=000100101.

РСМ												Р2												СЧТ1	СЧТ2	Комментарий
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1		2	РСМ:=0; Р2:=Y; СЧТ:=2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	5	2	СЧТ1:=5
0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	5	2	+Р1
							0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	5	2	сложение мл. тетрад
0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	4	2	сложение ст. тетрад; СЧТ1:=СЧТ1–1
0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	4	2	+Р1
							0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	4	2	сложение мл. тетрад
0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	3	2	сложение ст. тетрад; СЧТ1:=СЧТ1–1

0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	3	2	+P1
								1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	3		сложение мл. тетрад
							0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	3		коррекция +110
							1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	3	2	результат
0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	2	2	сложение ст. тетрад
0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	2	2	+P1
							0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	2	2	сложение мл. тетрад
0	0	0	0	1	0	0	1					0	0	0	0	0	0	1	0	0	1	0	1	1	2	сложение ст .тетрад; СЧТ1:=СЧТ1 – 1
0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	2	+P1
								1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	1	2	сложение мл. тетрад
								0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	1	2	коррекция +110
							1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	2	результат
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	2	сложение ст. тетрад
			0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	2	коррекция +110
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	2	результат; СЧТ1:=СЧТ1 – 1
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	AR((PCM,P2),1тет); СЧТ:=СЧТ – 1
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	2	1	СЧТ1:=2
0	0	0	0	0	0	1	0	0	1	0	0					0						2	1	+P1
							0	0	1	1	0					0						2	1	сложение мл. тетрад
0	0	0	0	0	0	1	1									0						2	1	сложение ст. тетрад
0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	результат; СЧТ1:=СЧТ1 – 1
0	0	0	0	0	0	1	0	0	1	0	0											1	1	+P1
							0	1	0	1	0											1	1	сложение мл. тетрад
								0	1	1	0											1	1	коррекция +110
							1	0	0	0	0											1	1	результат

			0	0	1	1	0																1	1	сложение ст. тетрад;
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	результат; СЧТ1:=СЧТ1 – 1
0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	AR((PCM,P2),1тет); СЧТ:=СЧТ – 1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0		AR((PCM,P2),1тет); n+1 сдвиг
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	результат умножения, равный $600_{(10)}$

Второй алгоритм умножения позволяет существенно сократить количество операций сложения по сравнению с первым. В его основу положено разложение множителя по степеням двойки, т.е. на 1-, 2-, 4-, 8-кратные составляющие. Таким образом, множитель Y представляется в следующем виде:

$$Y^{8421}=8*z_1+4*z_2+2*z_3+z_4=2*(2*(2*z_1+z_2)+z_3)+z_4,$$

где z_i – значение, полученное объединением битов одинакового веса каждой из тетрад, начиная со старшей.

Например, $Y=3872_{(10)}$, тогда $[Y]^{8421}=0011100001110010$.

Тетрады				
Старшая	0	0	1	1
	1	0	0	0
	0	1	1	1
Младшая	0	0	1	0
	Получаем сомножители, взяв биты одинакового веса каждой из тетрад сверху вниз			
	$z_1=0100$	$z_2=0010$	$z_3=1011$	$z_4=1010$

Алгоритм умножения состоит из следующих шагов:

1. Множимое умножается на z_1. Результат сдвигается влево на 1 бит.
2. Множимое умножается на z_2. Результат сдвигается влево на 1 бит.
3. Множимое умножается на z_3. Результат сдвигается влево на 1 бит.
4. Множимое умножается на z_4.

Следует отметить, что для реализации такого алгоритма требуется 2*n-разрядный сумматор, где n – разрядность процессора, что можно отнести к его недостатку. Рассмотрим пример выполнения такого алгоритма.

Умножить X на Y в процессоре, работающем в формате целых чисел со знаком в коде 8421 (под знак отводится одна тетрада, а под разряды – две тетрады).

$X=24_{(10)}$; $[X^{8421}]_1=000100100$;

P1=000000100100;

$Y=25_{(10)}$; $[Y^{8421}]_1=0000100101$;

P2=000000100101;

$z_1=00$;

$z_2=01$;

$z_3=10$;

$z_4=01$.

РСМ																				Комментарий
Знак				Тетрада 4				Тетрада 3				Тетрада 2				Тетрада 1				
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	РСМ:=0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	умножение на $z_1=00$
								0	0	0	0	0	0	0	0					* на старший 0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	сложение
												0	0	0	0	0	0	0	0	* на младший 0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	сложение
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	AL(PCM, 1)
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	умножение на $z_2=01$
								0	0	0	0	0	0	0	0					* на старший 0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	сложение
												0	0	1	0	0	1	0	0	* на младший 1

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	сложение
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	AL(PCM, 1)
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	умножение на z_3=10
								0	0	1	0	0	1	0	0					* на старший 1
0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	сложение
												0	0	0	0	0	0	0	0	* на младший 0
0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	сложение
0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	AL(PCM, 1)
																0	1	1	0	в тетраде 1 коррекция $+110_{(2)}$, так как при сдвиге был перенос из этой тетрады
																0	1	1	0	результат коррекции
												0	1	1	0					в тетраде 2 коррекция $+110_{(2)}$, так как при сдвиге был перенос из этой тетрады
												0	1	1	1					результат коррекции
0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0	1	1	0	результат умножения на z_3
0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0	1	1	0	умножение на z_4=01
								0	0	0	0	0	0	0	0					* на старший 0
0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0	1	1	0	сложение
												0	0	1	0	0	1	0	0	* на младший 1

0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	1	0	сложение
																0	1	1	0	в тетраде 1 коррекция $+110_{(2)}$, так как ее значение $\geq 1001_{(2)}$
															1	0	0	0	0	результат коррекции
												1	0	1	0					добавление 1 в тетраду2
												0	1	1	0					в тетраде 2 коррекция $+110_{(2)}$, так как ее значение $\geq 1001_{(2)}$
											1	0	0	0	0					результат коррекции
							0	0	1	1	0									добавление 1 в тетраду 3
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	результат умножения на z_4
В РСМ результат умножения, равный $600_{(10)}$																				

4.12. Алгоритм деления чисел в BCD-кодах

Деление операндов в BCD-кодах выполняется по алгоритму, аналогичному алгоритму деления двоичных операндов с восстановлением остатка. Отличие состоит в том, что в BCD-коде очередная цифра частного может принимать значение от 0 до 9. Эта цифра частного получается следующим образом. На первом шаге алгоритма из делимого (на следующих шагах из остатка) вычитается делитель. Если полученный остаток положительный или равен 0, то к очередной получаемой цифре частного (ее начальное значение равно 0) прибавляется 1. Вычитание делителя в цикле выполняется до тех пор, пока остаток от вычитания не станет отрицательным. В этом случае значение цифры частного не увеличивается, а остаток восстанавливается путем добавления делителя. Все вышеперечисленное повторяется

количество раз, равное разрядности регистра частного. Результатом деления будет частное в Р2 и остаток в РСМ. Рассмотрим пример выполнения такого алгоритма.

Разделить X на Y в процессоре, работающем в формате целых чисел со знаком в коде 8421 (под знак отводится одна тетрада, а под разряды – две тетрады).

$X=625_{(10)}$; $[X^{8421}]_2=011000100101$;

РСМ=0000000011000100101;

$Y=25_{(10)}$; $[Y^{8421}]_2=000000100101$;

$-Y=-25_{(10)}$; $[-Y^{8421}]_2=111101110101$;

Р1=000000100101;

– Р1=111101110101.

