Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Ю.А. Луцик, И.В. Лукьянова

АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

УЧЕБНОЕ ПОСОБИЕ
для студентов специальности
”Вычислительные машины, системы и сети”
всех форм обучения

Минск 2004

УДК 681.322 (075.8)
ББК 32.97 я 73
Л 86

Р е ц е н з е н т:
заведующий кафедрой математики и информатики ЕГУ,
кандидат технических наук В.И. Романов

Луцик Ю.А.

Л 86 Арифметические и логические основы вычислительной техники: Учеб. пособие для студ. спец. «Вычислительные машины, системы и сети» всех форм обуч. / Ю.А. Луцик, И.В. Лукьянова. − Мн.: БГУИР, 2004. − 121 с.: ил.

ISBN 985-444-595-X

Учебное пособие посвящено описанию способов представления числовой информации в ЭВМ, методов выполнения арифметических и логических операций в вычислительных машинах. Рассмотрены вопросы, связанные со способами контроля правильности функционирования вычислительного устройства и методами оптимизации устройств, выполняющих арифметические операции.

Пособие может быть использовано студентами всех форм обучения, магистрантами и аспирантами специальности 40 02 01 ”Вычислительные машины, системы и сети”.

УДК 681.322 (075.8)
ББК 32.97 я 73

ISBN 985-444-595-X

Содержание

Введение

Основная цель настоящего учебного пособия - помочь студенту, приступившему к изучению арифметики вычислительных машин, приобрести теоретические знания и практические навыки выполнения основных арифметических операций. Правильное понимание алгоритмов рассматриваемых операций подкрепляется знанием структурных и логических схем, реализующих эти алгоритмы и представляющих собой некоторые операционные устройства. В пособии уделяется внимание рассмотрению этих схемных решений. Достаточно подробно рассмотрен аппарат, основанный на правилах и законах булевой алгебры, ориентированный на упрощение (минимизацию) проектируемых логических схем. Кроме того, в пособии приводятся сведения об основных формах хранения и преобразования числовой информации, способах ее кодирования. Достаточное внимание уделено методам контроля правильности функционирования цифрового автомата, возможным ошибкам, возникающим при его работе, и способам их устранения.

Рассматриваемый в пособии теоретический материал сопровождается достаточным количеством примеров, что упрощает и делает более понятным излагаемый материал.

В заключение следует отметить, что в течение ряда лет литература, освещающая арифметику вычислительных машин, не выпускалась. В пособии сделана попытка устранить этот информационный пробел. Материал пособия базируется на работах [1-5].

Арифметические основы вычислительной техники

Системы счисления

В ЭВМ информация всегда представляется в виде чисел, записанных в той или иной системе счисления. Выбор системы счисления - один из важнейших вопросов. От правильности его решения зависят такие характеристики ЭВМ, как скорость вычислений, сложность алгоритмов реализации арифметических операций и др. Система счисления - совокупность цифр, приемов и правил для записи чисел цифровыми знаками.

Любая система счисления должна обеспечивать:

- возможность представления любого числа в рассматриваемом диапазоне величин;
- единственность этого представления;
- простоту оперирования числами.

Различают два типа систем счисления - непозиционные и позиционные.

Непозиционная система счисления - система, для которой значение символа не зависит от его положения в числе. Примером может служить система счисления с одной цифрой 1. Для записи любого числа в ней необходимо написать количество единиц, равное числу. Другой пример - это римская система счисления.

Позиционной системой счисления называется система записи любых по величине чисел, в которой значение цифры зависит от ее положения в числе, т.е. *веса*. Число цифр в позиционной системе счисления ограниченно.

Основание (базис) r позиционной системы счисления - максимальное количество различных знаков или символов, используемых для изображения числа в данной системе счисления. Таким образом, основание может быть любым числом, кроме 1 и бесконечности.

Любое число в системе счисления с основанием r может быть записано в общем виде:

$$A=a_n\cdot r^n+ a_{n-1}\cdot r^{n-1}+...+a_1\cdot r^1+a_0\cdot r^0+ a_{-1}\cdot r^{--1}+...+a_{-rn-1}\cdot r^{-(rn-1)}+a_{-rn}\cdot r^{-rn}, \quad (1)$$

или

$$A_i = \sum_{i=-m}^{n} a_i r^i, \quad (2)$$

где любая разрядная цифра $a_i \in \{0,\ldots,r-1\}$, а r^i - вес соответствующего разряда.

Запись числа в форме (1) назовем записью числа в развернутой форме. Свернутой формой записи чисел называется запись чисел в виде

$A=a_1a_2 \ldots a_k$.

Для любой системы счисления основание представляется как 1 (один) и 0 (ноль).

Например:

9	1	F	7
$+\underline{1}$	$+\underline{1}$	$+\underline{1}$	$+\underline{1}$
10_{10}	10_2	10_{16}	10_8

Вес разряда p_i числа выражается соотношением

$$p_i = r^i /r^0 = r^i,$$

где i - номер разряда при отсчете справа налево.

Если в i-м разряде накопилось значение единиц, равное или большее r, то должна происходить передача единицы в старший i+1 разряд. При сложении такая передача информации называется переносом. При вычитании передача из i+1 разряда в i-й – заем.

Длина числа – количество позиций (разрядов) в записи числа. В технической реализации под длиной числа понимается длина разрядной сетки.

Диапазон представления чисел в заданной системе счисления – интервал числовой оси, заключенный между максимальным и минимальным числами, представленными при заданной длине разрядной сетки.

В вычислительной технике для представления данных и выполнения арифметических операций над ними удобно использовать двоичную, восьмеричную и шестнадцатеричную системы счисления. Ниже коротко остановимся на них.

Двоичная система счисления

Для записи числа в двоичной системе счисления используются две цифры: 0 и 1. Основание системы записывается как $10_{(2)}$ ($2_{10}=1\cdot2^1+0\cdot2^0$). Используя данную систему, любое число можно выразить последовательностью высоких и

низких потенциалов или группой запоминающих элементов, способных запоминать одно из двух (0,1) значений. Арифметические операции в двоичной системе счисления выполняются по тем же правилам, что и в десятичной системе счисления.

Сложение	Вычитание	Умножение
0+0= 0	0-0=0	0 · 0=0
0+1= 1	1-0=1	0 · 1=0
1+0= 1	1-1=0	1 · 0=0
1+1=10	10-1=1	1 · 1=1

Рассмотрим несколько примеров, демонстрирующих выполнение арифметических операций:

Пример:

```
  01010110        10101101          10011
+ 10010111      - 10010110        *  1011
  --------        --------          -----
  11101101        00010111          10011
                                  + 10011
                                    10011
                                    -----
                                    11010001
```

Восьмеричная система счисления

В восьмеричной системе счисления используется восемь цифр: 0,1,2 … 7, а основание записывается как $10_{(8)}$ ($8_{10}=1\cdot8^1+0\cdot8^0$). Рассмотрим выполнение операций в восьмеричной системе счисления. При их выполнении используются правила, представленные в таблицах сложения и умножения восьмеричных цифр.

Сложение

	1	2	3	4	5	6	7
1	2	3	4	5	6	7	10
2	3	4	5	6	7	10	11
3	4	5	6	7	10	11	12
4	5	6	7	10	11	12	13
5	6	7	10	11	12	13	14
6	7	10	11	12	13	14	15
7	10	11	12	13	14	15	16

Умножение

	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	4	6	10	12	14	16
3	3	6	11	14	17	22	25
4	4	10	14	20	24	30	34
5	5	12	17	24	31	36	43
6	6	14	22	30	36	44	52
7	7	16	25	34	43	52	61

Пример:

```
  35047326        43670154          20314
+ 21764254      - 17352326        *   264
  --------        --------          -----
  57033602        24315626          101460
                                  + 142310
                                    40630
                                    -----
                                    5607560
```

Шестнадцатеричная система счисления

В шестнадцатеричной системе счисления используются шестнадцать символов: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Основание записывается как $10_{(16)}$ ($16_{10}=1*16^1+0*16^0$).

Сложение

	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10
2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11
3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12
4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13
5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14
6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15
7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16
8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17
9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18
a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19
b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a
c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b
d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c
e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d
f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e

Умножение

	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
2	2	4	6	8	a	c	e	10	12	14	16	18	1a	1c	1e
3	3	6	9	c	f	12	15	18	1b	1e	21	24	27	2a	2d
4	4	8	c	10	14	18	1c	20	24	28	2c	30	34	38	3c
5	5	a	f	14	19	1e	23	28	2d	32	37	3c	41	46	4b
6	6	c	12	18	1e	24	2a	30	36	3c	42	48	4e	54	5a
7	7	e	15	1c	23	2a	31	38	3f	46	4d	54	5b	62	69
8	8	10	18	20	28	30	38	40	48	50	50	60	68	70	78
9	9	12	1b	24	2d	36	3f	48	51	5a	63	6c	75	7e	87
a	a	14	1e	28	32	3c	46	50	5a	64	6e	78	82	8c	96
b	b	16	21	2c	37	42	4d	58	63	6e	79	84	8f	9a	a5
c	c	18	24	30	3c	48	54	60	6c	78	84	90	9c	a8	84
d	d	1a	27	34	41	4e	5b	68	75	82	8f	9c	a9	b6	c3
e	e	1c	2a	38	46	54	62	70	7e	8c	9a	a8	b6	c4	d2
f	f	1e	2d	3c	4b	5a	69	78	87	96	a5	b4	c3	d2	e1

Пример:

```
  9A4DBE6          4A6B0F52          1B2A3
+                  -                 *
  A2864AF          2F392F2F            3C2
 13CD4095          1B31E023          36546
                                   + 145FA4
                                    517E 9
                                   6614886
```

Критерии выбора системы счисления

Сформулируем требования, которым должна удовлетворять система счисления для ЭВМ.

1. **Простота технической реализации**. Для хранения чисел в той или иной системе счисления используются n-позиционные запоминающие элементы. Элемент будет тем проще, чем меньше состояний требуется для запоминания цифры числа, то есть чем меньше основание системы счисления. Двухпозиционными элементами, имеющими два состояния, являются, например:

- электромеханическое реле (контакты замкнуты – 1, разомкнуты – 0);
- конденсатор (заряжен – 1, разряжен – 0);
- полупроводниковый элемент (если открыт, то хранит 0, иначе − 1) и

др.

Трехпозиционные элементы более редки. Например, конденсатор для запоминания цифр 0 – разряжен, 1– заряжен в одном направлении, 2 – в другом. Таким образом, реализация n-позиционных элементов более сложна, чем двух-

позиционных.

2. **Наибольшая помехоустойчивость кодирования цифр**. Положим, что при технической реализации любой системы счисления диапазон изменения электрического значения наибольшего и наименьшего числа одинаков. Очевидно преимущество систем с меньшим основанием, так как представление соседних цифр в этих системах отличается друг от друга больше, чем для систем с большим основанием.

Очевидно, что при наложении помехи на основной сигнал, соответствующий некоторой цифре, наиболее вероятна ошибка в устройствах, для которых используется система счисления с наибольшим основанием (рис.1). Следовательно, при увеличении основания помеха может привести к искажению числа.

