Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

А. С. Кобайло

АРИФМЕТИКО-ЛОГИЧЕСКИЕ ОСНОВЫ ЦИФРОВЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН И

АРХИТЕКТУРА КОМПЬЮТЕРА

Конспект лекций

Часть 1

Арифметические и логические основы цифровых вычислительных машин

Минск 2014

УДК 004.2.031(075.8)
ББК 32.97.я73
К55

Кобайло, А. С.
К 55 Арифметические и логические основы цифровых вычислительных машин и архитектура компьютера: конспект лекций для

студентов специальности «Информационные системы и технологии (издательско-полиграфический комплекс)». Часть 1. Арифметические и логические основы цифровых вычислительных машин / А. С. Кобайло. – Минск : БГТУ, 2013. – 79 с.

Конспект лекций содержит основные сведения в области двоичной арифметики, алгебры логики, методику синтеза цифровых узлов ЭВМ на основе логических выражений.

Издание предназначено для студентов, изучающих дисциплину «Арифметические и логические основы цифровых вычислительных машин и архитектура компьютера». Пособие также может быть полезно аспирантам, изучающим избранные аспекты вышеуказанных проблем.

УДК 004.2.031(075.8)
ББК 32.97.я73

ВВЕДЕНИЕ

Конспект лекций по дисциплине «Арифметические и логические основы цифровых вычислительных машин» разработан для студентов высших учебных заведений, обучающихся по специальности 1–40 01 02–03 «Информационные системы и технологии» (издательско-полиграфический комплекс). Данное пособие составлено в соответствии с требованиями Образовательного стандарта и типового учебного плана специальности.

Дисциплина предусматривает изучение арифметических основ вычислительной техники, алгебры логики, схемотехники ЭВМ. Цель изучения – освоение студентами:

– арифметических основ вычислительной техники на основе двоичной арифметики;

– логических основ вычислительной техники на базе изучения алгебры логики;

– схемотехнических основ и архитектурной организации ЭВМ и ВС.

Конспект лекций предназначен для оказания помощи студентам в ознакомлении с основами организации и функционирования ЭВМ, приобретении навыков синтеза цифровых устройств с помощью законов булевой алгебры.

Предметом дисциплины является фундамент арифметической и логической организации и функционирования средств цифровой вычислительной техники. Дисциплина занимает важное место в подготовке современного инженера, специализирующегося в области разработки и использования современных информационных технологий и систем

Содержание конспекта соответствует учебным программам для студентов специальности 1–40 01 02–03 «Информационные системы и технологии» (издательско-полиграфический комплекс). При подготовке издания использовались материалы изданного в БГУИР учебно-методического пособия «Основы организации и функционирования ЭВМ» в 3-х частях авторов А. Т. Пешков и А. С. Кобайло.

1. АРИФМЕТИЧЕСКИЕ ОСНОВЫ ЦИФРОВЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Лекция 1. Системы счисления

1.1. Понятие системы счисления

В своей повседневной деятельности человек использует различные системы счисления, к которым относятся десятичная система счисления, римская система, система исчисления времени и т. д. Все их можно подразделить на позиционные и непозиционные.

В непозиционных системах счисления «доля» цифры или ее вес в количественном измерении записанного числа не зависит от местоположения данной цифры в записи этого числа. Типичным примером такой системы счисления является римская. В ней используются цифры:

I	V	X	L	C	D	M	– римские цифры
1	5	10	50	100	500	1000	– их десятичные эквиваленты

При количественной оценке числа его значение определяется как сумма значений цифр, составляющих запись числа, кроме пар, состоящих из цифры меньшего веса, предшествующей цифре большего веса, значение которой определяется как разность веса большей и меньшей цифр.

Пример

Значение числа MMMCMLIX определяется как сумма:

$$1000 + 1000 + 1000 + (1000 – 100) + 50 + (10 – 1),$$

что соответствует десятичному эквиваленту 3959.

Количественная оценка числа, записанного в позиционной системе счисления, определяется как сумма произведений значения цифр, составляющих запись числа, умноженных на вес позиции, в которой располагается цифра.

Примером такой системы счисления является широко используемая десятичная система счисления.

Пример

Количественная оценка десятичного числа 3959_{10} определяется как

$3 \times 1000 + 9 \times 100 + 5 \times 10 + 9 \times 1,$

где 1000, 100, 10, 1 – соответственно веса четвертого, третьего, второго, первого разрядов записи оцениваемого числа.

Десятичная *система счисления* является также системой с равномерно распределенными весами, которые характеризуются тем, что соотношение весов двух любых соседних разрядов имеет для такой системы одинаковое значение. Это соотношение называется основанием системы счисления, которое далее будем обозначать как «q».

Общая запись числа в системе с равномерно распределенными весами имеет вид

$$N_q = A_n A_{n-1} \ldots A_2 A_1 A_0 . \quad (1.1)$$

Значение такого числа определяется как

$$N_q = A_n \times q^n + A_{n-1} \times q^{n-1} + A_{n-2} \times q^{n-1} + \ldots A_2 \times q^2 + A_1 \times q^1 + A_0 \times q^0, \quad (1.2)$$

где A_i – цифра записи числа, удовлетворяющая условию

$$0 \le A_i \le (q-1),$$

где q – основание системы счисления.

При q =10 A изменяется в диапазоне от 0 до 9.

Запись числа N в виде (1.1) называется кодированной, а запись в форме (1.2) – расширенной.

Помимо q = 10 (десятичная система счисления), возможны другие значения для основания системы счисления:

- двоичная система счисления;
- восьмеричная система счисления;
- шестнадцатеричная система счисления и т. д.

В различных системах счисления в качестве цифр используются обозначения соответствующих цифр десятичной системы счисления – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, а в случае, когда десятичных цифр «не хватает» (для систем счисления с основанием q, большим чем 10), для цифр, превышающих 9, вводятся дополнительные обозначения, например, для q = 16, это будут обозначения A, B, C, D, E, F, которые соответствуют шестнадцатеричным цифрам (десятичные эквиваленты их равны, соответственно 10, 11, 12, 13, 14, 15).

В связи с тем, что в дальнейшем изложении будут использоваться различные системы счисления, примем такое обозначение:

N_q – число N, представленное в системе счисления с основанием q.

Примеры записи чисел в различных системах счисления:

$N_2 = 10011011 = 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + + 1 \times 2^1 + 1 \times 2^0$;

$N_8 = 471025 = 4 \times 8^5 + 7 \times 8^4 + 1 \times 8^3 + 0 \times 8^2 + 2 \times 8^1 + 5 \times 8^0$;

$N_{16} = 84FE4A = 8 \times 16^5 + 4 \times 16^4 + F \times 16^3 + E \times 16^2 + 4 \times 16^1 + + A \times 16^0$;

$N_{10} = 35491 = 3 \times 10^4 + 5 \times 10^3 + 4 \times 10^2 + 9 \times 10^1 + 1 \times 10^0$.

На основании вышеизложенного можно заключить, что запись одного и того же числа в различных системах счисления будет тем длиннее, чем меньше основание системы счисления. Например,

$N = 2063_{10} = 100000001111_2 = 4017_8 = 80F_{16}$.

При работе с различными системами счисления полезно помнить соотношения, приведенные в табл. 1.1. и 1.2.

Таблица 1.1

Соответствие показателя степени двоичного числа значению десятичного числа

n	0	1	2	3	4	5	6	7	8	9	10	11
2^n	1	2	4	8	16	32	64	128	256	512	1024	2048

Таблица 1. 2

Соответствие символов различных систем счисления

№ п/п	$q = 2$	$q = 8$	$q = 16$	$q = 10$	Десятичный эквивалент	Двоичный эквивалент
1	0	0	0	0	0	0000
2	1	1	1	1	1	0001
3	–	2	2	2	2	0010
4	–	3	3	3	3	0011
5	–	4	4	4	4	0100
6	–	5	5	5	5	0101
7	–	6	6	6	6	0110
8	–	7	7	7	7	0111
9	–	*10*	8	8	8	1000
10	–	*11*	9	9	9	1001
11	–	*12*	A	*10*	10	1010
12	–	*13*	B	*11*	11	1011

13	–	*14*	C	*12*	12	1100
14	–	*15*	D	*13*	13	1101
15	–	*16*	E	*14*	14	1110
16	–	*17*	F	*15*	15	1111

Человек в своей практической деятельности наиболее часто использует *десятичную* систему счисления. *Двоичная система счисления* является удобной для обработки информации в ЭВМ. Промежуточное место между ними занимает *двоично-десятичная система* счисления, которая, в принципе, является десятичной, но отдельные десятичные цифры в ней записываются в виде набора двоичных разрядов. Существуют разные двоично-десятичные системы.

Например, десятичное число 804714 в двоично-десятичной системе представляется в виде 1000 0000 0100 0111 0001 0100. В дальнейшем для сокращения будем использоваться название «двоично-десятичная система», имея в виду двоично-десятичную систему 8, 4, 2, 1. Цифры в обозначении разновидности двоично-десятичной системы указывают веса соответствующих двоичных разрядов.

1.2. Перевод чисел из одной системы счисления в другую

Наличие различных систем счисления предполагает использование способов перевода записи числа из одной системы в другую. Для этой цели применяются следующие методы преобразований:

- преобразования с использованием весов разрядов в исходной и в искомой записи числа;
- деления (умножения) на новое основание;
- с использованием особого соотношения заданной и искомой систем счисления.

Метод преобразования с использованием весов разрядов

Метод преобразования с использованием весов разрядов записи числа в исходной и в искомой системах предполагает применение расширенной записи числа (1.2) в некоторой системе счисления.

Метод имеет две разновидности в зависимости от того, какая система счисления (исходная или искомая) является более привычной. Если более привычной является искомая система, то на основании

расширенной записи исходного числа подсчитываются значения ее отдельных разрядов в новой системе счисления. Далее полученные значения суммируются. Например, при преобразовании целого двоичного числа N_2 = 110011010 в десятичную систему счисления исходное число представляется в расширенной записи $N = 2^8 + 2^7 + + 2^4 + 2^3 + 2^1$ и рассчитывается вес отдельных (ненулевых) двоичных разрядов в десятичной системе счисления:

256, 128, 16, 8, 2.

Затем искомая запись числа определяется как сумма весов всех ненулевых разрядов записи числа в заданной системе счисления:

256 + 128 + 16 + 8 + 2 = 410.

При преобразовании правильных дробей в принципе используется тот же подход, но при расчете весов отдельных разрядов берутся отрицательные степени основания счисления.

Пример

Найти двоичный эквивалент десятичного числа:

436_{10} = ____?$_{2.}$

Решение

Первый (старший) разряд, имеющий значение 1 в искомой двоичной записи числа, будет разряд весом 2^8 = 256. С помощью остальных (младших) разрядов искомой записи числа необходимо представить значение 180 (180 – остаток, полученный как 436 – 256).

Второй разряд с весом 2^7 = 128 будет иметь в искомой двоичной записи числа значение 1. С помощью остальных (более младших) разрядов искомой записи числа необходимо представить значение 52 (остаток, полученный как 180 – 128).

Третий разряд с весом 2^6 = 64 будет иметь в искомой двоичной записи числа значение 0.

Четвертый разряд с весом 2^5 = 32 будет иметь в искомой двоичной записи числа значение 1, а остаток – 20.

Пятый разряд с весом 2^4 = 16 будет иметь в искомой двоичной записи числа значение 1, а остаток – 4.

Шестой разряд с весом 2^3 = 8 будет иметь в искомой двоичной записи числа значение 0.

Седьмой разряд с весом 2^2 = 4 будет иметь в искомой двоичной записи числа значение 1, а остаток – 0.

Восьмой разряд с весом $2^1 = 2$ будет иметь в искомой двоичной записи числа значение 0.

Девятый разряд с весом $2^0 = 1$ будет иметь в искомой двоичной записи числа значение 0.

Таким образом

$436_{10} = 110110100_2$.

Пример

Найти двоичный эквивалент числа $0{,}7_{10} = \underline{?}_2$.

Решение

Предварительный результат ищется с точностью до пяти двоичных разрядов, причем пятый разряд используется только для округления при переходе к четырехразрядному окончательному результату.

Первый (старший) разрядом весом $2^{-1} = 0{,}5$ искомой двоичной записи числа будет иметь значение 1. С помощью остальных (младших) разрядов искомой записи числа необходимо представить значение 0,2 (0,2 – остаток, полученный как 0,7 – 0,5 = 0,2).

Второй (старший) разряд с весом $2^{-2} = 0{,}25$ в искомой двоичной записи числа будет иметь значение 0.

Третий разрядом с весом $2^{-3} = 0{,}13$ в искомой двоичной записи числа будет иметь значение 1. С помощью остальных (более младших) разрядов искомой записи числа необходимо представить значение 0,07 (0,07 – остаток, полученный как 0,2 - 0,13).

Четвертый разрядом с весом $2^{-4} = 0{,}06$ в искомой двоичной записи числа будет иметь значение 1, а остаток 0,01.

Пятый разряд с весом $2^{-5} = 0{,}03$ искомой двоичной записи числа будет иметь значение 0.

Таким образом, десятичное число $0{,}7_{10} = 0{,}10110_2$.

После округления имеет место $0{,}7_{10} = 0{,}1011_2$.

Метод деления (умножения) на новое основание

Метод деления (умножения) имеет две разновидности соответственно для преобразования целых и дробных чисел.

1. Преобразование целых чисел.

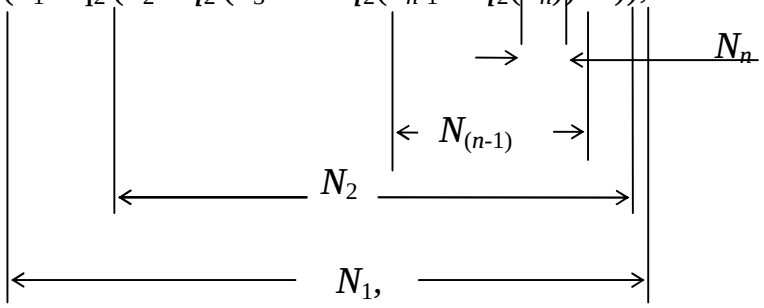
Задачу представления числа N, заданного в системе q_1, в системе счисления с основанием q_2 можно рассматривать как задачу поиска

коэффициентов полинома, представляющего собой расширенную запись числа N в системе счисления q_2:

$$N_{q1} = a_0 + a_1 \times q_2^{1} + a_2 \times q_2^{2} + ... + a_{n-2} \times q_2^{n-2} + a_{n-1} \times q_2^{n-1} + a_n \times q_2^{n} = Nq_2 . \quad (1.3)$$

Введем скобочную форму для выражения (1.3):

$$N_{q1} = N_{q2} = a_0 + q_2 (a_1 + q_2 (a_2 + q_2 (a_3 + . + q_2 (a_{n-1} + q_2 (a_n)) \ldots));$$



Обозначим выражение в первой скобке как N_1, выражение во второй скобке как N_2, в третьей – как N_3 и т. д., выражение в $(n–1)$-й скобке – как $N_{(n–1)}$, выражение в n-й скобке – как N_n. Теперь, основываясь на выражении (1.3), можно утверждать, что при делении N_{q1}/q_2 будет получена целая часть частного $\text{int}(N_{q1}/q_2)$ и остаток $\text{rest}(N_{q1}/q_2)$. Это можно записать:

$N_{q1}/q_2 \rightarrow \text{int}(N_{q1}/q_2)$ — целая часть частного N_1, и остаток $\text{rest}(N_{q1}/q_2)$, равный a_0.

Аналогично для остальных скобок:

$N_1/q_2 \rightarrow \text{int}(N_1/q_2)$ равное N_2 и остаток $\text{rest}(N_1/q_2)$, равный a_1;

$N_2/q_2 \rightarrow \text{int}(N_2/q_2)$ равное N_3, и остаток $\text{rest}(N_2/q_2)$, равный a_2;

$N_{(n–2)}/q_2 \rightarrow \text{int}(N_{(n–2)}/q_1)$ равное $N_{(n–1)}$, остаток $\text{rest}(N(n = 2)/q_1)$, равный $a_{(n–2)}$;

$N_{(n–1)}/q_2 \rightarrow \text{int}(N_{(n–1)}/q_2) = N_n = a_n$ и остаток $\text{rest}(N_{(n–1)}/q_2)$, равный $a_{(n–1)}$, при этом $N_n < q_2$.

Отсюда вытекает правило формирования коэффициентов полинома (1.3) или разрядов записи заданного числа N в системе счисления с основанием q_2:

- необходимо разделить исходное число N_{q1} на новое основание q_2, при этом получив целое частное и остаток;
- полученный остаток снова необходимо разделить на q_2, процесс деления продолжается до тех пор, пока частное будет не меньше нового основания q_2. Если очередное сформированное частное будет меньше, чем q_2, то процесс формирования записи заданного числа в новой системе с основанием q_2 считается законченным, а в качестве искомых разрядов новой записи числа используются результаты выполненных операций деления следующим образом:
- в качестве старшего разряда берется значение последнего частного, для остальных разрядов используются значения остатков в порядке, обратном порядку их получения.

Пример

Найти запись в двоичной форме десятичного числа N_{10} = 436.

Решение

Делим сначала исходное число N_{10}, а затем получаемые частные на значение нового основания 2 до получения частного со значением, меньше чем 2:

436/2 → int(436/2) = 218 и rest (436/2) = 0;

218/2 → int(218/2) = 109 и rest (218/2) = 0;

109/2 →int(109/2) = 54 и rest (109/2) = 1;

54/2 → int(54/2) = 27и rest (54/2) = 0;

27/2 → int(27/2) = 13 и rest (27/2) = 1;

13/2→int(13/2) = 6 и rest (13/2) = 1;

6/2 → int(6/2) = 3 и rest (6/2) = 0;

3/2 → int(3/2) = 1 и rest (3/2) = 1.

Таким образом: 436 = 11 0110100.

2. Преобразование дробных чисел

Задача представления дробного числа M_{q1}, заданного в системе q_1, в системе счисления с основанием q_2, можно рассматривать как задачу поиска коэффициентов полинома, представляющего собой расширенную запись числа M в системе счисления q_2:

$$B_1 \times q_2^{-1} + B_2 \times q_2^{-2} + B_3 \times q_2^{-3} + ... + B_{n-2} \times q_2^{-(n-2)} + B_{n-1} \times q_2^{-(n-1)} + B_n \times q_2^{-n} = M_q \quad (1.4)$$

Введем скобочную форму для выражения (1.4). Обозначим выражение в первой скобке как M_1, выражение во второй – как M_2, в третьей скобке – как M_3 и т. д., выражение в $(n-1)$-й скобке как M_{n-1}, выражение в n-й скобке – как M_n:

$$M_{q2} = M_{q1} = q_2^{-1}(B_1 + q_2^{-1}(B_2 + q_2^{-1}(B_3 + ... + q_2^{-1}(B_{n-1} + q_2^{-1}(B_n))...))).$$

Число M_{q1} – правильная дробь, поэтому при умножении $M_{q1} \times q_2$ будет получено произведение, в общем случае состоящее из целой части $\text{int}(M_{q1} \times q_2)$ и дробной части $\text{DF}(M_{q1} \times q_2)$. Использование введенных обозначений позволяет записать:

$$M_{q1} \times q_2 = (\text{int}(M_{q1} \times q_2) = B_1) + (\text{DF}(M_{q1} \times q_2) = M_1),$$

аналогично для остальных скобок будем иметь следующее:

$$M_1 \times q_2 = (\text{int}(M_1 \times q_2) = B_2) + (\text{DF}(M_1 \times q_2) = M_2);$$

$$M_2 \times q_2 = (\text{int}(M_2 \times q_2) = B_3) + (\text{DF}(M_2 \times q_2) = M_3);$$

$$M_3 \times q_2 = (\text{int}(M_3 \times q_2) = B_4) + (\text{DF}(M_3 \times q_2) = M_4);$$

$$M_{n-2} \times q_2 = (\text{int}(M_{n-2} \times q_2) = B_{n-1}) + (\text{DF}(M_{n-2} \times q_2) = M_{n-1});$$

$$M_{n-1} \times q_2 = (\text{int}(M_{n-1} \times q_2) = B_n) + (\text{DF}(M_{n-1} \times q_2) = M_n);$$

$$M_n \times q_2 = (\text{int}(M_n \times q_2) = B_{n+1}) + (\text{DF}(M_n \times q_2) = M_{n+1}).$$

Отсюда вытекает следующее правило формирования коэффициентов полинома, которые одновременно являются разрядами записи заданного числа M в системе счисления с основанием q_2:

- определяется количество разрядов «n» в записи числа M_{q2} в новой системе счисления;
- исходное число M_{q1} умножается на q_2, при этом будет получено смешанное число;
- дробная часть полученного произведения снова умножается на q_2 и т. д.; процесс умножения повторяется n +1 раз. В качестве искомых разрядов новой записи числа используются результаты выполненных операции деления следующим образом:
- в качестве первого старшего разряда искомой записи числа в новом основании берется значение целой части первого произведения, в качестве второго старшего разряда искомой записи числа в новом основании берется значение целой части второго произведения и т. д.

Пример

Найти запись в двоичной форме десятичного числа M_{10} = 0,7.

Решение

Определяем количество разрядов числа M_2. Так как исходная запись числа содержит один десятичный разряд, то запись данного числа в двоичном основании должна содержать четыре разряда. Учитывая округление, ищется предварительный двоичный эквивалент с пятью разрядами.

Умножаем исходное число M_{10}, а затем дробные части последовательно получаемых произведений на новое основание 2. Выполняется пять таких операций умножения, в результате получаем:

0,7 ×2 = 1,4 (int(0,7 ×2) = 1 и DF (0,7 ×2) = 0,4);

0,4 ×2 = 0,8 (int(0,4 ×2) = 0 и DF(rest (0,4 ×2) = 0,8);

0,8 ×2 = 1,6 (int(0,8 ×2) = 1 и DF(rest (0,8 ×2) = 0,6);

0,6 ×2 =1,2 (int(0,6 ×2) = 1 и DF(rest (0,6 ×2) = 0,2);

0,2 ×2 = 0,4 (int(0,2 ×2) = 0 и DF(rest (0,2 ×2) = 0,4).

Таким образом, 0,7 = 0,10110, а окончательный результат перехода в двоичную систему будет $0,7_{10} = 0,1011_2$.

1. 3. Метод с использованием особого соотношения оснований заданной и искомой систем счисления

Данный метод применим в тех случаях, когда исходное q_1 и новое q_2 основания могут быть связаны через целую степень, т.е. когда выполняются условия: $q_1^m = q_2$ (*условие 1*) или $q_2^m = q_1$ (*условие 2*).

Если имеет место *условие 2*, то для заданного в системе с основанием q_1 числа $N_{q1} = a_n a_{n-1} a_{n-2} \ldots a_1 a_0$ запись его в системе с новом основании q_2 определяется следующим образом:

- каждому разряду a_i исходной записи числа ставится в соответствие его m-разрядный эквивалент в системе счисления с основанием q_2;
- искомая запись всего заданного числа формируется за счет объединения всех полученных m-разрядных групп.

Если имеет место *условие 1*, то запись заданного числа

$N = a_n a_{n-1} a_{n-2} \ldots a_1 a_0$

в системе с новом основании q_2 формируется следующим образом:

- исходная запись числа разбивается на группы по m разрядов, двигаясь от точки вправо и влево (недостающие разряды в крайних группах (слева и справа) дополняются нулями;
- каждой полученной группе ставится в соответствие цифра новой системы счисления;
- искомая запись заданного числа в новой системе счисления образуется из цифр, соответствующих группам, на которые была разбита исходная запись.

Пример

Найти двоичный эквивалент восьмеричного числа 67401.64_8.

Решение

Основания исходной и новой систем счисления можно выразить через целую степень:

$2^3 = 8$.

Поэтому применяем третий метод для случая перехода из системы с большим основанием в систему с меньшим основанием. Ставим в соответствие каждой цифре исходной записи числа трехразрядный двоичный код (*триаду*):

6 7 4 0 1 6 4

110 111 100 000 001 110 100

Формируем окончательный результат посредством объединения полученных трехразрядных двоичных чисел в единый двоичный эквивалент:

$67401.64_8 = 110111100000001.110100.$

Пример

Найти шестнадцатеричный эквивалент двоичного числа

$N = 11100101110110.111011001_2.$

Решение

Основания исходной и новой систем счисления можно выразить через целую степень:

$2^4=16.$

Поэтому применяем третий метод для случая перехода из системы с меньшим основанием в систему с большим основанием. Разбиваем исходную запись числа на группы по четыре разряда (*тетрады*) вправо и влево от точки, в крайних левой и правой группах недостающие разряды заполняем нулями и каждой полученной группе из четырех разрядов ставим в соответствие цифру шестнадцатеричной системы счисления

0011 1001 0111 0110 . 1110 1100 1000
3 9 7 6 E C 8

Формируем окончательный результат посредством объединения полученных цифр в единый шестнадцатеричный эквивалент

$11100101110110.111011001_2 = 3976.EC8_{16}.$

Пример

Найти шестнадцатеричный эквивалент числа 67401.64_8, представленного в восьмеричной системе счисления.