РСМ												Р2								СЧТ	Комментарий
Знак				Тетрада 2				Тетрада 1				Тетрада 2				Тетрада 1					
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*		
0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1	0	1	2	(РСМ,Р2):=X; СЧТ:=2
1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	0	1	0	1	2	–Р1; пробное вычитание
							0	1	0	1	1	0	0	1	0	0	1	0	1	2	вычитание в тетраде 1
								0	1	1	0	0	0	1	0	0	1	0	1	2	коррекция $+110_{(2)}$
							1	0	0	0	1	0	0	1	0	0	1	0	1	2	результат коррекции
1	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	2	вычитание в тетраде 2
1	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	2	результат пробного вычитания; так как остаток ≤0, деление состоится
0	0	0	0	0	0	1	0	0	1	0	1									2	+Р1; восстановление отрицательного остатка
1	1	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0	1	0	1	2	результат сложения
				0	1	1	0													2	коррекция $+110_{(2)}$
0	0	0	0	0	0	0	0	0	1	1	0			1	0	0	1	0	1	2	результат коррекции
1	1	1	1	0	1	1	1	0	1	0	1									2	–Р1
							0	0	1	1	1	0	1	0	1	0	0	0	0	2	вычитание в тетраде 1
			0	1	1	0	1					0	1	0	1	0	0	0	0	2	вычитание в тетраде 2
				0	1	1	0					0	1	0	1	0	0	0	0	2	коррекция $+110_{(2)}$
0	0	0	0	0	0	1	1					0	1	0	1	0	0	0	0	2	результат коррекции

0	0	0	0	0	0	1	1	0	1	1	1	0	1	0	1	0	0	0	1	2	результат вычитания ≥0, поэтому к цифре частного P2 (тетрада 1) прибавляется 1
1	1	1	1	0	1	1	1	0	1	0	1									2	–P1
								1	1	0	0									2	вычитание в тетраде
								0	1	1	0									2	коррекция $+110_{(2)}$
								0	0	1	0									2	результат коррекции
			0	1	0	1	1													2	вычитание в тетраде 2
				0	1	1	0													2	коррекция $+110_{(2)}$
0	0	0	0	0	0	0	1													2	результат коррекции
0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	2	результат вычитания ≥0, поэтому к цифре частного P2 (тетрада 1) прибавляется 1
1	1	1	1	0	1	1	1	0	1	0	1	0	1	0	1	0	0	1	0	2	– P1
							0	0	1	1	1	0	1	0	1	0	0	1	0	2	вычитание в тетраде 1
			0	1	0	0	0					0	1	0	1	0	0	1	0	2	вычитание в тетраде 2
1	1	1	1	1	0	0	0	0	1	1	1	0	1	0	1	0	0	1	0	2	результат вычитания <0, поэтому цифра частного не увеличивается
0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	1	0	0	1	0	2	+P1; восстановление отрицательного остатка
							0	1	1	0	0	0	1	0	1	0	0	1	0	2	вычитание в тетраде 1
								0	1	1	0	0	1	0	1	0	0	1	0	2	коррекция $+110_{(2)}$
							1	0	0	1	0	0	1	0	1	0	0	1	0	2	результат коррекции
				1	0	1	1					0	1	0	1	0	0	1	0	2	вычитание в тетраде 2
				0	1	1	0					0	1	0	1	0	0	1	0	2	коррекция $+110_{(2)}$
			1	0	0	0	1					0	1	0	1	0	0	1	0	2	результат коррекции
0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	2	результат вычитания
0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	1	СЧТ:=СЧТ – 1
0	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	1	AL((PCM,P2),1тетраду);
1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	0	0	0	0	1	–P1
							0	1	0	1	0	0	0	1	0	0	0	0	0	1	вычитание в тетраде 1
								0	1	1	0	0	0	1	0	0	0	0	0	1	коррекция $+110_{(2)}$
							1	0	0	0	0	0	0	1	0	0	0	0	0	1	результат коррекции
			0	1	0	1	0					0	0	1	0	0	0	0	0	1	вычитание в тетраде 2

				0	1	1	0					0	0	1	0	0	0	0	1	коррекция $+110_{(2)}$
0	0	0	1	0	0	0	0					0	0	1	0	0	0	0	1	результат коррекции
0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	результат вычитания
0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	результат вычитания ≥0, поэтому к цифре частного P2 (тетрада 1) прибавляется 1
1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	0	0	0	1	1	–P1
								0	1	0	1	0	0	1	0	0	0	0	1	1	вычитание в тетраде 1
0	0	0	0	0	1	1	1					0	0	1	0	0	0	0	1	1	вычитание в тетраде 2
0	0	0	0	0	1	1	1	0	1	0	1	0	0	1	0	0	0	0	1	1	результат вычитания
0	0	0	0	0	1	1	1	0	1	0	1	0	0	1	0	0	0	1	0	1	результат вычитания ≥0, поэтому к цифре частного P2 (тетрада 1) прибавляется 1
1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	0	0	1	0	1	–P1
							0	1	0	1	0	0	0	1	0	0	0	1	0	1	вычитание в тетраде 1
								0	1	1	0	0	0	1	0	0	0	1	0	1	коррекция $+110_{(2)}$
							1	0	0	0	0	0	0	1	0	0	0	1	0	1	результат коррекции
				1	1	1	1					0	0	1	0	0	0	1	0	1	вычитание в тетраде 2
				0	1	1	0					0	0	1	0	0	0	1	0	1	коррекция $+110_{(2)}$
0	0	0	0	0	1	0	1					0	0	1	0	0	0	1	0	1	результат коррекции
0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	1	0	1	результат вычитания
0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	1	1	1	результат вычитания ≥0, поэтому к цифре частного P2 (тетрада 1) прибавляется 1
1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	0	0	1	1	1	–P1
								0	1	0	1	0	0	1	0	0	0	1	1	1	вычитание в тетраде 1
			0	1	1	0	0					0	0	1	0	0	0	1	1	1	вычитание в тетраде 2
				0	1	1	0					0	0	1	0	0	0	1	1	1	коррекция $+110_{(2)}$
0	0	0	0	0	0	1	0					0	0	1	0	0	0	1	1	1	результат коррекции
0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	1	1	1	результат вычитания
0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	результат вычитания ≥0, поэтому к цифре частного P2 (тетрада 1) прибавляется 1
1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	0	1	0	0	1	–P1
							0	1	0	1	0	0	0	1	0	0	1	0	0	1	вычитание в тетраде 1

								0	1	1	0	0	0	1	0	0	1	0	0	1	коррекция $+110_{(2)}$
							1	0	0	0	0	0	0	1	0	0	1	0	0	1	результат корреции
				1	0	1	0					0	0	1	0	0	1	0	0	1	вычитание в тетраде 2
				0	1	1	0					0	0	1	0	0	1	0	0	1	коррекция $+110_{(2)}$
0	0	0	0	0	0	0	0					0	0	1	0	0	1	0	0	1	результат корреции
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	результат вычитания
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	результат вычитания ≥0, поэтому к цифре частного P2 (тетрада 1) прибавляется 1
1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	0	1	0	0	1	–P1
							0	0	1	0	1	0	0	1	0	0	1	0	0	1	вычитание в тетраде 1
			0	0	1	1	1					0	0	1	0	0	1	0	0	1	вычитание в тетраде 2
1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	0	1	0	0	1	результат вычитания
1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	0	1	0	0	1	результат вычитания <0, поэтому цифра частного не увеличивается
0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	+P1; восстановление отрицательного остатка
							0	1	0	1	0	0	0	1	0	0	1	0	0	1	вычитание в тетраде 1
								0	1	1	0	0	0	1	0	0	1	0	0	1	коррекция $+110_{(2)}$
							1	0	0	0	0	0	0	1	0	0	1	0	0	1	результат коррекции
				1	0	1	0					0	0	1	0	0	1	0	0	1	вычитание в тетраде 2
				0	1	1	0					0	0	1	0	0	1	0	0	1	коррекция $+110_{(2)}$
0	0	0	0	0	0	0	0					0	0	1	0	0	1	0	0	1	результат коррекции
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	результат сложения
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	СЧТ:=СЧТ – 1
Так как СЧТ равен 0, деление завершено. В P2 частное, равное 25(10), а в РСМ остаток равен 0. Результат верен.																				