3. **Минимум оборудования**. Пусть r - количество цифр в числе, n - количество разрядов в каждом числе, тогда D = r·n - количество цифроразрядов на одно число. Надо найти такую систему счисления, которая имеет минимальное количество цифроразрядов при заданном количестве чисел N:

$$N = r^n,$$
$$n = \log_r N,$$
$$D = r \cdot \log_r N = \frac{r}{\log_N r}.$$

Будем считать, что основание системы счисления может принимать любые значения, а не только целочисленные, изменяясь непрерывно, а не дискретно. Соответственно количество цифроразрядов может быть также величиной непрерывной, связанной с основанием системы счисления логарифмической зависимостью:

$$D(r) = \frac{r}{\log_N r}.$$

Это позволит свести задачу нахождения $D(b)_{min}$ к исследованию функции на экстремум:

$$\frac{dD}{dr} = \frac{\log_N r - r\frac{1}{r}\log_N e}{(\log_N r)^2} = 0,$$

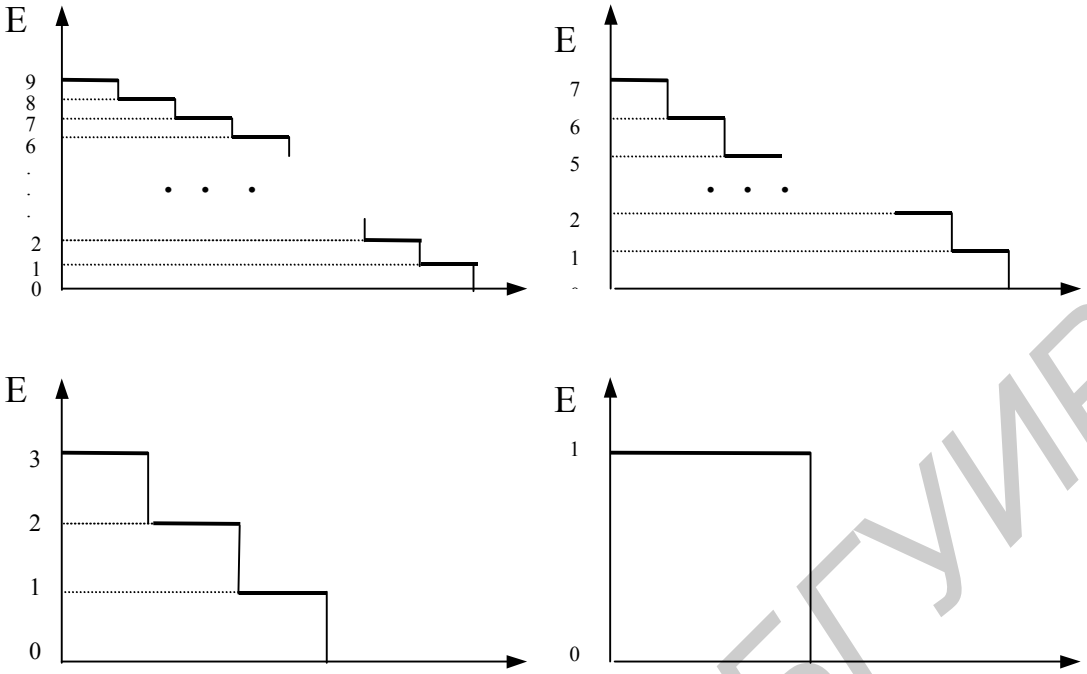


Рис. 1. Влияние помех на значение цифры

следовательно, $r_{опт}=e \cong 2{,}718$. Так как основание системы счисления должно быть целым числом, то основанием, наиболее близким к ***e,*** является основание r = 3. Но для реализации этого нужен элемент с тремя стабильными состояниями.

Выясним, насколько каждое из целочисленных оснований r_i уступает $r_{опт}$. Для этого оценим каждое основание r_i исходя из выражения

$$D_{i\,(отн)}=\frac{D_i}{D_{min}},$$

где $D_{min}= r_{опт} \log_{r\,опт} N=e \cdot \ln N$.

Следовательно, получим:

$$D(r)=\frac{r_i}{e \ln r_i}.$$

Выполнив расчеты для некоторых оснований, получим следующие результаты:

r_i	2	3	4	5	6	7	8	...
$D_{i\,(отн)}$	1,062	1,004	1,062	1,143	1,232	1,300	1,416	...

4. **Простота арифметических действий**. Чем меньше цифр в системе счисления, тем проще арифметические действия над ними. Таблицы для выполнения четырех арифметических операций будут усложняться с увеличением основания системы счисления. Это можно принять за косвенное доказательство выдвинутого положения.

5. **Наибольшее быстродействие**. Как будет показано далее, операции

вычитания, умножения и деления могут быть выполнены посредством операции алгебраического сложения. Алгебраическое сложение чисел часто сводится к их арифметическому сложению. Таким образом, взяв его за базовое, вычислим соотношение времени, необходимого на сложение:

$T_{сл} = n\, t_{пер} = t_{пер}\ \log_b N$.

Для упрощения оценки перейдем от абсолютной оценки к относительной:

$$\delta = \frac{T_{сл\ max}}{T_{сл}},$$

$T_{сл\ max} = T_{сл\ (2)} = t_{пер} \log_2 N$,

$$\delta = \frac{t_{пер} \log_2 N}{t_{пер} \log_b N} = \log_2 b,$$

r_i	2	3	4	5	...
δ	1,00	1,58	2,00	2,32	...

6. **Простота аппарата для выполнения анализа и синтеза цифровых устройств**. Математическим аппаратом, позволяющим относительно просто и экономно строить цифровые схемы, является алгебра логики. Наибольшее распространение и законченность вследствие своей простоты получила двузначная логика.

7.**Удобство работы с ЭВМ**. Наиболее удобной системой счисления для работы человека является десятичная система счисления. Но внутри ЭВМ для выполнения арифметических операций числа из десятичной системы счисления требуется переводить во внутреннюю систему счисления.

Для системы счисления с основанием, большим 10, появляются новые цифры. Таким образом, система счисления должна иметь минимальное число цифр, так как в этом случае можно пользоваться младшими цифрами десятичной системы счисления.

8.**Возможность представления любого числа в рассматриваемом диапазоне величин**.

9.**Единственность представления числа**.

Перевод чисел из одной системы счисления в другую

Так как числа, участвующие в операциях, могут быть представлены в различных позиционных системах счисления, то для выполнения действий над ними требуется привести их к одной системе счисления. Необходимо отметить, что целая и дробная части числа переводятся отдельно. Следовательно, все методы перевода чисел можно подразделить на две группы: перевода целых и дробных чисел.

Перевод целых чисел

Метод подбора степеней основания. В соответствии с (2) целые числа в системах счисления с основаниями r_1 и r_2 могут быть представлены:

$$A_{r1} = \sum_{i=0}^{n} a_i r_1^i = \sum_{j=0}^{k} b_j r_2^j = A_{r2}.$$

В общем случае перевод числа из системы счисления с основанием r_1 в систему счисления с основанием r_2 можно представить как задачу определения коэффициентов b_i нового ряда, изображающего число в системе счисления с основанием r_2. Основная трудность в выборе максимальной степени основания r_2, которая еще содержится в числе A_{r1}. Все действия должны выполняться по правилам r_1-арифметики (то есть исходной системы счисления). После нахождения максимальной степени и соответствующего ей коэффициента необходимо найти коэффициенты для всех остальных (младших) степеней.

Пример: $A_{10}=37$, $A_2=?$

$$37=1\cdot2^5 + 0\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0=100101.$$

Нечетным двоичным числом 100101 является число, содержащее единицу в младшем разряде.

Метод деления на основание системы счисления. На основании (1) число A_{r1} в системе счисления с основанием r_2 запишется в виде

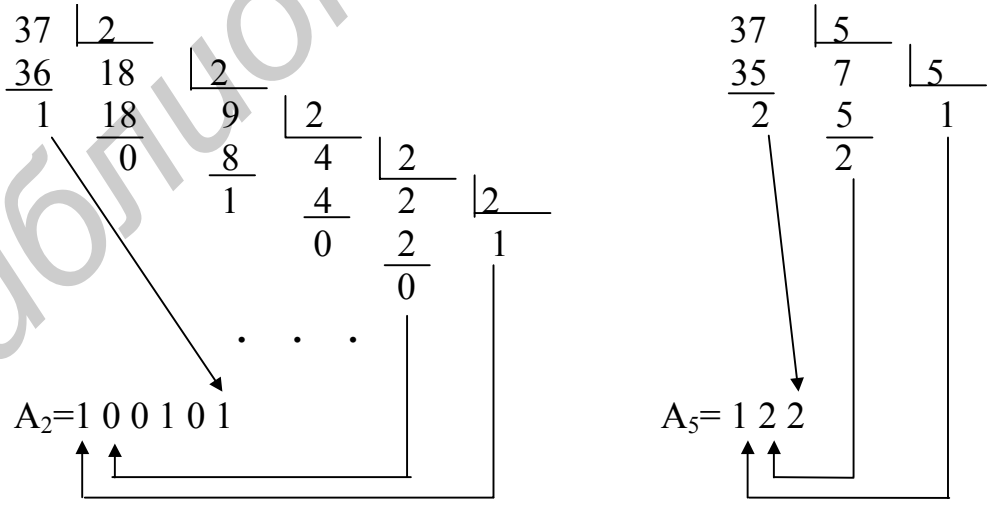
$$A_{r2} = a_n\cdot r_2^n + a_{n-1}\cdot r_2^{n-1} + \ldots + a_1\cdot r_2^1 + a_0\cdot r^0.$$

Переписав это выражение по схеме Горнера, получим:

$$A_{r2} = (\ldots(a_n r_2 + a_{n-1}) r_2 + \ldots + a_1) r_2 + a_0.$$

Разделив правую часть на r_2, получим первый остаток a_0 и целую часть $(\ldots(a_n r_2 + a_{n-1}) r_2 + \ldots + a_1)$. Разделив целую часть на r_2, получим остаток a_1 и новую целую часть. Выполнив деление n+1 раз, получим последнее целое частное $a_n < r_2$, являющееся старшей цифрой числа.

Пример: $A_{10} = 37$, $A_2 = ?$, $A_5 = ?$



Перевод правильных дробей

Метод подбора величин, обратных степеням основания

$A_{10} = 0{,}716$

$A_2 = 0{,}1011\ldots$

$\frac{1}{r}=\frac{1}{2}=0{,}5;\ \ \frac{1}{r^2}=\frac{1}{4}=0{,}25;\ \ \frac{1}{r^3}=\frac{1}{8}=0{,}125;\ \ \frac{1}{r^4}=\frac{1}{16}=0{,}025...$

Количество разрядов после запятой зависит от точности, с которой требуется представить число.

Метод умножения на основание r_2 новой системы счисления. Из выражения (1) дробное число A_{r1} в системе счисления с основанием r_2 запишется в виде

$A_{r2} = a_{-1}\, r_2^{-1} + ... + a_{-n}\, r_2^{-n}$.

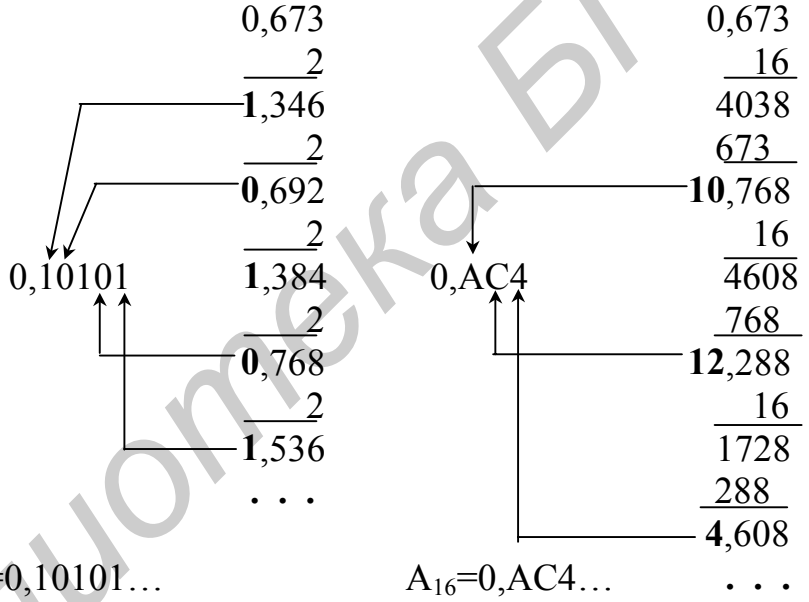
Переписав это выражение по схеме Горнера, получим:

$A_{r2} = r_2^{-1}\,(a_{-1} + r_2^{-1}\,(a_{-2} + ... + a_{-n}\, r_2^{-1})...)$.

Умножив правую часть на r_2, получим новую неправильную дробь, целая часть которой есть a_{-1} (старшая цифра числа A_{r2}). Продолжим процесс умножения дробной части на r_2 n-1 раз, получим цифры $a_{-2}, a_{-3}, \ldots$ числа A_{r2}.