Решение

Основания исходной q_1 и новой q_2 систем счисления не могут быть связаны через целую степень, поэтому напрямую третий метод перехода неприменим. Однако существует система с двоичным основанием, для которой допустим третий метод перехода и восьмеричную (исходную для данного примера), и в шестнадцатеричную (новую систему для данного примера) системы счисления, т. к.

$2^3 = 8$ и $2^4 = 16$.

Поэтому в данном случае для решения поставленной задачи целесообразно использовать два быстрых перехода из восьмеричной системы счисления в двоичную (промежуточную), а затем из двоичной системы счисления в шестнадцатеричную третьим методом. Это будет гораздо быстрее, чем использовать для заданного преобразования второй или третий метод.

Таким образом, поставленная задача решается в следующем порядке:

$67401.64_8 = 110\ 111\ 100\ 000\ 001\ .\ 110\ 100_2$;

$110\ 111\ 100\ 000\ 001\ .\ 110\ 100_2 = 0110\ 1111\ 0000\ 0001.\ 1101\ 0000_2 = 6\ F\ 0\ 1.\ D\ 0_{16}$, т.е.:

$67401.64_8 = 6F01.D0_{16}$.

Пример

Найти двоичный эквивалент числа 6740_{10}.

Решение

Основания исходной q_1 и новой q_2 систем счисления не могут быть связаны через целую степень, поэтому третий метод перехода неприменим. В принципе здесь целесообразно использовать второй метод – метод деления на новое основание. Однако в этом случае потребуется большое количество операций деления на два. Для сокращения количества операций деления может оказаться целесообразным решить эту задачу за счет перехода с использованием второго метода в промежуточную шестнадцатеричную систему счисления, а затем, используя третий метод, быстро перейти в заданную двоичную систему счисления:

- выполняем переход в промежуточную систему счисления:

6740/16= 421 (остаток 4);

421/16=26 (остаток 5);

26/16 = 1 (остаток 10);

- в промежуточной системе счисления имеем:

$6740_{10} = 1A54_{16}$,

- выполняем переход из промежуточной системы счисления в заданную:

$1A54_{16} = 0001\ 1010\ 0101\ 0100_2$.

Как видно из вышеприведенного, для заданного перехода потребовалось выполнить только 3 операции деления на «16» вместо 13-ти операций деления на 2.

Лекция 2. Двоичная арифметика

2.1. Операция сложения в двоичной системе счисления

При выполнении любой операции результат ищется согласно соответствующим правилам, которые удобно представлять в табличной форме, где для всех возможных комбинаций значений одноразрядных операндов приводятся значения результата.

Правила сложения в двоичной системе счисления

+	0	1
0	0	1

1	1	0*

Все возможные значения первого слагаемого задаются во второй и третьей строках первой колонки; все возможные значения второго слагаемого – во второй и третьей колонках первой строки. На пересечении отмеченных значениями операндов строк и колонок располагается результат их сложения. В таблице знаком «*» отмечен случай, когда в текущем разряде результата получен ноль и имеет место перенос в ближайший старший разряд.

Пример

$$\begin{array}{r} 1011100110 \\ +\ 0101101101 \\ \hline 10001010011 \end{array}$$

В общем случае при формировании значения в текущем разряде результата приходится дважды применять приведенную таблицу сложения: первый раз при сложении соответствующих разрядов операндов, формируя так называемую поразрядную сумму, и второй раз – при сложении разряда сформированной поразрядной суммы и переноса, пришедшего из ближайшего младшего разряда.

При машинной реализации операции сложения сначала формируется поразрядная сумма операндов без учета переноса, далее формируется код переноса и затем с помощью специальных логических цепей учитываются возникшие переносы. При этом перенос, возникший в некотором разряде, может изменить не только ближайший старший разряд, но и целую группу старших разрядов. В худшем случае перенос, возникший в самом младшем разряде, может изменить значение старших разрядов сформированной поразрядной суммы вплоть до самого старшего.

При формировании поразрядной суммы и учете возникших переносов используется следующая классификация разрядов складываемых операндов:

- разряд, генерирующий перенос (оба операнда в этом разряде имеют «1»);

- разряд, пропускающий перенос (операнды в этом разряде имеют разные значения);

- разряд, блокирующий распространение переноса (операнды в этом разряде имеют одинаковые значения).

2.2. Операция вычитания

Правила вычитания представлены ниже в форме таблицы.

Правила вычитания в двоичной системе счисления

–	0	1
0	0	1
1	1*	0

Все возможные значения вычитаемого задаются во второй и третьей строках первой колонки; все возможные значения уменьшаемого – во второй и третьей колонках первой строки. На пересечении отмеченных значениями операндов строк и колонок располагается результат вычитания второго операнда из первого. В таблице знаком «*» отмечен случай, когда в текущем разряде результата получена единица при займе из ближайшего старшего разряда.

Пример

$$\begin{array}{r} 1000111001 \\ -\ 0101101101 \\ \hline 0011001100 \end{array}$$

Как видно из таблицы и приведенного примера, реализация операции вычитания не сложнее операции сложения.

В ЭВМ никогда в перечне выполняемых операций арифметического устройства не присутствует одновременно операция сложения и операция вычитания. При этом, как правило, присутствует только операции сложения. Что же касается операции вычитания, то она реализуется за счет прибавления к уменьшаемому значения вычитаемого, взятого с противоположным знаком.

2.3. Операция умножения

Умножение в двоичной системе счисления задается в табличной форме.

Правила умножения в двоичной системе счисления

*	0	1
0	0	0
1	0	1

Все возможные значения множимого задаются во второй и третьей строках первой колонки; все возможные значения множителя – во второй и третьей колонках первой строки. На пересечении отмеченных значениями операндов строк и колонок располагается результат умножения первого операнда на второй.

При умножении многоразрядных операндов, как правило (особенно в десятичной системе счисления), используется метод, при котором формирование произведения выполняется за счет суммирования частичных произведений, которые оформляются посредством умножения множимого на отдельные разряды множителя с учетом веса соответствующего разряда множителя.

Таблица умножения одноразрядных операндов в двоичной системе существенно упрощает задачу формирования частичного произведения:

– частичное произведение для разряда множителя равняется нулю, если этот разряд равен нулю;

– частичное произведение для разряда множителя равняется множимому, взятому с соответствующим весом, если разряд множителя равен единице.

При последовательном способе формирования частичных произведений последние могут рассчитываться поочередно для отдельных разрядов множителя начиная с младшего или старшего разряда. При десятичном основании, как правило, формирование частичных произведений осуществляется начиная с младшего разряда множителя.

Пример1

Найти произведение двоичных чисел 1011 и 1101, начиная формирование частичных произведений со старшего разряда множителя.

Решение

При формировании частичных произведений, начиная со старшего разряда множителя, процесс формирования произведения заданных операндов можно представить следующим образом:

$$
\begin{array}{r}
1\,0\,1\,1 \\
\times 1\,1\,0\,1 \\
\hline
1\,0\,1\,1 \\
+\,1\,0\,1\,1 \\
+\quad 0\,0\,0\,0 \\
+\quad\;\; 1\,0\,1\,1 \\
\hline
1\,0\,0\,0\,1\,1\,1\,1
\end{array}
$$

При реализации умножения рассматриваемым способом требуется использование $2n$-разрядного сумматора для подсчета промежуточных и конечного произведения и $2n$-разрядного сдвигающего регистра для хранения множителя.

Пример 2

Найти произведение двоичных чисел 1011 и 1101, начиная формирование частичных произведений с младшего разряда множителя и применяя учет сформированных частичных произведений по мере их формирования.

Решение

```
    1 0 1 1
  × 1 1 0 1
    0 0 0 0
  + 1 0 1 1
    1 0 1 1
 + 0 0 0 0_
   0 1 0 1 1
 + 1 0 1 1___
   1 1 0 1 1 1
 + 1 0 1 1_____
 1 0 0 0 1 1 1 1
```

Реализация данного метода умножения требует использовать $2n$-разрядный сумматор для последовательного, от такта к такту, формирования $2n$-разрядного произведения и $2n$-разрядный регистр для хранения и сдвига влево множимого.

В данном примере для того, чтобы учесть то, что очередной разряд множителя имеет вес, в два раза больший, чем предыдущий разряд, его частичное произведение учитывается со сдвигом множимого на один разряд влево при суммировании с промежуточным результатом. В таком случае говорят, что умножение выполняется со сдвигом множимого.

Однако для учета того, что очередное частичное произведение имеет вес, в два раза больший веса предыдущего частичного произведения, можно при суммировании сдвигать вправо промежуточный ре-

зультат. В таком случае говорят: умножение выполняется со сдвигом промежуточного результата. При использовании данного подхода умножение чисел 1101 и 1011 представляется в виде следующих действий:

		множимое		
		множитель		
1 разряд		промежуточный результат		
2 разряд	× 1011 1101	(начальное значение)		
	0000 + 1011	1	результат сдвига	
3 разряд	1011 + 0101			
	0000	1		
4 разряд	0101 + 0010	11	результат сдвига	
	1011			
	1101 + 0110	11		
В колонке	1011 10001	111	результат сдвига	первой
	1000			
		111		
		1111	результат сдвига	

приведены номера отрабатываемых разрядов множителя начиная с младшего. Эти номера отмечают строки, в которых учитывается частичное произведение, соответствующее этому разряду множителя. В этой же первой колонке расположены единицы переполнения, возникающие при суммировании промежуточного результата и очередного частичного произведения, сформированного для соответствующего отрабатываемого разряда множителя (в данном случае единица переполнения имеется при отработке четвертого, самого старшего, разряда множителя).

Вторая колонка отражает длину основной разрядной сетки ($n = 4$).

В третей колонке представлены разряды промежуточных и конечного произведений, «вытолкнутых» за пределы основной разрядной сетки в процессе выполнения очередного сдвига промежуточного произведения.

Из приведенного примера видно, что выталкиваемые за пределы разряды промежуточных произведений в дальнейшем не изменяются и их значение не влияет на значение суммы, формируемой в пределах основной разрядной сетки. Поэтому для реализации этого метода умножения требуется n-разрядный сумматор, обеспечивающий суммирование только в пределах основной разрядной сетки.

Аналогично умножению начиная с младшего разряда множителя при умножении со старших разрядов можно заменить сдвиг вправо множителя на сдвиг влево промежуточного произведения.

Операция умножение в общем случае дает точный результат – $2n$-разрядное произведение, где n – разрядность операндов.

В ЭВМ при выполнении различных операций, в том числе и операции умножения, разрядность операндов и результатов одинаковая. Это означает, что при умножении правильных дробей последние (младшие) n разрядов отбрасываются, а старшие n разрядов округляются с учетом отбрасываемых младших. Например, произведение $0{,}1011 \times 0{,}1101 = 0{,}10001111$ представляется в n-разрядном варианте как 0,1001. В этом случае операция умножения считается приближенной.

Возможные методы реализации операции умножения можно классифицировать по двум признакам:

– начиная с какого разряда (со старшего или младшего) выполняется отработка множителя;

– что сдвигается – множимое или промежуточное произведение.

Используя эти два классификационных признака, можно выделить четыре метода умножения:

– *умножение с младших* разрядов множителя со сдвигом множимого; при реализации данного метода требуется $2n$- разрядный сумматор, $2n$- разрядный регистр промежуточного произведения, $2n$ - разрядный регистр для хранения и сдвига множимого и n- разрядный регистр для хранения множителя;

– *умножение с младших* разрядов множителя со сдвигом промежуточного произведения; при реализации данного метода требуется n-разрядный сумматор, $2n$-разрядный регистр промежуточного произведения, n-разрядный регистр для хранения множимого и n-разрядный регистр для хранения множителя;

– *умножение со старшего* разряда множителя со сдвигом множимого; при реализации данного метода требуется $2n$-разрядный сумматор, $2n$-разрядный регистр промежуточного произведения, $2n$-

разрядный регистр для хранения и сдвига множимого и n-разрядный регистр для хранения множителя;

– *умножение со старшего* разряда множителя со сдвигом промежуточного произведения; при реализации данного метода требуется n-разрядный сумматор, $2n$-разрядный регистр промежуточного произведения, n-разрядный регистр для хранения множимого и n-разрядный регистр для хранения множителя.

2.4. Деление двоичных чисел

Деление в принципе является неточной операцией, поэтому при её выполнении прежде всего устанавливается количество разрядов частного, которые подлежат определению.

Деление в двоичной системе счисления может выполняться точно так же, как и в десятичной, однако формирования частного двоичных операндов реализуется гораздо проще, чем в десятичной системе, т.к.:

- упрощается процедура подбора очередной цифры вследствие того, что в двоичной системе очередной цифрой может быть одна из двух - либо 0, либо 1;
- упрощается процедура умножения найденной цифры частного на делитель.

Пример

Найти частное от деления двоичных чисел 0.1001 на 0.1101.

Решение

По умолчанию считается, что разрядность результата и операндов одинаковая, поэтому окончательный результат должен иметь в данном случае 4 разряда. Учитывая необходимость округления, найдем дополнительный пятый разряд, на основании которого выполним округление.

```
0. 1 0 0 1        | 0. 1101   – делитель
 -  1 1 0 1        0. 10110   – частное
    1 0 1 0
  -  1 1 0 1
       1 1 1
   -  1 1 0 1
            1   – остаток
```

После округления получаем окончательный результат: 0.1001 / 0.1101 = 0.1011.

Таким образом, правило деления с восстановлением остатков формулируется следующим образом. Делитель вычитается из делимого и определяется знак нулевого (по порядку) остатка. Если остаток положительный, т. е. $|A| > |B|$, то в псевдознаковом разряде частного проставляется 1, при появлении которой формируется признак переполнения разрядной сетки и операция прекращается. Если остаток отрицательный, то в псевдознаковом разряде частного записывается 0, а затем производится восстановление делимого путем добавления к остатку делителя. Далее выполняется сдвиг восстановленного делимого на один разряд влево и повторное вычитание делителя. Знак получаемого таким образом остатка определяет первую значащую цифру частного: если остаток положителен, то в первом разряде частного записывается 1, если отрицательный, то записывается 0. Далее, если остаток положителен, то он сдвигается влево на 1 разряд и из него вычитается делитель для определения следующей цифры частного. Если остаток отрицателен, то к нему прибавляется делитель для восстановления предыдущего остатка, затем восстановленный остаток сдвигается на 1 разряд влево и от него вычитается делитель для определения следующей цифры частного и т. д. до получения необходимого количества цифр частного с учетом одного разряда для округления, т. е. до обеспечения требуемой точности деления.

Таким образом, цифры частного получаются как инверсное значение знаковых разрядов текущего остатка, которые принимают значение 00 или 11. Однако при сдвиге остатка влево в знаковых разрядах может возникнуть сочетание 01. В некоторых случаях для того чтобы цифры частного формировались как прямое значение знакового разряда текущего остатка, деление выполняют с инверсными знаками. При этом делимое передается в сумматор не прямым, а инверсным кодом, а на нулевом шаге выполняется операция «$+B$», вместо операции «$-B$».

4.9. ДЕЛЕНИЕ БЕЗ ВОССТАНОВЛЕНИЯ ОСТАТКОВ

Рассмотренный способ деления с восстановлением остатков является аритмичным процессом с переменным числом шагов того или иного вида в каждом конкретном случае (3 шага при $2R_i < B$ и 2 шага при $2R_i > B$). Для ритмизации процесса на каждую цифру частного необходимо затратить по 3 шага, в результате чего увеличивается время выполнения операции. Вместе с тем, операцию можно упростить и получить каждую цифру частного за 2 шага.

Рассмотрим случай, когда $R_i < 0$. В предыдущем способе в этом случае выполнялись следующие операции.

Таким образом, чтобы определить следующую цифру частного, необходимо сдвинуть текущий остаток влево на один разряд, а затем алгебраически прибавить к нему модуль делителя, которому приписывается знак, противоположный знаку текущего остатка. Знак полученного таким образом следующего остатка и определяет следующую цифру частного: если остаток положительный, то в частном записывается 1, если отрицательный — записывается 0. Операция сдвигов и алгебраических сложений повторяется до тех пор, пока в частном не получится требуемое количество цифр.

В настоящее время во всех ЭВМ деление производится по способу без восстановления остатков. Это, во-первых, упрощает схему управления процессом деления и, во-вторых, увеличивает быстродействие ЭВМ, так как длительность операции деления без восстановления остатков равна минимальной длительности операции деления с восстановлением остатков.

При выполнении операции деления результат получится одинаковым, если сдвигать остатки от деления влево либо делитель вправо. Следовательно, возможны две схемы выполнения деления:

1) деление без восстановления остатков со сдвигом делителя вправо;

2) деление без восстановления остатков со сдвигом влево.

Для реализации первого варианта необходимы $2n$-разрядный регистр делителя со сдвигом вправо, $2n$-разрядный СМ, $(n + 1)$-разрядный регистр частного со сдвигом влево (если округление результата выполняется в сумматоре, то частное пересылается в него по окончании операции) и схема управления (рис. 4.12).

Передача в сумматор делителя или дополнения модуля делителя обеспечивается блоком управления делением (БУД), который анализирует знаковые цифры остатков, снимаемые с сумматора с помощью блока съема знаков остатков (БСЗО). Эти знаковые цифры остатков, проходя через блок инверсии цифр (БИЦ), инвертируются и подаются в младший разряд сдвигающего регистра частного уже как цифры частного.

Для реализации второго варианта (рис. 4.13) необходимы: n-разрядный регистр делителя; $(n+1)$-разрядный регистр частного со сдвигом влево; n- или $(n+1)$-разрядный сумматор со сдвигом влево и схема управления. Сумматор производит сдвиг текущего остатка влево и алгебраическое сложение его с делителем. Передача в СМ модуля делителя или его дополнения обеспечивается БУД, который анализирует сдвигаемые из СМ знаковые цифры остатков. Эти же цифры инвертируются в БИЦ и подаются в младший разряд Рг частного как цифры частного.

Анализ обеих схем показывает, что второй вариант примерно на 40 % экономичнее по оборудованию по сравнению с первым.

Выбор типа длительного устройства при проектировании машины обычно не является самостоятельной задачей. Поэтому на практике вначале по заданным техническим условиям выбирается схема множительного устройства вследствие того, что умножение является примерно в 10 раз более частой операцией. После этого выбирается наиболее совместимая с устройством умножения схема делительного блока. Однако при проектировании специализированных ЭВМ может быть принят другой порядок выбора структур отдельных устройств.

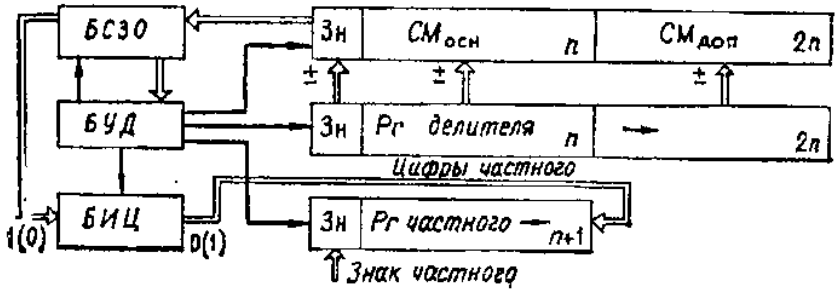


Рис. 4.12

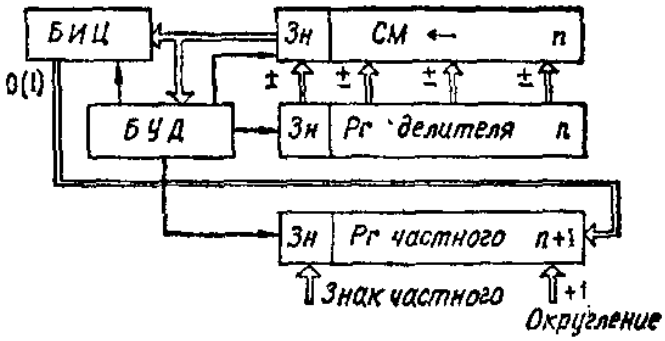


Рис. 4.13

Если сравнивать приведенные схемы деления со схемами множительных устройств, то оказывается, что схема первого варианта деления во многом совпадает с четвертой схемой умножения. Второй вариант схемы деления хорошо совместим с третьей схемой умножения.

4.11. ДЕЛЕНИЕ ЧИСЕЛ В ДОПОЛНИТЕЛЬНОМ КОДЕ

Так же как и при умножении, здесь возможны два пути:

1) перевод операндов в прямой код, получение частного обычным способом и перевод его в дополнительный код перед записью в память;

2) деление операндов в дополнительном коде с получением частного в требуемом виде.

Второй путь так же прост, как и первый. Необходимо только управлять логикой образования цифр частного, чему способствуют два обстоятельства:

1) делимое в процессе операции участвует всего лишь один раз, а дальше текущие остатки получаются автоматически;

2) нетрудно организовать получение цифр дополнительного кода отрицательного частного непосредственно в процессе деления путем передачи знаковых цифр остатков из СМ в регистр частного напрямую, минуя БИЦ.

Рассмотрим четыре возможных случая, определяемые комбинациями знаков дополнительных кодов операндов.

2.5. Арифметика с положительными двоично-десятичными числами

В ЭВМ часто предусматривается обработка чисел не только в двоичной системе счисления, но в двоично-десятичной. При этом, как правило, стремятся реализовать двоично-десятичную арифметику по правилам двоичной с введением ограниченного количества коррекций.

Сложение двоично-десятичных чисел

Рассмотрим на конкретном примере реализацию этой операции.

Пример

Найти сумму двух десятичных чисел с использованием двоично-десятичной системы счисления:

$A = D + C$, где $D = 3927$; $C = 4856$.

Решение

Составляем двоично-десятичную запись для чисел D и C:

D =3927 = 0011 1001 0010 0111:

C = 4856 =0100 1000 0101 0110.

Найти значение A можно, реализовав следующую последовательность операций из двоичного сложения и операции коррекции:

* **

0011 1001 0010 0111 – D

\+ 0100 1000 0101 0110 – *C*

1000 0001 0111 1101 – двоичная сумма

\+ 0110 0110 – коррекция

1000 0111 1000 0011 – двоично-десятичная сумма

Для получения двоично-десятичной суммы *A* на основании результата сложения операндов по правилам двоичной арифметики необходимо добавить шестерку (0110) в те тетрады, из которых был перенос. В данном примере это вторая тетрада (отмечена *). Необходимость такой коррекции обусловливается тем, что перенос, сформированный по правилам двоичного суммирования, унес из тетрады шестнадцать, а для десятичного сложения перенос должен был унести десять, т.е. перенос, сформированный по правилам двоичной арифметики, унес лишнюю шестерку.

Кроме этого шестерка добавляется в те тетрады, в которых получено значение, большее девяти. Такая коррекция обуславливается тем, что по правилам десятичной арифметики в таких тетрадах должен быть выработан перенос и, чтобы его выработать по правилам двоичной арифметики, в тетраду нужно добавить шестерку. Для рассмотренного примера такой тетрадой является и четвертая тетрада (отмечена **).

Пример

Найти разность двух десятичных чисел с использованием двоично-десятичной системе счисления:

A = *C* - *D*, где *D* = 3927, *C* = 4856.

Решение

Составляем двоично-десятичную запись для чисел *D* и *C*:

D = 3927 = 0011 1001 0010 0111:

C = 4856 = 0100 1000 0101 0110.

Найти значение *B* можно, реализовав следующую последовательность операций из двоичного сложения и операции коррекции:

* *

0100 1000 0101 0110 – *C*

\- 0011 1001 0010 0111 – *D*

0000 1111 0010 1111 – двоичная сумма

\- 0110 0110 – коррекция

0000 1001 0010 1001 – двоично-десятичная сумма

Для получения двоично-десятичной разности «*A*» на основании результата вычитания операндов по правилам двоичной арифметики необходимо вычесть шестерку (0110) из тетрад, в которые пришел заем. Это обусловливается тем, что заем, сформированный по правилам двоичного вычитания, приносит в тетраду шестнадцать, а для десятичного сложения заем должен был принести в тетраду десять, т.е. заём, сформированный по правилам двоичной арифметики, принес лишнюю шестерку. Для рассмотренного примера тетрадами, в которые пришел заем и в которых необходимо выполнить коррекцию (вычесть шестерку), являются вторая и четвертая тетрады (отмечены *).

Лекция 3. Арифметика с алгебраическими числами

3.1. Кодирование алгебраических чисел

Для представления чисел со знаком используются специальные коды:

– *прямой код*;

– *дополнительный код*;

– *обратный код*.

Во всех трёх случаях используется следующий формат представления числа, содержащий два поля – поле знака и поле модуля (Рис 1.1).