5. Контроль выполнения арифметических операций

Цифровые процессоры используются и в таких областях применения, где ошибки в работе должны быть исключены, в том числе и при выполнении арифметических операций. К этим областям относятся управление космическими объектами, управление ядерными реакторами, управление военными объектами. Чтобы полностью исключить ошибку, при вычислениях используется метод «дублирования», который состоит в том, что цифровое устройство повторяется (дублируется) с кратностью 3, 5 и т.д. Любая операция выполняется одновременно на всех устройствах. Полученные результаты от каждого из устройств сравниваются между собой. Если они все совпали, ошибки нет. Если не совпали, то, например для трехкратного дублирования, возможны два случая:

– результаты от двух устройств одинаковы, и тогда этот результат принимается за правильный (так называемый мажоритарный выбор);

– результаты от всех трех устройств разные, результат неверный и вычисление выполняется заново.

Такой контроль требует больших аппаратных затрат, а значит, и экономических, энергетических и конструктивных. Но контролировать правильность выполнения арифметических операций с меньшей достоверностью обнаружения ошибки можно и более простыми методами.

Рассмотрим такой метод, который получил наибольшее распространение. Это метод контроля по модулю, основанный на следующих положениях теории чисел.

Если задан модуль q, то для любого числа N существует соотношение

$N \equiv R_N (\mathrm{mod}\ q)$.

Данное выражение читается как «число N сравнимо с R_N по модулю q», где R_N – наименьший вычет числа N по модулю q, или другими словами, наименьший остаток от целочисленного деления N на q. Тогда

$$N = p*q + R_N,$$

где p – частное от целочисленного деления N на q.

R_N используется в качестве контрольного кода в методе контроля по модулю. Рассмотрим пример получения вычета от числа $N=13_{(10)}$ по модулю q=3. Целочисленное деление дает

p=4; R_N=1.

Тогда можно записать 13≡1(mod3).

Из теории чисел известно, что два числа – N_1 и N_2 – сравнимы по модулю q, если равны их вычеты по этому модулю:

$N_1 \equiv N_2(\text{mod}q)$, если $R_{N1}=R_{N2}$.

При выполнении контроля арифметических операций по модулю q исходное число N представляют в цифровом процессоре, состоящем из двух частей: самого числа и его вычета по модулю q.

число N→	<операнд>	<вычет по модулю q
	m разрядов процессора	k разрядов процессора
	(m+k)-разрядный процессор	

Для эффективного контроля должно соблюдаться условие k<<m.

Принципы контроля выполнения арифметических операций основаны на следующих свойствах вычетов:

- для сложения $N_1+N_2=N_3$ выполняется условие $R_{N1}+R_{N2}=R_{N3}$;
- для вычитания $N_1-N_2=N_3$ выполняется условие $R_{N1}-R_{N2}=R_{N3}$;
- для умножения $N_1*N_2=N_3$ выполняется условие $R_{N1}*R_{N2}=R_{N3}$;
- для деления $N_1/N_2=N_3(N_4)$ выполняется условие $R_{N1}=R_{N2}*R_{N3}+R_{N4}$.

Функциональная схема контроля операции сложения по модулю приведена на рис. 5.1.

5.1. Выбор значения модуля и вычисление вычета

От выбранного значения модуля q зависят:

- контролирующая способность контрольного кода;
- сложность контрольного оборудования.

С ростом модуля q увеличивается контролирующая способность кода, но увеличиваются и затраты на контрольное оборудование. Исследования показали, что наилучшим вариантом выбора модуля q будет выбор значения q из числового ряда, определяемого выражением $2^k - 1$, где (k = 2, 3, 4, 5 …), т.е. 3, 7, 15, 31…

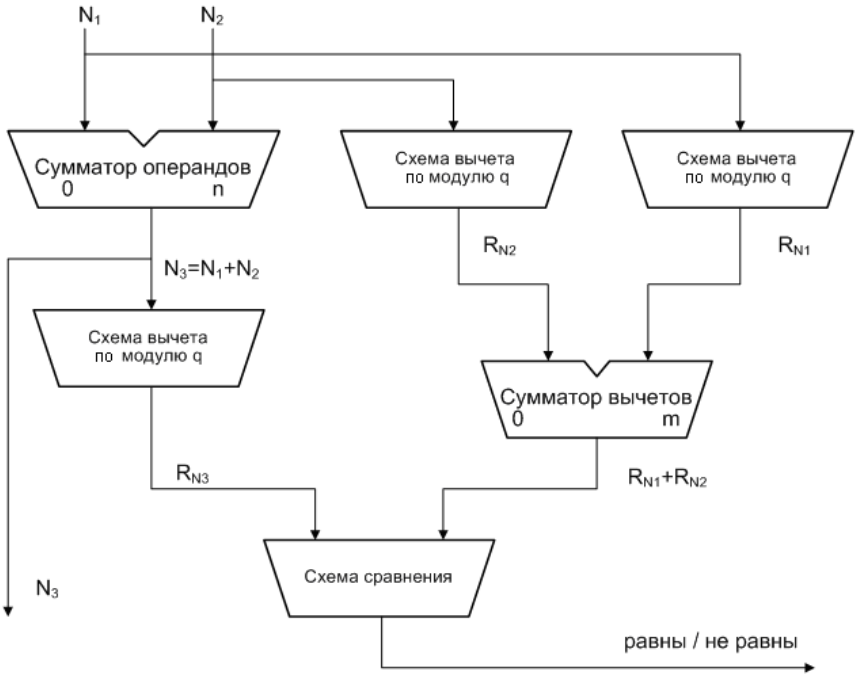

Рис. 5.1. Функциональная схема ОА с контролем операции сложения по модулю

При таком выборе модуля q вычет от операнда получается суммированием по модулю q разрядов исходного числа, группируемых по k разрядов. Для целого числа группировка выполняется справа налево, а для дробного числа слева направо. То, что вычет можно получить суммированием («короткой операцией»), а не делением («длинной операцией»), существенно сказывается на скорости операции контроля. Рассмотрим пример получения вычета числа $N=185_{(10)}$ в восьмиразрядном процессоре

Пример 1. $N=185_{(10)}=10111001_{(2)}$.

Получим вычет по модулю $7=(2^3 - 1)$, для чего операнд разобьем на триады справа налево, а затем суммируем триады, начиная с младшей по модулю 7.

Разбиваем: (10)(111)(001). Суммируем:

Триада			Комментарий
0	0	1	младшая
1	1	1	средняя
0	0	1	результат сложения по модулю 7
0	1	0	старшая
0	1	1	сложение по модулю 7; конечный результат
Получен вычет, равный 3			

Действительно, наименьший остаток от целочисленного деления $185_{(10)}$ на 7 равен 3.

Пример 2. Получим вычет того же операнда, но по модулю $3=(2^2-1)$, для чего операнд разбивается по два разряда справа налево, а затем они суммируются по модулю 3.

Разбиваем: (10)(11)(10)(01). Суммируем:

Два разряда		Комментарий
0	1	младшая (первая)
1	0	вторая
0	0	результат сложения по модулю 3;
1	1	третья
1	1	сложение по модулю 3;
1	0	четвертая
1	0	сложение по модулю 3; конечный результат
Получен вычет, равный 2		

Действительно, наименьший остаток от целочисленного деления $185_{(10)}$ на 3 равен 2.

Рассмотрим выполнение контроля операции сложения на примере сложения операндов в восьмиразрядном процессоре:

$N_3=N_1+N_2$;

$N_1=53(8)=00101011_{(2)}$;

$N_2=61(8)=00110001_{(2)}$.

Пример 1. Контроль осуществим по модулю 7. Вычисление вычетов операндов дает следующий результат:

$R_{N1}=1$;

$R_{N2}=0$.