Процесс умножения может быть прекращен, если во всех разрядах после очередного умножения получены нули либо достигнута требуемая точность.

Пример: A_{10}=0,673, A_2=?, A_{16}=?



Для перевода неправильных дробей отдельно выделяется целая и дробная части числа и с использованием соответствующих методов выполняется их перевод. Результаты записываются в виде новой неправильной дроби.

Перевод чисел из одной системы счисления в другую, основание которой кратно степени 2

К таким системам относятся двоичная, четверичная, восьмеричная и т.д. системы счисления.

$$A_8 = \sum_{i=0}^{n} a_{m_i}\, 8^i = \sum_{i=0}^{n} (b_{2i}\, 2^2 + b_{1i}\, 2^1 + b_{0i})\, 2^{3i}\ .$$

Ограничимся тремя восьмеричными разрядами, придавая i значения 0,1,2.

$\underbrace{(b_{20}\, 2^2 + b_{10}\, 2 + b_{00})}_{i=0} +$ (младшая восьмеричная цифра с весом 8^0)

$$+\underbrace{(b_{20}2^5 + b_{10}2^4 + b_{00}2^3)}_{i=0} +$$
$$+\underbrace{(b_{20}2^8 + b_{10}2^7 + b_{00}2^6)}_{i=0} = A_{2/8}$$

При переводе числа из двоичной системы счисления в восьмеричную необходимо разделить его разряды на триады, начиная с младших разрядов, и каждую триаду заменить восьмеричной цифрой.

Пример: $A_8 = 45$ $A_2 = 0010\ 0101$

$100\ 101 = A_2$ $2\ \ 5 = A_{16}$

Кодирование чисел

Кодирование знака числа. Кодирование чисел позволяет заменить операцию арифметического вычитания операцией алгебраического сложения с помощью двоичного сумматора. Для кодирования знака числа используется специальный двоичный разряд, называемый *знаковым*. При этом знак плюс кодируется двоичной цифрой 0, а минус – цифрой 1 (для системы счисления с основанием r – цифрой r-1). Для машинного представления отрицательных чисел используют три основных вида кодов: прямой, обратный и дополнительный. Общая схема кода числа: *код знака . код числа.*

Прямой код числа. При этом способе кодирования чисел кодируется только знак числа, а значащая часть остается без изменения.

$$[A]_{пр} = \begin{cases} 0, A & \text{, если } A \geq 0, \\ r-1, |A| & \text{, если } A < 0, \end{cases}$$ – для правильных дробей.

Пример: $A = +0{,}1101$ $A = -0{,}1101$

$[A]_{пр} = 0{,}1101$ $[A]_{пр} = 1{,}1101$

$$[A]_{пр} = \begin{cases} 0.A & \text{, если } A \geq 0, \\ r-1.|A| & \text{, если } A < 0, \end{cases}$$ – для целых чисел.

Пример: $A = +1101$ $A = -1101$

$[A]_{пр} = 0.1101$ $[A]_{пр} = 1.1101$

Диапазон изменения машинных изображений для прямого кода лежит в пределах: $-(1-2^{-n}) \leq [A]_{пр} \leq (1-2^{-n})$.

Недостатком прямого кода является сложность выполнения операции сложения чисел с разными знаками.

Для арифметических операций над числами в прямом коде используется сумматор прямого кода. В этом сумматоре отсутствует цепь поразрядного переноса между старшим значащим и знаковым разрядами, то есть на этом сумматоре невозможно выполнение операции алгебраического сложения.

Дополнительный код числа. Число A' называется *дополнением* к числу A, если выполняется соотношение: $A + A' = r^n$ для целых чисел или $A + A' = r^0$ для дробных чисел, где n – количество цифр в записи числа A.

Пример: $A_{10} = 378$

$$A_{10}' = 10^{n=3} - A_{10} = 1000 - 378 = 622$$

378

621 - все разряды дополняются до младшей цифры системы счисления

1 - младший разряд дополняется до основания системы счисления

1000

$$A_2 = 1011, \Rightarrow A_2' = 2^{n=4} - A = 10000 - 1011 = 0101, \text{ или } A_2' = 0101$$

Замена операции вычитания операцией сложения. В ЭВМ достаточно сложно выполнить операцию вычитания (A-B). Для этого требуется:

1) сравнить числа и выявить наибольшее из них по абсолютной величине;
2) наибольшее число разместить на входах вычитающего устройства;
3) выполнить операцию вычитания;
4) присвоить значению разности знак наибольшего по абсолютной величине числа.

Для сложения чисел в дополнительных кодаах требуется сумматор и неважно, какие слагаемые подаются на его входы A или B. Пусть необходимо сложить

A = 487	A = 487
B = -348	B = 652
A-B = 139	A-B = 1 139

$A + (10^3 - B) = A-B+10^3$ (10^3 игнорируется).

A = 348	A = 348
B = -487	B = 513
A-B = -139	A-B = 861

Дополнительный код отрицательных чисел является математическим дополнением абсолютной величины числа до основания r системы счисления для дробных чисел и до r^n для целых чисел.

$|A| + [A]_{доп} = r$ - для дробных чисел, $|A| + [A]_{доп} = r^n$ - для целых чисел,

где $|A|$ - абсолютное значение числа A, n – число цифр числа.

Положительные числа в дополнительном коде не меняют своего изображения. Правило преобразования числа в дополнительный код можно записать:

$$[A]_{доп} = \begin{cases} 0, A, & \text{если } A \geq 0, \\ r - |A|, & \text{если } A < 0, \end{cases}$$ – для правильных дробей,

$$[A]_{доп} = \begin{cases} 0. A, & \text{если } A \geq 0, \\ r^n - |A|, & \text{если } A < 0, \end{cases}$$ – для целых чисел.

Рассмотрим несколько примеров сложения чисел в дополнительных кодах.

A= 0,1001	$[A]_{доп}$ = 0,1001	A= - 0,1001	$[A]_{доп}$ = 1,0111
B= - 0,0100	$[B]_{доп}$ = 1,1100	B= 0,0100	$[B]_{доп}$ = 0,0100

10,0101 1,1011

Теорема. Сумма дополнительных кодов чисел есть дополнительный код результата.

Доказательство теоремы приведено в [1].

Теорема справедлива для всех случаев, в которых не возникает переполнения разрядной сетки, что позволяет складывать машинные представления чисел по правилам двоичной арифметики, не разделяя знаковую и значащую части числа. Для выполнения арифметических операций над числами в дополнительном коде используется *двоичный сумматор дополнительного кода*, характерной особенностью которого является наличие поразрядного переноса из старшего значащего в знаковый разряд.

Обратный код числа. Обратный код двоичного числа является инверсным изображением числа, в котором все разряды исходного числа принимают инверсное (обратное) значение. Правила преобразования чисел в обратный код аналитически можно определить следующим образом:

$$[A]_{обр} = \begin{cases} 0, A & , \quad \text{если } A \geq 0, \\ r - |A| - r^{-n} & , \quad \text{если } A < 0, \end{cases} \quad - \text{ для правильных дробей,}$$

$$[A]_{обр} = \begin{cases} 0. A & , \text{ если } A \geq 0, \\ r^{n} - |A| - 1 & , \text{если } A < 0, \end{cases} \quad - \text{ для целых чисел.}$$

Выполнение арифметических операций над числами в обратном коде осуществляется на *сумматоре обратного кода.* Этот код имеет несущественный недостаток: требует наличия в сумматоре цепи циклического переноса из знакового разряда в младший значащий. Это может привести к увеличению времени выполнения арифметических операций. Ниже приведены несколько примеров выполнения арифметических операций над числами, записанными в обратном коде.

A= 0,1001 $[A]_{обр}$ = 0,1001 A= - 0,1001 $[A]_{обр}$ = 1,0110

B= - 0,0100 $[B]_{обр}$ = 1,1011 B= 0,0100 $[B]_{обр}$ = 0,0100

10,0100 1,1010

↳ 1

0,0101

Теорема. Сумма обратных кодов чисел есть обратный код результата.

Доказательство теоремы приведено в [1].

Переполнение разрядной сетки

При выполнении некоторых арифметических операций может возникать явление переполнения разрядной сетки. Причиной переполнения может служить суммирование двух чисел с одинаковыми знаками (для чисел с разными знаками переполнение не возникает), которые в сумме дают величину, большую или равную 1 (при сложении правильных дробей), или величину r^n (при сложении целых чисел).

Пример: A=+0,101 $[A]_{доп} = 0,101$

B=+0,110 $[B]_{доп} = \underline{0,110}$

$[A+B]_{доп} = 1,011$

В результате сложения двух положительных чисел получено отрицательное число, что является ошибкой. Результат неверен также и по величине.

Для обнаружения переполнения можно использовать следующие признаки:

- знаки слагаемых не совпадают со знаком суммы;
- есть перенос только в знаковый или только из знакового разряда.

Функция переполнения имеет вид: $f=П_1\overline{П}_2 + \overline{П}_1П_2 = П_1 \oplus П_2$.

Если при сложении чисел с фиксированной запятой возникло переполнение, то вырабатывается сигнал переполнения разрядной сетки и вычисления прекращаются.

Следует отметить, что при сложении чисел в дополнительном коде возможен случай, когда переполнение не фиксируется. Это происходит тогда, когда сумма модулей двух отрицательных чисел равна удвоенному весу единицы старшего разряда числа.

Пример: A=- 0,101 $[A]_{доп} = 1,011$

B=- 0,011 $[B]_{доп} = \underline{1,101}$

$[A]_{доп}+[B]_{доп} = 1,000$

Модифицированные коды

Для обнаружения переполнения разрядной сетки можно использовать модифицированные коды. Модифицированные коды отличаются от обычных кодов тем, что знак числа кодируется двумя разрядами. При выполнении алгебраического сложения или вычитания два знаковых разряда участвуют в операции как равноправные цифровые разряды. После выполнения операции содержимое знаковых разрядов определяет знак результата (левый знаковый разряд) и наличие переполнения (несовпадение знаковых разрядов): комбинация 01 фиксирует переполнение при сложении положительных чисел (положительное переполнение), а 10 – отрицательных (отрицательное переполнение).

A=+0,101 $[A]^{мод}_{доп} = 00,101$

B=+0,110 $[B]^{мод}_{доп} = \underline{00,110}$

$[A]^{мод}_{доп}+[B]^{мод}_{доп} = 01,011$

A=-0,101 $[A]^{мод}_{доп} = 11,011$

B=-0,110 $[B]^{мод}_{доп} = \underline{11,010}$

$[A]^{мод}_{доп}+[B]^{мод}_{доп} = 10,101$

Функция переполнения имеет вид: $f=Зн1\,\overline{Зн2} + \overline{Зн1}\,Зн2 = Зн1 \oplus Зн2$.

Логическая схема формирования единичного сигнала при возникновении переполнения приведена на рис 2.

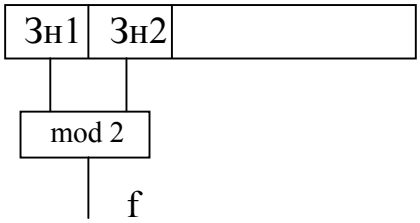


Рис.2. Схема выявления переполнения

Машинные формы представления чисел

Существуют два основных способа представления данных в ЭВМ: с фиксированной и плавающей запятой.

Представление чисел в форме с фиксированной запятой. Для сокращения длины разрядной сетки и упрощения обработки данных положение запятой может быть зафиксировано схемотехнически. При этом в слове данных сохраняются только две структурных компоненты: поле знака и поле цифр.

±	целая часть	дробная часть

Определим диапазон представления чисел для этого формата.

$A_{max}=(2^k-1)+(1-2^{-l})$.

В зависимости от размеров целой и дробной частей возможно следующее:

1) k=0, l=n $A_{max}=1-2^{-n}$

2) k=n, l=0 $A_{max}=2^n-1$

3) k=0, l=n $A_{min}=2^{-n}$

4) k=n, l=0 $A_{min}=1$

±	. 1	1	. . .	1	1
±	1	1	. . .	1	1.
±	. 0	0	. . .	0	1
±	0	0	. . .	0	1.