Поле знака	Поле модуля

Рис.1.1. Общий формат представления числа в ЭВМ

Поле знака представлено одним разрядом, в котором устанавливается 0, если число положительное, и 1, если число отрицательное.

Поле модуля отражает количественную оценку числа и для каждого кода формируется по–разному. Количество разрядов поля модуля определяется диапазоном изменения отображаемых чисел или точностью их представления.

В *прямом коде* запись целого числа A формируется по следующему правилу:

$$[A]_{\text{пк}} = \begin{cases} 0.A, \text{если } A \geq 0; \\ 1.|A|, \text{если } A < 0. \end{cases}$$

В *дополнительном коде* запись целого числа A формируется по следующему правилу:

$$[A]_{\text{дк}} = \begin{cases} 0.A, \text{если } A \geq 0; \\ 1.q^n + A, \text{если } A < 0, \end{cases}$$

где n – разрядность модульного поля;

q – основание системы счисления;

q^n – максимальная невключенная граница диапазона изменения представляемых чисел, т. к. диапазон изменение чисел A определяется как $q^n > |A| \geq 0$.

Для случая правильной дроби запись числа A в дополнительном коде имеет вид:

$$[A]_{\text{дк}} = \begin{cases} 0.A, \text{если } A \geq 0; \\ 1.(1+A), \text{если } A < 0, \end{cases}$$

где 1 – максимальная невключенная граница диапазона изменения представляемых чисел, т. е. диапазон изменение чисел A определяется как $1 > |A| \geq 0$.

В *обратном коде* запись целого числа A формируется по следующему правилу:

$$[A]_{\text{ок}} = \begin{cases} 0.A, \text{если } A \geq 0; \\ 1.((q^n - 1) + A), \text{если } A < 0, \end{cases}$$

где n – разрядность модульного поля;

q – основание системы счисления;

$(2^n - 1)$ – максимальная включенная граница диапазона изменения представляемых чисел, т. е. диапазон изменения чисел A определяется как $(q^n - 1) \geq |A| \geq 0$.

Для случая правильной дроби запись числа A в обратном коде имеет вид

$$[A]_{\text{ок}} = \begin{cases} 0.A, \text{если } A \geq 0; \\ 1.((1 - q^{-n}) + A), \text{если } A < 0, \end{cases}$$

где n – разрядность поля модуля;

$1 - q^{-n}$ – верхняя включенная граница представляемых чисел.

Т. о. диапазон изменения чисел A определяется с.о.:

$$(1 - q^{-n}) \geq |A| \geq 0.$$

Легко показать, что перевод отрицательного числа из обратного или дополнительного кода в прямой выполняется по тем же правилам, что и перевод числа из прямого кода в обратный или *дополнительный*:

- для перевода отрицательного числа из обратного в *прямой код* необходимо дополнить его модуль до включенной границы;

- для перевода отрицательного числа из обратного в *прямой код* необходимо дополнить его модуль до невключенной границы.

3.2. Дополнительный и обратные коды двоичных чисел

При переводе двоичных чисел в качестве включенной и не включенной границы диапазона изменения абсолютных значений представляемых чисел используется соответственно 2^n и $2^n - 1$.

Представление двоичных чисел в прямом и обратном кодах поясняется следующими примерами.

Пример

Найди запись чисел $A = 532$ и $B = -150$ в прямом, дополнительном и обратном двоичных кодах.

Решение

Найдем запись заданных чисел в двоичной системе:

$A = 532_{10} = 1000010100_2$, $B = -150_{10} = -10010110_2$.

Если считать, что представляются в заданных кодах только A и B, то разрядность n модульного поля должна соответствовать разрядности двоичной записи большего числа, т.е. $n = 10$.

Найдем запись заданных чисел в прямом коде:

$[A]_{пк} = 0.1000010100$, $[B]_{пк} = 1.0010010110$.

Найдем запись заданных чисел в дополнительном коде:

$[A]_{дк} = 0.1000010100$,

$[B]_{дк}$: для определения модульной части прибавим к невключенной границе диапазона ($2^n = 10000000000$) число B:

10000000000 +(- 10010110) = 1101101010.

Тогда $[B]_{дк} = 1.1101101010$.

Найдем запись заданных чисел в обратном коде:

$[A]_{ок} = 0.1000010100$,

$[B]_{ок}$: для определения модульной части прибавим к включенной границе диапазона ($2^n - 1 = 1111111111$) число B:

1111111111 + (- 10010110) = 1101101001,
и тогда $[B]_{ок}$ = 1.1101101001.

Анализируя запись модульной части отрицательного числа *C* в обратном коде, можно заключить, что она представляет собой инверсию модульной части записи этого числа в прямом коде, т.е. 0 заменяется на 1, а 1 заменяется на 0. Отсюда вытекает правило формирования модуля обратного кода отрицательного двоичного числа:

для формирования модульной части записи отрицательного числа в обратном коде достаточно в модульной части записи этого числа в прямом коде взять обратные значения всех двоичных разрядов, т.е. необходимо проинвертировать модуль прямого кода.

Переход от обратного кода отрицательного числа к представлению в прямом коде осуществляется по тому же правилу, т.е. необходимо проинвертировать модуль записи числа в дополнительном коде.

Если сравнить запись модульных частей дополнительного и обратного кодов отрицательного числа *B*, то можно заметить, что они отличаются на значение, соответствующее единицы младшего разряда. Отсюда вытекает правило формирования модуля дополнительного кода отрицательного числа:

для формирования модульной части записи отрицательного числа в дополнительном коде достаточно в модульной части записи этого числа в прямом коде взять обратные значения всех двоичных разрядов, т.е. необходимо проинвертировать модуль прямого кода и к полученному коду прибавить 1 в младший разряд.

Переход от дополнительного кода отрицательного числа к прямому осуществляется по тому же правилу, т.е. необходимо проинвертировать модуль записи числа в дополнительном коде, и к полученному коду прибавить 1 в младший разряд.

При выполнении операций над числами со знаком в ЭВМ используются прямой, обратный и дополнительный коды. Как правило, информация в памяти хранится в прямом коде, а при выполнении операций применяется или обратный, или дополнительный код.

3.3. Операции с двоичными числами в дополнительном коде

При использовании *дополнительного* или о*братного* кода операция вычитания заменяется операцией сложения с изменением знака второго операнда. При сложении чисел, представленных в дополнительном коде, выполняется сложение разрядов, представляющих за-

пись операндов, по правилам двоичной арифметики по всей длине записи чисел, не обращая внимание на границу, разделяющую знаковое и модульные поля. Переполнение знакового поля, т.е. перенос, возникший из крайнего левого разряда, игнорируется. В результате такого сложения будет получен *дополнительный код* суммы заданных операндов.

Пример

Найти значения для $C1$, $C2$, $C3$, $C4$, определяемых выражениями:

$C1 = A + B$, $C2 = A - B$, $C3 = B - A$, $C4 = - A - B$,

если $A = 57_{10}$, $B = - 210_{10}$. При выполнении операций использовать двоичный *дополнительный* код. Результат представить в *прямом* коде.

Решение

Преобразуем заданные числа в двоичную систему счисления:

$A = 57_{10} = 111001_2$, $B = - 210_{10} = - 11010010_2$.

Определим количество разрядов для модульной части записи чисел, учитывая не только значения используемых операндов, но и ожидаемые результаты выполнения заданных операций. Исходя из абсолютного значения операндов, разрядность представления модульной части n должна быть равна 8. Учитывая то, что в подлежащих реализации выражениях над числами выполняется только одна операция (или сложения, или вычитания), при этом возможно переполнение (возникновение переноса из старшего разряда), длину модульной части необходимо взять на один разряд больше, т.е. $n = 9$.

Избавляясь от операции вычитания, приводим заданные выражения к виду

$C1 = A + B$, $C2 = A + (- B)$, $C3 = B + (- A)$, $C4 = (- A) + (- B)$.

Таким образом, в подлежащих реализации выражениях в качестве операндов присутствуют следующие величины: A , $- A$, B, $- B$. Представим их в прямом и дополнительном коде

$[A]_{пк} = 0.000111001$, $[A]_{дк} = 0.000111001$,

$[- A]_{пк} = 1.000111001$, $[- A]_{дк} = 1.111000111$,

$[B]_{пк} = 1.011010010$, $[B]_{дк} = 1.100101110$,

$[- B]_{пк} = 0.011010010$, $[- B]_{дк} = 0.011010010$.

Используя сформированный дополнительный код, реализуем выражения для $C1$,$C2$, $C3$, $C4$.

$C1$: $0.0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 - [A]_{дк}$

$+\ \underline{1.1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0} - [B]_{дк}$

$1.1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1 - [C1]_{дк}$

$$1.0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1 - [C1]_{пк}$$

$C2$: $0.0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 - [A]_{дк}$

$+\ \underline{0.0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0} - [-B]_{дк}$

$0.1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1 - C2]_{дк} = [C2]_{пк}$

$C3$: $1.1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1 - [-A]_{дк}$

$+\ \underline{1.1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0} - [B]_{дк}$

$\underline{1}\ 1.0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1 - [C3]_{пк}$

$1.1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1 - [C3]_{пк.}$.

При выполнении сложения в данном случае возникла единичка переполнения знакового поля. При работе с дополнительным кодом она игнорируется (в примере она перечеркнута).

$C4$: $1.1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1 - [-A]_{дк}$

$+\ \underline{0.\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0} - [-B]_{дк}$

$\underline{1}\ 0.\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1 - [C4]_{дк} = [C4]_{пк}$

В данном случае также возникло переполнение знакового поля, которое игнорируется.

3.4. Операции с двоичными числами в обратном коде

При сложении чисел, представленных в *обратном* коде, выполняется сложение разрядов, представляющих запись операндов, по правилам двоичной арифметики по всей длине записи чисел, не обращая внимания на границу, разделяющую знаковое и модульные поля. Переполнение знакового поля, т.е. перенос, возникший из крайнего левого разряда, должен быть учтен как +1 в младший разряд полученной суммы. В результате такого сложения будет получен *обратный* код суммы заданных операндов.

Пример

Найти значения для $C1$, $C2$, $C3$, $C4$, определяемых выражениями

$C1 = A + B$, $C2 = A - B$, $C3 = B - A$, $C4 = - A - B$,

если $A=57_{10}$, $B = - 210_{10}$. При выполнении операций использовать двоичный обратный код. Результат представить в прямом коде.

Решение

В данном примере используются те же выражения и те же операнды, что и в предыдущем примере, поэтому при его решение используются уже найденные ранее двоичные представления операндов и их прямые коды.

Обратные коды операндов имеют вид

$[A]_{ок}$ = 0.000111001, $[-A]_{ок}$ = 1.111000110,
$[B]_{ок}$ = 1.100101101, $[-B]_{ок}$ = 0.011010010.

Используя сформированный дополнительный код, реализуем выражения для *C*1, *C*2, *C*3, *C*4.

*C*1: 0.0 0 0 1 1 1 0 0 1 – $[A]_{ок}$

\+ 1.1 0 0 1 0 1 1 0 1 – $[B]_{ок}$

1.1 0 1 1 0 0 1 1 0 – $[C1]_{ок}$

1.0 1 0 0 1 1 0 0 1 – $[C1]_{пк}$

*C*2: 0.0 0 0 1 1 1 0 0 1 – $[A]_{ок}$

\+ 0.0 1 1 0 1 0 0 1 0 – $[-B]_{ок}$

0.1 0 0 0 0 1 0 1 1 – $[C2]_{ок} = [C2]_{пк}$

*C*3: 1.1 1 1 0 0 0 1 1 0 – $[-A]_{ок}$

\+ 1.1 0 0 1 0 1 1 0 1 – $[B]_{ок}$

1 1.0 1 1 1 1 0 0 1 1

\+ 1

1.0 1 1 1 1 0 1 0 0 – $[C3]_{ок}$.

1.1 0 0 0 0 1 0 1 1 – $[C3]_{пк}$

C4: 1.1 1 1 0 0 0 1 1 0 – $[-A]_{ок}$

\+ 0.0 1 1 0 1 0 0 1 0 – $[-B]_{ок}$

1 0.0 1 0 0 1 1 0 0 0

\+ 1

0. 0 1 0 0 1 1 0 0 1 – $[C4]_{ок} = [C4]_{пк}$

В данном случае также возникло переполнение знакового разряда, которое должно быть учтено как +1 в младший разряд сформированной суммы.

1. Операция сложения в обратном и дополнительном кодах

Сложение и вычитание чисел в обратном и дополнительном кодах выполняется с использованием обычного правила арифметического сложения многоразрядных чисел. Общей для этих ко-

дов особенностью (и очень удобной особенностью) является лишь то, что при поразрядном сложении чисел разряды, изображающие знаки чисел рассматриваются как равноправные разряды двоичного числа, которые складываются друг с другом и с единицей переноса из предыдущего разряда числа по обычным правилам арифметики. Различия же обратного и дополнительного кодов связаны с тем, что делается с единицей переноса из старшего разряда (изображающего, как неоднократно говорилось, знак числа).

При сложении чисел в дополнительном коде единица переноса из старшего разряда игнорируется (теряется), а в обратном коде эту единицу надо прибавить к младшему разряду результата.

Пример: Сложить числа +12 и -5.

а) В обратном коде

Десятичная форма	Двоичная форма	Прямой код	Обратный код
+12	+1100	00001100	00001100
-5	-101	10000101	11111010

Выполним сложение:

	+	0	0	0	0	1	1	0	0
		1	1	1	1	1	0	1	0
+	1	0	0	0	0	0	1	1	0
									1
	0	0	0	0	0	0	1	1	1

Результат в обратном коде – 00000111. Поскольку знаковый разряд равен 0, результат положительный, и, следовательно, запись кода числа совпадает с записью прямого кода. Теперь можно восстановить алгебраическую запись результата. Он равен +111 (незначащие нули отброшены), или в десятичной форме +7. Проверка (+12-5=+7) показывает, что результат верный.

б) В дополнительном коде

Десятичная форма	Двоичная форма	Прямой код	Обратный код	Дополнительный код
+12	+1100	00001100	00001100	00001100
-5	-101	10000101	11111010	11111011

Выполним сложение в дополнительном коде:

	+	0	0	0	0	1	1	0	0
		1	1	1	1	1	0	1	1
	1	0	0	0	0	0	1	1	1
	0	0	0	0	0	0	1	1	1

Результат в дополнительном коде – 00000111. Поскольку знаковый разряд равен 0, результат положительный. Теперь можно восстановить алгебраическую запись результата. Он равен +111 (незначащие нули отброшены), или в десятичной форме +7. Проверка (+12-5=+7) показывает, что результат верный.

Умножение и деление двоичных чисел производится в ЭВМ в прямом коде, а знаки их используются лишь для определения знака результата.

1.1 Двоичное число: прямой, обратный и дополнительный коды

Прямой код двоичного числа
Обратный код двоичного числа
Дополнительный код двоичного числа

Прямой, обратный и дополнительный коды двоичного числа *— способы представления двоичных чисел с фиксированной запятой в компьютерной (микроконтроллерной) арифметике, предназначенные для записи отрицательных и неотрицательных чисел*

Мы знаем, что десятичное число можно представить в двоичном виде. К примеру, десятичное число 100 в двоичном виде будет равно 1100100, или в восьмибитном представлении

0110 0100. А как представить отрицательное десятичное число в двоичном виде и произвести с ним арифметические операции? Для этого и предназначены разные способы представления чисел в двоичном коде.

*Сразу отмечу, что положительные числа в двоичном коде вне зависимости от способа представления (**прямой, обратный или дополнительный коды**) имеют одинаковый вид.*

1.2 Прямой код

***Прямой код** — способ представления двоичных чисел с фиксированной запятой. Главным образом используется для записи неотрицательных чисел*

Прямой код используется в двух вариантах.

В первом (основной) — для записи только неотрицательных чисел:

Десятичное число	Двоичное число в прямом коде (в 8-битном представлении)
0	0000 0000
10	0000 1010
100	0110 0100
255	1111 1111

В этом варианте (для восьмибитного двоичного числа) мы можем записать максимальное число 255 (всего чисел 256 — от 0 до 255)

Второй вариант — для записи как положительных, так и отрицательных чисел.

В этом случае старший бит (в нашем случае — восьмой) объявляется знаковым разрядом (знаковым битом). При этом, если:

— знаковый разряд равен 0, то число положительное

— знаковый разряд равен 1, то число отрицательное

— знаковый разряд равен 1, то число отрицательное

Разряды восьмибитного двоичного числа							
7	6	5	4	3	2	1	0
Знаковый разряд	Значение числа						

В этом случае диапазон десятичных чисел, которые можно записать в прямом коде составляет от — 127 до +127:

Десятичное число	Двоичное число в прямом коде (в 8-битном представлении)
127	0111 1111
100	0110 0100
0	0000 0000
- 0	1000 0000
- 100	1110 0100
- 127	1111 1111

Подводя итоги вопроса, не влезая в его дебри, скажу одно:
Прямой код используется главным образом для представления неотрицательных чисел.
Использование прямого кода для представления отрицательных чисел является неэффективным — очень сложно реализовать арифметические операции и, кроме того, в прямом коде два представления нуля — положительный ноль и отрицательный ноль (чего не бывает):

1.3 Обратный код

Обратный код *— метод вычислительной математики, позволяющий вычесть одно число из другого, используя только операцию сложения.*
Обратный двоичный код положительного числа *состоит из одноразрядного кода знака (битового знака) — двоичной цифры 0, за которым следует значение числа.*
Обратный двоичный код отрицательного числа *состоит из одноразрядного кода знака (битового знака) — двоичной цифры 1, за которым следует инвертированное значение положительного числа.*
Для неотрицательных чисел обратный код двоичного числа имеет тот же вид, что и запись неотрицательного числа в пря-

мом коде.
Для отрицательных чисел обратный код получается из неотрицательного числа в прямом коде, путем инвертирования всех битов (1 меняем на 0, а 0 меняем на 1).
Для преобразования отрицательного числа записанное в обратном коде в положительное достаточного его проинвертировать.
При 8-битном двоичном числе — знаковый бит (как и в прямом коде) старший (8-й)

Положительное десятичное число	Двоичное число в обратном коде	Отрицательное десятичное число	Двоичное число в обратном коде
0	0000 0000	- 0	1111 1111
10	0000 1010	- 10	1111 0101
100	0110 0100	- 100	1001 1011
127	0111 1111	- 127	1000 0000

Диапазон десятичных чисел, который можно записать в обратном коде от -127 до + 127

Арифметические операции с отрицательными числами в обратном коде:
(Арифметические операции с двоичными числами)
1-й пример (для положительного результата)
Дано два числа:
100 = 0110 0100
-25 = — 0001 1001
Необходимо их сложить:
100 + (-25) = 100 — 25 = 75
1-й этап
Переводим число -25 в двоичное число в обратном коде:
25 = 0001 1001
-25= 1110 0110
и складываем два числа:
0110 0100 (100) + 1110 0110 (-25) = 1 0100 1010, отбрасываем старшую 1 (у нас получился лишний 9-й разряд — переполнение), = 0100 1010
2-й этап
Отброшенную в результате старшую единицу прибавляем к результату:

0100 1010 + 1 = 0100 1011 (знаковый бит =0, значит число положительное), что равно 75 в десятичной системе

2-й пример (для отрицательного результата)

Дано два числа:

5 = 0000 0101

-10 = — 0000 1010

Необходимо их сложить:

5 + (-10) = 5 — 10 = -5

1-й этап

Переводим число -10 в двоичное число в обратном коде:

10 = 0000 1010

-10= 1111 0101

и складываем два числа:

0000 0101 (5) + 1111 0101 (-10) = 1111 1010 (знаковый бит =1, значит число отрицательное)

2-й этап

Раз результат получился отрицательный, значит число представлено в обратном коде.

Переводим результат в прямой код (путем инвертирования значения, знаковый бит не трогаем):

1111 1010 —-> 1000 0101

Проверяем:

1000 0101 = — 0000 0101 = -5

Обратный код *решает проблему сложения и вычитания чисел с различными знаками, но и имеет свои недостатки:*

— арифметические операции проводятся в два этапа

— как и в прямом коде два представления нуля — положительный и отрицательный

1.4 Дополнительный код

Дополнительный код *— наиболее распространенный способ представления отрицательных чисел. Он позволяет заменить операцию вычитания на операцию сложения и сделать операции сложения и вычитания одинаковыми для знаковых и беззнаковых чисел.*

В дополнительном коде (как и в прямом и обратном) старший разряд отводится для представления знака числа (знаковый бит).

Диапазон десятичных чисел которые можно записать в дополнительном коде от -128 до +127. Запись положительных двоичных чисел в дополнительном коде та-же, что и в прямом и обратном кодах.

Десятичное представление	Двоичное представление (8 бит)		
	прямой	обратный	дополнительный
127	01111111	01111111	01111111
1	00000001	00000001	00000001
0	00000000	00000000	00000000
-0	10000000	11111111	---
-1	10000001	11111110	11111111
-2	10000010	11111101	11111110
-3	10000011	11111100	11111101
-4	10000100	11111011	11111100
-5	10000101	11111010	11111011
-6	10000110	11111001	11111010
-7	10000111	11111000	11111001
-8	10001000	11110111	11111000
-9	10001001	11110110	11110111
-10	10001010	11110101	11110110
-11	10001011	11110100	11110101
-127	11111111	10000000	10000001
-128	---	---	10000000

Дополнительный код отрицательного числа можно получить двумя способами
1-й способ:
— инвертируем значение отрицательного числа, записанного в прямом коде (знаковый бит не трогаем)
— к полученной инверсии прибавляем 1
Пример:
Дано десятичное число -10
Переводим в прямой код:
10 = 0000 1010 —-> -10 = 1000 1010

Инвертируем значение (получаем обратный код):
1000 1010 —-> 1111 0101
К полученной инверсии прибавляем 1:
1111 0101 + 1 = 1111 0110 — десятичное число -10 в дополнительном коде
2-й способ:
Вычитание числа из нуля
Дано десятичное число 10, необходимо получить отрицательное число (-10) в дополнительном двоичном коде
Переводим 10 в двоичное число:
10 = 0000 1010
Вычитаем из нуля:
0 — 0000 1010 = 1111 0110 — десятичное число -10 в дополнительном коде

Получение дополнительного кода отрицательного числа

Десятичное число	Двоичное число в прямом коде	Инвертирование значения (обратный код)	Прибавляем к инверсии единицу	Двоичное число в дополнительном коде
127	0111 1111	Положительные значения не меняются		0111 1111
10	0000 1010			0000 1010
0	0000 0000			0000 0000
- 0	1000 0000	-----	-----	-----
- 5	1000 0101	1111 1010	+1	1111 1011
- 10	1000 1010	1111 0101	+1	1111 0110
- 127	1111 1111	1000 0000	+1	1000 0001
- 128	-----			1000 0000

Арифметические операции с отрицательными числами в дополнительном коде
Дано: необходимо сложить два числа -10 и 5
-10 + 5 = -5
Решение:
5 = 0000 0101
-10 = 1111 0110 (в дополнительном коде)
Складываем:
1111 0110 + 0000 0101 = 1111 1011, что соответствует числу -5 в дополнительном коде

Как мы видим на этом примере — дополнительный код отрицательного двоичного числа наиболее подходит для выполнения арифметических операций сложения и вычитания отрицательных чисел.

Вывод:
1. Для арифметических операций сложения и вычитания положительных двоичных чисел наиболее подходит применение прямого кода
2. Для арифметических операций сложения и вычитания отрицательных двоичных чисел наиболее подходит применение дополнительного кода

3.5. Модифицированные коды

При расчете разрядности n модульного поля весьма трудно бывает учесть диапазон значений результатов, особенно когда последовательность операции, представленных в подлежащих реализации выражениях, достаточно сложны.

При несоответствии выбранной разрядности *n* диапазону изменения представляемых чисел при выполнении операции сложении чисел с одинаковыми знаками возможно появление ситуации переполнения, когда подлежащий представлению результат выходит за диапазон представления, определенный некорректно выбранной разрядностью n поля модуля.

Например, в случае сложения двух чисел, представленных в обратном коде:

$[D1]_{ок} = 1.00110$ и $[D2]_{ок} = 1.00110$.

Сумма этих чисел $F1 = D1 + D2$ будет подсчитана следующим образом:

$F1$:

$$\begin{array}{r} 1.\,0\,0\,1\,1\,0 \\ \underline{+1.\,0\,0\,1\,1\,0} \\ 10.\,0\,1\,1\,0\,0 \end{array}$$

$$\underline{+ \qquad 1}$$

$$0.0\,1\,1\,0\,1$$

Пример, выполненный по всем формальным правилам, дал абсурдный результат, так как получена положительная сумма двух отрицательных операндов. Аналогичная ситуация может возникать и при использовании дополнительного кода.