Сложение операндов и сложение вычетов операндов для случая отсутствия ошибки при сложении:

Операнд								Вычет			Комментарий
0	0	1	0	1	0	1	1	0	0	1	операнд N_1; вычет R_{N1}
0	0	1	1	0	0	0	1	0	0	0	операнд N_2; вычет R_{N2}
0	1	0	1	1	1	0	0	0	0	1	сложение операндов и вычетов
(0	1)	(0	1	1)	(1	0	0)	0	0	1	вычисление вычетов R_{N3} и $R_{N1}+R_{N2}$
1								1			
Так как вычет суммы операндов равен сумме вычетов операндов, считается, что результат сложения верен											

Сложение операндов и сложение вычетов операндов для случая возникновения ошибки при сложении:

Операнд								Вычет			Комментарий
0	0	1	0	1	0	1	1	0	0	1	операнд N_1; вычет R_{N1}
0	0	1	1	0	0	0	1	0	0	0	операнд N_2; вычет R_{N2}
0	1	0	0 ошибка	1	1	0	0	0	0	1	сложение операндов и вычетов
(0	1)	(0	0	1)	(1	0	0)	0	0	1	вычисление вычетов R_{N3} и $R_{N1}+R_{N2}$
6								1			
Так как вычет суммы операндов не равен сумме вычетов операндов, считается, что результат сложения не верен											

Пример 2. Контроль сложения тех же операндов осуществим по модулю 3. Вычисление вычетов операндов дает следующий результат:

$R_{N1}=1$;

$R_{N2}=1$.

Сложение операндов и сложение вычетов операндов для случая отсутствия ошибки при сложении:

Операнд								Вычет		Комментарий
0	0	1	0	1	0	1	1	0	1	операнд N_1; вычет R_{N1}
0	0	1	1	0	0	0	1	0	1	операнд N_2; вычет R_{N2}
0	1	0	1	1	1	0	0	1	0	сложение операндов и вычетов
(0	1)	(0	1)	(1	1)	(0	0)	1	0	вычисление вычетов R_{N3} и $R_{N1}+R_{N2}$
2								2		
Так как вычет суммы операндов равен сумме вычетов операндов, считается, что результат сложения верен										

Сложение операндов и сложение вычетов операндов для случая возникновения ошибки при сложении:

Операнд								Вычет		Комментарий
0	0	1	0	1	0	1	1	0	1	операнд N_1; вычет R_{N1}
0	0	1	1	0	0	0	1	0	1	операнд N_2; вычет R_{N2}
0	1	0	1	1	0 ошибка	0	0	1	0	сложение операндов и вычетов
(0	1)	(0	1)	(1	0)	(0	0)	1	0	вычисление вычетов R_{N3} и $R_{N1}+R_{N2}$
1								2		
Так как вычет суммы операндов не равен сумме вычетов операндов, считается, что результат сложения не верен										

5.2. Контроль арифметических операций над беззнаковыми числами

Контроль арифметических операций над беззнаковыми числами выполняется в соответствии со всеми вышеизложенными положениями, но только при получении вычета суммы (разности) учитывается значение флага CF.

Рассмотрим выполнение контроля **операции сложения** на примере сложения операндов в восьмиразрядном процессоре:

$N_3=N_1+N_2$;

$N_1=246_{(8)}=10100110_{(2)}$;

$N_2=177_{(8)}=01111111_{(2)}$.

Пример 1. Контроль осуществляется по модулю 7. Вычисление вычетов операндов дает следующий результат:

$R_{N1}=5$;

$R_{N2}=1$.

Сложение операндов и сложение вычетов операндов для случая отсутствия ошибки при сложении:

CF	Операнд								Вычет			Комментарий
*	1	0	1	0	0	1	1	0	1	0	1	операнд N_1; вычет R_{N1}
*	0	1	1	1	1	1	1	1	0	0	1	операнд N_2; вычет R_{N2}
1	0	0	1	0	0	1	0	1	1	1	0	сложение операндов и вычетов
(1	0	1)	(0	1	1)	(1	0	1)	1	1	0	вычисление вычета R_{N3} с учетом CF и $R_{N1}+R_{N2}$
6									6			
Так как вычет суммы операндов равен сумме вычетов операндов, считается, что результат сложения верен												

Сложение операндов и сложение вычетов операндов для случая возникновения ошибки при сложении:

CF	Операнд								Вычет			Комментарий
*	1	0	1	0	0	1	1	0	1	0	1	операнд N_1; вычет R_{N1}
*	0	1	1	1	1	1	1	1	0	0	1	операнд N_2; вычет R_{N2}
1	0	0	1	0	0	0 ошибка	0	1	1	1	0	сложение операндов и вычетов
(1	0	1)	(0	1	1)	(0	0	1)	1	1	0	вычисление вычета R_{N3} с учетом CF и $R_{N1}+R_{N2}$
2									6			
Так как вычет суммы операндов не равен сумме вычетов операндов, считается, что результат сложения не верен												

Пример 2. Контроль сложения тех же самых операндов осуществляется по модулю 3. Вычисление вычетов операндов дает следующий результат:

$R_{N1}=1$;

$R_{N2}=1$.

Сложение операндов и сложение вычетов операндов для случая отсутствия ошибки при сложении:

CF	Операнд								Вычет		Комментарий
*	1	0	1	0	0	1	1	0	0	1	операнд N_1; вычет R_{N1}
*	0	1	1	1	1	1	1	1	0	1	операнд N_2; вычет R_{N2}
	0	1	0	1	1	1	0	0	1	0	сложение
(1)	(0	0)	(1	0)	(0	1)	(0	1)	1	0	вычисление вычета
2									2		
Так как вычет суммы операндов равен сумме вычетов операндов, считается, что результат сложения верен											

Сложение операндов и сложение вычетов операндов для случая возникновения ошибки при сложении:

CF	Операнд								Вычет		Комментарий
*	1	0	1	0	0	1	1	0	0	1	операнд N_1; вычет R_{N1}
*	0	1	1	1	1	1	1	1	0	1	операнд N_2; вычет R_{N2}
	0	1	0	1	1 ошибка	1	0	0	1	0	сложение операндов и вычетов
(1)	(0	0)	(1	0)	(1	1)	(0	1)	1	0	вычисление вычета R_{N3} с учетом CF и $R_{N1}+R_{N2}$
1									2		
Так как вычет суммы операндов не равен сумме вычетов операндов, считается, что результат сложения не верен											

Рассмотрим выполнение контроля **операции вычитания** на примере вычитания операндов в восьмиразрядном процессоре в дополнительном коде:

$N_3=N_1–N_2=N_1+(–N_2)$;

$N_1=44_{(8)}=00100100_{(2)}$;

$N_2=42_{(8)}=00100010_{(2)}$;

$–N_2= –42_{(8)}= –00100010_{(2)}$; $[–N_2]_2=11011110$.

Пример. Контроль осуществляется по модулю 7. Вычисление вычетов операндов дает следующий результат:

$R_{N1}=1$;

$–R_{N2}=5$.

Вычитание операндов и вычитание вычетов операндов для случая отсутствия ошибки при сложении (вычитание заменено на сложение с $–N_2$):

CF	Операнд								Вычет			Комментарий
*	0	0	1	0	0	1	0	0	0	0	1	операнд N_1; вычет R_{N1}
*	1	1	0	1	1	1	1	0	1	0	1	операнд N_2; вычет R_{N2}
1	0	0	0	0	0	0	1	0	0	0	1	сложение
(1	0	0)	(0	0	0)	(0	1	0)	1	1	0	вычисление вычета
6									6			
Так как вычет разности операндов равен разности вычетов операндов, считается, что результат сложения верен												

Вычитание операндов и вычитание вычетов операндов для случая возникновения ошибки при сложении:

CF	Операнд								Вычет			Комментарий
*	0	0	1	0	0	1	0	0	0	0	1	операнд N_1; вычет R_{N1}
*	1	1	0	1	1	1	1	0	1	0	1	операнд N_2; вычет R_{N2}
1	0	1 ошибка	0	0	0	0	1	0	1	1	0	вычитание операндов и вычетов
(1	0	1)	(0	0	0)	(0	1	0)	1	1	0	вычисление вычета R_{N3} с учетом CF и $R_{N1}+R_{N2}$
0									6			
Так как вычет суммы операндов не равен сумме вычетов операндов, считается, что результат сложения не верен												

Рассмотрим выполнение контроля **операции умножения** на примере умножения операндов в восьмиразрядном процессоре в прямом коде:

Пример. $N_3=N_1*N_2$;

$N_1=121_{(8)}=01010001_{(2)}$;

$N_2=41_{(8)}=00100001_{(2)}$.

Контроль осуществляется по модулю 7. Вычисление вычетов операндов дает следующий результат:

$R_{N1}=4$;

$R_{N2}=5$.