Очевидно, что ограничение длины разрядной сетки приводит к ограничению диапазона хранимых чисел и потере точности их представления. Поэтому на практике широко используется и другая форма представления чисел.

Представление чисел в форме с плавающей запятой. В общем виде числа с плавающей запятой имеют следующий вид:

$A=\pm m_A r^{\pm p_A}$,

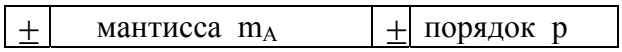
где m_A- мантисса, а p_A- порядок числа А. Порядок (с учетом знака) показывает, на сколько разрядов и в какую сторону сдвинута запятая при замене формы записи числа с естественной на нормальную.

Например, $A_{10} = 239{,}745 = 0{,}239745\cdot 10^3 = 239745\cdot 10^{-3}$.

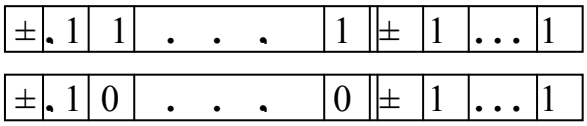
Наиболее распространено и удобно для представления в ЭВМ ограничение вида $r^{-1}\le |m_A| <1$.

Форма представления чисел, для которых справедливо данное ограничение, называется *нормализованной*. Так как абсолютное значение мантиссы в этом случае лежит в диапазоне от r^{-1} до $1-r^{-n}$, где n – число разрядов мантиссы без знака, то положение разрядов числа в его машинном изображении непостоянно. Отсюда и название этой формы представления чисел – с плавающей запятой. Формат машинного изображения чисел с плавающей запятой должен вклю-

включать знаковые поля (мантиссы и порядка), поле мантиссы и поле порядка числа и имеет следующий вид:



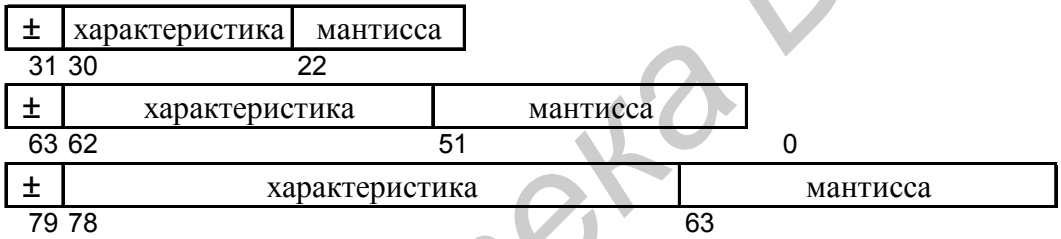
Для данного формата разрядной сетки можно записать следующий диапазон представления чисел:



$$A_{max} = (1 - 2^{-n})\, 2^{(2^q - 1)}$$

$$A_{min} = 2^{-2^q}$$

Для упрощения операций над порядками применяют представление чисел с плавающей запятой со смещенным порядком: $p' = p + N$, где N – целое положительное число (смещение), $N = \max(-p)$. Обычно $N = 2^k$, где k- число двоичных разрядов в поле цифр несмещенного порядка. В этом случае поле знака порядка избыточно, так как p′ всегда положительно. Такие смещенные порядки называют *характеристиками*. В зависимости от типа данных числа с плавающей запятой в памяти ЭВМ хранятся в одном из следующих трех форматов:



При выполнении арифметических операций над числами с плавающей запятой может получаться результат, выходящий за пределы диапазона представления чисел, при этом выход за правую границу диапазона принято называть переполнением порядка (получение очень большого числа), а выход за левую границу – исчезновением порядка (потерей порядка) – получение очень малого числа, близкого к нулю.

В стандарте IEEE крайние значения порядка (характеристики) зарезервированы и не используются для представления обычных чисел. Максимальное значение характеристики, представленное всеми единицами при положительном знаке числа, зарезервировано для представления значения (+ ∞) при нулевой мантиссе. При знаке минус число с максимальной характеристикой используется для представления (- ∞) и неопределенности. Значение с минимальной характеристикой, равной нулю, зарезервировано для представления денормализованных чисел (положительных и отрицательных), а также для представления нуля (представляется всеми нулями), причем различают +0 и –0.

Погрешность выполнения арифметических операций

Выбор длины разрядной сетки и формы представления чисел тесно связан с точностью получаемых при арифметических операциях результатов. При вы-

полнении операций над числами с фиксированной запятой можно считать, что результат точен (при условии отсутствии переполнения).

При выполнении операций над числами, представленными в форме с плавающей запятой, требуется выравнивание порядков. Это может приводить к потере некоторых разрядов мантиссы.

Рассмотрим арифметические операции над операндами, заданными с абсолютными погрешностями: $A=[A]+\Delta A$ и $B=[B]+\Delta B$.

$A+B=[A]+[B]+(\Delta A+\Delta B)$,

где абсолютная погрешность суммы $\Delta(A+B)=\Delta A+\Delta B$.

$A-B=[A]-[B]+(\Delta A-\Delta B)$,

где абсолютная погрешность разности $\Delta(A-B)=\Delta A-\Delta B$.

$A\cdot B=[A][B]+[A]\Delta B+[B]\Delta A+\Delta A\Delta B$.

Произведением $\Delta A\Delta B$ можно пренебречь, следовательно,

$A\cdot B\approx[A][B]+[A]\Delta B+[B]\Delta A$,

то есть абсолютная погрешность произведения $\Delta(AB)\approx[A]\Delta B+[B]\Delta A$.

При выполнении операции деления

$$\frac{A}{B}=\frac{[A]+\Delta A}{[B]+\Delta B}=\frac{[A]+\Delta A}{[B]}\left(\frac{1}{1+\Delta B/[B]}\right)$$

абсолютная погрешность частного $\Delta(A/B)=\Delta A/[B]-[A]\Delta B/([B])^2$.

Округление

Речь идет об округлении только дробных чисел, целые не округляются. Так как в ЭВМ используются числа с конечным числом разрядов, а также часто выполняются операции приведения данных одной размерности к данным другой, то операция округления выполняется достаточно часто.

В общем виде число с плавающей запятой, размещенное в разрядной сетке размерностью k, имеет вид $A_r=m_a r^k$. Если для записи мантиссы используются только n разрядов, то число может быть представлено в виде двух частей: $A_r=[m_a]r^n+[A_0]r^{k-n}$, где $[A_0]r^{k-n}=A_0$ – часть числа, не вошедшая в разрядную сетку размерностью k.

В зависимости от того, как учитывается A_0 при записи числа A в n–разрядную сетку, можно выделить несколько способов округления чисел.

1. *Отбрасывание A_0.* При этом возникает относительная погрешность $\delta_{окр}=|A_0|r^{k-n}/(|m_A|r^n)$, так как $r^{-1}\le|m_A|<1$, $0\le|A_0|<1$, то $\delta_{окр}=r^{-(n-1)}$.

2. *Симметричное округление*. При этом производится анализ величины A_0:

$$[A]=\begin{cases}[m_A]r^n, & \text{если } |A_0|<r^{-1},\\ [m_A]r^n+r^{k-n}, & \text{если } |A_0|\ge r^{-1}.\end{cases}$$

При условии $|A_0|\ge r^{-1}$ единица добавляется к младшему разряду мантиссы. Данный способ округления наиболее часто используется на практике.

3. *Округление по дополнению*. В этом случае для округления используется (n+1)-й разряд. Если в нем находится единица, то она передается в n-й разряд, иначе разряды начиная с (n+1)-го отбрасываются.

4. *Случайное округление*. Генератор случайных чисел формирует нулевое или единичное значение, посылаемое в младший разряд мантиссы.

Оценка точности вычислений зависит как от вида выполняемых операций, так и от последовательности их следования друг за другом.

Нормализация чисел

Число называется нормализованным, если его мантисса удовлетворяет условию $r^{-1} \le |M_A| < 1$.

Нормализация – процесс, относящийся к числам, записанным в форме с плавающей запятой. Число A =0,00101…1 – денормализованное (признак нарушения нормализации вправо). Для нормализации число нужно сдвинуть в сторону, противоположную направлению нарушения нормализации. Таким образом, в примере мантиссу числа A необходимо сдвинуть влево на два разряда. При этом порядок необходимо уменьшить на два. Различают два вида сдвигов: простой и модифицированный.

Простой сдвиг – сдвиг, выполняемый по правилу:

Исходная комбинация	Сдвиг влево	Сдвиг вправо
$0,a_1a_2\ldots a_n$	$a_1,a_2\ldots a_n0$	$0,0a_1a_2\ldots a_{n-1}$
$1,a_1a_2\ldots a_n$	$a_1,a_2\ldots a_n\alpha$	$0,1a_1a_2\ldots a_{n-1}$

Модифицированный сдвиг - сдвиг, при котором в сдвигаемый разряд заносится значение, совпадающее со значением знакового разряда.

Исходная комбинация	Сдвиг влево	Сдвиг вправо
$00,a_1a_2\ldots a_n$	$0a_1,a_2\ldots a_n0$	$00,0a_1a_2\ldots a_{n-1}$
$01,a_1a_2\ldots a_n$	$1a_1,a_2\ldots a_n0$	$00,1a_1a_2\ldots a_{n-1}$
$10,a_1a_2\ldots a_n$	$0a_1,a_2\ldots a_n\alpha$	$1,1a_1a_2\ldots a_{n-1}$
$11,a_1a_2\ldots a_n$	$1a_1,a_2\ldots a_n\alpha$	$1,1a_1a_2\ldots a_{n-1}$

Нарушение нормализации вправо может быть более глубоким при вычитании, например, одного числа из другого, если они близки по величине.

Последовательное и параллельное сложение чисел

Параллельный способ передачи информации является более быстродействующим по сравнению с последовательным, но менее экономичным (требуется вместо одного проводника (и усилителя сигналов) n проводников).

В соответствии со способом приема/передачи информации устройства,

обрабатывающие эту информацию, могут быть либо параллельного, либо последовательного действия.

Последовательный сумматор (рис. 3**)**.

Сумматор – устройство, предназначенное для выполнения арифметического сложения чисел в двоичном коде. Простейший случай - это суммирование двух одноразрядных чисел.

$T_{с(посл)} \cong n\ \Delta t$, Δt-время задержки сигнала переноса на элементе задержки.

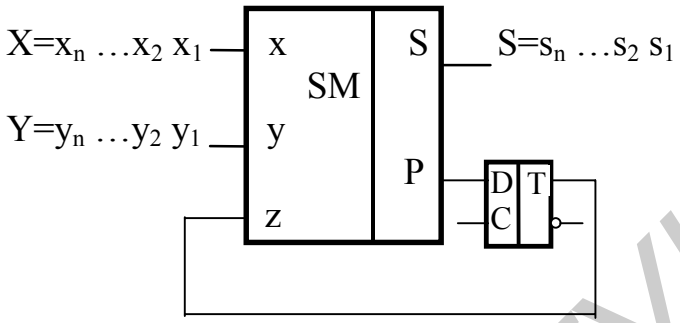


Рис. 3. Схема последовательного суммирования

В этой схеме на входы x и y последовательно подаются попарно разряды слагаемых x_i и y_i. На выходе S формируются s_i разряды суммы. Триггер введен в схему для хранения значения переноса до следующего такта (следующей пары разрядов).

Параллельный сумматор (рис. 4)**.**

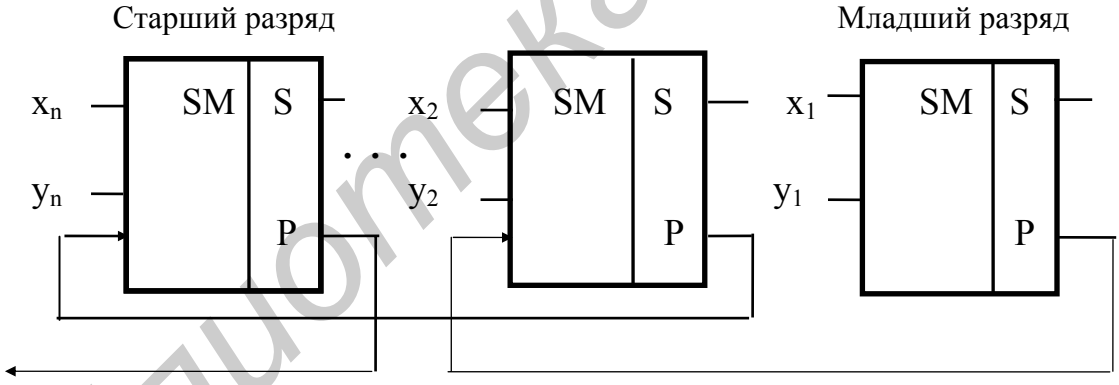


Рис. 4. Схема параллельного суммирования

$T_{с(пар)} \cong n\tau_{лэ}$.