Ситуацию переполнения можно обнаруживать по факту появления «абсурдного» результата, но для этого необходимо помнить то, что в суммировании принимают участие операнды с одинаковыми знаками и знак полученного при этом результата отличен от знака операндов.

Более просто ситуация переполнения определяется при применении *модифицированного* кода (обратного или дополнительного). Модифицированные коды отличаются от базовых кодов только тем, что поле знака операндов имеет два разряда, и эти разряды имеют одинаковые значения:

00 – для положительных чисел;

11 – для отрицательных чисел.

Если в результате сложения чисел в модифицированном коде полученный результат имеет в поле знака одинаковые значения в обоих разрядах (00 или 11), то переполнения нет, если же разряды знакового поля имеют не одинаковые значения (10 или 01), то имеет место переполнение. При этом, если в поле знака имеет место значение 01 – результат положительный, а если 10, то полученный результат отрицательный (основным носителем знака числа является левый разряд знакового поля).

Пример

Найти значения выражений

$C1 = A + B$, $C2 = A - B$, $C3 = B -$ A, $C4 = - A - B$,

используя *модифицированный* обратный код, если

$[A]_{пк} = 0.1010011$,

$[B]_{пк} = 1.0111001$.

Решение

Модифицированный обратный код для всех операндов, используемых в приведенных выражениях, имеет вид

$[A]_{мпк}$ = 00.1010011, $[A]_{мок}$ = 00.1010011, $[-A]_{мок}$ = 11.0101100,

$[B]_{мпк}$ = 11.0111001, $[B]_{мок}$ =11.1000110, $[-B]_{мок}$ = 00.0111001.

Выполним действия, указанные в приведенных выражениях:

$C1$:

00.1010011 – $[A]_{мок}$

+11.1000110 – $[B]_{пок}$

100.0011001

\+ 1

00.0011010 – $[C1]_{мок} = [C1]_{мпк}$

$C2$:

00.1010011 – $[A]_{мок}$

+00.0111001 – $[-B]_{пок}$

01.0001100

Результат положительный и имеет место переполнение.

$C3$:

11.0101100 – $[-A]_{мок}$

+11.1000110 – $[B]_{пок}$

110.1110010

\+ 1

10.1110011

Результат отрицательный и имеет место переполнение.

$C4$:

11.0101100 – $[-A]_{мок}$

+00.0111001 – $[-B]_{пок}$

11.1100101 – $[C4]_{мок}$

11.0011010 – $[C1]_{мпк}$

При формировании *С*1 был получен положительный результат (без переполнения).

При формировании *С*4 был получен отрицательный результат (без переполнения).

Факт переполнения при формировании *С*3 и *С*2 устанавливается по наличию в разрядах знакового поля различных значений.

Лекция 4. Логические операции с двоичными кодами

Над двоичными кодами могут выполняться различные логические операции, среди которых особое место занимают:

- *логическое суммирование* (обозначения – ИЛИ, OR, «Ú»);
- *логическое умножение* (обозначения – И, AND, «Ù»);
- *отрицание* (обозначения – НЕТ, NOT, «$\bar{x}$», т.е. штрих над отрицыемым *x*);
- *суммирование по модулю* 2 (обозначается mod 2, « Å »);
- *операции сдвига*.

4.1. Логические операции

Операция *логического суммирования* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если хотя бы в одном операнде в *i*-м разряде имеет место единица.

Пример:

10001101 Ú 11110000 = 11111101.

Операция *логического умножения* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если оба операнда в этом *i*-м разряде имеются единицу, и ноль во всех других случаях.

Пример:

10001101 Ù 11110000 = 10000000

Операция *суммирования по модулю 2* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если два задан-

ных операнда в *i*-м разряде имеют противоположные значения. Иногда эта операция называется «исключающее ИЛИ».

Пример

10001101 Å 11110000 = 01111101.

Операция логического отрицания выполняется над одним кодом и генерирует результирующий код той же разрядности, что и операнд, в некотором *i*-м разряде которого находится значение, противоположное значению в *i*-м разряде отрицаемого кода.

Операции *сдвига* в свою очередь, подразделяются на:

- логические сдвиги, которые имеют разновидности – сдвиг вправо, сдвиг влево, циклический сдвиг вправо, циклический сдвиг влево;
- арифметические сдвиги вправо и влево, выполнение которых зависит от знака и кода сдвигаемого числа.

4.2. Логические сдвиги

Сдвиг *влево* выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем младшем разряде (освобождающийся самый правый т.е. самый младший, разряд заполняется 0, а «выталкиваемый» разряд пропадает). Например, код 11001110 после сдвига влево будет иметь вид 10011100.

Сдвиг *вправо* выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем старшем разряде (в освобождающийся самый левый, т.е. самый старший, разряд заполняется 0, «выталкиваемый» разряд пропадает). Например, код 11001110 после сдвига влево будет иметь вид 01100111.

Циклический сдвиг влево выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем младшем разряде (в освобождающийся самый правый, т.е. самый младший, разряд заносится значение старшего, т.е. самого левого разряда исходного кода). Например, код 11001110 после сдвига влево будет иметь вид 10011101.

Циклический сдвиг вправо выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем старшем разряде (в освобождающийся самый левый т.е. самый старший, разряд заполняется значение в самом младшем

разряде исходного кода). Например, код 11001110 после сдвига влево будет иметь вид 01100111.

4.3. Арифметические сдвиги

Арифметические сдвиги обеспечивают выполнение умножения (сдвиги влево) или операции деления (сдвиги вправо) двоичных кодов на два, точно так же, как сдвиги вправо и влево десятичного числа обеспечиваю выполнение деления и умножение на 10.

Арифметические сдвиги влево двоичного прямого кода выполняются в зависимости от того, какое сдвигается число – положительное или отрицательное.

Если *сдвигается положительное* число, то сдвиг (вправо или влево) выполняется как соответствующий логический сдвиг (влево или вправо), с той лишь разницей, что предусматриваются средства определения факта переполнения при сдвиге влево, что реализуется и при всех других арифметических операциях. При любом сдвиге вправо предусматриваются средства для округления после завершения нужного количества сдвигов и средства обнаружения обнуления сдвигаемой величины после очередного сдвига.

Арифметические сдвиги влево положительных двоичных чисел выполняются не зависимо от используемого кода (прямого обратного, дополнительного). Его реализация иллюстрируются следующими примерами.

Примеры

1. Найти результат арифметического сдвига влево на три разряда двоичного прямого кода числа $[A]_{пк}$= 00.00000101

Решение

Процесс выполнения заданного сдвига дает следующие промежуточные и конечное значения:

первый сдвиг: 00.00000101 **à** 00.00001010;
второй сдвиг: 00.00001010 **à** 00.00010100;
третий сдвиг: 00.00010100 **à** 00.00101000.

2. Найти результат арифметического сдвига влево на четыре разряда двоичного прямого кода числа $[A]_{пк}$= 00.00101.

Решение

Процесс заданного сдвига дает следующие промежуточные и конечное значения:

первый сдвиг: 00.00101 **à** 00.01010;
второй сдвиг: 00.01010 **à** 00.10100;
третий сдвиг: 00.10100 **à** 01.01000.

После третьего сдвига будет выработан сигнал переполнения, так как после очередного сдвига в разрядах знакового поля появятся разные значения. Таким образом, не считая процедуры определения переполнения, арифметический сдвиг влево выполняется точно так же, как и логический сдвиг влево.

3. Найти результат арифметического сдвига вправо на два разряда двоичного прямого кода числа $[A]_{пк}$ = 00.00000110.

Решение

Процесс заданного сдвига дает следующие промежуточные и конечное значение:

первый сдвиг: 00. 00000110 **à** 00. 00000011;

второй сдвиг: 00. 00000011 **à** 00. 00000001;

После выполнения заданного количества сдвигов выполняется округление на основании последнего «вытолкнутого» разряда; в данном случае последний «вытолкнутый» разряд равен 1, поэтому конечный результат выполнения заданного сдвига будет равен

00.00000010.

4. Найти результат арифметического сдвига вправо на четыре разряда двоичного прямого кода числа $[A]_{пк}$= 00.00000110.

Решение

Процесс заданного сдвига дает следующие промежуточные и конечное значения:

первый сдвиг: 00.00000110 **à** 00.00000011;

второй сдвиг: 00.00000011 **à** 00.00000001;

третий сдвиг: 00.00000001 **à** 00.00000000.

После выполнения третьего сдвига будет выработан сигнал о получении нулевого результата. Оставшиеся сдвиги могут не выполняться.

Арифметические сдвиги отрицательных двоичных чисел, представленных в прямом коде

Арифметические сдвиги влево и вправо реализуются по-разному в зависимости как от знака числа, так и от используемого кода (прямого обратного, дополнительного).

При арифметическом сдвиге отрицательного двоичного числа, представленного в прямом коде, осуществляется соответствующий сдвиг только модульного поля записи числа.

Реализация этого типа сдвига иллюстрируется следующими примерами.

Пример 1

Выполнить *арифметический сдвиг* влево двоичного числа A = 11. 001010 (соответствует 10_{10}), представленного в модифицированном прямом коде.

Решение

Заданный сдвиг, имеющий своей целью получение результата, в два раза превышающего по абсолютному значению значение исходного кода, дает в результате 11.010100 (20_{10}), которое получается за счет логического сдвига влево только модульной части исходного кода.

Факт получения переполнения устанавливается по наличию единичного значения старшего разряда в сдвигаемом коде перед очередным сдвигом.

Пример 2

Выполнить арифметический сдвиг вправо двоичного числа A = 11.01110 (14_{10}), представленного в модифицированном прямом коде.

Решение

Заданный сдвиг, имеющий своей целью получение кода, в два раза меньшего по абсолютному значению по отношению к значению исходного кода, дает в результате число 11.00111 (7_{10}), которое получается за счет логического сдвига вправо только модульной части исходного кода.

При арифметическом сдвиге влево отрицательного двоичного числа, представленного в обратном коде, осуществляется циклический сдвиг исходного кода с контролем за переполнением, например, сдвиг влево отрицательного двоичного числа 11.1100110 (25_{10}), представленного в обратном коде, дает в результате 11.1001101 (50_{10}).

При арифметическом сдвиге вправо отрицательного двоичного числа, представленного в обратном коде, осуществляется сдвиг только модульной части записи числа с установкой единицы в освобождающийся разряд. При этом может осуществляется контроль за обнулением результата сдвига (появление единичных значений во всех разрядах) и округление результата после выполнения заданного количества сдвигов.

Пример3

Выполнить сдвиг вправо на четыре разряда двоичного числа 11.1001101 (десятичный эквивалент – 50_{10}), представленного в обратном коде.

Первый сдвиг дает 11.11001101 (50_{10}) **à** 11.11100110 (25_{10}).

Второй сдвиг дает 11.11100110 (25_{10}) **à** 11.11110011 (12_{10}).

Третий сдвиг дает 11.11110011 (12_{10}) **à** 11.11111001 (6_{10}).

Четвертый сдвиг дает 11.11111001 (6_{10}) **à** 11.11111100 (3_{10}).

При выполнении сдвига вправо нечетного числа результат получается с точностью до младшего разряда кода, причем ошибка отрицательная.

После выполнения последнего, четвертого сдвига выполняется округление, при котором, если последний «вытолкнутый» разряд имел значение 0, к результату последнего сдвига прибавляется – 1. Данное округление можно выполнить за счет прибавления единицы к прямому коду, соответствующему результату последнего сдвига исходного обратного кода.

В рассмотренном примере корректировать на единицу результат четвертого сдвига не надо, так как «вытолкнутый» разряд при последнем (четвертом) сдвиге равен единице. В данном случае конечный результат сдвига заданного отрицательного числа, представленного в обратном коде, равен 11.11111100.

При арифметическом сдвиге влево отрицательного двоичного числа, представленного в дополнительном коде, осуществляется логический сдвиг влево модуля исходного кода (освобождающийся разряд заполняется нулем) с контролем за переполнением, например, сдвиг влево отрицательного двоичного числа 11.11001110 (50_{10}), представленного в дополнительном коде, дает в результате 11.10011100 (100_{10}).

При арифметическом сдвиге вправо отрицательного двоичного числа, представленного в дополнительном коде, осуществляется логический сдвиг вправо модуля записи числа с установкой единицы в освобождающийся разряд. При этом может осуществляется контроль за обнулением результата сдвига (появление единичных значений во всех разрядах).

Пример

Выполнить сдвиг вправо на четыре разряда двоичного числа 11.11001110 (десятичный эквивалент – 50_{10}), представленного в дополнительном коде.

Решение

Первый сдвиг дает 11.11001110 **à** 11.11100111 (25_{10}).
Второй сдвиг дает 11.11100111 **à** 11.11110011 (13_{10}).
Третий сдвиг дает 11.11110011 **à** 11.11111001 (7_{10}).
Четвертый сдвиг дает 11.11111001 **à** 11.11111100 (4_{10}).

При выполнении сдвига вправо нечетного целого числа результат получается с точностью до младшего разряда кода, причем ошибка положительная.

Арифметический сдвиг вправо может выполняться над отрицательными числами с переполнением (такие числа в модифицированном прямом, обратном или дополнительном коде имеют в знаковом поле 10). В этом случае после сдвига в знаковом поле будет 11, а в старшем разряде – 0, если число представлено в обратном или дополнительном коде, или 1, если число представлено в прямом коде.

Пример1

Выполнить сдвиг вправо на 2 разряда числа $[A]_{пк}$ = 10.01000110 (A_{10} = - 326).

Решение

1-й сдвиг: 10. 01000110 **à** 11.10100011 (- 163_{10});

2-й сдвиг: 11.10100011 **à** 11.11010001 (- 81_{10}) и последний вытолкнутый разряд равен 1).

С учетом округления имеем окончательный результат:

$[A2]_{пк}$=11.10010010.

Пример2

Выполнить сдвиг вправо на 2 разряда числа $[A]_{ок}$ = 10.10111001 (A_{10} = – 326).

Решение

1-й сдвиг: 10.10111001 **à** 11.01011100 (- 163_{10});

2-й сдвиг: 11.01011100 **à** 11.10101110 (- 82_{10}).

Пример3

Выполнить сдвиг вправо на 2 разряда число $[A]_{ок}$= 10. 10111010 (A_{10} – = - 326).

Решение

1-й сдвиг: 10.0111010 **à** 11.01011101 (- 163_{10});

2-й сдвиг: 11.01011101 **à** 11.10101110 (- 81_{10}) и последний вытолкнутый разряд равен 1).

С учетом округления имеем окончательный результат

$[A2]_{ок}$ = 11.10101101.

Лекция 5. Представление чисел в ЭВМ

5.1. Представление чисел с фиксированной точкой

Числовая информация представляется в машине в форме с фиксированной или с плавающей точкой. При представлении с фиксированной точкой положение последней в записи числа фиксировано.

Как правило, при использовании фиксированной точки числа представляются в виде целого числа или правильной дроби, форматы которых приведены на Рис. 1..

К заданному виду (целым числам или правильной дроби) исходные числа приводятся за счет введения масштабных коэффициентов.

Зн	1р	2р	3р	4р	....	(*n*-1)	*n*р	«.»

а

Зн	«.»	1р	2р	3р	4р	....	(*n*-1)	*n*р

б

Рис. 1.3. Представление чисел с фиксированной точкой: *а* – формат целого числа; *б* – формат дробного числа

Точка в записи числа не отображается, а так как она находится всегда в одном месте, то указание на её положение в записи числа отсутствует. При *n*-разрядном представлении модульной части формат с фиксированной точкой обеспечивает диапазон изменения абсолютного значения числа *A*, для которого выполняется неравенство

$2^n > |A| \geq 0$.

Одним из важнейших параметров представления чисел является ошибка представления. Ошибка представления может быть абсолютной (Δ) или относительной (δ). Для фиксированной точки максимальные значения этих ошибок определяются следующим образом.

В случае целых чисел:

$\Delta_{max} = 0.5$; $\delta_{max} = \Delta_{max} / A_{min} = 0.5$, где A_{min} – минимальное, отличное от нуля, значение числа.

В случае дробных чисел:

$\Delta_{max} = 0.5 \times 2^n = 2^{(n+1)}$; $\delta_{max} = \Delta_{max} / A_{min} = 2^{(n+1)} / 2^{-n} = 0.5$,

т.е. в худшем случае относительная ошибка при фиксированной точке может достигать сравнительно большого значения – 50%.

5.2. Арифметические операции над числами, представленными с фиксированной точкой

К числу основных арифметических операций, непосредственно реализуемых в ЭВМ, относятся операции сложения, умножения, деления. Остальные операции (например, такие, как возведение в степень, извлечение квадратного корня) реализуются программным способом.

Выполнение операций с числами, представленными с фиксированной точкой, рассмотрено в рамках материала по выполнению операций с алгебраическими числами (подраздел 1.3).

Выполнение длинных операций, таких, как умножение и деление, реализуется в два этапа:

– на первом этапе формируется знак искомого результата,

- на втором этапе, используя абсолютные значения операндов, ищем результата (произведение или частное), которому затем присваивается предварительно определенный знак.

Операнды, как правило, представлены в прямом коде, и знак результата, не зависимо от того, частное это или произведение, ищется за счет сложения по модулю 2 знаковых разрядов операндов. В результате этого знак результата положителен, если операнды имеют одинаковые знаки, или отрицательный, если операнды имеют разные знаки.

Выполнение второго этапа, т.е. умножение положительных чисел достаточно подробно изложено в пункте 1.2.3.

5.3. Представление чисел с плавающей точкой

При представления числа *с плавающей точкой* число в общем случае представляет собой смешанную дробь и имеет формат, приведенный на Рис.

1р	2р.	3р.	...	*к* р.	«.»	(*к*+1) р.	(*к*+2) р.	...	(*n*-1) р.	*n*р

Рис. 1.4. Формат представления числа с плавающей точкой

Местоположение точки в записи числа может быть различным, а так как сама точка в записи числа не присутствует, то для однозначного задания числа необходима не только его запись, но и информация о том, где в записи числа располагается точка, отделяющая целую и дробную части.

Поэтому в случае с плавающей точкой число X представляется в виде двух частей:

мантисса ($x_м$), отображающая запись числа, представляется в виде правильной дроби с форматом фиксированной точкой;

порядок ($x_п$), отображающий местоположение в этой записи точки, представляется в виде целого числа с форматом фиксированной точки.

Количественная оценка числа X определяется как

$$X = q^{x_п} \times x_м ,$$

где q – основание системы счисления.

Для двоичной системы счисления имеет место

$$X = 2^{x_п} \times x_м .$$

При s-разрядном представлении модуля записи мантиссы и k-разрядном представлении модуля записи порядка форма с плавающей точкой обеспечивает диапазон изменения абсолютного значения числа A, для которого выполняется неравенство:

$$2^{|x_п|_{max}} \times |x_м|_{max} = 2^p \times (1 - 2^{-s}) \geq |X| \geq 0 ,$$

где $p = 2^k - 1$.

В ЭВМ числа с плавающей точкой представляются в так называемой нормализованной форме, при которой в прямом коде мантиссы нормализованного числа в старшем разряде модуля имеет ненулевое значение, а для двоичной системы счисления – нормализованная мантисса должна иметь в старшем разряде модуля прямого кода значение 1, т.е. для двоичной системы мантисса должна удовлетворять неравенству:

$$1 > |x_м| \geq 0.5.$$

Для плавающей точки максимальные значения абсолютной и относительной ошибок определяются следующим образом.

Максимальная абсолютная погрешность представления чисел:

$$\Delta_{max} = 2^{-(s+1)} \times 2^p;$$

Максимальная относительная погрешность:

$$\delta_{max} = \Delta_{max} / A_{min} = 2^{-(s+1)} \times 2^p / (x_{м\,min} \times 2^p) = 2^{-(s+1)} \times 2^p / (2^{-1} \times 2^p) =$$
$$= 2^{-(s+1)} / (2^{-1}) = 2^{-s}.$$

Отсюда видно, что относительная ошибка при представлении чисел в форме с плавающей точкой существенно меньше, чем в случае с фиксированной точкой. Это, а также больший диапазон изменения представляемых чисел, является основным преимуществом представления чисел с плавающей точкой.

5.4. Арифметика с плавающей точкой

Результат сложения двух чисел $A = ap^{ma}$ и $B = bp^{mb}$ представленных в форме с плавающей запятой, должен быть тоже числом вида $C = cp^{mc}$. При этом для слагаемых и результата должно выполняться равенство

$$ap^{ma} + bp^{mb} = cp^{mc}.$$

Так как числа с разными порядками суммировать нельзя, то для сложения двух чисел, представленных в нормальной форме, необходимо предварительно их привести к общему порядку, т. е. преобразовать одно из слагаемых, например *B,* следующим образом:

$$B = bp^{mb} = b'p^{ma}$$

После этого можно вынести степень основания системы за скобки и выполнить сложение мантисс

Преобразованная мантисса должна быть правильной дробью с тем чтобы при сложении ничем не отличаться от обычных (непреобразованных) мантисс. Поэтому преобразованию подвергается всегда меньшее слагаемое, так как в противном случае происходит переполнение разрядной сетки мантиссы преобразуемого числа. При этом часть разрядов преобразуемой мантиссы может теряться. Поэтому арифметические действия над числами с плавающей запятой являются по своей сути приближенными, а не точными.

При сложении чисел в нормальной форме можно выделить четыре этапа.

На первом этапе уравниваются порядки слагаемых: меньший порядок увеличивается до большего, а мантисса преобразуемого числа сдвигается вправо на соответствующее количество разрядов. С этой целью производится вычитание порядков чисел. Знак и модуль разности будут определять соответственно, какое из слагаемых нужно преобразовывать и на сколько разрядов следует сдвинуть их мантиссу. При этом младшие разряды мантиссы могут пропадать, вследствие чего в слагаемое, сдвигаемое вправо, вносится погрешность.

Сложить два числа, представленные в формате с плавающей запятой:

$A = 0{,}315290 \cdot 10^{-2}$, $B = 0{,}114082 \cdot 10^{+2}$.

Обратите внимание, мантиссы чисел нормализованы. Очевидно, прежде чем складывать мантиссы, требуется преобразовать числа таким образом, чтобы они имели *одинаковые порядки*. Выравнивание порядков можно выполнить двумя способами — уменьшением большего порядка до меньшего или увеличением меньшего до большего (рис. 3.31).

$A =$	$0{,}315290 \cdot 10^{-2}$	
$B =$	$0{,}820000 \cdot 10^{-2}$	1140
$C =$	$1{,}135290 \cdot 10^{-2}$	

а

$A =$	$0{,}000031$	$5290 \cdot 10^{+2}$
$B =$	$0{,}114082$	$\cdot 10^{+2}$
$C =$	$0{,}114114$	$\cdot 10^{+2}$

б

На втором этапе производится преобразование мантисс слагаемых в один из модифицированных кодов: дополнительный или обратный.

На третьем этапе выполняется сложение мантисс по правилам сложения чисел с фиксированной запятой.

На четвертом этапе производится нормализация результатов (в случае необходимости), затем результат переводится в прямой код к нему приписывается общий порядок слагаемых и выполняется округление мантиссы результата.

В зависимости от абсолютных величин мантисс слагаемых сумма может получиться: нормализованной, денормализованной вправо, денормализованной влево (переполнение).

Следует отметить, что положительные нормализованные числа всегда имеют 1 в разряде p^{-1} *а* отрицательные числа, записанные инверсным кодом, имеют 0 в разряде p^{-1} Поэтому у нормализованных чисел значения цифр разрядов с весами p и p^{-1} не совпадают ни для положительных чисел, ни для изображений отрицательных чисел. Вследствие этого несовпадение цифр в знаковых разрядах свидетельствует о денормализации влево (переполнение), а совпадение цифр знакового и старшего разряда мантиссы — о нарушении нормализации вправо.

При переполнении (денормализации результата влево) мантисса результата сдвигается на один разряд вправо, а порядок увеличивается на единицу, так как сумма двух мантисс может быть денормализована влево не более чем на 1 разряд. В то же время при денормализации вправо мантисса сдвигается влево до появления в старшем разряде 1, при значении знака 0, или 0 при 1 в знаковом разряде, а из порядка вычитается число единиц, рав-

ное числу сдвигов мантиссы. Такой порядок действий обусловливается тем, что количество разрядов, на которое может нарушиться нормализация вправо, ничем не ограничено.