Результат умножения $N_3=N_1*N_2$ равен $0000101001110001_{(2)}$ (выполнение алгоритма здесь не приводится). Вычет произведения (0)(000)(101)(001)(110)(001) равен 6.

Произведение вычетов равно $R_{N1}*R_{N2}=4*5=20_{(10)}$, $\mathrm{mod}_7(20)=6$.

Так как вычет произведения равен вычету от произведения вычетов, результат операции считается верен.

Если при умножении произошла ошибка и получен неверный результат, например, $N_3=N_1*N_2=0000101001110000$, то вычет произведения будет равен 5 и не равен вычету от произведения вычетов, т.е. результат операции не верен.

Рассмотрим выполнение контроля **операции деления** на примере деления операндов в восьмиразрядном процессоре в прямом коде.

Пример. $N_3=N_1/N_2$;

$N_1=241_{(8)}=10100001_{(2)}$;

$N_2=24_{(8)}=00010100_{(2)}$.

Контроль осуществляется по модулю 3. Вычисление вычетов операндов дает следующий результат:

$R_{N1}=2$;

$R_{N2}=2$.

Результат деления, т.е. частное $N_3=N_1/N_2$, равен $00001000_{(2)}$, остаток N_4 равен $00000001_{(2)}$ (выполнение алгоритма здесь не приводится). Вычет частного R_{N3} от $N_3=(00)(00)(10)(00)$ равен 2. Вычет остатка

R_{N4} от N_4=(00)(00)(00)(01) равен 1. Тогда вычет от $R_{N2}*R_{N3}+R_{N4}$= =2*2+1=$5_{(10)}$ по $mod_3(5)=2$.

Так как вычет делимого равен вычету от произведения вычетов делителя и частного с добавлением вычета от остатка, результат операции считается верным.

Если при делении произошла ошибка и получен неверный результат, например, $N_3=N_1/N_2=000001001_{(2)}$ и $N_4=00000001_{(2)}$, то вычет делимого не равен вычету от произведения вычетов делителя и частного с добавлением вычета от остатка, результат операции считается неверным:

$R_{N2}*R_{N3}+ R_{N4}=2*0+1=1_{(10)}$ по $mod_3(1)=1$.

5.3. Выполнение контроля арифметических операций над числами со знаком

Рассмотрим такой контроль только для процессора, работающего в дополнительном коде. В этом случае при получении вычета знаковый разряд учитывается как числовой. Все остальные положения по контролю совпадают с контролем арифметических операций без знака.

Рассмотрим выполнение контроля **операции сложения** на примере сложения операндов в восьмиразрядном процессоре в дополнительном коде:

$N_3=N_1+N_2$;

$N_1= -44_{(8)}= -00100100_{(2)}$;

$N_2= -42_{(8)}= -00100010_{(2)}$;

$[N_1]_2$=11011100;

$[N_2]_2$=11011110.

Пример. Контроль осуществляется по модулю 7. Вычисление вычетов дополнительного кода операндов дает следующий результат:

$R_{N1}=3$;

$R_{N2}=5$.

Сложение операндов и сложение вычетов операндов для случая отсутствия ошибки при сложении:

CF	Операнд								Вычет			Комментарий
*	1	1	0	1	1	1	0	0	0	1	1	операнд N_1; вычет R_{N1}
*	1	1	0	1	1	1	1	0	1	0	1	операнд N_2; вычет R_{N2}
1	1	0	1	1	1	0	1	0	0	0	1	сложение операндов и вычетов
(1	1	0)	(1	1	1)	(0	1	0)	0	0	1	вычисление вычета R_{N3} с учетом CF и $R_{N1}+R_{N2}$
1									1			
Так как вычет суммы операндов равен сумме вычетов операндов, считается, что результат сложения верен												

Сложение операндов и сложение вычетов операндов для случая возникновения ошибки при сложении:

CF	Операнд								Вычет			Комментарий
*	1	1	0	1	1	1	0	0	0	1	1	операнд N_1; вычет R_{N1}
*	1	1	0	1	1	1	1	0	1	0	1	операнд N_2; вычет R_{N2}
1	1	1 ошибка	1	1	1	0	1	0	0	0	1	сложение операндов и вычетов
(1	1	1)	(1	1	1)	(0	1	0)	0	0	1	вычисление вычета R_{N3} с учетом CF и $R_{N1}+R_{N2}$
2									1			
Так как вычет суммы операндов не равен сумме вычетов операндов, считается, что результат сложения не верен												

6. Контроль передачи информации в цифровых каналах связи

В цифровых каналах связи (рис. 6.1) при передаче информации по разным причинам (неисправность оборудования, помехи) могут возникать ошибки. Поэтому на приемной стороне используются различные способы обнаружения ошибок, а также способы, позволяющие выполнить как обнаружение, так исправление ошибки.

Рис. 6.1. Канал связи

Если контроль передачи информации выполняется только с обнаружением ошибки, то в случае ошибки приемник запрашивает повторную передачу искаженной информации. Если же контроль выполняется с обнаружением и исправлением ошибки, то приемник сам выполняет коррекцию искаженной информации без обращения к передатчику.

Из теории кодирования информации известно, что вероятность двойной или тройной ошибки (в двух или трех битах одновременно) на один или два порядка меньше, чем вероятность возникновения одиночной ошибки. Поэтому при контроле передачи информации основное внимание уделяется обнаружению, а также обнаружению и исправлению одиночной ошибки. Различают коды, которые позволяют только обнаруживать ошибку, и коды, которые позволяют как обнаруживать, так и исправлять ее.

6.1. Коды для обнаружения ошибок

Наиболее часто для обнаружения ошибок используется контроль на четность/нечетность передаваемой информации. При таком контроле блок информации, передаваемый от источника к приемнику, должен содержать соответственно четное/нечетное число единиц. Источник формирует передаваемый блок информации так, чтобы это выполнялось. Приемник проверяет выполнение этого условия. Если четность/нечетность нарушена, то считается, что произошла ошибка

при передаче информации. Приемник сообщает об этом источнику, и источник заново посылает этот блок информации. Передаваемый блок информации состоит из информационных битов и одного контрольного:

Передаваемый блок информации	
Информационные биты	Контрольный бит
$x_1, x_2, \ldots x_n$	k

При контроле на четность в бит k записывается сумма по модулю 2 информационных разрядов. При контроле на нечетность записывается инверсное значение этой суммы.

Рассмотрим пример такого контроля. Пусть от источника к приемнику надо переслать 8 бит информации с контролем на четность. Тогда пересылаемый блок информации должен содержать 9 бит.

Пример. Переслать код 01110101.

Побитное сложение по модулю для этого кода дает значение, равное 1. Тогда значение контрольного бита должно равняться 1, чтобы передаваемый код содержал четное число единиц.

Передатчик									Приемник									Комментарий
Информационные биты								Контрольный бит	Принятые биты								Сумма по mod2	
0	1	1	1	0	1	0	1	1	0	1	1	1	0	1	0	1	1	0	нет ошибки
									0	1	1	1	**1**	1	0	1	1	1	есть ошибка

6.2. Код Хэмминга для обнаружения и исправления ошибки

Исправлять ошибку при приеме информации труднее, чем ее обнаруживать. Исправление ошибки предполагает два совмещенных процесса: обнаружение факта, что есть ошибка, и определение ее места. После решения этих двух задач, исправление тривиально – надо инвертировать значение ошибочного бита. В наземных каналах связи, где вероятность ошибки невелика, обычно используется только метод обнаружения ошибки и повторной пересылки блока информации, содержавшего ошибку. Для спутниковых каналов с типичны-

ми для них большими задержками коррекция ошибки становится необходимой. Здесь используется код Хэмминга.

Код Хэмминга представляет собой блочный код, который позволяет выявить и исправить ошибочно переданный бит в пределах переданного блока. Блочными называются коды, в которых информационный поток символов разбивается на отрезки, и каждый из них преобразуется в определенную последовательность (блок) кодовых символов. В блочных кодах кодирование при передаче (формирование проверочных элементов) и декодирование при приеме (обнаружение и исправление ошибок) выполняются в пределах каждой кодовой комбинации (блока) в отдельности по соответствующим алгоритмам.