Очевидно, $T_{с(посл)} \leq T_{с(пар)}$, так как

$\Delta t \cong (\overline{3,6})\ k\tau_{лэ}$,

где k - коэффициент запаса, обеспечивающий полное окончание всех переходных процессов в сумматоре последовательного действия $k \in [1{,}2,\ 1{,}3]$.

Недостатком такого параллельного суммирования является большое время распространения сигналов переноса P_i. Параллельные безрегистровые сумматоры обеспечивают наибольшую скорость суммирования, если снабжены схемой ускоренного переноса.

Сложение чисел с плавающей запятой

При сложении чисел складываемые цифры (разряды) должны иметь одинаковый вес. Это требование выполняется, если складываемые числа имеют

одинаковые порядки. Пусть имеются два числа с плавающей запятой:

$A=\pm m_A r^{\pm p_A}$,

$B=\pm m_B r^{\pm p_B}$.

Алгоритм сложения чисел с произвольными знаками состоит в следующем.

1. Производится сравнение порядков p_A и p_B. Для этого из порядка числа A вычитается порядок числа B. Разность $p=p_A-p_B$ указывает, на сколько разрядов требуется сдвинуть вправо мантиссу числа с меньшим порядком. Если $p=p_A-p_B>0$, то $p_A>p_B$ и для выравнивания порядков необходимо сдвинуть вправо мантиссу M_B. Если $p=p_A-p_B<0$, то $p_B>p_A$ и для выравнивания порядков необходимо сдвинуть вправо мантиссу M_A. Если $p=p_A-p_B=0$, то $p_A=p_B$ и порядки слагаемых выравнивать не требуется.

2. Выполняется сдвиг соответствующей мантиссы до тех пор, пока $p\neq 0$.

3. Выполняется сложение мантисс M_A и M_B по правилу сложения правильных дробей.

4. Если при сложении мантисс произошло переполнение, то производится нормализация путем сдвига мантиссы суммы вместе со знаковым разрядом вправо на один разряд с увеличением порядка на единицу. Если же происходит денормализация, то выполняется сдвиг мантиссы результата на соответствующее количество разрядов в сторону, противоположную нарушению нормализации с соответствующим изменением порядка суммы.

Пример: M_A=-0,10110 p_A=+0111

M_B=-0,11011 p_B=+0101

$[M_A]_{доп}$=1,01010 $p=[p_A]_{доп}+[-p_B]_{доп}=$ 0.0111

$[M_B]_{доп}$=1,00101 $+$ 1.1011

1 0.0010

Так как $[p_A-p_B]_{доп}>0$, то сдвигу подвергается мантисса M_B.

В рассматриваемом примере при каждом сдвиге мантиссы на один разряд из положительной разности порядков производим последовательное вычитание единицы до тех пор, пока в результате не будет получен ноль. При этом выполняется анализ разности порядков на каждом шаге. Если она отлична от нуля, то производится очередной сдвиг соответствующей мантиссы. В случае если разность $[p_A-p_B]_{доп}<0$, необходимо либо прибавлять единицу до нулевого результата, либо изменить знак разности на противоположный и, как и выше, выполнять вычитание единицы.

$[M_B]_{доп}$=1,00101 0.0010

$[-1]_{доп}=$ 1.1111

$[M_B]_{доп}$=1,**1**0010 1 0.0001

$[M_B]_{доп}$=1,**11**001 01 $[-1]_{доп}=$ 1.1111

0.0000

$[M_B]_{доп}$=1,11001 01

$[M_A]_{доп}$=1,01010

11,00011 01 = $[M_{A+B}]$ $p_{A+B}=\max(p_A,p_B)=p_A$=+0.0111

Полученный результат нормализован. После выполнения операции округления получим $[M_{A+B}]= 1{,}00011$.

Машинные методы умножения чисел в прямых кодах

Операция умножения состоит из ряда последовательных сложений. Сложением управляют разряды множителя: если в очередном разряде множителя содержится единица, то к сумме добавляется множимое. При этом, в зависимости от метода умножения, выполняется сдвиг либо множимого, либо частичной суммы. Наряду с этим умножение можно начинать как с младших, так и со старших разрядов множителя. Для умножения используют модули сомножителей. Знак произведения определяется сложением по модулю 2 знаковых разрядов сомножителей.

Введем некоторые обозначения, используемые ниже: $\Pi_i^ч$ - частичное произведение, $\Sigma_i^ч$ - частичная сумма.

Ниже приводится схема четырех алгоритмов умножения (рис. 5).

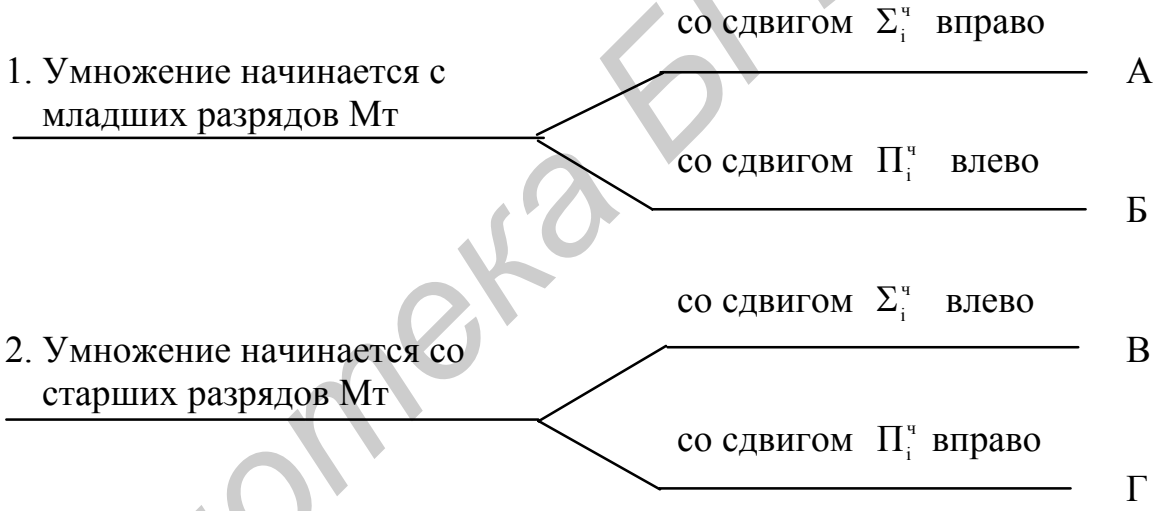


Рис. 5. Схема алгоритмов умножения

Остановимся более подробно на реализации умножения согласно *алгоритму А*.

Представим $Мн = A = 0,a_1a_2\ldots a_n$

$$Мт = B = 0,b_1b_2\ldots.b_n = b_12^{-1}+b_22^{-2}+\ldots+b_n2^{-n}+b_n2^{-n}$$

$$Мн\cdot Мт = C=A\cdot B= 0,a_1a_2\ldots a_n\ (b_12^{-1}+b_22^{-2}+\ldots+b_n2^{-n}) =$$

$$= 0+(b_1\cdot 0,a_1a_2\ldots a_n)2^{-1}+\ldots+(b_{n-1}\cdot 0,a_1a_2\ldots a_n)2^{-(n-1)}+(b_n\cdot 0,a_1a_2\ldots a_n)2^{-n} =$$

$$= 0+b_1\cdot A2^{-1}+\ldots+b_{n-1}\cdot A2^{-(n-1)}+\ldots+b_n\cdot A2^{-n} = 0+b_n\cdot A2^{-n}+b_{n-1}\cdot A2^{-(n-1)}+\ldots+b_1\cdot A2^{-1} =$$

$$= (\ldots((0+b_n\cdot A)2^{-1}+b_{n-1}\cdot A)2^{-1}+\ldots+b_1\cdot A)2^{-1}$$

Ниже приведены (без вывода) остальные три реализации алгоритмов (Б, В и Г) умножения.

$$Мн\cdot Мт = C=A\cdot B = 0+b_n\cdot A+b_{n-1}\cdot A\cdot 2^2+\ldots+b_1\cdot A\cdot 2^{n-1} \quad \text{(алгоритм Б)}$$

$$Мн\cdot Мт = C=A\cdot B = (\ldots(0+b_1\cdot A)\cdot 2^1+b_2\cdot A)\cdot 2^1+\ldots+b_n\cdot A) \quad \text{(алгоритм В)}$$

$$Мн\cdot Мт = C=A\cdot B = 0+b_1\cdot A\cdot 2^{-1}+b_2\cdot A\cdot 2^{-2}+\ldots+b_n\cdot A\cdot 2^{-n} \quad \text{(алгоритм Г)}$$

Структурные схемы операционных устройств, выполняющих умножение по алгоритмам А, Б, В и Г, приведены на рис 6.

Рассмотрим пример умножения чисел согласно алгоритму А.

Пример: Мн = 0,1011
Мт = 0,1101
$b_1 \ldots b_4$

0,0000	$\Sigma_0^ч$	начальное содержимое сумматора
+ 0,1011	$П_1^ч = Мн \cdot b_4$	первое частичное произведение
0,1011	$\Sigma_1^ч$	первая частичная сумма
0,**0**101 1	$\Sigma_1^ч \cdot 2^{-1}$	сдвиг первой частичной суммы
+ 0,0000	$П_2^ч = Мн \cdot b_3$	второе частичное произведение
0,0101 1	$\Sigma_2^ч$	вторая частичная сумма
0,**0**010 11	$\Sigma_2^ч \cdot 2^{-1}$	сдвиг второй частичной суммы
+ 0,1011	$П_3^ч = Мн \cdot b_2$	третье частичное произведение
0,1101 11	$\Sigma_3^ч$	третья частичная сумма
0,**0**110 111	$\Sigma_3^ч \cdot 2^{-1}$	сдвиг третьей частичной суммы
+ 0,1011	$П_4^ч = Мн \cdot b_1$	четвертое частичное произведение
1,0001 111	$\Sigma_4^ч$	(возникло переполнение)
0,**1**000 1111	$\Sigma_4^ч \cdot 2^{-1}$	сдвиг, получение верного результата Мн·Мт

Заметим, что при умножении чисел по алгоритму А на отдельных этапах операции возможно переполнение (попадание значащей единицы в знаковый разряд). Однако при последующем сдвиге переполнение устраняется. При использовании других алгоритмов (Б, В, Г) переполнения не возникает.