Обычно количество сдвигов влево ограничивают числом разрядов сумматора, так как процесс сдвигов при таком порядке действий может оказаться бесконечным, если в результате алгебраического сложения мантисс получен нуль. В этом случае после выполнения предельного числа сдвигов мантиссу результата представляют машинным нулем. Мантиссу результата представляют также машинным нулем, если в процессе ее сдвига влево порядок числа окажется меньше допустимого, т. е. абсолютная величина результата будет меньше, чем минимально возможное машинное число. Однако приравнивание результата нулю при отрицательном переполнении порядка допустимо только в отдельных случаях, когда результат должен складываться со зна» чительно большей величиной. Если исчезновение порядка не отмечается каким-то образом, то могут возникнуть неожиданные ситуации, когда (AB) C = 0, а (AC) B ¹ 0, обусловленные ограниченностью разрядной сетки мантиссы и округлением результатов.

$$
\begin{array}{rl}
[A]_{\text{пр}} = & \underbrace{0\,011}_{m_a}\ \underbrace{1\,10101}_{a} \\
[B]_{\text{пр}} = & \underbrace{0\,101}_{m_b}\ \underbrace{0\,11001}_{b} \\
[A]'_{\text{пр}} = & 0\,101\ 1\,0010101 \\
[a]^{\text{м}}_{\text{д}} = & 11,1101011 \\
+\ [b]^{\text{м}}_{\text{д}} = & 00,11001 \\
\hline
[c]^{\text{м}}_{\text{д}} = & 00,1001111 \quad \text{рез}
\end{array}
$$

результат нормализован, поэтому производим только его усечение: $[C]_{\text{пр}} = 0\ 101\ 1\ 10101$.

2) $[A]_{пр} = 0\ 101\ 0\ 10101$

$[B]_{пр} = 0\ 011\ 0\ 11001$

$[B]'_{пр} = 0\ 10100011001$

$[a]^{м}_{д} = 11{,}0101100$

$+\ [b]^{м}_{д} = 00{,}0011001$

$[c]^{м}_{д} = 11{,}1000101.$

Нормализация: $[C]_{д} = 1{,}000101$, так как результат денормализован вправо на 1 разряд. Коррекция порядка: $m' = m_{общ} - 1 = 100$. Перевод в прямой код и усечение мантиссы результата:

$[C]пр = 1.\ 111011.$

Окончательно имеем

$[C]_{пр} = 0\ 100\ 1\ 11101.$

При выполнении сдвига влево необходимо следить, чтобы справа не было введено ошибок

$[A]_{пр} = 0\ 101\ 1\ 10101$

$[B]_{пр} = 0\ 100\ 1\ 11001$

$[B]_{пр} = 0\ 101\ 1\ 011001$

$[a]^{м}_{д} = 11{,}010110$

$+\ [b]^{м}_{д} = 11{,}100111$

$[c]^{м}_{д} = 10{,}111101$

Результат денормализован влево, поэтому мантиссу сдвигаем вправо на один разряд [c) = 1,0111101. Переводим в прямой код и производим усечение маитиссы

$$[C]_{пр} = 0\ 1101\ 10000.$$

В системах команд современных ЭВМ имеется большой набор команд для выполнения операций сложения и вычитания как с длинными, так и с короткими операндами с нормализацией и без нормализации результата. Выполнение операций вычитания отличается от сложения тем, что в начале операции знак второго операнда искусственно меняется на обратный.

Так как нередко основание системы счисления берется равным $2^4 = 16$, то при выравнивании порядков сдвиг производится на количество разрядов, кратное 4, В основной памяти и в АУ порядки и мантиссы хранятся в

прямом коде. Поэтому в АУ мантиссы с одинаковыми знаками всегда складываются как положительные числа, а результату присваивается знак первого операнда. Если же знаки не равны, то при алгебраическом сложении мантисса вычитаемого преобразуется в инверсный код. Если результат от сложения кодов мантисс получается в дополнительном коде, то в конце выполнения операции он переводится в прямой код.

Переполнение, т. е. денормализация влево, устраняется сдвигом мантиссы вправо на четыре разряда и увеличением порядка на единицу. Если возникает при этом переполнение порядка, то формируется требование прерывания. При нормализации влево может появиться характеристика меньше нуля (порядок меньше — 64). В этом случае также вырабатывается требование прерывания, а порядку и мантиссе результата присваиваются нулевые значения.

Если в результате алгебраического сложения получилась мантисса, равная 0, то реакция ЭВМ на этот результат зависит от того, разрешено или нет прерывание при потере значимости. Знак суммы при нулевой мантиссе результата всегда положителен.

Пример

Найти разность $C1$ чисел A и B, представленных с плавающей точкой, если A и B представлены в виде порядков, соответственно $[a_п]_{пк}$ и $[в_п]_{пк}$ и мантисс, соответственно $[a_м]_{пк}$ и $[в_м]_{пк}$, где $[a_п]_{пк}$ = = 1.001, $[a_м]_{пк}$ = 1.11001, $[в_п]_{пк}$ = 0.001, $[в_м]_{пк}$ = 0.11100.

При выполнении операций использовать дополнительный модифицированный код.

Решение

Начнем с выравнивания порядков.

Для этого из порядка первого числа вычитается порядок второго числа:

1.111 – $[a_п]_{дк}$

\+ 1.111 – $[- в_п]_{дк}$

1.110 – разность порядков в дополнительном коде,

1.010 – разность порядков в прямом коде.

Так как знак разности порядков отрицательный, то в качестве общего порядка, а следовательно, и предварительного значения порядка искомого результата $C1_п$`, берется порядок второго числа ($в_п$). Для того чтобы взять в качестве порядка первого числа порядок второго числа, т.е. увеличить его порядок на 2, необходимо мантиссу этого меньшего числа умножить на 2^{-2}, т.е. выполнить её арифметический сдвиг на два разряда вправо.

Таким образом, будем иметь после выравнивания следующую форму представления операндов:

$[a_м`]_{пк}$ = 1.00110,

$[в_м`]_{пк}$ = 0.11100.

После выравнивания порядков можно определить предварительное значение мантиссы $C1`$ как

$C1` = [a_м`]_{пк} - [в_м`]_{пк}$.

$$\begin{array}{ll} & 11.11010 - [a_м`]_{мдк} \\ + & \underline{11.00100} - [- в_м`]_{мдк} \\ & 10.11110 - [C1`]_{мдк} \\ & 10.00010 - [C1`]_{мпк}. \end{array}$$

Из записи $[C1`]_{дк}$, полученной после вычитания мантисс операндов с выравненными порядками, видно, что нормализация представления результата нарушена. Поэтому для данного примера необходимо выполнить этап устранения нарушения нормализации.

В данном случае нарушение нормализации слева от точки, так как получено $[C1`]_{пк}$ с ненулевой целой частью (неодинаковые разряды в поле знака использованного модифицированного дополнительного кода). Для того чтобы привести полученную предварительную мантиссу к нормализованной форме, достаточно её разделить на 2, то есть выполнить её арифметический сдвиг вправо. В результате будем иметь окончательное значение мантиссы:

$C1 = C1` \times 2^{-1} = 10.00010 \times 2^{-1} = 11.10001$.

Деление мантиссы $C1`$ на 2 сопровождается изменением ранее найденного предварительного значения порядка результата $C1_п`$ на +1.

$$\begin{array}{ll} & 00.001 - [c_п`]_{мпк} \\ + & \underline{00.001} - +1 \\ & 00.010 - [c1_п]_{мпк} \end{array}$$

После устранения нарушения нормализации окончательный результат будет иметь вид

$C1$ **à** $\{[c1_п]_{пк} = 00.010, [c1_м]_{пк} = 11.10001\}$.

<u>Операция умножения</u>

4.6. УМНОЖЕНИЕ ЧИСЕЛ В МАШИНАХ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Если заданы $A = a \cdot 2^{m_1}$ и $B = b \cdot 2^{m_2}$ в нормальной форме, то их произведение составит

$$AB = ab \cdot 2^{m_1+m_2}.$$

Следовательно, умножение нормализованных чисел состоит из следующих этапов:

1. Определение знака произведения путем сложения по mod 2 знаковых цифр мантисс сомножителей.
2. Алгебраическое сложение порядков сомножителей в инверсном коде с целью определения порядка произведения.
3. Умножение модулей мантисс сомножителей по правилам умножения чисел с фиксированной запятой.
4. Нормализация и округление мантиссы результата. Следует учесть, что сомножители являются нормализованными числами. Поэтому денормализация мантиссы произведения возможна только на один разряд вправо. Она устраняется путем сдвига мантиссы на один разряд влево и вычитания единицы из порядка результата.
5. Присвоение знака результату.

Первые три операции могут выполняться одновременно, так как они независимы. Наличие операции определения знака произведения предполагает, что умножение мантисс выполняется в прямом коде. Однако можно умножать непосредственно изображения мантисс с последующей коррекцией результата, как и при числах с фиксированной запятой.

При выполнении операции умножения в машине с плавающей запятой может получиться переполнение отрицательного порядка, которое будет интерпретировано как машинный нуль, если программой пользователя игнорируется признак исчезновения порядка. Может также возникнуть и переполнение положительного порядка. В этом случае в первую очередь необходимо нормализовать мантиссу результата. Если и после этого переполнение порядка не устраняется, то машиной формируется признак переполнения порядка.

Округление мантиссы результата производится следующим образом. Если мантисса результата получилась без нарушения нормализации, то округление производится обычно добавлением единицы в $(n + 1)$-й разряд сумматора. Если мантисса произведения денормализована вправо, то при $2n$-м разрядном результате вначале производится нормализация, а затем округление; при $(n + 1)$-разрядном результате вначале производится округление, а затем нормализация.

Во всех случаях единица округления добавляется в $(n + 1)$-й разряд сумматора только при наличии в нем кода «1».

С точки зрения представления чисел с плавающей точкой поиск произведения

$C2 = A \times B$

сводится к поиску $C2_{\text{п}}$ и $C2_{\text{м}}$, соответственно порядку и мантиссе произведения на основании порядка $a_{\text{п}}$ и мантиссы $a_{\text{м}}$ множимого и порядка $в_{\text{п}}$ и мантиссы $в_{\text{м}}$ множителя. Учитывая общую запись чисел с плавающей точкой, произведение двух операндов представляется в виде

$$C2 = A \times B = 2^{a\text{п}} \times a_{\text{м}} \times 2^{в\text{п}} \times в_{\text{м}} = 2^{a\ \text{п}+в\text{п}} \times a_{\text{м}} \times в_{\text{м}} = 2^{c}$$

$^{\text{п}} \times c2_{\text{м}}$.

Отсюда вытекает, что порядок произведения определяется как сумма порядков сомножителей, а мантисса произведения – как произведение мантисс сомножителей. Однако, учитывая возможность нарушения нормализации при умножении мантисс, в результате указанных действий будет найдено предварительное значения порядка и мантиссы искомого произведения, и окончательное значение произведения будет найдено только после устранения нарушения нормализации.

Таким образом, имеем:

$C2_{\text{п}}` = a_{\text{п}} + в_{\text{п}}$;

$C2_{\text{м}}` = a_{\text{м}} \times в_{\text{м}}$.

Отсюда последовательность действий, обеспечивающих получение произведение двух чисел, заключается в следующем:

- определяется знак произведения как сумма по модулю два знаковых разрядов мантисс сомножителей;
- определяется предварительное значение порядка произведения посредством суммирования порядков сомножителей;
- определяется предварительное значение мантиссы произведения как произведения мантисс операндов;
- устраняется нарушение нормализации мантиссы произведения (если нарушение имеет место) соответствующей корректировкой предварительного значения порядка и мантиссы искомого произведения.

При формировании мантиссы произведения нормализованных чисел с плавающей точкой возможен только один вид нарушения нормализации – нарушение нормализации справа от точки с появлением нуля только в старшем разряде мантиссы.

Пример

Найти произведение C чисел A и B, представленных с плавающей точкой, если A и B представлены в виде порядков, соответственно $[a_п]_{пк}$ и $[в_п]_{пк}$ и мантисс, соответственно $[a_м]_{пк}$ и $[в_м]_{пк}$,

где $[a_п]_{пк} = 1.010$, $[a_м]_{пк} = 1.1010$, $[в_п]_{пк} = 0.001$, $[в_м]_{пк} = 0.1001$.

При выполнении операций использовать *обратный код*. При умножении мантисс использовать метод умножения, начиная *со старшего разряда* множителя со сдвигом промежуточного результата.

Решение

Знак искомого произведения, представляемого знаком его мантиссы, отрицательный, так как знаки мантисс сомножителей неодинаковые.

Предварительное значение порядка произведения определяется следующим образом:

$C_п' = a_п + в_п$:

11.101 – $[a_п]_{мок}$

+ 00.001 – $[в_п]_{мок}$

11.110 – $[C_п`]_{мок}$

11.001 – $[C_п`]_{мпк}$, т.е. $[C_п`]_{пк} = 1.001$.

Абсолютное значение предварительного значения мантиссы произведения определяется следующим образом:

$[C_м`]$:

0.1010 – ú$a_м$ú

× 0.1001 – ú$в_м$ú

.0000 – начальное значение промежуточного произведения

+ 1010 – первый младший разряд множителя равен единице

1010 – промежуточное произведение с учетом первого разряда

01010 – сдвинутое промежуточное произведение

001010 – второй разряд множителя равен нулю, поэтому выполняется только сдвиг

0001010 – третий разряд равен нулю, поэтому выполняется только сдвиг

+ 1010 – четвертый разряд равен единице

1011010 – промежуточное произведение с учетом старшего разряда

01011 010 – сдвинутое промежуточное произведение.

Таким образом,

$[C_м`]_{пк}$ = 0.01011010.

С учетом округления имеем

$[C_м]_{пк}$ = 0.01011.

Мантисса произведения ненормализованная, поэтому необходимо сдвинуть мантиссу влево на один разряд, а предварительное значение порядка произведения уменьшить на единицу. После нормализации с учетом ранее полученного знака окончательные значения мантиссы и порядка произведения будут следующими:

$[C_м]_{пк}$ = 1.1011.

$[C_п]_{пк}$ = 1.010.

Операция деления

С точки зрения формирования частного представления чисел с плавающей точкой поиск частного $C З = A / B$ сводится к поиску $C З_п$ и $C З_м$, соответственно порядку и мантиссы частного на основании порядка $a_п$ и мантиссы $a_м$ делимого и порядка $в_п$ и мантиссы $в_м$ делителя. Учитывая общую запись чисел с плавающей точкой, произведение двух операндов представляется в виде

$$C З = A / B = 2^{an} \times a_м / (2^{вп} \times в_м) = 2^{aп - вп} \times (a_м / в_м) = 2^{сп} \times c_м.$$

Отсюда следует, что порядок частного определяется как разность порядка делимого и делителя, а мантисса – как частное от деления мантиссы делимого на мантиссу делителя. Однако, учитывая то, что при делении мантисс может произойти нарушение нормализации, в результате указанных действий будет найдено предварительное значения порядка и мантиссы искомого частного. Окончательные значения порядка и мантиссы частного будут определены после устранения нарушения нормализации в предварительном результате.

При формировании мантиссы частного нормализованных чисел с плавающей точкой возможен только один вид нарушения нормализации - нарушение нормализации слева от точки.

Пример

Найти частное $C З$ от деления чисел A на B, представленных с плавающей точкой, если A и B представлены в виде порядков, соответственно

$[a_п]_{пк}$ и $[в_п]_{пк}$ и мантисс, соответственно $[a_м]_{пк}$ и $[в_м]_{пк}$,

где $[a_п]_{пк}$ = 1.010, $[a_м]_{пк}$ = 1.1010, $[в_п]_{пк}$ = 0.001, $[в_м]_{пк}$= 0.1001.

При выполнении операций использовать обратный код. При делении мантисс использовать метод деления без восстановления ос-

татка. При вычитании порядков и формирования мантиссы частного использовать модифицированный обратный код.

Решение

Знак искомого частного, представляемого знаком его мантиссы, отрицательный, так как знаки мантисс сомножителей не одинаковые.

Предварительное значение порядка $[C\text{З}_\text{п}`]_\text{ок}$ частного определяется следующим образом:

$C\text{З}_\text{п}` = a_\text{п} - в_\text{п}$:

11.101 – $[a_\text{п}]_\text{мок}$

+ 11.110 – $[в_\text{п}]_\text{мок}$

111.011

\+ 1

11.011 – $[C\text{З}_\text{п}`]_\text{мок}$, т.е. $[C\text{З}_\text{п}`]_\text{пк} = 1.011$.

Абсолютное значение предварительного значения мантиссы частного ищется за счет выполнения шести тактов деления следующим образом:

00.1010 – $[\acute{у}a_\text{м}\acute{у}]_\text{ок}$,

+ 11.0110 – $[-\acute{у}в_\text{м}\acute{у}]_\text{ок}$,

100.0000

\+1 – учет переноса (переполнения знакового поля) при сложении в обратном коде,

00.0001 – положительный остаток первого такта,

00.0010 – сдвинутый остаток,

+ 11.0110 – $[-\acute{у}в_\text{м}\acute{у}]_\text{мок}$,

11.1000 – отрицательный остаток второго такта,

11.0001 – остаток после арифметического сдвига влево,

+ 00.1001 – $[\acute{у}в_\text{м}\acute{у}]_\text{мок}$,

11.1010 – отрицательный остаток третьего такта,

11.0101 – остаток после арифметического сдвига влево,

+ 00.1001 – $[\acute{у}в_\text{м}\acute{у}]_\text{мок}$,

11.1110 – отрицательный остаток четвертого такта,

11.1101 – остаток после арифметического сдвига влево,

+ 00.1001 – $[\acute{у}в_\text{м}\acute{у}]_\text{мок}$,

100.0110

\+ 1

00.0111 – положительный остаток пятого такта

00.1110 – остаток после арифметического сдвига влево,

+ 11.0110 – $[-\acute{у}в_\text{м}\acute{у}]_\text{мок}$,

100.0100

+ 1

00.0101 – положительный остаток шестого такта,

00.1010 – остаток после арифметического сдвига влево.

Таким образом, учитывая знаки остатков, полученных на шести тактах, абсолютное предварительное значение мантиссы искомого частного равно:

$[\acute{u}\,C\mathrm{З}_{\mathrm{м}}\grave{}\,\acute{u}]_{\mathrm{пк}} = 1.00011,$

с учетом округления:

$[\acute{u}\,C\mathrm{З}_{\mathrm{м}}\grave{}\,\acute{u}]_{\mathrm{пк}} = 1.0010.$

Мантисса частного не нормализованная (нарушение нормализации слева от точки), поэтому необходимо сдвинуть мантиссу вправо на один разряд, а предварительное значение порядка частного увеличить на единицу. После нормализации окончательное значение мантиссы и порядка частного равны:

$[C\mathrm{З}_{\mathrm{м}}]_{\mathrm{пк}} = 0.1001,$

$[C\mathrm{З}_{\mathrm{п}}]_{\mathrm{пк}} = 0.000.$

Если числа A и B заданы в нормальной форме, то их частное будет равно

$$Y = a : b = (a : b) \cdot 2^{m_1 - m_2},$$

где a и b — мантиссы, а m_1 и m_2 — порядки соответственно чисел A и B. Отсюда следует, что операция деления в машинах с плавающей запятой выполняется в пять этапов.

1-й этап. Определение знака частного путем сложения по модулю 2 знаковых цифр мантисс операндов.

2-й этап. Деление модулей мантисс операндов по правилам деления чисел с фиксированной запятой.

3-й этап. Определение порядка частного путем вычитания порядка делителя из порядка делимого.

4-й этап. Нормализация результата и его округление.

5-й этап. Присвоение знака мантиссе результата.

Два первых этапа полностью совпадают с этапами деления чисел с фиксированной запятой. Третий этап представляет собой обычное сложение в инверсных кодах.

При делении нормализованных чисел денормализация результата возможна только влево и только на один разряд. Это обусловлено тем, что мантисса любого нормализованного числа лежит в пределах

$$2^{-1} \leqslant |a| < 1 - 2^{-n}.$$

Тогда наименьшая и наибольшая возможные величины мантиссы частного равны соответственно

$$|y_{\min}| = \frac{2^{-1}}{1-2^{-n}} > 2^{-1}; \quad |y_{\max}| = \frac{1-2^{-n}}{2^{-1}} = 2 - 2^{-(n-1)} < 2,$$

т. е. мантисса частного лежит в пределах

$$2^{-1} < |y| < 2.$$

Поэтому на четвертом этапе может возникнуть необходимость нормализации мантиссы частного путем ее сдвига вправо на один разряд и увеличения порядка частного на единицу. Если же перед делением сдвинуть делимое на один разряд вправо, то на 4-м этапе может потребоваться нормализация результатов влево на 1 разряд.

Пример. Заданы $[A]_{\text{пр}} = 1\,110\,1\,101$:

$[B]_{\text{пр}} = \underbrace{1\,101}_{\text{порядок}}\ \underbrace{0\,110}_{\text{мантисс.}}$

1-й этап: определение знака частного: $1 \oplus 0 = 1$.

2-й этап: деление мантисс:

$$|y| = |a| : |b| \approx 0.1101.$$

3-й этап: определение порядка частного

$$\begin{aligned} &[m_a]^{\text{м}}_{\text{д}} = 11{,}010 \\ +\ &[m_b]^{\text{м}}_{\text{д}} = 00{,}101 \\ &[m_y]^{\text{м}}_{\text{д}} = 11.111. \end{aligned}$$

Перевод порядка частного в прямой код:

$$[m_y]_{\text{пр}} = 1.001.$$

4-й этап: частное получилось нормализованным, поэтому производится только округление мантиссы.

5-й этап: результат $[Y]_{\text{пр}} = A/B = 1\,001\,1\,111$.

НЕОСНОВНЫЕ АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

5.1. ОПЕРАЦИЯ ИЗВЛЕЧЕНИЯ КВАДРАТНОГО КОРНЯ

Основными арифметическими операциями ЭВМ являются алгебраическое сложение, умиожеиие и деление. Время выиолиеиия этих операций в основиом определяет быстродействие машин. Однако в ходе решеиия задачи очеиь часто иеобходимо выполиять другие, так иазываемые иеосиовиые операции. такие, как извлечеиие квадратного кория, вычисление тригоиометрических и других элемеитарных функций. Неосиовиые арифметические операции реализуются обычио с помощью стаидартных программ, которые входят в состав математического обеспечеиия (МО) ЭВМ и вызываются простым обращением к соответствующей библиотеке подпрограмм. Однако для реализации этих подпрограмм требуется значительио больше времени, чем для выполнения основных арифметических операций.

Вместе с тем, в настоящее время существенно повысилась степень интеграции и снизилась стоимость электронных компонентов ЭВМ. В связи с этим актуальным стал вопрос о передаче части функций МО машин аппаратным средствам. Прежде всего это касается выполнения сложных арифметических операций, таких, как перевод из одной системы счисления в другую, вычисление численного значения многочлена, вычисление элементарных функций, извлечение корня квадратного, выполнение операций комплексной арифметики. При этом интерес представляет исследование таких методов вычисления указанных операций, которые допускают аппаратную реализацию за время, сонзмеримое со временем выполнения основных операций.

Естественно, что алгоритмы выполнения неосновных арифметических операций, ориентированные на аппаратурную реалнзацию, как правило, существенно отличаются от алгоритмов программной реализации, так как первые строятся обычно на основе многократного выполнения операций сложения — вычитания и сдвигов. Рассмотрение алгоритмов неосновиых арифметических операций начнем с операции извлечения корня квадратного.

Эта операция включается как самостоятельная в систему комаид ЭВМ в том случае, когда она составляет не менее 2 % от общего числа операций или является составной частью алгоритмов, которые необходимо выполнять в реальном масштабе времени, т. е. с высоким быстродействием.

Имеются два пути решения рассматриваемой задачи. Первый путь связан с разработкой микропрограммы извлечения квадратного корня с использованием набора основных арифметических операций. При этом микропрограмма реализует один из известных итерационных методов извлечения квадратного корня с помощью базовой аппаратуры. Например, в универсальных ЭВМ для приближенного вычисления квадратного корня применяется обычно известная формула Ньютона:

$$B_{i+1} = 0{,}5\,(B_i + A/B_i),$$

где B_{i+1} есть $(i+1)$-е приближение $B = \sqrt{A}$, а $i = 0, 1, 2 \ldots$

Другой путь состоит в создании алгоритма извлечения квадратного корня, похожего по структуре на алгоритмы основных арифметических операций, например деления.

Наиболее простой алгоритм сводится к подбору цифр в результате разряд за разрядом, начиная со старшего, т. е. с 2^{-1}.

При этом вычисление i-й цифры B происходит следующим образом. После получения $(i-1)$-й цифры b_{i-1} в i-й разряд B для пробы помещается 1. Вычисляется разность $(A - B_i^2) = R_i$. Если $R_i > 0$, то B_i есть число, у которого цифры всех i разрядов совпадают с цифрами искомого результата B. Если $R_i < 0$, то в i-м разряде b_i нужно поставить 0 и переходить к вычислению $(i+1)$-го разряда.

Так как вычисление этого разряда снова начнется с подстановки пробной 1, то в случае $(A - B_i^2) = R_i < 0$ можно вместо «стирания» 1 в i-м разряде вычесть 1 из $(i+1)$-го разряда.

Аналогичным образом можно вычислять $\sqrt[3]{A}$, $\sqrt[4]{A}$ и т. д.