Обычно код Хэмминга характеризуется двумя целыми числами, например, код (11,7), используемый при передаче семибитных ASCII-кодов. Такая запись говорит, что при передаче семибитного кода используется дополнительно четыре контрольных бита (7+4=11). При этом предполагается, что может иметь место ошибка в одном бите и что ошибка в двух или более битах существенно менее вероятна. С учетом этого исправление ошибки осуществляется с определенной вероятностью.

При рассмотрении кода Хэмминга требуется знать, что такое кодовое расстояние. Кодовое расстояние между двумя двоичными кодами одинаковой длины определяется количеством битов, в которых эти коды отличаются.

Пример 1. Кодовое расстояние между «кодом 1» и «кодом 2» равно 1.

код 1	0	1	0
код 2	0	1	1

Пример 2. Кодовое расстояние между «кодом 1» и «кодом 2» равно 2.

код 1	0	0	1
код 2	1	1	1

Можно обнаружить ошибку только, если между используемыми кодовыми комбинациями есть необходимое для этого кодовое расстояние, т.е. между соседними используемыми кодовыми комбина-

циями есть другие комбинации. Эти кодовые комбинации не используются для передачи информации, и их получение на приеме свидетельствует об ошибке передачи информации в канале связи. Для заданного кода минимальное кодовое расстояние – это минимальное из всех расстояний всех пар кодовых слов.

В обычном равномерном непомехоустойчивом коде число разрядов n в кодовых комбинациях определяется числом сообщений и основанием кода. Коды, у которых все кодовые комбинации разрешены к передаче, называются простыми или равнодоступными и являются полностью безызбыточными. Безызбыточные первичные коды обладают большой «чувствительностью» к помехам. Внесение избыточности при использовании помехоустойчивых кодов обязательно связано с увеличением числа разрядов (длины) кодовой комбинации. В этом случае все множество $N=2^n$ комбинаций можно разбить на два подмножества: подмножество разрешенных комбинаций, т.е. обладающих определенными признаками, и подмножество запрещенных комбинаций, этими признаками не обладающих. Помехоустойчивый код отличается от обычного тем, что в канал передаются не все кодовые комбинации N, которые можно сформировать из имеющегося числа разрядов n, а только их часть N_K, которая составляет подмножество разрешенных комбинаций

Если при приеме выясняется, что кодовая комбинация принадлежит к запрещенным, то это свидетельствует о наличии ошибки, т.е. таким образом решается задача обнаружения ошибок. При этом принятая комбинация не декодируется (не принимается решение о приеме сообщения). Помехоустойчивые коды называют корректирующими кодами. Корректирующие свойства избыточных кодов зависят от правила их построения, определяющего структуру кода, и параметров кода.

Первые работы по корректирующим кодам принадлежат Хэммингу, который ввел понятие минимального кодового расстояния и предложил код, позволяющий однозначно указать ту позицию в кодовой комбинации, где произошла ошибка. К «М» (message) информационным битам в коде Хэмминга добавляется «С» (control) проверочных битов для определения местоположения ошибочного бита.

Таким образом, код Хэмминга состоит из информационных и контрольных битов. Информационные биты будем обозначать через М, контрольные биты – через С:

$M=\{M_1,M_2,\ldots M_n\}$;

$C=\{C_1, C_2,\ldots C_k\}$;

N=M+C – общее количество передаваемых битов в канал связи (передаваемый блок).

Сущность кода Хэмминга заключается в том, что местоположение контрольных и информационных битов определяется по правилу, которое устанавливает зависимость, позволяющую в случае ошибки определить ее местоположение, т.е. номер бита.

Основные положения для получения кода Хэмминга состоят в следующем:

1. Число, составленное из контрольных разрядов, должно указывать номер бита, в котором произошла ошибка, или 0, если нет ошибки. Исходя из этого, если произошла ошибка, проверочное С – битовое число – должно быть в диапазоне от 1 до $2^C - 1$. Из этого следует, что $2^C - 1 \geq N$; тогда $2^C \geq N+1$; умножим левую и правую части на 2^M, получим $2^M * 2^C \geq 2^M * (N+1)$, откуда $2^{(M+C)} \geq 2^M * (N+1)$, тогда $2^N \geq 2^M * (N+1)$; делим левую и правую части на (N+1), получаем условие $(2^N/(N+1)) \geq 2^M$.

С использованием этого условия и определяется минимально необходимое число контрольных битов для передачи заданного количества информационных.

2. Контрольные биты в коде Хэмминга занимают позиции с номерами, равными 2^n, где n=0,1,3,… и т.д. Причем позиции кода нумеруются справа налево, начиная с 1. Информационные биты располагаются в остающихся позициях справа налево по возрастанию весов разрядов.

3. Значение каждого контрольного бита получается сложением по модулю 2 тех информационных битов, для которых в номере кодовой позиции информационных битов, записанном через контрольные разряды, значение бита равно 1. Эта формулировка будет более понятна, если посмотреть ее использование по табл. 6.1 или 6.2.

Вначале построим код Хэмминга для передачи четырехбитового кода.

Условие 1. Это условие будет выполняться при N=7:

$\{2^7/(7+1)=128/8=16\}\geq\{2^4=16\}$.

Таким образом, для четырех информационных битов (M=4) требуются три контрольных бита (C=3).

Условие 2. Определяем местоположение информационных и контрольных битов в коде.

Положение информационных и контрольных битов	M_4	M_3	M_2	C_3	M_1	C_2	C_1
Номер позиции кода	7	6	5	4	3	2	1

Условие 3. Определяем логические выражения для вычисления контрольных битов. Для этого построим таблицу.

Таблица 6.1

Вычисление контрольных битов для кода (7,4)

Позиция кода	Контрольные биты		
	C_3	C_2	C_1
3 (бит M_1)	0	1	1
5 (бит M_2)	1	0	1
6 (бит M_3)	1	1	0
7 (бит M_4)	1	1	1
Суммируемые по модулю 2 биты	M_2 M_3 M_4	M_1 M_3 M_4	M_1 M_2 M_4

Таким образом, передатчик вычисляет значения контрольных разрядов по следующим логическим выражениям:

$C_1=M_1\text{^}M_2\text{^}M_4$;

$C_2=M_1\text{^}M_3\text{^}M_4$;

$C_3=M_2\text{^}M_3\text{^}M_4$.

Приемник проверяет правильность принятого кода, вычисляя следующие выражения:

$C_{11}=C_1\text{^}M_1\text{^}M_2\text{^}M_4$;

$C_{12}=C_2\text{^}M_1\text{^}M_3\text{^}M_4$;

$C_{13}=C_3\text{^}M_2\text{^}M_3\text{^}M_4$.

Построим код Хэмминга для передачи семибитового кода.

Условие 1. Это условие будет выполняться при N=11:

$\{2^{11}/(11+1)=2048/12\approx170\}\geq\{2^{7}=128\}$.

Таким образом, для семи информационных битов (M=7), требуются четыре контрольных бита (C=4).

Условие 2. Определяем местоположение информационных и контрольных битов в коде.

Положение информационных и контрольных битов	M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1
Номер позиции кода	11	10	9	8	7	6	5	4	3	2	1

Условие 3. Определяем логические выражения для вычисления контрольных битов. Для этого построим таблицу.

Таблица 6.2

Вычисление контрольных битов для кода (11,4)

Позиция кода	Контрольные биты			
	C_4	C_3	C_2	C_1
3 (бит M_1)	0	0	1	1
5 (бит M_2)	0	1	0	1
6 (бит M_3)	0	1	1	0
7 (бит M_4)	0	1	1	1
9 бит (M_5)	1	0	0	1
10 (бит M_6)	1	0	1	0
11 (бит M_7)	1	0	1	1
Суммируемые по модулю 2 биты	M5 M6 M7	M2 M3 M4	M1 M3 M4 M6 M7	M1 M2 M4 M5 M7

Таким образом, передатчик вычисляет значения контрольных разрядов по следующим логическим выражениям:

$C_1=M_1\wedge M_2\wedge M_4\wedge M_5\wedge M_7$;

$C_2=M_1\wedge M_3\wedge M_4\wedge M_6\wedge M_7$;

$C_3=M_2\text{^}M_3\text{^}M_4$;

$C_3=M_5\text{^}M_6\text{^}M_7$.

Приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1\text{^}M_1\text{^}M_2\text{^}M_4\text{^}M_5\text{^}M_7$;

$C_{12}=C_2\text{^}M_1\text{^}M_3\text{^}M_4\text{^}M_6\text{^}M_7$;

$C_{13}=C_3\text{^}M_2\text{^}M_3\text{^}M_4$;

$C_{14}=C_4\text{^}M_5\text{^}M_6\text{^}M_7$.

Рассмотрим примеры передачи информации с использованием кода Хэмминга для случаев, когда ошибки при передаче нет и когда есть.

Начнем с кода (7,4).

Пример. В канал связи нужно передать следующий блок информации:

$1101_{(2)}$.

Передатчик формирует код Хэмминга:

M_4	M_3	M_2	C_3	M_1	C_2	C_1
1	1	0	*	1	*	*

$C_1=M_1\text{^}M_2\text{^}M_4=1\text{^}0\text{^}1=0$;

$C_2=M_1\text{^}M_3\text{^}M_4=1\text{^}1\text{^}1=1$;

$C_3=M_2\text{^}M_3\text{^}M_4=0\text{^}1\text{^}1=0$,

тогда код, передаваемый в канал, будет:

M_4	M_3	M_2	C_3	M_1	C_2	C_1
1	1	0	0	1	1	0

Рассмотрим случай, когда не было ошибки при передаче кода. Тогда точно такой же код принял приемник. Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1\text{^}M_1\text{^}M_2\text{^}M_4=0\text{^}1\text{^}0\text{^}1=0$;

$C_{12}=C_2\text{^}M_1\text{^}M_3\text{^}M_4=1\text{^}1\text{^}1\text{^}1=0$;

$C_{13}=C_3\text{^}M_2\text{^}M_3\text{^}M_4=0\text{^}0\text{^}1\text{^}1=0$.

Поскольку значение, составленное из контрольных битов, равно нулю, ошибки нет, и код передается от приемника далее.

Рассмотрим случай, когда была ошибка при передаче кода. Пусть ошибка произошла в третьем бите, т.е. принят код:

M_4	M_3	M_2	C_3	M_1	C_2	C_1
1	1	0	0	0	1	0

Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1\text{^}M_1\text{^}M_2\text{^}M_4=0\text{^}0\text{^}0\text{^}1=1;$

$C_{12}=C_2\text{^}M_1\text{^}M_3\text{^}M_4=1\text{^}0\text{^}1\text{^}1=1;$

$C_{13}=C_3\text{^}M_2\text{^}M_3\text{^}M_4=0\text{^}0\text{^}1\text{^}1=0.$

Поскольку, значение, составленное из контрольных разрядов, равно $011_{(2)}$, ошибка в бите № 3. Приемник меняет значение этого бита на противоположное.

Теперь рассмотрим код (11,7).

Пример. В канал связи нужно передать следующий блок информации:

$1101010_{(2)}$.

Передатчик формирует код Хэмминга:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1
1	1	0	*	1	0	1	*	0	*	*

$C_1=M_1\text{^}M_2\text{^}M_4\text{^}M_5\text{^}M_7=0\text{^}1\text{^}1\text{^}0\text{^}1=1;$

$C_2=M_1\text{^}M_3\text{^}M_4\text{^}M_6\text{^}M_7=0\text{^}0\text{^}1\text{^}1\text{^}1=1;$

$C_3=M_2\text{^}M_3\text{^}M_4=1\text{^}0\text{^}1=0;$

$C_4=M_5\text{^}M_6\text{^}M_7=0\text{^}1\text{^}1=0,$

тогда код, передаваемый в канал, будет:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1
1	1	0	0	1	0	1	0	0	1	1

Рассмотрим случай, когда не было ошибки при передаче кода. Тогда точно такой же код принял приемник. Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1\text{^}M_1\text{^}M_2\text{^}M_4\text{^}M_5\text{^}M_7=1\text{^}0\text{^}1\text{^}1\text{^}0\text{^}1=0;$

$C_{12}=C_2\text{^}M_1\text{^}M_3\text{^}M_4\text{^}M_6\text{^}M_7=1\text{^}0\text{^}0\text{^}1\text{^}1\text{^}1=0;$

$C_{13}=C_3\text{^}M_2\text{^}M_3\text{^}M_4=0\text{^}1\text{^}0\text{^}1=0;$

$C_{14}=C_4\text{^}M_5\text{^}M_6\text{^}M_7=0\text{^}0\text{^}1\text{^}1=0.$

Поскольку значение, составленное из контрольных битов, равно нулю, ошибки нет, и код передается от приемника далее.

Рассмотрим случай, когда была ошибка при передаче кода. Пусть ошибка произошла во втором бите, т.е. принят код:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1
1	1	0	0	1	0	1	0	0	0	1

Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1\text{^}M_1\text{^}M_2\text{^}M_4\text{^}M_5\text{^}M_7=1\text{^}0\text{^}1\text{^}1\text{^}0\text{^}1=0;$

$C_{12}=C_2\text{^}M_1\text{^}M_3\text{^}M_4\text{^}M_6\text{^}M_7=0\text{^}0\text{^}0\text{^}1\text{^}1\text{^}1=1;$

$C_{13}=C_3\text{^}M_2\text{^}M_3\text{^}M_4=0\text{^}1\text{^}0\text{^}1=0;$

$C_{14}=C_4\text{^}M_5\text{^}M_6\text{^}M_7=0\text{^}0\text{^}1\text{^}1=0.$

Поскольку значение, составленное из контрольных битов, равно $0010_{(2)}$, ошибка во втором бите. Приемник меняет значение этого бита на противоположное.

Код Хэмминга является равномерно защищенным, исправляет ошибки как в информационных, так и в контрольных битах. С его помощью возможно исправление одиночной и обнаружение двойной ошибки. Для того чтобы обнаружить двойную ошибку, в код Хэмминга вводят дополнительный контрольный бит E_0. Значение E_0 вычисляется как сумма по модулю 2 всех разрядов кода Хэмминга, т.е. он дополняет код Хэмминга по четности. На стороне приемника, кроме получения контрольных разрядов, которые определяют позицию с ошибкой, вычисляется контрольный разряд E_{10}:

$E_{10}=E_0\text{^}<$сложенные по модулю 2 все остальные биты принятого кода$>$.

Если $E_{10}=0$, то ошибок нет. Если значение, полученное из контрольных разрядов, не равно 0 и $E_{10}=1$, то имеет место одиночная ошибка, позиция которой определяется значением контрольных разрядов. Если значение $C\neq 0$ и $E_{10}=0$, то имеет место двойная ошибка.

Рассмотрим пример для кода, исправляющего одиночную ошибку и обнаруживающего двойную.

Для кода (11,7) формат блока информации:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1	E_0

Пример. В канал связи нужно передать следующий блок информации:

$1101010_{(2)}$.