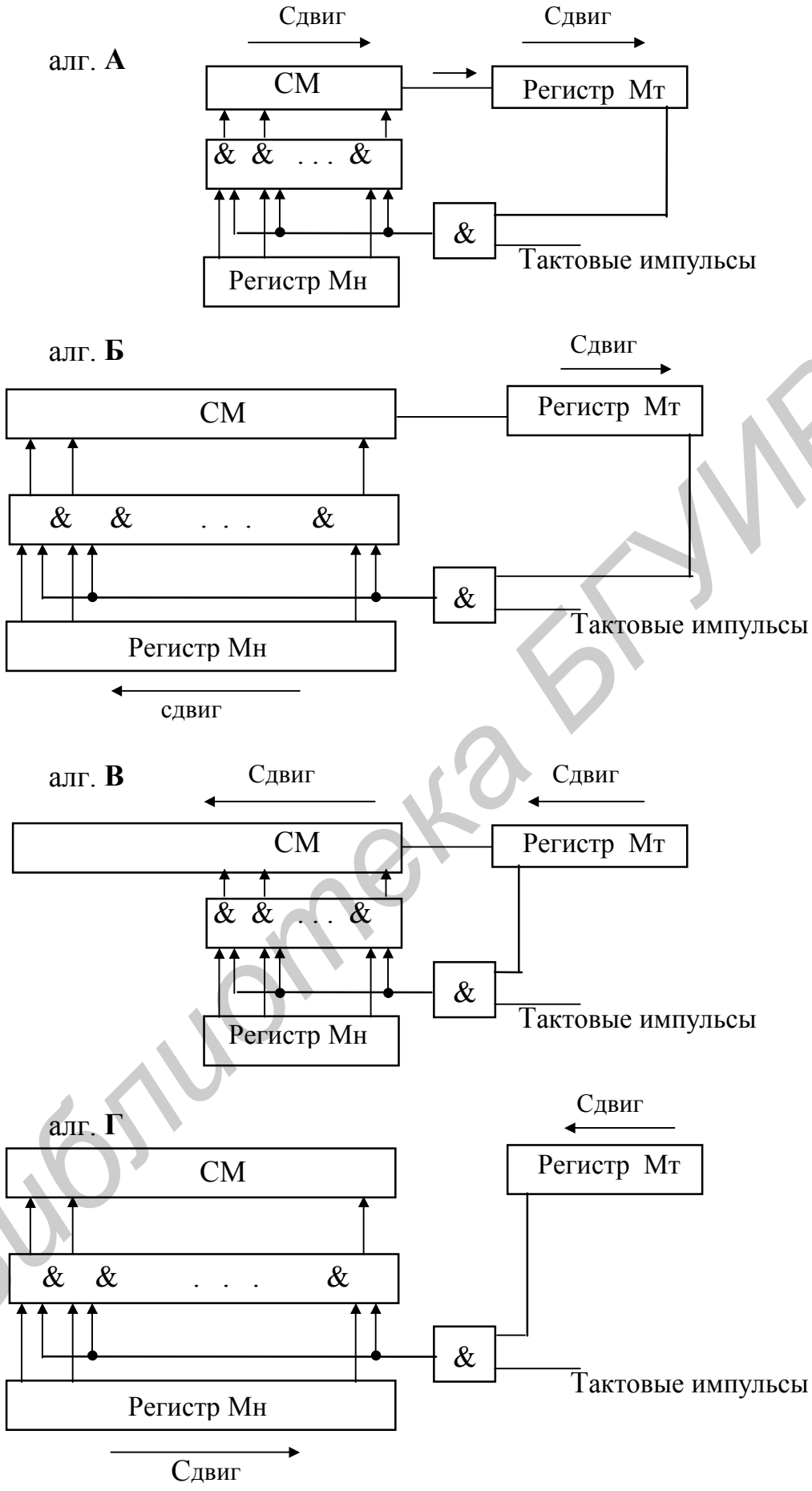


Рис. 6. Структурные схемы устройств умножения по алгоритмам А,Б,В и Г

Время умножения чисел по алгоритму А $t_{умн} = (t_{сл} + t_{сдв}) \cdot n$, где n - число разрядов Мт. Следовательно, сдвиг и сложение нельзя выполнять в одном автоматном такте. Это наглядно показано на рис 7.

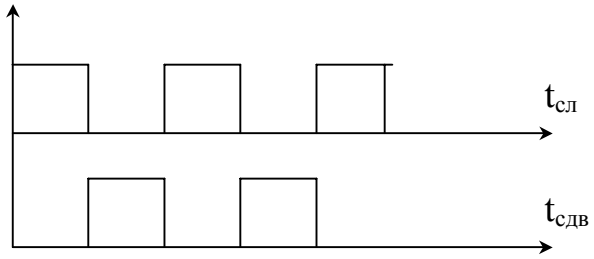


Рис. 7. Временные диаграммы при умножения по алгоритму А

Ускорение операции умножения

Арифметические операции, к числу которых относится умножение, часто встречаются при решении задач на ЭВМ. Умножение является длинной операцией. Временные затраты на умножение чисел в прямых кодах можно оценить по формуле

$$T_{умн} = \sum_{i=1}^{n} (p_i t_{сл} + t_{сдв}), \qquad (3)$$

где p_i – вероятность появления единицы в разрядах множителя; $t_{сл}$ – время формирования очередной частичной суммы; $t_{сдв}$ – время выполнения сдвига числа на один разряд.

Анализируя выражение (3), можно предложить различные пути сокращения величины $T_{умн}$: уменьшение времени на сдвиг, на формирование очередной суммы, уменьшение числа разрядов множителя. Этого можно достигнуть логическими или аппаратными методами. Рассмотрим логические методы ускорения умножения.

Один из наиболее простых способов состоит в том, чтобы при наличии нулевого разряда в множителе не выполнять формирование очередного (нулевого) частичного произведения, не изменяющего содержимое сумматора. В зависимости от используемого алгоритма умножения выполняется сдвиг либо частичной суммы, либо частичного произведения без выполнения суммирования.

Умножение с хранением переносов

Время, затрачиваемое на сложение двоичных чисел, состоит из времени, необходимого для поразрядного сложения, и времени на формирование переноса

$t_{сл} = t_{\oplus} + t_{пер}$.

Поразрядное сложение является элементарной операцией, и время на эту операцию может быть сокращено путем использования более быстродействующих элементов. В то же время если исключить необходимость вы-

полнения межразрядных переносов при сложении, то время умножения уменьшится на $t_{пер}$. Переносы, формируемые при сложении, записываются в отдельный регистр. Содержимое этого регистра добавляется в сумматор вместе с очередым частичным произведением. При этом сложение может выполняться паралельно по всем разрядам. На последнем такте (при умножении на последний разряд множителя) сложение выполняется с учетом межразрядных переносов. В заключение следует отметить, что этот метод используется с алгоритмом А.

Пример: Мн = 0,1011
Мт= 0,1101

0,0000	$\Sigma_0^ч$
0,0000	регистр переносов
$^+$ 0,1011	$П_1^ч = Мн \cdot b_4$
0,1011	$\Sigma_1^ч$
0,0000	регистр переносов
0,**0**101 1	$\Sigma_i^ч \cdot 2^{-1}$
$^+$ 0,0000	$П_2^ч = Мн \cdot b_3$
0,0101 1	$\Sigma_2^ч$
0,0000	регистр переносов
0,**0**010 11	$\Sigma_2^ч \cdot 2^{-1}$
$^+$ 0,1011	$П_3^ч = Мн \cdot b_2$
0,1001 11	$\Sigma_3^ч$
0,0010	регистр переносов
0,**0**100 111	$\Sigma_3^ч \cdot 2^{-1}$
$^+$ 0,1011	$П_4^ч = Мн \cdot b_1$
1,0001 111	$\Sigma_4^ч$
0,**1**000 1111	$\Sigma_4^ч \cdot 2^{-1}$

Умножение на два разряда множителя одновременно

Разбиение множителя на группы длиной k разрядов означает переход к новой системе счисления с основанием 2^k. Если при этом удается сократить количество элементарных действий, выполняемых при умножении (сложение и сдвиги), то сокращается время умножения. Остановимся более подробно на умножении на два разряда множителя за один такт (k=2). Это связано с анализом пар разрядов множителя.

Возможны четыре случая сочетания разрядов множителя: 00, 01, 10, 11. Умножение на каждую из пар разрядов множителя должно выполняться за один такт автоматного времени, то есть в каждом такте умножения должно выпол-

няться не более одного сложения. Рассмотрим умножение на эти пары на примере алгоритма А.

В случае пары 00 необходимо выполнить только сдвиг частичной суммы на два разряда - $\Sigma^{ч}_{i-1} \cdot 2^{-2}$.

Для пары 01 выполняется добавление множимого в сумматор с последующим сдвигом суммы на два разряда - $(\Sigma^{ч}_{i-1} + Мн) \cdot 2^{-2}$.

При наличии пары 10 возможны следующие варианты действий:

а) $(\Sigma^{ч}_{i-1} + Мн + Мн) \cdot 2^{-2}$, то есть в этом случае происходят два сложения, что противоречит требованию;

б) $(\Sigma^{ч}_{i-1} + 2Мн) \cdot 2^{-2}$, в этом случае требуется дополнительный регистр для хранения удвоенного Мн;

в) $(\Sigma^{ч}_{i-1} + Мн \cdot 2^{1}) \cdot 2^{-2}$, что соответствует добавлению к частичной сумме сдвинутого на один разряд влево множимого;

г) $(\Sigma^{ч}_{i-1} \cdot 2^{-1} + Мн) \cdot 2^{-1}$, то есть частичная сумма сдвигается на один разряд вправо до и после добавления к ней множимого.

При умножении на пару 11 (к частичной сумме необходимо добавить утроенное множимое) ее можно представить в виде

$11 = (2^2 - 1)$

$Мн \cdot 11 = Мн \cdot (2^2 - 1) = Мн \cdot 2^2 - Мн$, то есть в текущем такте к частичной сумме добавляется множимое, взятое со знаком минус. Добавление $Мн \cdot 2^2$ реализуется путем увеличения на единицу следующей старшей пары разрядов.

В табл.1 представлены правила преобразования множителя для системы (0,1,1).

Таблица 1

Анализируемая пара разрядов Мт	Перенос из предыдущей пары	Преобразованная пара
00	0	00
01	0	01
10	0	10
11	0	$0\bar{1}$
00	1	01
01	1	10
10	1	$0\bar{1}$
11	1	00

Пример: Мн = 0101

Мт = 11000111

$Мт^{п} = 01\bar{0}1001\bar{0}01$

Умножение будем осуществлять согласно алгоритму А.

$[- Мн]_{доп} = 1.1011$

$2 Мн = 0.1010$

0.0000	$\Sigma_0^ч$
$^+$ 1.1011	$П_1^ч = –Мн$
1.1011	$\Sigma_1^ч$
1.**11**10 11	$\Sigma_1^ч \cdot 2^{-2}$
$^+$ 0.1010	$П_2^ч = 2Мн$
0.1000 11	$\Sigma_2^ч$
0.**00**10 0011	$\Sigma_2^ч \cdot 2^{-2}$
0.**00**00 100011	$\Sigma_3^ч \cdot 2^{-2}$ $(\Sigma_2^ч \cdot 2^{-4})$
$^+$ 1.1011	$П_4^ч = -Мн$
1.1011 100011	$\Sigma_4^ч$
1.**11**10 11100011	$\Sigma_4^ч \cdot 2^{-2}$
$^+$ 0.0101	$П_5^ч = Мн$
0.0011 11100011	$\Sigma_5^ч$
0.**00**00 1111100011	$\Sigma_5^ч \cdot 2^{-2}$

Время умножения на два разряда множителя одновременно

Появление любой из рассматриваемых пар множителей равновероятно. Следовательно, время умножения на два разряда множителя может быть выражено следующим соотношением: $T_{умн}^{2\,разр} = (n/2 + 1)\,[0{,}75\cdot(t_{сл} + t_{сдв}) + (0{,}25\cdot t_{сдв}]$, где n – количество разрядов множителя.

Умножение на четыре разряда одновременно

В этом случае с помощью приема, аналогичного приему, использованному в случае умножения на два разряда одновременно, можно рассматривать сразу тетраду двоичных разрядов. Может быть выведено общее правило сокращенного умножения:

1. Если цифра множителя $b_{i-1} < r/2$, то $\Sigma_{i-1}^ч + П_i^ч$, где $П_i^ч = Мн\ b_i$.

2. Если цифра множителя $b_{i-1} \geq r/2$, то $\Sigma_{i-1}^ч + П_i^ч$, где $П_i^ч = [Мн\cdot(r - b_i)]_{доп}$.