Пример. Вычислить $B = \sqrt{A} = \sqrt{0{,}10111}$ с точностью до 5-ти знаков после запятой

$-\ \begin{array}{r}00{,}10111\\00{,}01\end{array}$		
00,01111		0,1
$-\ \begin{array}{r}00{,}11110\\00{,}101\end{array}$	Сдвиг	
00,01010		0,11
$-\ \begin{array}{r}00{,}10100\\00{,}1101\end{array}$	Сдвиг	
11,11010		0,110
$+\ \begin{array}{r}11{,}10100\\00{,}11011\end{array}$	Сдвиг	
00,01111		0,1101
$-\ \begin{array}{r}00{,}11110\\00{,}110101\end{array}$	Сдвиг	
00,000111		$B = \sqrt{A} = 0{,}11011$

Результат вычислений всегда получается с недостачей, поэтому желательно его округление. Для этого необходимо определить $(n + 1)$ разряд корня.

Таким образом, процесс вычисления квадратного корня состоит из ряда однотипных циклов, выполняемых последовательно, и шага округления. В каждом цикле находится очередная цифра корня, значение которой определяет знак результата алгебраического сложения остатка с «переменным делителем», которому приписывается знак, противоположный знаку предыдущего остатка (в первом цикле остаток — это подкоренное число). Если новый остаток положительный, то в регистр результата операции заносится 1 и формируется новое значение переменного делителя путем приписывания к результату операции справа пары цифр 01 (в первом цикле результат равен 0). Затем остаток сдвигается на один разряд влево и выполняется следующий цикл. Если же новый остаток отрицательный, то в регистр

результата операции заносится 0, а новое значение делителя формируется путем приписывания к результату справа пары цифр 11. После чего текущий остаток сдвигается влево на один разряд и выполняется следующий шаг алгоритма.

Для извлечения корня квадратного из числа с плавающей запятой необходимо порядок числа разделить на два, а из мантиссы извлечь корень по правилам для чисел с фиксированной запятой. Действительно, число с плавающей запятой имеет вид

$$B = a \cdot 2^m, \text{ откуда } B = \sqrt{A} = \sqrt{a} \cdot 2^{m/2}.$$

Для деления порядка на два его необходимо сдвинуть на один разряд вправо, если он четный. Если же он нечетный, то необходимо прибавить к порядку единицу, затем сдвинуть порядок и мантиссу на один разряд вправо. То есть, если $m = 2k - 1$, то

$$A = a \cdot 2^{2k-1} = 2^{2k}(a \cdot 2^{-1}) \text{ и } B = \sqrt{A} = 2^k \sqrt{a \cdot 2^{-1}}.$$

Так как мантиссы всегда нормализованы и в первом цикле из мантиссы производится вычитание числа 0,01, то первый остаток будет всегда положительным, т. е первая цифра результата всегда будет единица Следовательно, при выполнении операции извлечения корня квадратного в машине с плавающей запятой никогда не может произойти нарушение нормализации результата.

5.2. ВЫЧИСЛЕНИЕ СУММ ПАРНЫХ ПРОИЗВЕДЕНИЙ

К получению сумм парных произведений приходится обращаться, например, при решении задач линейной алгебры. Оно сводится к вычислению выражения вида

$$\sum_{i=1}^{m} A_i B_i = A_1 B_1 + A_2 B_2 + \cdots + A_m B_m. \qquad (5.1)$$

Трудоемкость данной операции характеризует величина m, которая в зависимости от вида решаемой задачи может меняться в широких пределах и принимать очень большие значения. В ЭВМ эта операция обычно выполняется программно путем вычисления отдельных произведений и последующего их накопления, что требует значительных затрат машинного времени Длительность операции можно сократить с помощью дополнительных аппаратных средств. Далее будем считать, что числа A_i и B_i заданы в форме с запятой, фиксированной перед старшим разрядом.

5.3. АРИФМЕТИКА КОМПЛЕКСНЫХ ЧИСЕЛ

В последнее время существенно возрос интерес к теории функций комплексного переменного, особенно в связи с появлением теории цифровой обработки сигналов как самостоятельной дисциплины. В рамках этой теории возможно не только выполнение задач спектрального анализа, но и решение дифференциальных и интегральных уравнений при помощи преобразования Лапласа или Фурье, исследование систем автоматического управления, расчет электрических цепей и т. п. Алгоритмы решения задач перечисленных классов требуют выполнения действий над комплексными числами.

В ЭВМ комплексное число может быть представлено минимум парой вещественных чисел. Этому удовлетворяют три формы: алгебраическая, полярная и показательная. Предпочтение отдают обычно алгебраической форме (т. е. представлению комплексного числа A в виде

$$A = \mathrm{Re}\, A + j\, \mathrm{Im}\, A,$$

где $\mathrm{Re}\, A$ и $\mathrm{Im}\, A$ — вещественные числа, а j — мнимая единица, удовлетворяющая условию $j^2 = -1$), так как в этой форме просто выполняется наиболее часто встречающаяся операция алгебраического сложения комплексных чисел A и B как две операции сложения в инверсных кодах соответствующих действительных и мнимых частей этих чисел:

$$A + B = (\mathrm{Re}\, A + \mathrm{Re}\, B) + j\,(\mathrm{Im}\, A + \mathrm{Im}\, B).$$

узкую область. Методическая погрешность знакопеременна и равномерно распределена на интервале изменения аргумента. Для вычисления ЭФ с произвольной разрядностью в некоторых ЭВМ используется комбинированный таблично-полиномиальный алгоритм. Приближение любой ЭФ в приведенном интервале ведется с помощью подпрограмм не одним ортогональным полиномом, а их набором, каждый из которых применяется на подынтервалах с возрастанием степени аппроксимации от одного подынтервала к следующему.

Табличные методы основаны главным образом на кусочно-линейной и криволинейной аппроксимации. Для вычисления ЭФ этим методом требуется выполнить малое число арифметических операций, однако объем таблиц и время поиска в них может быть большим. Поэтому этот метод применяется в машинах с небольшой разрядностью слов.

При методе рационального приближения ЭФ функцию представляют в виде отношения двух полиномов, причем число членов в каждом полиноме намного меньше, чем при соответствующем разложении в ряд Тейлора. Однако коэффициенты полиномов должны обязательно храниться в памяти. Для вычисления ЭФ следует вычислить два полинома и выполнить операцию деления.

5.4. МЕТОДЫ ВЫЧИСЛЕНИЯ ЭЛЕМЕНТАРНЫХ ФУНКЦИЙ

В цифровой вычислительной технике применяют следующие методы вычисления элементарных функций (ЭФ) [27, 38]: разложение в ряд Тейлора (степенные полиномы), аппроксимацию с помощью различных полиномов, табличные методы, рациональные приближения ЭФ, использование цепных дробей, итерационные (рекуррентные).

Степенные полиномы (отрезок ряда Тейлора, полином Чебышева и т. д.) вычисляются в ЭВМ чаще всего по схеме Горнера. При этом требуется выполнить m операций умножения и m операций сложения (m — степень полинома). К сожалению ряд Тейлора очень медленно сходится для некоторых функций (натуральный логарифм, обратные тригонометрические и гиперболические функции), и поэтому время вычисления будет большим, а инструментальная погрешность увеличивается. Методическая погрешность этого метода монотонно увеличивается с ростом аргумента, и поэтому приходится предварительно сводить аргумент в более узкую область с помощью соответствующих преобразований. Достоинством разложения в ряд Тейлора (в отличие от аппроксимации полинома Чебышева) является то, что можно вычислять коэффициенты членов ряда непосредственно при вычислении функций и не хранить их в памяти ЭВМ. Однако при этом возрастает время вычисления ЭФ. Единообразие вычисления всех ЭФ трудно обеспечить из-за плохой сходимости ряда для некоторых функций.

Метод полиномиальной аппроксимации используется в ЭВМ наиболее часто. Он характеризуется достаточно высоким единообразием вычисления всех ЭФ, однако при этом в памяти необходимо хранить большое количество коэффициентов всех полиномов. Для ускорения сходимости полинома аргумент предварительно сводится в более узкую область. Методическая погрешность знакопеременна и равномерно распределена на интервале изменения аргумента. Для вычисления ЭФ с произвольной разрядностью в некоторых ЭВМ используется комбинированный таблично-полиномиальный алгоритм. Приближение любой ЭФ в приведенном интервале ведется с помощью подпрограмм не одним ортогональным полиномом, а их набором, каждый из которых применяется на подынтервалах с возрастанием степени аппроксимации от одного подынтервала к следующему.

Табличные методы основаны главным образом на кусочно-линейной и криволинейной аппроксимации. Для вычисления ЭФ этим методом требуется выполнить малое число арифметических операций, однако объем таблиц и время поиска в них может быть большим. Поэтому этот метод применяется в машинах с небольшой разрядностью слов.

Для большинства других методов характерно отсутствие единообразной методики вычисления всех ЭФ. Это приводило к тому, что выбирался набор так называемых базовых функций, вычисляемых выбранным методом, а остальные ЭФ выражались через базовые и вычислялись на их основе. Например, в машине МИР базовый набор ЭФ состоит из функций $\ln x$; a^x; $\cos x$; $\arcsin x$. Такая методика приводит к многоуровневой организации управления, усложнению структуры управляющего устройства и к увеличению времени вычисления.

Разработаны новые эффективные итерационные алгоритмы вычисления ЭФ. Этот метод чаще всего называется методом «цифра за цифрой», так как после n итераций алгоритма получается значение функции с точностью до единицы n-го разряда. Однако следует заметить, что в отличие от традиционных методов «цифра за цифрой» (например, деление) в данном методе на каждом шаге итерации получается полноразрядный приближенный результат, а приращение прибавляется к нему, так что на любом шаге из-за переносов может измениться любая цифра результата, вплоть до старшего разряда. Тем не менее этот метод обычно называется методом «цифра за цифрой». Важным преимуществом этого метода является его эффективность для непосредственного вычисления почти всех ЭФ. С помощью этого метода можно также находить корни полиномов, выполнять преобразования координат и преобразования систем счисления чисел. Погрешность вычислений легко компенсируется с помощью дополнительных разрядов. Области сходимости достаточно широкие, а для некоторых функций вообще не требуется приведения аргумента. Так как алгоритмы метода построены на простых операциях сдвига и сложения, то их аппаратная реализация является простой и эффективной. Метод позволяет вычислить большинство ЭФ за время трех операций деления (если не учитывать приведение аргумента), причем его алгоритмам присущ внутренний параллелизм, позволяющий еще более повысить быстродействие за счет совмещения операций в одном процессоре. Недостатками метода является большое количество используемых констант и применение операционного устройства со специфической структурой.

Таким образом, метод «цифра за цифрой» обладает следующими преимуществами при структурной реализации: высокое быстродействие алгоритмов, основанных на операциях сдвига и сложения; единообразие вычисления почти всех ЭФ и оправданные аппаратурные затраты; простая организация вычислительного процесса, малое число уровней управления; удобство аппаратной компенсации погрешностей, возникающих при реализации алгоритмов на ЭВМ.

Метод «цифра за цифрой»

В основе данного метода лежит процедура преобразования прямоугольных координат точки в полярную, которую приспособили для вычисления большинства ЭФ. Метод «цифра за цифрой» в геометрическом смысле есть последовательность преобразований вектора в плоскости XY, т. е. последовательность поворотов радиуса вектора на стандартные углы вокруг начала координат с одновременным изменением длины вектора. В машине этот процесс представляется арифметическими преобразованиями координат вектора.

Например, для вычисления функций $\sin\varphi$, $\cos\varphi$ берется исходный вектор с координатами (1, 0) и поворачивается последовательно на углы α_i ($i = 0, 1\ldots$) так, чтобы алгебраическая сумма всех этих углов была равна φ. Тогда, очевидно, координаты итогового вектора равны искомым функциям $y = \sin\varphi$, $x = \cos\varphi$. Согласно известным формулам, при каждом повороте вектора на угол α_i новые координаты вектора выражаются через предыдущие [27]:

$$y_{i+1} = \cos\alpha_i (y_i + x_i \operatorname{tg}\alpha_i); \tag{5.9}$$

$$x_{i+1} = \cos\alpha_i (x_i - y_i \operatorname{tg}\alpha_i). \tag{5.10}$$

КОНТРОЛЬ ВЫПОЛНЕНИЯ ОПЕРАЦИИ

8.1. ОБЩИЕ ПОЛОЖЕНИЯ

Контроль правильности выполнения операций может быть осуществлен применением специальных арифметических кодов, идея построения которых базируется на свойствах сравнения по модулю.

Дело в том, что при рассмотрении различных арифметических выражений исходные числа, входящие в эти выражения, можно заменять на другие, сравнимые с ними по выбранному модулю p. В частности, каждое число может быть заменено своим вычетом. При этом все машинные числа считаются условно целыми.

Различают два метода получения контрольного кода: числовой и цифровой. При числовом методе контроля контрольный код заданного числа определяется как наименьший положительный остаток от деления числа A на выбранный модуль p:

$$r_A = A - \{A/p\}\, p, \tag{8.1}$$

где $\{A/p\}$ — целая часть от деления числа A на p.

Величина модуля p существенно влияет на качество контроля. Если при числовом контроле $p = q$, где q — основание системы счисления, в которой представлено число, то контролируется только младший разряд числа и контроль как таковой не имеет смысла. Для $p = q^m$ справедливы аналогичные соображения, так как опять не все разряды числа (при $m < n$) участвуют в контроле и ошибки в разрядах старше m вообще не воспринимаются.

Величина модуля p существенно влияет на качество контроля. Если при числовом контроле $p = q$, где q — основание системы счисления, в которой представлено число, то контролируется только младший разряд числа и контроль как таковой не имеет смысла. Для $p = q^m$ справедливы аналогичные соображения, так как опять не все разряды числа (при $m < n$) участвуют в контроле и ошибки в разрядах старше m вообще не воспринимаются.

Для числового метода контроля по mod p справедливы основные свойства сравнений. Поэтому, если

$$A \equiv r_A \pmod{p};$$

$$B \equiv r_B \pmod{p},$$

где $0 \leqslant r_A \leqslant p-1$; $0 \leqslant r_B \leqslant p-1$, то $A + B \equiv r_A + r_B \pmod{p}$, откуда

$$r_{A+B} \equiv r_A + r_B \pmod{P}. \tag{8.2}$$

Аналогичным образом доказывается справедливость и следующих соотношений:

$$r_{A-B} \equiv r_A - r_B \pmod{p}; \tag{8.3}$$

$$r_{AB} \equiv r_A r_B \pmod{p}. \tag{8.4}$$

Пример. Заданы $A = 125$; $B = 89$; $p = 11$. Найти $r_A \pmod{p}$ и $r_B \pmod{p}$.

$$r_A = 125 - [125/11] \cdot 11 = 4 \pmod{11};$$

$$r_B = 89 - [89/11] \cdot 11 = 1 \pmod{11}.$$

Недостатком числового метода контроля по mod p является использование операции деления для определения остатка, что требует больших затрат машинного времени.

При цифровом методе контроля контрольный код числа образуется делением суммы цифр числа на выбранный модуль при выполнении условий

$$r'_A = \sum_i a_i - \left\{\frac{\sum_i a_i}{p}\right\} \cdot p \qquad (8.5)$$

или

$$r'_A \equiv \sum_i a_i \pmod{p}.$$

2. ЛОГИЧЕСКИЕ ОСНОВЫ ЦВМ

Лекция 6. Основные понятия алгебры логики

Алгебра логики используется при анализе и синтезе схем ЭВМ по двум причинам. Во-первых, это объясняется соответствием представления переменных и функций алгебры логики. Во-вторых, двоичным представлением информации и характером работы отдельных компонентов вычислительной техники. Эти компоненты могут пропускать или не пропускать ток, иметь на выходе высокий или низкий уровень сигнала (напряжения или тока).

Приведем основные понятия алгебры логики.

Логическая переменная — это такая переменная, которая может принимать одно из двух значений: истинно или ложно (да или нет, единица или ноль).

Логическая константа — это такая постоянная величина, значением которой может быть истинно или ложно (да или нет, единица или ноль).

Логическая функция — это такая функция, которая может принимать одно из двух значений: истинно или ложно (да или нет, единица или ноль) в зависимости от текущих значений ее аргументов, в качестве которых используются логические переменные.

Логическая функция может быть одного ($n = 1$) или нескольких ($n > 2$) аргументов. Значение логической функции определяется комбинацией конкретных значений переменных, от которых она зависит. Комбинация конкретных значений переменных (аргументов функции) называется набором. Количество различных наборов N для «n» переменных вычисляется по формуле $N = 2^n$.

Зависимость логической функции от переменных может задаваться по–разному:

– словесным описанием;

– таблицей истинности;

– логическим выражением.

Словесное описание используется в случае сравнительно несложной логической функции.

Таблица истинности является универсальным средством задания логической функции. Она включает все наборы для заданного количества переменных, определяющих значение логической функции, с указанием значений, которые принимает функция для каждого набора. В одной таблице истинности может задаваться несколько логических функций, зависящих от одних и тех же переменных. Таблица истинности для нескольких функций y_i трех переменных x_1, x_2, x_3 может быть задана следующим образом (табл. 2.1)

Таблица 2.1

Таблица истинности трех переменных

№	x_1	x_2	x_3	y_1	y_2	y_3	…	y_n
0	0	0	0	0	1	1		0
1	0	0	1	1	1	0		1
2	0	1	0	1	1	1		0
3	0	1	1	0	1	0		–
4	1	0	0	1	0	1		0
5	1	0	1	0	0	0		1
6	1	1	0	0	0	1		–
7	1	1	1	1	1	0		1

В приведенной таблице истинности во второй, третей и четвертой колонках, помеченных соответственно x_1, x_2, x_3 , приведены все возможные наборы этих переменных. В следующих колонках приводятся значения функций y_1, y_2, y_n для каждого набора.

Логическая функция называется «*полностью определенной*», если для нее заданы значения по всем возможным наборам. Функция называется «*частично определенной*», если для некоторых наборов значения функции не заданы. В приведенной таблице истинности функции y_1, y_2, y_3 являются полностью определенными, а функция y_n – частично определенная (знак «–» означает неопределенность значения функции).

Максимальное количество полностью определенных функций от «n» переменных определяется как $M = (2^2)^n$.

Логическим выражением называется комбинация логических переменных и констант, связанных элементарными базовыми логическими функциями (или логическими операциями), которые могут разделяться скобками.

Например, логическую функцию y_1, определенную в вышеприведенной таблице истинности, можно представить в виде логического выражения

$y_1 = \overline{(x_1 \cdot x_2 + x_1 \cdot x_3 + x_2 \cdot x_3)} \cdot (x_1 + x_2 + x_3) + x_1 \cdot x_2 \cdot x_3$, где «+», «·», а также верхнее надчеркивание – знаки базовых логических функций.

Набор элементарных логических операций, с помощью которых можно задать любую, сколь угодно сложную логическую функцию, называется *функционально полной системой логических функций*. Иногда такую систему называют *базисом*.

В качестве элементарных логических функций функционально полных систем этих функций используются функции одной или двух логических переменных.

Все возможные функции одной переменой приведены в табл. 2.2.

Таблица 2.2

Функции одной переменной

x	y_0	y_1	y_2	y_3
0	0	0	1	1
1	0	1	0	1

Из таблицы видно, что:

$y_0 = 0$ – константа; y_1 равна значению переменной; y_2 равна значению, обратному значению переменной « x »; $y_3 = 1$ – константа.

С точки зрения базовых функций интерес представляет только функция y_2, она называется *функцией отрицания*, читается как «не x» и обозначается как « $\overline{x}$ », т. е. можно записать $y_2 = \overline{x}$.

Все возможные функции двух переменных приведены в табл. 2.3

Таблица 2.3

Функции двух переменных

№	x_1	x_2	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
2	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
3	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Информация по функциям двух переменных приведена в табл.2.4.

Таблица 2.4

Булевы выражения для функций двух переменных

y_i	Название функции	Чтение функции	Запись в виде булевого выражения
1	2	3	4
y_0	Const «0»		0
y_1	Конъюнкция	и x_1, и x_2	$x_1 \cdot x_2$; x_1x_2; x_1 & x_2
y_2	Запрет по x_2	неверно, что, если x_1, то x_2	$x_1\bar{x}_2$
y_3	$F(x_1)$	функция одной переменной	x_1
y_4	Запрет по x_1	неверно, что, если x_2 , то x_1	$\bar{x}_1x_2$
y_5	$F(x_2)$	функция одной переменной	x_2
y_6	Неравнозначности	x_1 не равно x_2	$\bar{x}_1x_2 + x_1\bar{x}_2$
y_7	Дизъюнкция	или x_1, или x_2	$x_1 + x_2$
y_8	Пирса	ни x_1, ни x_2	$\overline{x_1 + x_2}$
y_9	Равнозначности	x_1 равно x_2	$\bar{x}_1\bar{x}_2 + x_1x_2$
y_{10}	$F(x_2)$	функция одной переменной	$\bar{x}_2$

y_{11}	Импликация	если x_2 , то x_1	$\bar{x}_1\bar{x}_2 + x_1$
y_{12}	$F(x_1)$	функция одной переменной	$\bar{x}_1$
y_{13}	Импликация	если x_1, то x_2	$\bar{x}_1\bar{x}_2 + x_2$
y_{14}	Шеффера	неверно, что и x_1, и x_2	$\overline{x_1 x_2}$
y_{15}	Const (= 1)		1

Наиболее распространенной в алгебре логики является функционально полная система логических функций, которая в качестве базовых логических функций использует функцию одной переменной «НЕ» (функция отрицания) и две функции двух переменных – «И» (конъюнкция или логическое умножение) и «ИЛИ» (дизъюнкция или логическое сложение). Эта система получила название *система булевых функций*, или *булевый базис*. В алгебре логики имеется целый раздел *Алгебра Буля*, посвященный этому базису.

В вышеприведенной таблице, описывающей функции от двух переменных, в последней колонке приведены варианты записи этих функций в булевом базисе. Среди перечисленных формул наиболее известными являются так называемые «Стрелка Пирса» и «Штрих Шеффера». Они выступают как функционально полные системы и могут записываться в следующем виде:

$$y_8 = \overline{x_1 + x_2} = x_1 \bar{} \; x_2,$$
$$y_{14} = \overline{x_1 x_2} = x_1 | x_2 \quad .$$

Лекция 7. Основные понятия булевой алгебры

В алгебре Буля логические выражения включают логические операции И, ИЛИ, НЕ, которые могут быть использованы в самых различных сочетаниях. При оценке значения такого выражения необходимо решить его для конкретного набора переменных. В алгебре Буля применяется следующая приоритетность выполнения операций: сначала рассчитываются значения имеющих место отрицаний и скобок, затем выполняется операция И (логическое умножение); самый низший приоритет имеет операция ИЛИ (логическая сумма).

При работе с булевыми логическим выражениями используются следующие законы, правила и операции.

Переместительный (коммутативный) *закон*. Закон справедлив как для конъюнкции, так и для дизъюнкции.

$x_1 + x_2 + x_3 + x_4 = x_4 + x_3 + x_2 + x_1$ – от перемены мест логических слагаемых сумма не меняется.

$x_1x_2x_3x_4 = x_4x_3x_2x_1$ – от перемены мест логических сомножителей их произведение не меняется.

Этот закон справедлив для любого количества логических операндов.

Сочетательный (ассоциативный) *закон* справедлив как для конъюнкции, так и для дизъюнкции.

$x_1 + x_2 + x_3 + x_4 = (x_2 + x_3) + x_1 + x_4 = (x_1 + x_4) + (x_2 + x_3)$ – при логическом сложении отдельные слагаемые можно заменить их суммой.

$x_1\, x_2\, x_3\, x_4 = (x_2\, x_3)\, x_1x_4 = (x_1\, x_4)\, (x_2\, x_3)$ – при логическом умножении отдельные логические сомножители можно заменить их произведением.

Распределительный (дистрибутивный) *закон.*

$(x_1 + x_2)\, x_3 = x_1x_3 + x_2x_3$;
$(x_1 + x_2)\, (x_1 + x_3) = x_1 + x_2\, x_3$.

Правило де Моргана.

$\overline{x_1 + x_2} = \overline{x}_1\overline{x}_2$ – отрицание суммы равно произведению отрицаний;

$\overline{x_1x_2} = \overline{x}_1 + \overline{x}_2$ – отрицание произведения равно сумме отрицаний.

Операция склеивания.

$\overline{x}_i A + x_i A = A$ – операция *склеивания* для конъюнкций, где A – переменная или любое логическое выражение.

$(\overline{x}_i + A)(x_i + A) = A$ – операция склеивания для дизъюнкций.