Передатчик формирует код Хэмминга:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1	E_0
1	1	0	*	1	0	1	*	0	*	*	*

$C_1=M_1\text{^}M_2\text{^}M_4\text{^}M_5\text{^}M_7=0\text{^}1\text{^}1\text{^}0\text{^}1=1$;

$C_2=M_1\text{^}M_3\text{^}M_4\text{^}M_6\text{^}M_7=0\text{^}0\text{^}1\text{^}1\text{^}1=1$;

$C_3=M_2\text{^}M_3\text{^}M_4=1\text{^}0\text{^}1=0$;

$C_4=M_5\text{^}M_6\text{^}M_7=0\text{^}1\text{^}1=0$;

$E_0=C_1\text{^}C_2\text{^}M_1\text{^}C_3\text{^}M_2\text{^}M_3\text{^}M_4\text{^}C_4\text{^}M_5\text{^}M_6\text{^}M_7=1\text{^}1\text{^}0\text{^}0\text{^}1\text{^}0\text{^}1\text{^}0\text{^}0\text{^}1\text{^}1=0$,

тогда код, передаваемый в канал, будет:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1	E_0
1	1	0	0	1	0	1	0	0	1	1	0

Рассмотрим случай, когда не было ошибки при передаче кода. Тогда точно такой же код принял приемник. Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1\text{^}M_1\text{^}M_2\text{^}M_4\text{^}M_5\text{^}M_7=1\text{^}0\text{^}1\text{^}1\text{^}0\text{^}1=0$;

$C_{12}=C_2\text{^}M_1\text{^}M_3\text{^}M_4\text{^}M_6\text{^}M_7=1\text{^}0\text{^}0\text{^}1\text{^}1\text{^}1=0$;

$C_{13}=C_3\text{^}M_2\text{^}M_3\text{^}M_4=0\text{^}1\text{^}0\text{^}1=0$;

$C_{14}=C_4\text{^}M_5\text{^}M_6\text{^}M_7=0\text{^}0\text{^}1\text{^}1=0$;

$E_{10}=E_0\text{^}C_1\text{^}C_2\text{^}M_1\text{^}C_3\text{^}M_2\text{^}M_3\text{^}M_4\text{^}C_4\text{^}M_5\text{^}M_6\text{^}M_7=$
$=0\text{^}1\text{^}1\text{^}0\text{^}0\text{^}1\text{^}0\text{^}1\text{^}0\text{^}0\text{^}1\text{^}1=0$.

Поскольку значение, составленное из контрольных битов, равно 0, и $E_{10}=0$, ошибки нет, и код передается от приемника далее.

Рассмотрим случай, когда была одиночная ошибка при передаче кода. Пусть ошибка произошла во втором бите (не считая E_0, т.е. в C_2) и принят код:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1	E_0
1	1	0	0	1	0	1	0	0	0	1	0

Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1\wedge M_1\wedge M_2\wedge M_4\wedge M_5\wedge M_7=1\wedge 0\wedge 1\wedge 1\wedge 0\wedge 1=0;$

$C_{12}=C_2\wedge M_1\wedge M_3\wedge M_4\wedge M_6\wedge M_7=0\wedge 0\wedge 0\wedge 1\wedge 1\wedge 1=1;$

$C_{13}=C_3\wedge M_2\wedge M_3\wedge M_4=0\wedge 1\wedge 0\wedge 1=0;$

$C_{14}=C_4\wedge M_5\wedge M_6\wedge M_7=0\wedge 0\wedge 1\wedge 1=0;$

$E_{10}=E_0\wedge C_1\wedge C_2\wedge M_1\wedge C_3\wedge M_2\wedge M_3\wedge M_4\wedge C_4\wedge M_5\wedge M_6\wedge M_7=$
$=0\wedge 1\wedge 0\wedge 0\wedge 0\wedge 1\wedge 0\wedge 1\wedge 0\wedge 0\wedge 1\wedge 1=1.$

Поскольку значение $E_{10}=1$ и значение, составленное из контрольных разрядов, равно $0010_{(2)}$, то ошибка во втором бите. Приемник меняет значение этого бита на противоположное.

Рассмотрим случай, когда была двойная ошибка при передаче кода. Пусть ошибка произошла во втором (т.е. в C_2) и в пятом битах (т.е. M_2) и принят код:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1	E_0
1	1	0	0	1	0	0	0	0	0	1	1

Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1\wedge M_1\wedge M_2\wedge M_4\wedge M_5\wedge M_7=1\wedge 0\wedge 0\wedge 1\wedge 0\wedge 1=1;$

$C_{12}=C_2\wedge M_1\wedge M_3\wedge M_4\wedge M_6\wedge M_7=0\wedge 0\wedge 0\wedge 1\wedge 1\wedge 1=1;$

$C_{13}=C_3\wedge M_2\wedge M_3\wedge M_4=0\wedge 0\wedge 0\wedge 1=1;$

$C_{14}=C_4\wedge M_5\wedge M_6\wedge M_7=0\wedge 0\wedge 1\wedge 1=0;$

$E_{10}=E_0\wedge C_1\wedge C_2\wedge M_1\wedge C_3\wedge M_2\wedge M_3\wedge M_4\wedge C_4\wedge M_5\wedge M_6\wedge M_7=$
$=0\wedge 1\wedge 0\wedge 0\wedge 0\wedge 0\wedge 1\wedge 0\wedge 0\wedge 1\wedge 1=0.$

Поскольку значение $E_{10}=0$ и значение, составленное из контрольных разрядов, равно $0111_{(2)}\neq 0$, то произошла двойная ошибка. Приемник не передает код далее, а запрашивает от источника повторную передачу этого блока информации.

Заключение

При создании пособия авторы руководствовались идеей, что предлагаемый теоретический материал, иллюстрированный подробными практическими примерами, окажет существенную помощь читателю в изучении основ арифметики цифровых процессоров. Именно основ, так как в современных процессорах используются более сложные алгоритмы выполнения арифметических операций. Но чтобы разобраться в них, надо получить базовые знания, которые и дает это пособие.

Авторы надеются, что после изучения пособия читатель будет уверенно чувствовать себя в таких вопросах, как форматы представления операндов, кодирование операндов, алгоритмы арифметических операций, контроль арифметических операций, контроль передачи данных. Надеемся, что полученные знания будут успешно использованы для решения практических задач.

Авторами тщательно выверялись все примеры пособия, но если внимательный читатель обнаружит ошибки или сделает замечания по структуре пособия, мы примем и рассмотрим их для исправления в возможных последующих изданиях.

Список литературы

1. Борисова, М. В. Основы информатики и вычислительной техники / М. В. Борисова. – М. : Феникс, 2006. – 544 с.

2. Каймин, В. А. Информатика / В. А. Каймин. – М. : ИНФРА-М, 2006. – 284 с.

3. Цилькер, Б. Я. Организация ЭВМ и систем : учеб. для вузов / Б. Я. Цилькер. – СПб. : Питер, 2006. – 668 с.

4. Хамахер, К. Организация ЭВМ / К. Хамахер, З. Вранешич, С. Заки. – 5-е изд. – СПб. : Киев : Питер; Издательская группа BHV, 2003. – 848 с.

5. Гук, М. Аппаратные средства IBM PC : энциклопедия / М. Гук. – СПб. : Питерком, 1999. – 816 с.

6. Сергеев, Н. П. Основы вычислительной техники / Н. П. Сергеев, Н. П. Вашкевич. – 2-е изд. – М. : Высш. шк., 1988. – 360 с.

7. Майоров, С. А. Структура электронных вычислительных машин / С. А. Майоров, Г. И. Новиков. – Л. : Машиностроение, 1979. – 384 с.

8. Соловьев, Г. Н. Арифметические устройства ЭВМ / Г. Н. Соловьев. – М. : Энергия, 1978. – 176 с.

9. Соренков, Э. И. Точность вычислительных устройств и алгоритмов / Э. И. Соренков, А. И. Телича, А. С. Шаталов. – М. : Машиностроение, 1976. – 200 с.

10. Кнут, Д. Искусство программирования для ЭВМ. Т. 2. Получисленные алгоритмы / Д. Кнут. – М. : Мир, 1977. – 724 с.

11. Карцев, М. А. Вычислительные системы и синхронная арифметика / М. А. Карцев, В. А. Брик. – М. : Радио и связь, 1981. – 360 с.

12. Рабинер, Л. Теория и применение цифровой обработки сигналов / Л. Рабинер, Б. Гоулд. – М. : Мир, 1978. – 848 с.

Учебное издание

Вашкевич Николай Петрович, **Калиниченко** Евгений Иванович

Основы арифметики цифровых процессоров

Учебное пособие

Редактор *О. Ю. Ещина*
Технический редактор *Н. А. Вьялкова*
Корректор *Н. А. Сидельникова*
Компьютерная верстка *М. Б. Жучковой*

Сдано в производство 22.01.10. Формат 60x84 1/16.
Усл. печ. л. 9,3. Уч.-изд. л. 11,1.
Тираж 50. Заказ № 19.

Издательство ПГУ.
440026, Пенза, Красная, 40.