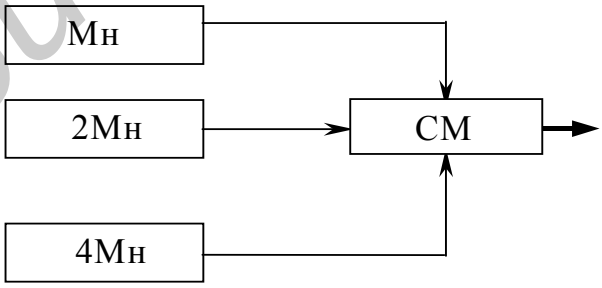


Рис. 8. Схема формирования сомножителей при умножении на четыре разряда множителя

Анализ четырех двоичных разрядов одновременно дает возможность осуществить сдвиг на четыре разряда за один такт.

Пример: Мн = 011

Мт = С49

$Мт^п = 1457$

Можно заранее заготовить кратные множители: Мн,

2Мн, 4Мн, поместив их в дополнительные регистры. Это позволит сократить время, необходимое для формирования частичного произведения $П_i^ч$, равного от нуля до семи множимых.

$[+7Мн]_{доп} = 0.10101$ $[+5Мн]_{доп} = 0.01111$ $[+4Мн]_{доп} = 0.01100$
$[-7Мн]_{доп} = 1.01011$ $[-4Мн]_{доп} = 1.10100$

0.00000	$Σ_0^ч$
$^+$ 1.01011	$П_1^ч = -7Мн$
1.01011	$Σ_1^ч$
1.**1111**0 1011	$Σ_1^ч \cdot 2^{-4}$
$^+$ 0.01111	$П_2^ч = +5Мн$
0.01101 1011	$Σ_2^ч$
0.**0000**0 1101 1011	$Σ_2^ч \cdot 2^{-4}$
$^+$ 1.10100	$П_3^ч = -4Мн$
1.10100 1101 1011	$Σ_3^ч$
1.**1111**1 0100 1101 1011	$Σ_3^ч \cdot 2^{--4}$
$^+$ 0.00011	$П_4^ч = +Мн$
0.00010 0100 1101 1011	$Σ_4^ч$
0.**0000**0010 0100 1101 1011	$Σ_4^ч \cdot 2^{--4} = Мн \cdot Мт$

Умножение в дополнительных кодах

Умножение чисел в дополнительных кодах может выполняться по любому из рассмотренных выше четырех алгоритмов, но отличается тем, что для получения верного результата необходимо вводить поправки. Умножение правильных дробей и целых чисел имеет некоторые различия.

Рассмотрим вначале **умножение дробных чисел**. Возможны четыре случая знакосочетания сомножителей.

1) Мн>0,
Мт>0.

В этом случае дополнительный код сомножителей совпадает с прямым кодом и умножение выполняется по правилам умножения в прямых кодах, в результате чего получается верный результат.

2) Мн>0,
Мт<0.

Так как Мн и Мт имеют разные знаки, то результат будет иметь отрицательный знак. Следовательно, результат должен быть представлен в дополнительном коде $[Мн \cdot Мт]_{доп}=2-Мн \cdot Мт$. Для формирования произведения выполним умножение Мн на Мт′ (дополнение):

$Мн \cdot Мт' = Мн \cdot (1 - Мт) = Мн - Мн \cdot Мт.$

Таким образом, погрешность Δ умножения равна разности $[Мн \cdot Мт]_{доп}$ и $Мт \cdot Мн'$

$\Delta = 2 - Мн \cdot Мт - Мн + Мн \cdot Мт = 2 - Мн = [-Мн]_{доп} = [\,[Мн]_{доп}]_{доп}$

и должна быть внесена в полученный результат в качестве поправки.

Пример: Мн = 0,1011 (алгоритм умножения А)

Мт = - 0,1101

$\Delta = [-Мн]_{доп} = 1,0101$

$[-Мт]_{доп} = 1,0011$

0,0000	$\Sigma_0^ч$
$^{+}$ 0,1011	$П_1^ч = Мн \cdot b_4$
0,1011	$\Sigma_1^ч$
0,**0**101 1	$\Sigma_1^ч \cdot 2^{-1}$
$^{+}$ 0,1011	$П_2^ч = Мн \cdot b_3$
1,0000 1	$\Sigma_2^ч$ (произошло переполнение)
0,**1**000 01	$\Sigma_2^ч \cdot 2^{-1}$ (коррекция)
0,**0**10 0001	$\Sigma_3^ч \cdot 2^{-1}$ $(\Sigma_2^ч \cdot 2^{-1}) \cdot 2^{-1}$
0,**0**010 0001	$\Sigma_4^ч \cdot 2^{-1}$ $((\Sigma_2^ч \cdot 2^{-1}) \cdot 2^{-1}) \cdot 2^{-1}$
$^{+}$ 1,0101	$\Delta = [-Мн]_{доп}$ (поправка)
1,0111 0001	$\Sigma_5^ч = [Мн \cdot М_т]_{доп}$
- 0,1000 1111	$Мн \cdot Мт$

В этом примере, а также в некоторых примерах, следующих ниже, происходит переполнение разрядной сетки, в результате которого искажаются и знаковая и значащая часть числа. Для коррекции (восстановления верного значения) производится немодифицированный сдвиг. В результате этого знаковый разряд сдвигается в значащую часть, а знаковый разряд меняет свое значение на противоположное.

3) $Мн < 0$,

$Мт > 0$.

Аналогично предыдущему случаю

$[Мн \cdot Мт]_{доп} = 2 - Мн \cdot Мт$,

$Мн' \cdot Мт = (1 - Мн) \cdot Мт = Мт - Мн \cdot Мт.$

Таким образом, погрешность умножения равна разности $[Мн\ Мт]_{доп}$ и $Мт \cdot Мн'$.

$\Delta = 2 - Мн \cdot Мт - Мт + Мн \cdot Мт = 2 - Мт = [-Мт]_{доп} = [\,[Мт]_{доп}]_{доп.}$

Использовать этот вариант неудобно, так как нужно вводить поправку $[-Мт]_{доп}$ в конце умножения, а в результате сдвигов Мт постепенно исчезает на регистре множителя и для поправки нужно либо вводить дополнительный регистр, либо вносить поправку в сумматор на первом такте умножения.

При этом знакосочетании возможно умножение без ввода поправки. Рассмотрим этот вариант на примере умножения по алгоритму Г (это справедливо и для других алгоритмов).

$Мн \cdot Мт = A \cdot B = [\, A \cdot b_1 \cdot 2^{-1} \,]_{доп} + [\, A \cdot b_2 \cdot 2^{-2} \,]_{доп} + \ldots + [A \cdot b_n \cdot 2^{-n} \,]_{доп}$.

На основании теоремы, доказывающей что сумма дополнительных кодов есть дополнительный код суммы, получаем:

$[\, A \cdot b_1 \cdot 2^{-1} + A \cdot b_2 \cdot 2^{-2} + \ldots + A \cdot b_n \cdot 2^{-n} \,]_{доп} = [Мн \cdot Мт]_{доп}$.

В этом случае поправка вводится автоматически на каждом этапе умножения.

Пример: Мн = -0,1011
Мт = 0,1101
$[Мн]_{доп}$ = 1,0101
$b_1 \ldots b_4$

0,00000000	$\Sigma_0^ч$
+ 1,**1**0101000	$П_1^ч = [Мн \cdot b_1 \cdot 2^{-1}]_{доп}$
1,10101000	$\Sigma_1^ч$
+ 1,**11**010100	$П_2^ч = [Мн \cdot b_2 \cdot 2^{-2}]_{доп}$
1,01111100	$\Sigma_2^ч$
+ 1,**1111**0101	$П_3^ч = [Мн \cdot b_4 \cdot 2^{-4}]_{доп}$
1,01110001	$[Мн \cdot Мт]_{доп}$
-0,10001111	$Мн \cdot Мт$

4) Мн<0,
Мт<0.

$Мн \cdot Мт = 2 - [Мн \cdot Мт]_{доп}$

$[Мн]_{доп} \cdot (1 - Мт) = [Мн]_{доп} - [Мн]_{доп} \cdot Мт = [Мн]_{доп} - [Мн \cdot Мт]_{доп}$

$\Delta = 2 - [Мн \cdot Мт]_{доп} - [Мн]_{доп} + [Мн \cdot Мт]_{доп} = 2 - [Мн]_{доп} = [[Мн]_{доп}]_{доп}$

Пример: Мн = - 0,1011 умножение будем выполнять
Мт = - 0,1101 по алгоритму умножения Г
$[Мн]_{доп}$ = 1,0101
$[Мт]_{доп}$ = 1,0011
$\Delta = [-Мн]_{доп}$ = 0,1011

0,00000000	$\Sigma_0^ч$
+ 1,**111**01010	$П_1^ч = [Мн \cdot b_3 \cdot 2^{-3}]_{доп}$
1,11101010	$\Sigma_1^ч$
+ 1,**1111**0101	$П_4^ч = [Мн \cdot b_4 \cdot 2^{-4}]_{доп}$
1,11011111	$\Sigma_4^ч$
+ 0,**1011**0000	Δ (поправка)
0,10001111	$[Мн \cdot Мт]_{доп}$

Далее коротко остановимся на **умножении целых чисел**. При представлении целых чисел в дополнительном коде знаковый разряд входит в число n разрядов. Следовательно, при умножении целых чисел (в отличие от дробных) в дополнительных кодах знаковый разряд участвует в умножении наряду со значащими. То есть умножение ведется на $[Мт]_{доп}$, а не на Мт′.

1) Мн > 0,
   Мт > 0.

Как отмечалось выше, в этом случае умножение выполняется по правилам умножения чисел в прямых кодах.

2) Мн>0,
   Мт<0,
$[Мт]_{доп} = 2^n - Мт.$

Так как сомножители имеют разные знаки, то произведение Мн·Мт<0, следовательно, $[Мн·Мт]_{доп}=2^{2n}$ - Мн·Мт. Однако при умножении $Мн·[Мт]_{доп}$ получается $Мн·(2^n-Мт) =2^n$ Мн - Мн·Мт. Следовательно, погрешность в этом случае равна $\Delta=2^{2n}–Мн·Мт–2^n$ Мн+Мн·Мт $=2^{2n}–2^n$ Мн $=2^{2n}$ $[–Мт]_{доп}=2^{2n}[$ $[Мт]_{доп}]_{доп}$.

Пример: Мн = +110
Мт = -101
$[Мн]_{доп}$ = 0.110
$[Мт]_{доп}$ = 1.011
$\Delta = [-Мн]_{доп}$ = 1.010

0.000	$\Sigma_0^ч$	
+ 0.110	$П_1^ч = Мн·b_4$	
0.110	$\Sigma_1^ч$	
0.**0**11 0	$\Sigma_1^ч·2^{-1}$	
+ 0.110	$П_2^ч = Мн·b_3$	
1.001 0	$\Sigma_2^ч$	(возникло переполнение)
0.**1**00 10	$\Sigma_2^ч·2^{-1}$	(коррекция)
0.**01**0 010	$\Sigma_3^ч·2^{-1}$	
+ 0.110	$П_4^ч = Мн·b_1$	
1.000 010	$\Sigma_4^ч$	(возникло переполнение)
0.**1**00 0010	$\Sigma_4^ч·2^{-1}$	(коррекция)
+ 1.010	Δ	(поправка)
1.110 0010	$[Мн·Мт]_{доп}$	
- 001 1110	Мн·Мт	

3) Мн<0,
   Мт>0.

Здесь, как и при умножении дробных чисел, возможны два случая:

а) с вводом поправки в получаемое произведение

$[Мн]_{доп} = 2^n - Мн$.

Как и ранее, требуется получить $[Мн \cdot Мт]_{доп} = 2^{2n} - Мн \cdot Мт$. Получаем

$(2^n - Мн) \cdot Мт = 2^n \cdot Мт - Мн \cdot Мт$.

$\Delta = 2^{2n} - Мн \cdot Мт - 2^n \cdot Мт + Мн \cdot Мт = 2^n(2^n - Мт) = [Мт]_{доп} \cdot 2^n$;

б) вариант без ввода поправки рассмотрим применительно к алгоритму умножения **Г** (как и ранее это справедливо и для других алгоритмов):

$Мн \cdot Мт = A \cdot B = [A \cdot b_1 \cdot 2^{-1}]_{доп} + [A \cdot b_2 \cdot 2^{-2}]_{доп} + ... + [A \cdot b_n \cdot 2^{-n}]_{доп} =$

$= [A \cdot b_1 \cdot 2^{-1} + A \cdot b_2 \cdot 2^{-2} + ... + A \cdot b_n \cdot 2^{-n}]_{доп} = [Мн \cdot Мт]_{доп}$.