Если в качестве A используется *простая конъюнкция*, т. е. конъюнкция, представляющая собой логическое произведение переменных и их отрицаний, то имеет место

$\overline{x}_1x_2\overline{x}_3\overline{x}_4x_5\overline{x}_6 + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4x_5\overline{x}_6 = \overline{x}_1\overline{x}_3\overline{x}_4x_5\overline{x}_6$

Как видно, в результирующем выражении количество переменных на единицу меньше, чем в склеенных конъюнкциях. Количество переменных в простой конъюнкции называется *рангом конъюнкции*, т.е. операция склеивания, примененная к простым конъюнкциям, дает результат с рангом, на единицу меньшим ранга исходных конъюнкций.

Операции с отрицаниями.

$\overline{\overline{x}} = x$ – двойное отрицание равносильно отсутствию отрицания;

$\overline{x} \cdot x = 0$;

$\overline{x} + x = 1$.

Операции с константами.

$x_1 + 1 = 1, x_1 + 0 = x_1,$

$x_1 \cdot 1 = x_1, x_1 \cdot 0 = 0.$

Операции с одинаковыми операндами.

$x_1 + x_1 + x_1 + x_1 + \ldots + x_1 = x_1;$

$x_1 x_1 x_1 \ldots x_1 = x_1$ при любом числе повторений.

Законы, правила и операции алгебры *Буля* могут быть доказаны путем логического рассуждения, однако такое доказательство применимо только для простейших случаев. Доказать справедливость того или иного правила можно, если с помощью различных преобразований привести правую часть правила к выражению в левой части (или наоборот). Универсальным приемом доказательства является использование таблицы истинности. Это основано на том утверждении, что два выражения (правая и левая часть правила или закона) *эквивалентны*, если они принимают одинаковые значения на всех наборах логических переменных.

Например, правило двойного отрицания, которое справедливо не только относительно одной переменной, но и любого логического выражения, можно доказать следующим рассуждением: если неверно утверждение, что выражение ложно, то очевидно утверждение, что это выражение истинно.

Доказать справедливость распределительного закона в интерпретации выражением $(x_1 + x_2)(x_1 + x_3) = x_1 + x_2x_3$ можно за счет приведения левой части к выражению правой части, раскрыв скобки:

$(x_1 + x_2)(x_1 + x_3) = x_1x_2 + x_1x_1 + x_1x_3 + x_2x_3 = x_1x_2 + x_1 + x_1x_3 + x_2x_3 = x_1(x_2 + 1 + x_3) + x_2x_3.$

Помня, что логическая сумма с одним слагаемым, равным константе «1», равна «1», можно записать $x_1 + x_2x_3$.

Используем таблицу истинности для доказательства правила де Моргана в варианте $\overline{x_1 + x_2} = \overline{x}_1 \overline{x}_2$ – отрицание суммы равно произведению отрицаний.

Составим таблицу истинности для правой и левой частей и составляющих их функций (табл. 2.5).

Таблица 2.5

Таблица истинности для правила де Моргана

x_1	x_2	$x_1 + x_2$	$\overline{x_1 + x_2}$	$\overline{x_1}$	$\overline{x_2}$	$\overline{x_1}\,\overline{x_2}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Из таблицы истинности видно, что правая и левая части доказываемого правила принимают одинаковые значения на всех наборах, следовательно они эквивалентны.

Функционально полной системой булевых функций (ФПСБФ) называется совокупность таких булевых функций (f_1, f_2, ..., f_k), посредством которых можно записать произвольную булеву функцию f. Как уже было сказано, ФПСБФ являются «Стрелка Пирса» и «Штрих Шеффера».

Лекция 8. Записи функций алгебры логики (ФАЛ) в различных формах, их взаимосвязь

Соверше́нная конъюнкти́вная норма́льная фо́рма (СКНФ) — это такая КНФ, которая удовлетворяет трём условиям:

- в ней нет одинаковых элементарных дизъюнкций
- в каждой дизъюнкции нет одинаковых пропозициональных переменных
- каждая элементарная дизъюнкция содержит каждую пропозициональную букву из входящих в данную КНФ пропозициональных букв.

Любая булева формула, не являющаяся тождественно истинной, может быть приведена к СКНФ.

Совершённая дизъюнктивная нормальная форма (СДНФ) — это такая ДНФ, которая удовлетворяет трём условиям:

- в ней нет одинаковых элементарных конъюнкций
- в каждой конъюнкции нет одинаковых пропозициональных букв
- каждая элементарная конъюнкция содержит каждую пропозициональную букву из входящих в данную ДНФ пропозициональных букв, причём в одинаковом порядке.

Любая булева формула, не являющаяся тождественно ложной, может быть приведена к СДНФ, причем единственным образом[1], то есть для любой выполнимой функции алгебры логики существует своя СДНФ, причём единственная.

Одну и ту же логическую функцию можно представить различными логическими выражениями. Среди множества выражений, которыми представляется логическая функция, особое место занимают две канонические формы: *совершенная конъюнктивная нормальная форма* (СКНФ) и *совершенная дизъюнктивная нормальная форма* (СДНФ).

Совершенная дизъюнктивная нормальная форма представляет собой дизъюнкцию простых конъюнкций, где под термином простая *конъюнкция* имеется в виду конъюнкция переменных или их отрицаний. В СДНФ простые конъюнкции содержат все переменные в своей прямой или инверсной форме и отражают собой наборы, на которых представляемая функция имеет единичное значение. Такие конъюнкции называются *конституентами единицы* рассматриваемой функции. Поэтому СДНФ представляет собой дизъюнкцию (логическую сумму), слагаемыми которой являются конституенты единицы. Общая запись СДНФ функции y имеет вид

$$y = \bigvee x_1^{\delta 1} x_2^{\delta 2} x_3^{\delta 3} \ldots x_{(n-1)}^{\delta(n-1)} x_n^{\delta n},$$

$$x_i^{\delta_i} = \begin{cases} x_i, \text{если } \delta_i = 1, \\ \bar{x}_i, \text{если } \delta_i = 0. \end{cases}$$

СДНФ легко сформировать на основе таблицы истинности. Например, если функции задаются в виде таблицы истинности 2.6, то СДНФ для них будет иметь следующий вид:

Таблица 2.6

Таблица истинности СДНФ

№	x_1	x_2	x_3	y_1	y_2	y_3
0	0	0	0	1	1	1
1	0	0	1	1	0	1
2	0	1	0	0	0	0
3	0	1	1	1	1	1
4	1	0	0	0	0	1
5	1	0	1	0	0	0
6	1	1	0	1	0	1
7	1	1	1	1	0	1

$$y_1 = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2x_3 + x_1x_2\bar{x}_3 + x_1x_2x_3;$$
$$y_2 = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2x_3;$$
$$y_3 = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2\bar{x}_3 + x_1x_2\bar{x}_3 + x_1x_2x_3.$$

Совершенная конъюнктивная нормальная форма — это конъюнкция простых дизъюнкций, где под термином простая дизъюнкция имеется в виду дизъюнкция переменных или их отрицаний. В СКНФ простые дизъюнкции содержат все переменные в своей прямой или инверсной форме и представляют собой отрицание конституент нуля. Общая запись СКНФ функции y имеет вид

$$y = \text{Ù}(x_1^{\delta 1} + x_2^{\delta 2} + x_3^{\delta 3} + \ldots + x_{(n-1)}^{\delta(n-1)} + x_n^{\delta n}),$$

$$x_i^{\delta_i} = \begin{cases} x_i, \text{если} & \delta_i = 1, \\ \bar{x}_i, \text{если} & \delta_i = 0. \end{cases}$$

СКНФ легко сформировать на основе таблицы истинности. Например, для функций, из предыдущей таблицы (табл. 2.6) имеем:

$$y_1 = (x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + x_2 + x_3)(\bar{x}_1 + x_2 + \bar{x}_3);$$

$$y_2 = (x_1 + x_2 + \bar{x}_3)(x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + x_2 + x_3)(\bar{x}_1 + x_2 + \bar{x}_3)\,'$$
$$'\ (\bar{x}_1 + \bar{x}_2 + x_3)(\bar{x}_1 + \bar{x}_2 + \bar{x}_3);$$

$$y_3 = (x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + x_2 + x_3).$$

СКНФ строится на основе конституент нуля. *Конституента нуля* представляет набор логических переменных, на котором логическая функция принимает значение «0».

Каждая скобка в приведенных выражениях представляет собой отрицание конституенты нуля соответствующей функции, а запись функции в виде конъюнкции таких скобок представляет собой условие, при котором отсутствуют все конституенты нуля определяемой функции, при выполнении которого функция имеет единичное значение.

Например, для y_1 выражение первой скобки представляет собой отрицание набора значений переменных второй строки, на котором функция y_1 имеет нулевое значение. Выражение второй скобки представляет собой отрицание набора значений переменных четвертой строки, на которой функция y_1 также имеет нулевое значение, выражение третьей скобки – отрицание набора значений переменных пятой строки, на котором функция y_1 имеет нулевое значение.

Из вышеизложенного видим, что любую функцию можно представить или в СДНФ, или в СКНФ, а т. к. эти формы представлены в базисе Буля, то отсюда следует, что этот базис (базис И, ИЛИ, НЕ) является *функционально полным*.

Если функция задана в СДНФ и требуется найти ее СКНФ, то такой переход можно выполнить, составив по заданной СДНФ таблицу истинности для этой функции. На основе полученной таблицы составляется СКНФ заданной функции.

Однако в некоторых случаях может оказаться более удобным подход, который поясняется следующим примером.

Пример:

По заданной СДНФ функции:

$$y_3 = x_1 x_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 x_2 x_3 + \bar{x}_1 \bar{x}_2 \bar{x}_3 + \bar{x}_1 x_2 x_3$$

найти запись этой функции в СКНФ.

Решение

Запишем логические выражение отрицания заданной функции, т.е. найдем логическое условие, при котором эта функция имеет нулевое значение. В качестве такого выражения можно взять дизъюнкцию конъюнкций, где каждая конъюнкция представляет собой конституен-

ту нуля заданной функции. Очевидно, что конституенты нуля – это те наборы, которые не являются наборами, соответствующими конституентам единицы, использованным в СДНФ. Таким образом, можно записать

$$\bar{y}_3 = \bar{x}_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 x_3;$$

Эту запись можно интерпретировать как словесное описание функции: функция *y* равна нулю, если имеет место хотя бы одна из конституент нуля.

В этой записи представлена дизъюнкция тех наборов, которые не использовались в записи функции y_3. Возьмем отрицание правой и левой частей полученного уравнения и применим к правой части правило де Моргана:

$$\bar{\bar{y}}_3 = \overline{\bar{x}_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 x_3} = \overline{\bar{x}_1 x_2 \bar{x}_3} \cdot \overline{x_1 \bar{x}_2 x_3}.$$

Применим *правило де Моргана* к отрицаниям конъюнкций, полученным в правой части:

$$\bar{\bar{y}}_3 = (x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + x_2 + \bar{x}_3).$$

Полученная запись для *y* является искомой СКНФ.

Лекция 9. Минимизация функций алгебры логики

Учитывая то, что одну и ту же логическую функцию можно представить различными выражениями, перед реализацией функции в виде логической схемой весьма важным является выбор из всех возможных выражений, соответствующих данной функции, самого простого. Решить эту проблему можно за счет использования процедуры минимизации логического выражения.

Методы минимизации:

- *минимизация* методом Квайна;
- *минимизация* с использованием диаграмм Вейча (или карт Карно);
- *минимизация* не полностью определенных (частично определенных) функций;
- *минимизация* конъюнктивных нормальных форм;
- *минимизация* методом кубического задания функций алгебры логики;

- *минимизация* методом Квайна–Мак-Класски;
- *минимизация* с использованием алгоритма извлечения (Рота);
- *минимизация* ФАЛ методом преобразования логических выражений.

Далее мы рассмотрим методы, употребляемые наиболее часто.

9.1. Минимизация методом Квайна

В качестве исходной формы представления логического выражения используется СДНФ. Если подлежащее минимизации выражение имеет другую форму, то приведение к СДНФ осуществляется за счет открытия скобок, избавления от отрицаний логических выражений, более сложных чем отрицание переменной (используется правило де Моргана).

Метод Квайна выполняется в два этапа.

Первый этап имеет своей целью получение тупиковой формы, представляющей собой дизъюнкцию, в качестве слагаемых которой используются конъюнкции (каждая из них не склеивается ни с одной другой конъюнкцией, входящей в это выражение). Такие конъюнкции называются *простыми импликантами*.

Данный этап выполняется за счет реализации отдельных шагов. На каждом шаге на основании выражения, полученного на предыдущем шаге, выполняются все возможные операции склеивания для пар имеющихся конъюнкций. Каждый шаг понижает ранг исходных конъюнкций на единицу. Шаги повторяются до получения тупиковой формы.

Второй этап имеет своей целью устранение из тупиковой формы всех избыточных простых импликант, что дает в результате минимальное логическое выражение.

Пример:

Найти методом Квайна минимальное выражение для функции *y*:

$$y = \bar{x}_1\bar{x}_2x_3x_4 + x_1x_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_2\bar{x}_3x_4 + \bar{x}_1x_2x_3\bar{x}_4 + \bar{x}_1x_2\bar{x}_3\bar{x}_4 + x_1x_2\bar{x}_3x_4 +$$
$$+ x_1\bar{x}_2x_3\bar{x}_4 + x_1x_2x_3\bar{x}_4 + x_1\bar{x}_2x_3x_4 + \bar{x}_1\bar{x}_2x_3\bar{x}_4.$$

Решение

1-й этап:

$$y = \overset{1}{\bar{x}_1}\bar{x}_2 x_3 x_4 + x_1 x_2 \overset{2}{\bar{x}_3}\bar{x}_4 + \bar{x}_1 x_2 \overset{3}{\bar{x}_3} x_4 + \bar{x}_1 x_2 x_3 \overset{4}{\bar{x}_4} + \bar{x}_1 x_2 \overset{5}{\bar{x}_3}\bar{x}_4 + x_1 x_2 \overset{6}{\bar{x}_3} x_4 +$$

$$+ x_1 \overset{7}{\bar{x}_2} x_3 \bar{x}_4 + x_1 x_2 \overset{8}{x_3}\bar{x}_4 + x_1 \overset{9}{\bar{x}_2} x_3 x_4 + \overset{10}{\bar{x}_1}\bar{x}_2 x_3 \bar{x}_4 = \underset{1\text{-}9}{\overset{1}{\bar{x}_2} x_3 x_4} + \underset{1\text{-}10}{\overset{2}{\bar{x}_1}\bar{x}_2 x_3} + \underset{2\text{-}5}{x_2 \overset{3}{\bar{x}_3}\bar{x}_4} +$$

$$\underset{2\text{-}8}{\overset{4}{x_1 x_2 x_4}} + \underset{3\text{-}5}{\overset{5}{\bar{x}_1 x_2 \bar{x}_3}} + \underset{3\text{-}6}{\overset{6}{x_2 \bar{x}_3 x_4}} + \underset{4\text{-}5}{\overset{7}{\bar{x}_1 x_2 \bar{x}_4}} + \underset{4\text{-}8}{\overset{8}{x_2 x_3 \bar{x}_4}} + \underset{4\text{-}10}{\overset{9}{\bar{x}_1 x_3 \bar{x}_4}} + \underset{7\text{-}8}{\overset{10}{x_1 x_3 \bar{x}_4}} + \underset{7\text{-}9}{\overset{11}{x_1 \bar{x}_2 x_3}} + \underset{7\text{-}10}{\overset{12}{\bar{x}_2 x_3 \bar{x}_4}} =$$

$$= \underset{1\text{-}12}{\overset{1}{\bar{x}_2 x_3}} + \underset{2\text{-}11}{\overset{2}{\bar{x}_2 x_3}} + \underset{3\text{-}6}{\overset{3}{x_2 \bar{x}_3}} + \underset{3\text{-}8}{\overset{4}{x_2 \bar{x}_4}} + \underset{4\text{-}7}{\overset{5}{x_2 \bar{x}_4}} + \underset{8\text{-}12}{\overset{6}{x_3 \bar{x}_4}} + \underset{9\text{-}10}{\overset{7}{x_3 \bar{x}_4}} + \underset{5}{\overset{8}{\bar{x}_1 x_2 \bar{x}_3}} = \overset{1}{\bar{x}_2 x_3} + \overset{2}{x_2 \bar{x}_3} + \overset{3}{\bar{x}_2 x_4} +$$

$$+ \overset{4}{x_3 \bar{x}_4} + \overset{5}{\bar{x}_1 x_2 \bar{x}_3} = y \text{ - } \textit{тупиковая форма.}$$

Над конъюнкциями проставлены их номера; в скобках под каждой конъюнкцией (*i*–*j*) указывают, что данная конъюнкция является результатом склеивания *i*-й и *j*-й конъюнкций исходного выражения.

К результатам склеивания логически добавлен ни с чем не склеенный пятый член исходного выражения; несколько одинаковых конъюнкций представляются одной конъюнкцией.

Последнее выражение получено из предыдущего посредством удаления повторяющихся членов.

2-й этап:

На основании исходного выражения и полученной тупиковой формы составляется и заполняется импликантная таблица (табл.2.7).

Таблица 2.7

Импликантная таблица

	$\bar{x}_1\bar{x}_2x_3x_4$	$\bar{x}_1x_2\bar{x}_3x_4$	$\bar{x}_1x_2x_3\bar{x}_4$	$x_1\bar{x}_2\bar{x}_3x_4$	$x_1\bar{x}_2x_3\bar{x}_4$	$x_1x_2\bar{x}_3\bar{x}_4$	$x_1\bar{x}_2x_3x_4$	$x_1x_2\bar{x}_3x_4$	$x_1x_2x_3\bar{x}_4$	$x_1x_2x_3x_4$
	1	2	3	4	5	6	7	8	9	10
$\bar{x}_2x_3$	*						*		*	*
$\bar{x}_2x_3$		*	*		*	*				
$\bar{x}_2x_4$		*		*	*			*		
$\bar{x}_3x_4$				*			*	*		*
$\bar{x}_1\bar{x}_2x_3$			*		*					

Колонки приведенной таблицы помечены конституентами единицы, имеющимися в исходном логическом выражении.

Строки таблицы помечены простыми импликантами полученной тупиковой формы.

Звездочками в каждой строке отмечены те конституенты единицы, которые покрываются соответствующей простой импликантой (практически отмечаются те конституенты единицы, которые включают простую импликанту как свою составную часть).

Анализируя покрытия простыми импликантами конституент единицы заданной функции, составляем ее минимальное выражение:

$$y_{\min} = \overset{1}{\bar{x}_2}x_3 + \overset{2}{x_2}\bar{x}_3 + \overset{3}{x_2}\bar{x}_4 .$$

Минимальное выражение $y_{\min}$ формируется за счет последовательного включения простых импликант. При этом используется следующая приоритетность включения импликант в формируемое минимальное выражение:

- простая импликанта является единственной, покрывающей одну из колонок;

- если импликант вышеуказанного типа нет, то выбирается импликанта, покрывающая большее количество еще не покрытых колонок.

Последовательность включения простых импликант в приведенное минимальное выражение:

$\bar{x}_2 x_3$ – единственная импликанта, покрывающая колонку 1, при этом из дальнейшего рассмотрения исключаются все колонки, покрываемые этой импликантой, т. е. колонки 1, 7, 9, 10;

$x_2 \bar{x}_3$ – покрывает максимальное число колонок, оставшихся для рассмотрения (колонки 2, 3, 5, 6) эти колонки из дальнейшего рассмотрения исключаются;

$x_2 \bar{x}_4$ – покрывает оставшиеся две колонки (4, 8); после выбрасывания этих двух колонок для рассмотрения не останется ни одной колонки, не покрытой уже включенными в формируемое выражение простыми импликантами. Поэтому простые импликанты $x_3 \bar{x}_4$ и $\bar{x}_1 x_2 \bar{x}_3$ найденной тупиковой формы являются избыточными и в минимальном логическом выражении для заданной функции не присутствуют.

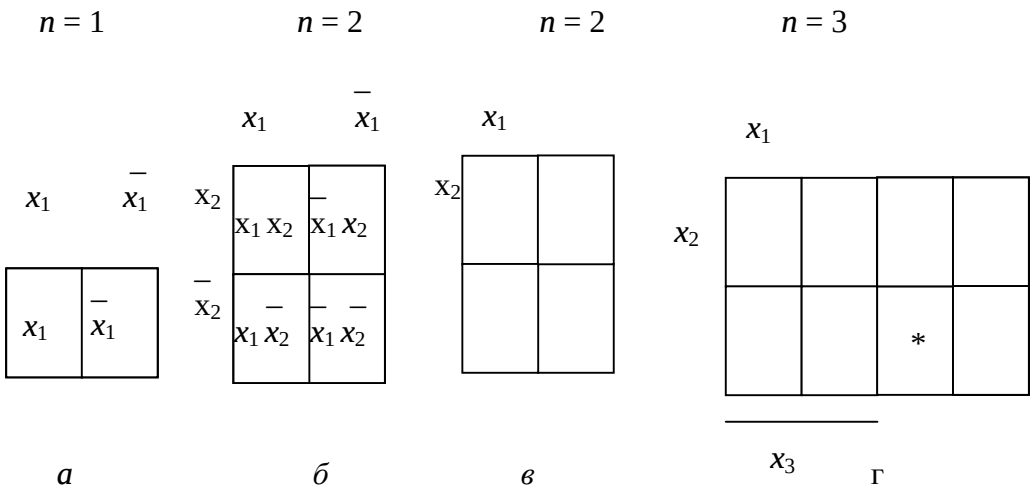
9.2. Минимизация с диаграммами Вейча

Минимизация этим методом предполагает использование специальных форм – диаграмм Вейча (или карт *Карно*).

Карта Карно для «*n*» логических переменных представляет собой множество квадратов (клеток), объединенных в близкую к квадрату прямоугольную форму. Каждая такая клетка соответствует одному набору логических переменных, причем наборы двух соседних клеток должны отличаться на значение одной переменной (их наборы образуют склеивающиеся конъюнкции).

На рис. 2.1 приведены карты Карно для n = 1, 2, 3. На рис.2.1, *а*, 2.1,*б* показана разметка колонок и строк, а также указан для каждой составляющей клетки соответствующие ей набор. Разметка колонок (строк) указывает, какие значения данная переменная имеет в клетках, находящихся в данной колонке (строке). На рис. 2.1,*в* приведен пример компактной разметки карты, соответствующей карте на рис.2.1, *б*. Здесь помечаются колонки (строки), в которых соответствующая переменная имеет прямое значение. На рис. 2.1, *г* приведена карта Карно для n = 3, сформированная посредством зеркального отображения карты Карно для n = 2 (рис. 2.1, *в*) относительно правой границы. Этот прием универсальный; его можно использовать для построения карты для заданного «*n*» на основании имеющейся карты Карно для «n - 1» переменной. Клетка, отмеченная знаком «*», соответствует набору $\bar{x}1\bar{x}2\bar{x}3$.

Карты Карно используются для представления и минимизации логических функций.



Записываемая функция должна быть представлена в СДНФ.

Рис. 2.1. Карта Карно: *а* – для одной переменной, *б* – для двух переменных, *в*)– сокращенная для 2-х переменных, *г* – для 3-хпеременных

Запись функции в карту осуществляется за счет установки «1» в клетки карты, соответствующие конституентам единиц записываемой функции.

Для выполнения минимизации представленной в карте Карно функции необходимо выполнить два этапа:

- охватить множество клеток карты Карно контурами;
- записать минимальное выражение для заданной функции в виде дизъюнкции конъюнкций, где каждая конъюнкция соответствует одному из введенных на карте контуров.

Охват клеток карты контурами выполняется с соблюдением следующих правил:

- контур должен иметь прямоугольную форму;
- в контур может входить количество клеток, равное целой степени числа «2»;
- в контур могут входить клетки, являющиеся логическими соседями;

- в контур необходимо включить максимальное количество клеток с учетом вышеприведенных требований;
- контурами необходимо охватить все клетки с единичными значениями;
- контуров должно быть минимальное количество;
- количество клеток в контуре должно быть равно 2^{DR}, где DR – разность ранга (дельта ранга) конституент единицы заданной функции и ранга конъюнкции, соответствующей контуру.

Логическими соседями являются такие две клетки, наборы которых отличаются только одной переменной – в одном эта переменная должна иметь прямое, в другом – обратное значение.

Для того чтобы быть логическими соседями, клеткам достаточно быть геометрическими соседями. Считая, что карта является пространственным объектом и заворачивается по горизонтали и вертикали, сливаясь своими крайними горизонтальными и крайними вертикальными границами, можно считать, что соответствующие крайние горизонтальные и вертикальные клетки являются геометрическими соседями. Логическими соседями могут быть клетки, которые не являются геометрическими соседями. К числу таких клеток относятся клетки, которые по горизонтали или вертикали симметричны относительно линий зеркального отображения, которые были использованы при переходе от «n» к «n+1» переменным.

Запись минимального выражения по заданной функции имеет вид дизъюнкции простых конъюнкций, соответствующих контурам на карте, и формируется следующим образом:

- конъюнкция, соответствующая контуру, должна включать только те переменные, которые имеют постоянное значение во всех клетках, охваченных рассматриваемым контуром,
- или по другому: в конъюнкцию, соответствующую контуру, не должны входить переменные, которые имеют разные значения для клеток, охваченных рассматриваемым контуром.

Например, если задана логическая функция «y» трех переменных в виде выражения

$$y = x_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 x_3 + x_1 x_2 x_3,$$

то её запись в карту Карно будет иметь вид, приведенный на рис.2.2.

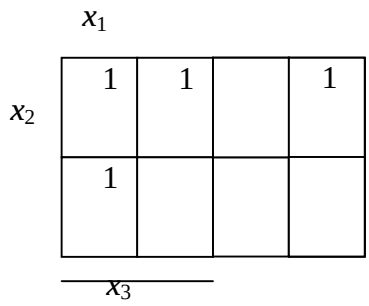


Рис. 2.2. Карта Карно для 3-хпеременных

Для функции, заданной в карте Карно, приведенной на рис.2.2, контуры имеют вид, приведенный на рис.2.3.

Для примера, контур 1 представлен на рисунке в виде двух клеток: клетки, соответствующей набору $x_1x_2x_3$, и клетки, соответствующей набору $x1x2\bar{x}3$, поэтому данному контуру будет соответствовать конъюнкция $x_1\ x_2$.

Минимальное логическое выражение для функции имеет вид:

$$y = x_1\ x_2 + x_1\ x_3 + x_2\ x_3.$$

1 2 3

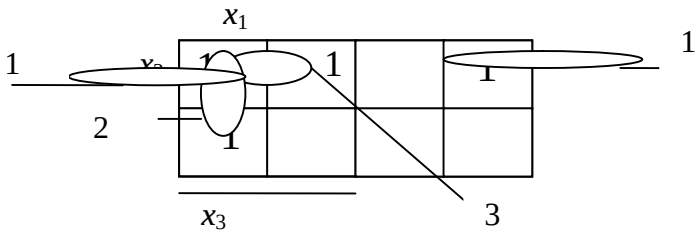


Рис. 2.2. Сформированные контуры для заданной функции

Конъюнкции минимального выражения помечены внизу цифрами, соответствующими номерам контуров, которые они представляют.

Лекция 10. Синтез логических схем по логическим выражениям

10.1. Синтез логических схем в базисе И, ИЛИ, НЕ

Логические схемы строятся на основе логических элементов, набор которых определяется заданным логическим базисом.

Для базиса Буля в качестве логических элементов используются элементы, реализующие базовые логические функции И, ИЛИ, НЕ, которые имеют приведенные на рис. 2.4 обозначения.

При синтезе схемы по логическому выражению, составляющие логические операции представляются в виде соответствующих логических элементов, связи между которыми определяются последовательностью выполнения логических операций в заданном выражении. определяется заданным логическим базисом.

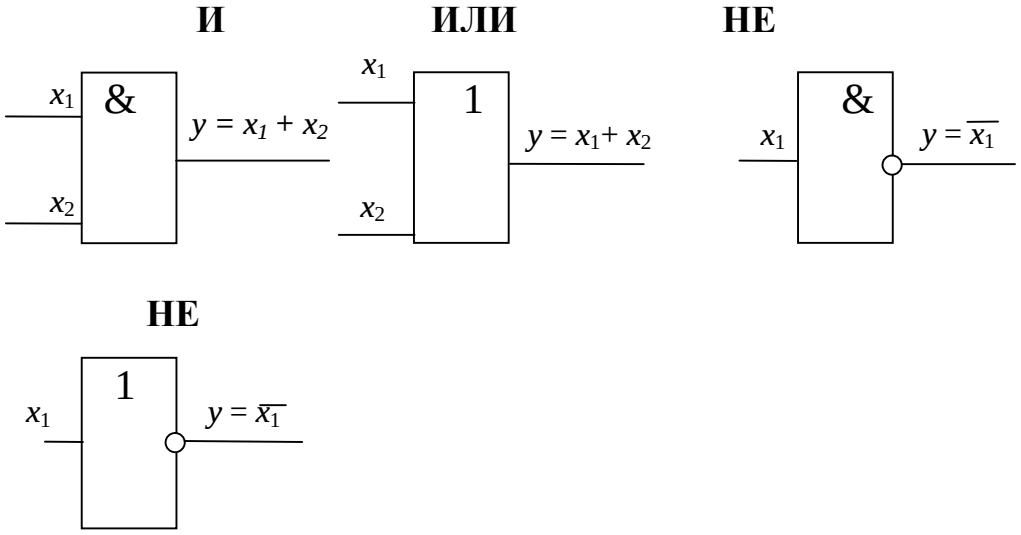


Рис. 2.3. Базовые логические элементы

Пример

Синтезировать логическую схему в базисе И, ИЛИ, НЕ, реализующую логическое выражение

$$y_1 = \overline{(x_1x_2 + x_1x_3 + x_2x_3)}(x_1 + x_2 + x_3) + x_1x_2x_3.$$

Решение

Входными сигналами синтезируемой схемы являются x_1, x_2, x_3, а выходным – y_1.

Реализацию заданного выражения в виде логической схемы можно начать или с последней операции, или с первой.

Последней операцией в заданном выражении является операция логического сложения двух операндов

$$\overline{(x_1x_2 + x_1x_3 + x_2 x_3)}(x_1 + x_2 + x_3) \text{ и } x_1 x_2 x_3,$$

поэтому для её реализации требуется элемент ИЛИ(1) с двумя входами, на выходе которого будет сформирован сигнал, соответствующий y_1, если на его входы будут поданы эти два слагаемые (например, первое слагаемое на второй вход, а второе слагаемое на первый вход).

На первый вход выходного элемента ИЛИ подается логическое произведение $x_1\ x_2\ x_3$, для реализации которого необходимо использовать логический элемент И с тремя входами, на которые подаются входные переменные x_1, x_2, x_3. Аналогичным образом рассматривается последовательность формирования выражения

$$\overline{(x_1x_2 + x_1x_3 + x_2x_3)}(x_1 + x_2 + x_3),$$

которое соответствует сигналу, подаваемому на второй вход элемента ИЛИ (1). В результате синтезируется схема для заданного выражения, приведенная на рис. 2.4.

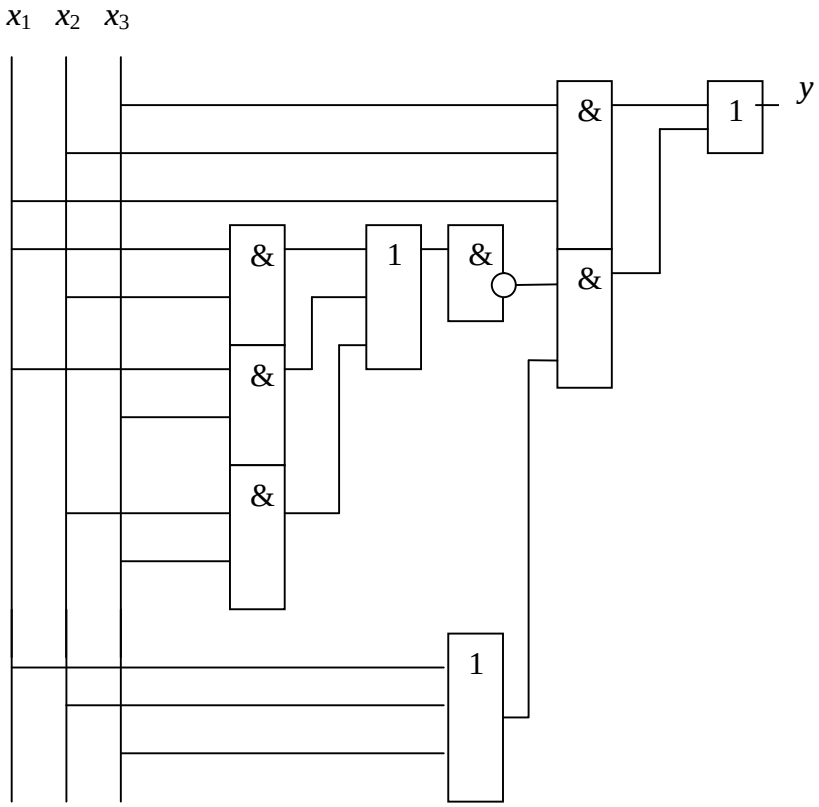


Рис. 2.4. Логическая схема для заданного выражения

10.2. Логические базисы И–НЕ, ИЛИ–НЕ

Булевый базис не является единственной функционально полной системой логических функций. Среди других наибольшее распространение получили базис И–НЕ и базис ИЛИ–НЕ.

Чтобы доказать логическую полноту любого базиса, достаточно показать, что в этом базисе можно реализовать базовые функции И, ИЛИ, НЕ.

Для базиса И-НЕ в качестве базового элемента используется элемент приведенный на рисунке рис. 2.5,а.

а *б*

Рис. 2.5. Базовые элементы: *а* – И–НЕ; *б* – ИЛИ–НЕ

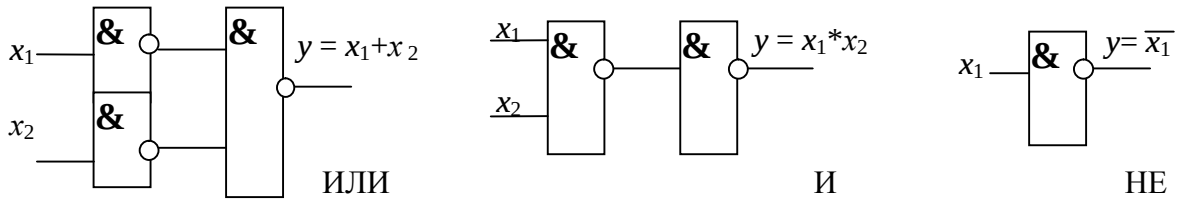
Реализация с помощью функции И-НЕ базовых функций алгебры Буля осуществляется следующим образом.

ИЛИ: $x_1 + x_2 = \overline{\overline{x_1 + x_2}} = \overline{x_1 x_2}$;

И: $x_1 \times x_2 = \overline{\overline{x_1 \times x_2}}$.

Функция НЕ реализуется с помощью схемы И-НЕ с одним входом.

На рис. 2.6. приведена схемная реализация функций И, ИЛИ, НЕ в базисе И-НЕ.



а *б* *в*

Рис. 2.6. Реализация булевых функций в базисе И–НЕ: *а* – функция ИЛИ; *б* – функция И; *в* – функция НЕ

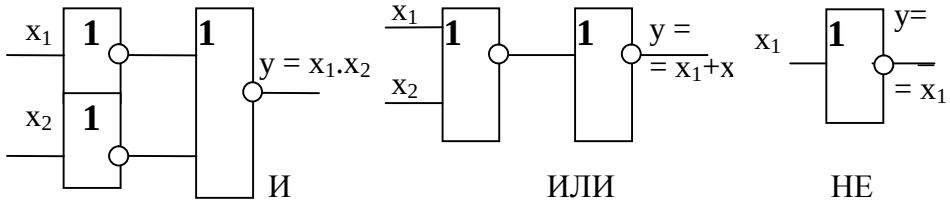
Реализация с помощью логической функции ИЛИ-НЕ базовых функций алгебры Буля осуществляется следующим образом.

ИЛИ: $x_1 + x_2 = \overline{\overline{x_1 + x_2}}$;

И: $x_1 \times x_2 = \overline{\overline{x_1 \times x_2}} = \overline{\overline{x_1} + \overline{x_2}}$

Функция НЕ реализуется с помощью схемы ИЛИ-НЕ с одним входом.

На рис.2.7. приведена схемная реализация операций И, ИЛИ, НЕ в базисе ИЛИ-НЕ



а *б* *в*

Рис. 2.7. Реализация булевых функций в базисе ИЛИ–НЕ: *а* – функция И; *б* – функция ИЛИ; *в* – функция НЕ

10.3.Синтез логических схем в базисах И–НЕ, ИЛИ–НЕ

При синтезе логических схем в заданном базисе логических элементов (например, в базисах И–НЕ, или ИЛИ–НЕ) целесообразно предварительно исходное выражение привести к форме, в которой в выражении будут использованы только логические операции, соответствующие используемым логическим элементам в заданном базисе.

Пример

Синтезировать логическую схему в базисе И–НЕ, соответствующую выражению

$$y = \overline{(x1x2 + x1x3 + x2x3)}(x1 + x2 + x3) + x1x2x3 .$$

Решение

Используя правило де Моргана преобразуем исходное выражение таким образом, чтобы последней операцией было отрицание и в выражение были бы только операции И.

$$\overline{(x1x2 + x1x3 + x2x3)}(x1 + x3 + x3) + x1x2x3 =$$

$$= \overline{\overline{(x1x2 + x1x3 + x2x3)}(x1 + x2 + x3)} + x1x2x3 =$$

$$= \overline{\overline{(x1x2 + x1x3 + x2x3)}(x1 + x2 + x3)}\,\overline{x1x2x3} =$$

$$= \overline{\overline{x1}\,\overline{x2}x1\overline{x3}x2\overline{x3}}\,\overline{(x1 + x2 + x3)}\,\overline{x1x2x3} =$$

$$= \overline{\overline{x1}\,\overline{x2}x1\overline{x3}x2\overline{x3}x1\overline{x2}\overline{x3}}\,\overline{x1x2x3}.$$

Полученное выражение, представленное в виде вложенных операции И-НЕ, позволяет синтезировать соответствующую логическую схему в заданном базисе, которая приведена на рис.2.8.

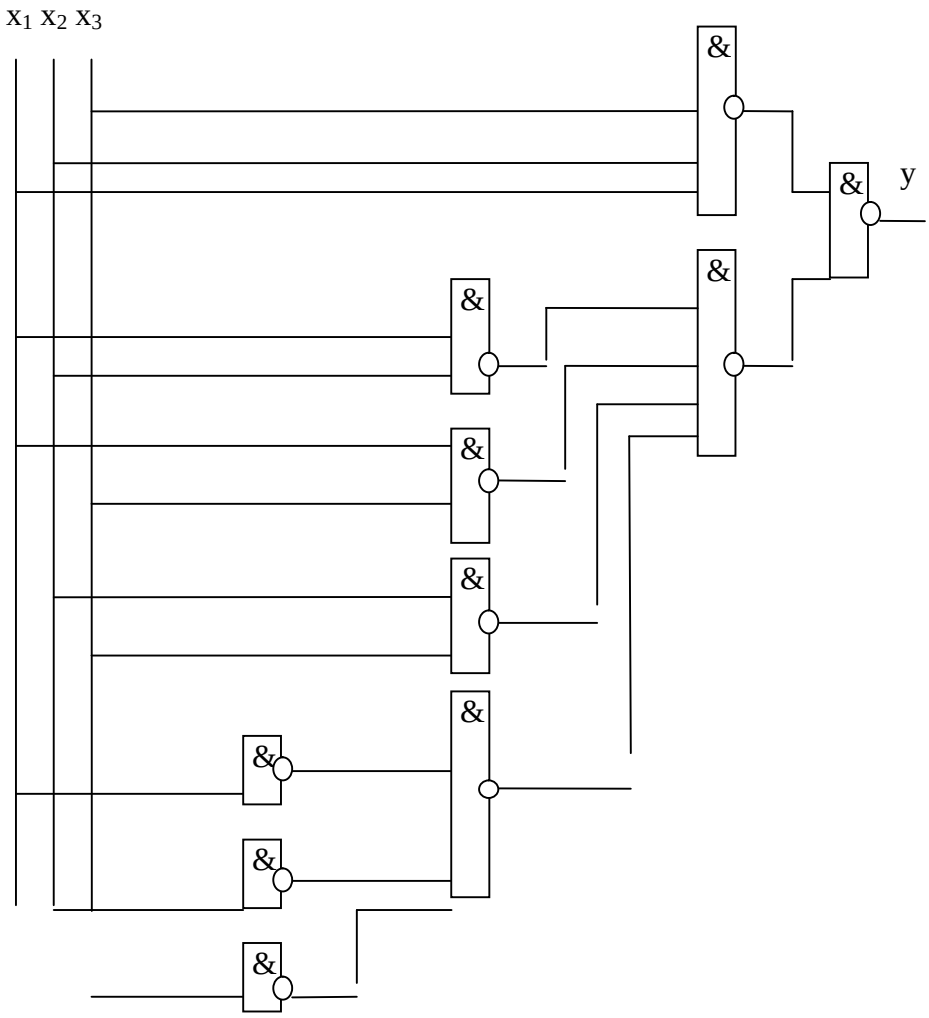


Рис. 2.8. Реализация логического выражения в базисе И–НЕ

Пример

Синтезировать логическую схему в *базисе* ИЛИ-НЕ, соответствующую выражению

$$y = \overline{(x1x2 + x1x3 + x2x3)}(x1 + x2 + x3) + x1x2x3$$

Решение

Используя правило де Моргана, преобразуем исходное выражение таким образом, чтобы последней операцией было отрицание и в выражение были бы только операции ИЛИ.

$$y = \overline{\overline{\overline{\overline{(x1x2 + x1x3 + x2x3)}(x1 + x2 + x3)}} + \overline{\overline{x1x2x3}}} =$$

$$= \overline{\overline{(x1x2 + x1x3 + x2x3)}(x1 + x2 + x3)} + \overline{\overline{x1x2x3}} =$$

$$= (x1x2 + x1x3 + x2x3) + \overline{(x1 + x2 + x3)} + \overline{\overline{x1} + \overline{x2} + \overline{x3}} =$$

$$= (\overline{\overline{x1x2}} + \overline{\overline{x1x3}} + \overline{\overline{x2x3}}) + \overline{(x1 + x2 + x3)} + \overline{\overline{x1} + \overline{x2} + \overline{x3}} =$$

$$= \overline{\overline{x1} + \overline{x2}} + \overline{\overline{x1} + \overline{x3}} + \overline{\overline{x2} + \overline{x3}} + \overline{\overline{x1} + \overline{x2} + \overline{x3}} + \overline{\overline{x1} + \overline{x2} + \overline{x3}}.$$

Полученное выражение, представленное в виде вложенных операций ИЛИ-НЕ, позволяет легко синтезировать соответствующую логическую схему в заданном базисе, которая приведена на рисунке рис. 2.9.

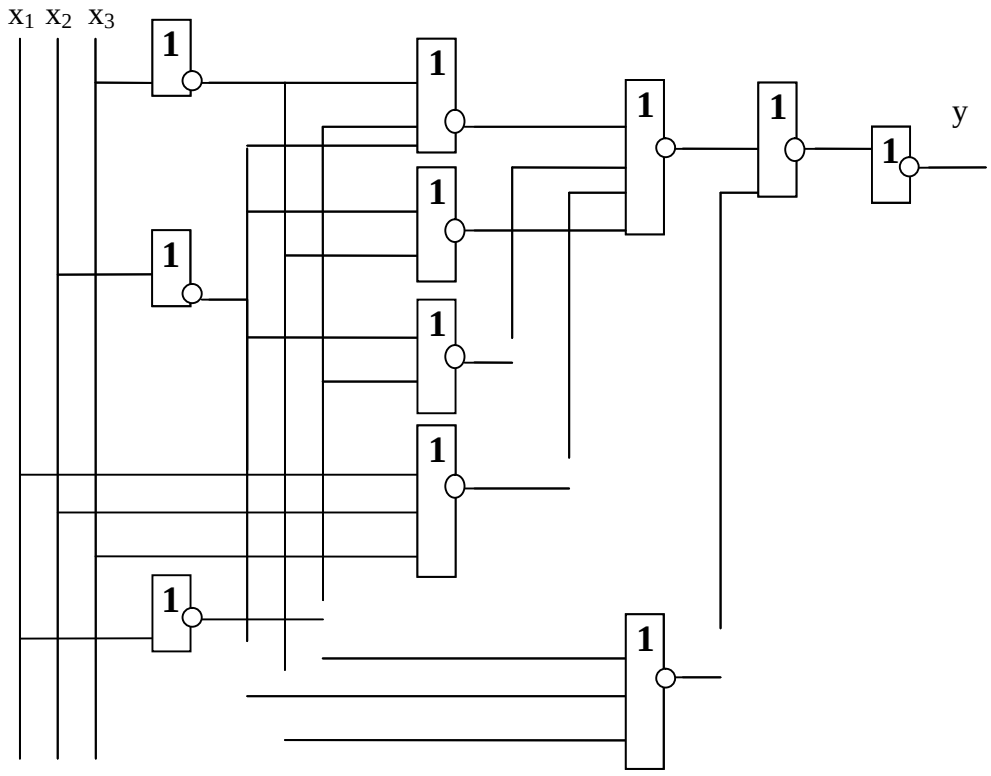


Рис. 2.9. Реализация логического выражения в базисе ИЛИ–НЕ

ЛИТЕРАТУРА

1. Кобайло, А. С. Логические основы цифровых вычислительных машин / А.С. Кобайло, А.Т. Пешков. – Минск: БГТУ, 2010. – 95 с.
2. Лысиков, Б. Г. Арифметические и логические основы цифровых автоматов / Б. Г. Лысиков. – Минск: Высшая школа, 1980. – 268 с.
3. Савельев, А. Я. Прикладная теория цифровых автоматов: учебник для вузов по специальности ЭВМ / А. Я. Савельев. – М.: Высшая школа, 1987. – 462 с.
4. Миллер, Р. Теория переключательных схем. Т. 1. / Р. Миллер, – М.: Наука, 1970. – 534 с.
5. Баранов, С. И. Синтез микропрограммных автоматов / С. И. Баранов.. – Л.: Энергия, 1979. – 271 с.
6. Скляров В. А. Синтез автоматов на матричных БИС / В. А. Скляров. – Минск: Наука и техника, 1984. – 288 с.
7. Морозевич, А. Н. МикроЭВМ, микропроцессоры и основы микропрограммирования / А. Н. Морозевич, А. Н. Дмитриев [и др.]. – Минск: Высшая школа, 1990. – 178 с.
8. Баранов, С. А. Цифровые устройства на программируемых БИС с матричной структурой / С. А. Баранов, В. А. Скляров. – М.: Радио и связь, 1986. – 272 с.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ…… 3
1. АРИФМЕТИЧЕСКИЕ ОСНОВЫ ЦИФРОВЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН…… 4
Лекция1. Системы счисления 4
1.1. Понятие системы счисления 4
1.2. Перевод чисел из одной системы счисления в другую 7
Лекция 2. Двоичная арифметика…….16
2.1. Операция сложения в двоичной системе счисления 16
2.2. Операция вычитания…… 18
2. 3. Операция умножения……18
2.4. Деление двоичных чисел……..22
2.5. Арифметика с положительными двоично-десятичными числами……23
Лекция 3. Арифметика с алгебраическими числами……25
3.1. Кодирование алгебраических чисел……25
3.2. Дополнительный и обратный коды двоичных чисел…27
3.3. Операции с двоичными числами в дополнительном коде……...29
3.4. Операции с двоичными числами в обратном коде…...30
3.5. Модифицированные коды……32
Лекция 4. Логические операции с двоичными кодами……35
4.1. Логические операции……35
4.2. Логические сдвиги……36
4.3. Арифметические сдвиги……...37
Лекция 5. Представление чисел в ЭВМ……42
5.1. Представление чисел с фиксированной точкой…….42
5.2. Арифметические операции над числами, представленными с фиксированной точкой……..43

5.3. Представление чисел с плавающей точкой 43
5.4. Арифметика с плавающей точкой……..45
2. ЛОГИЧЕСКИЕ ОСНОВЫ ЦИФРОВЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН..... 52
Лекция 6. Основные понятия алгебры логики 52
Лекция 7. Основные понятия булевой алгебры 56
Лекция 8. Записи функций алгебры логики (ФАЛ) в различных формах, их взаимосвязь..... 59
Лекция 9. Минимизация функций алгебры логики 62
9.1. Минимизация методом Квайна..... 63
9.2. Минимизация с диаграммами Вейча..... 66
Лекция 10. Синтез логических схем по логическим выражениям……..68
10.1. Синтез логических схем в базисе И, ИЛИ, НЕ…….68
10.2. Логические базисы И–НЕ, ИЛИ–НЕ…….70
10.3. Синтез логических схем в базисах И–НЕ, ИЛИ–НЕ…...72
ЛИТЕРАТУРА…….. 76

Учебное издание

Кобайло Александр Серафимович

АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОСНОВЫ ЦИФРОВЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Конспект лекций

Редактор
Компьютерная верстка

Подписано в печать _____. Формат 60•84$^{1}/_{16}$.
Усл. печ. л. 4. Уч.–изд. л.-4.
Тираж 70 экз. Заказ .
Отпечатано в Центре издательско–полиграфичных и информационных технологий учреждения образование «Белорусский государственный технологический университет».
220006. Минск, Свердлова, 13а.
ЛИ №02330/0549428 от 08.04.2009.
ЛП №,0330/0150477 от 16.01.2009.