Пример: $Мн = -110$

$Мт = 101$

$[Мн]_{доп} = 1.010$

$[Мт]_{доп} = 0.101$

$b_1 \ldots b_4$

0.0000000 $\quad \Sigma_0^ч$

\+ 0.**0**000000 $\quad \Pi_1^ч = [M_Н \cdot b_1]_{доп} \cdot 2^{-1}$

0.0000000 $\quad \Sigma_1^ч$

\+ 1.**11**01000 $\quad \Pi_2^ч = [M_Н \cdot b_2]_{доп} \cdot 2^{-2}$

1.1101000 $\quad \Sigma_2^ч$

\+ 1.**1111**010 $\quad \Pi_4^ч = [M_Н \cdot b_4]_{доп} \cdot 2^{-4}$

1.1100010 $\quad \Sigma_4^ч$

4) $Мн < 0$

$Мт < 0$

При этом сочетании знаков сомножителей в результате должно быть получено:

$[Мн]_{доп} = 2^n - Мн$,

$[Мт]_{доп} = 2^n - Мт$,

$Мн \cdot Мт = 2^{2n} - [Мн\ Мт]_{доп}$.

При умножении $[Мн]_{доп} \cdot [Мн]_{доп}$ получается:

$$[Мн]_{доп} \cdot [Мн]_д = [Мн]_{доп}(2^n - Мт) = 2^n [Мн]_{доп} - \underbrace{[Мн]_{доп} \cdot Мт}_{[Мн \cdot Мт]_{доп}}$$

Пример: $Мн = -110$

$Мт = -101$

$[Мн]_{доп} = 1.010$

$[Мт]_{доп} = 1.011$

$b_1 \ldots b_4$

$\Delta = [[Мн]_{доп}]_{доп} = 0.110$

При умножении используем алгоритм Г.

0.0000000 $\Sigma_0^{ч}$

<u>1.**1**010000</u> $\Pi_1^{ч} = [Мн \cdot b_1]_{доп} \cdot 2^{-1}$

1.1010000 $\Sigma_1^{ч}$

<u>1.**111**0100</u> $\Pi_1^{ч} = [Мн \cdot b_3]_{доп} \cdot 2^{-3}$

1.1000100 $\Sigma_3^{ч}$

<u>1.**1111**010</u> $\Pi_1^{ч} = [Мн \cdot b_4]_{доп} \cdot 2^{-4}$

1.0111110 $\Sigma_4^{ч}$

<u>0.110</u> Δ (поправка)

0.0011110 Мн Мт

Умножение на два разряда множителя в дополнительных кодах

В отличие от умножения в прямых кодах преобразованию подвергается не только пара 11, но также и пара 10. Это позволяет упростить анализ кода и формирование преобразованного множителя.

i	i+1	i	i+1
00	11	00	10
01	01	01	10
+4Мн	-Мн	+4Мн	-2Мн

В табл. 2 приведены преобразования i и i+1 пары множителя.

Таблица 2

i-я пара		i+1-я пара		Преобраз. пара		Алгоритм	
x	y	z	–	x′	y′	A	Г
0	0	0	-	0	0	1	6
0	0	1	-	0	1	2	7
0	1	0	-	0	1	2	7
0	1	1	-	1	0	3	8
1	0	0	-	$\overline{1}$	$\overline{0}$	4	9
1	0	1	-	0	$\overline{1}$	5	10
1	1	0	-	0	$\overline{1}$	5	10
1	1	1	-	0	0	1	6

Действия, осуществляемые при выполнении, например, алгоритмов А и Г, следующие:

(1) $\sum_{i-1}^{r} \cdot 2^{-2}$

(2) $(\sum_{i-1}^{r} + Мн) \cdot 2^{-2}$

(3) $(\sum_{i-1}^{r} \cdot 2^{-1} + Мн) \cdot 2^{-1}$

(4) $(\sum_{i-1}^{r} \cdot 2^{-1} - Мн) \cdot 2^{-1}$

(6) $Мн \cdot 2^{-2}$

(7) $\sum_{i-1}^{r} + Мн \cdot 2^{-2}$

(8) $\sum_{i-1}^{r} + 2Мн \cdot 2^{-2}$

(9) $\sum_{i-1}^{r} - 2Мн \cdot 2^{-2}$

(5) $(\sum_{i-1}^{r} - Мн) \cdot 2^{-2}$ (10) $\sum_{i-1}^{r} - Мн \cdot 2^{-2}$

Если i+1 пара имеет старшую цифру 1, то умножение на эту пару будет соответствовать вычитанию одного или двух множителей. На рис. 9 приведена логическая схема для формирования сигнала выполняемого действия при анализе пары (xy) и старшего разряда (z) предыдущей пары при умножении чисел согласно алгоритму Г.

Как было показано выше, при умножении чисел в дополнительных кодах в общем случае необходимо вводить поправку для получения верного произведения. Однако при умножении на два разряда множителя этого выполнять не требуется. Если на умножение поступает отрицательный множитель, то при преобразовании его старшей пары происходит формирование дополнительной пары:

Мт = -10110

$[Мт]_{доп}$= 1.0 $1\bar{0}$ $1\bar{0}$

$[Мт]_{доп}^{п}$= 01 $0\bar{1}$ $0\bar{1}$ $\bar{1}0$

При умножении на дополнительную пару ($0\bar{1}$) происходит вычитание одного множителя, а затем он должен быть добавлен в качестве поправки. Таким образом, тратится два такта на выполнение взаимоисключающих операций.

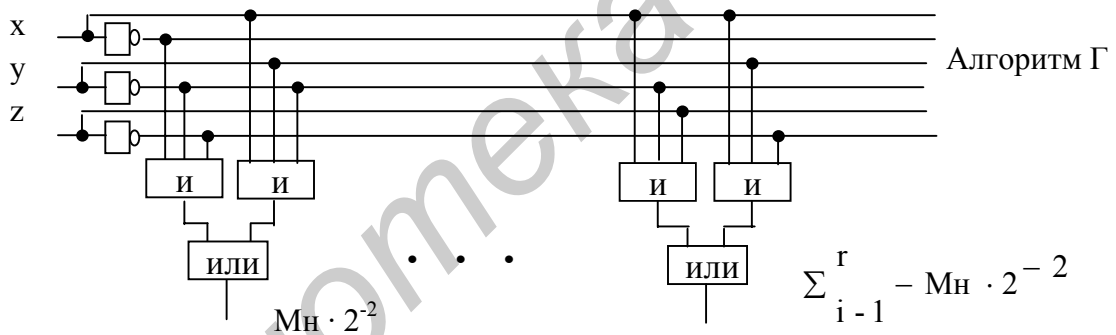


Рис. 9. Логическая схема формирования сигнала выполняемого действия

Пример:

Мн = + 0101 умножение выполним согласно

Мт = + 1100111 алгоритму Г

$[Мн]_{доп}$ = 0.0101

$[Мт]_{доп}$ = 0.1 10 01 11

Анализ пар Мт можно производить начиная от старших (при умножении по алгоритмам В и Г) и от младших разрядов (алгоритмы А и Б).

$[Мт]_{доп}^{пр}$ = 10 $\bar{1}$0 10 $0\bar{1}$

$[-2Мн]_{доп}$ = 1.0110 $[2Мн]_{доп}$ = 0.1010

0.0000 00000000 $\Sigma_0^ч$

\+ 0.**00**10 10000000 $П_1^ч = 2М_Н \cdot 2^{-2}$

0.0010 10000000 $\Sigma_1^ч$

$^{+}$ 1.**1111** 01100000	$\Pi_2^ч = -2M_Н \cdot 2^{-4}$
0.0001 11100000	$\Sigma_2^ч$
$^{+}$ 0.**0000 00**101000	$\Pi_3^ч = 2M_Н \cdot 2^{-6}$
0.0010 00001000	$\Sigma_3^ч$
$^{+}$ 1.**1111 1111**1011	$\Pi_4^ч = -M_Н \cdot 2^{-8}$
0.0010 00000011	$\Sigma_4^ч = [Мн\ Мт]_{доп}$

Пример:

Мн= - 0101 умножение выполним согласно
Мт = -1100111 алгоритму Б
$[Мн]_{доп}$ = 1.1011
$[Мт]_{доп}$ = 1.0 01 10 01
$[Мт]^{пр}_{доп}$ = 1.0 10 10 01
$[2Мн]_{доп}$ = 1.0110 $[-2Мн]_{доп}$ = 0.1010

0.000000 0000	$\Sigma_0^ч$
$^{+}$ 1.111111 1011	$\Pi_1^ч = Мн$
1.111111 1011	$\Sigma_1^ч$
$^{+}$ 0.000010 10**00**	$\Pi_2^ч = -2Мн \cdot 2^2$
0.000010 0011	$\Sigma_2^ч$
$^{+}$ 1.110110 **0000**	$\Pi_3^ч = 2Мн \cdot 2^4$
1.111000 0011	$\Sigma_3^ч$
$^{+}$ 0.101000 **0000**	$\Pi_4^ч = -2Мн \cdot 2^6$
0.100000 0011	$\Sigma_4^ч = Мн \cdot Мт$

Матричные методы умножения

Кроме рассмотренных методов ускоренного умножения существуют методы умножения, основанные на использовании матриц промежуточных результатов.

Пусть имеем сомножители:

Мн = A = a_n ... a_2 a_1
Мт = B = b_n ... b_2 b_1

$$
\begin{array}{r}
A = a_n \ldots a_2\ a_1 \\
{}^{*}\ B = b_n \ldots b_2\ b_1 \\
\hline
a_nb_1 \ldots a_3b,\ a_2b_1,\ a_1b_1 \\
{}^{+}a_nb_2 \quad . \ . \ . \quad a_2b_2,\ a_1b_2 \\
+ \qquad . \ . \ . \\
a_nb_n \ldots a_2b_n,\ a_1b_n \\
\hline
C = C_{2n} \ . \quad . \quad . \quad . C_2 \quad C_1
\end{array}
$$

Рассмотрим схему умножения чисел согласно алгоритму Б. Данная схема умножения может быть представлена в виде матрицы (табл.3).

Таблица 3

	a_n	. . .	a_2	a_1
b_1	$a_n b_1$	. . .	$a_2 b_1$	$a_1 b_1$
b_2	$a_n b_2$	. . .	$a_2 b_2$	$a_1 b_2$
. . .		. . .		
b_n	$a_n b_n$	. . .	$a_2 b_n$	$a_1 b_n$

Каждый элемент $a_i b_j$ (i, j = 1, n) принимает значение 0 или 1. Произведение A·B может быть получено, если суммировать элементы матрицы (по диагонали).

$$+ \ldots \left| \begin{matrix} a_3 b_1 \\ a_2 b_2 \\ a_1 b_3 \end{matrix} \right| \left| \begin{matrix} a_2 b_1 \\ a_1 b_2 \end{matrix} \right|$$

Для суммирования по столбцам могут быть использованы счетчики. Однако при достаточно большом значении величины n потребуются счетчики с большим числом входов, что существенно увеличит время сложения. Но этот принцип умножения может быть реализован на устройствах, имеющих не более трех входов. В качестве их могут быть использованы одноразрядные двоичные сумматоры и полусумматоры.

На рис. 10 приведена структурная схема устройства умножения для реализации матричного алгоритма.

Реализация методов матричного умножения требует большего количества оборудования, чем метод последовательного умножения, и дает больший выигрыш во времени. В связи с увеличением степени интеграции элементной базы ограничения по количеству оборудования становятся не столь строгими.

Машинные методы деления

Деление - простое многократное вычитание делителя вначале из делимого, затем из остатков. Введем обозначения, используемые ниже: Дм - делимое, Дт - делитель, A_i – очередной (i-й) остаток, Чт – частное. Известны два основных подхода к операции деления:

- с восстановлением остатков;
- без восстановления остатков.

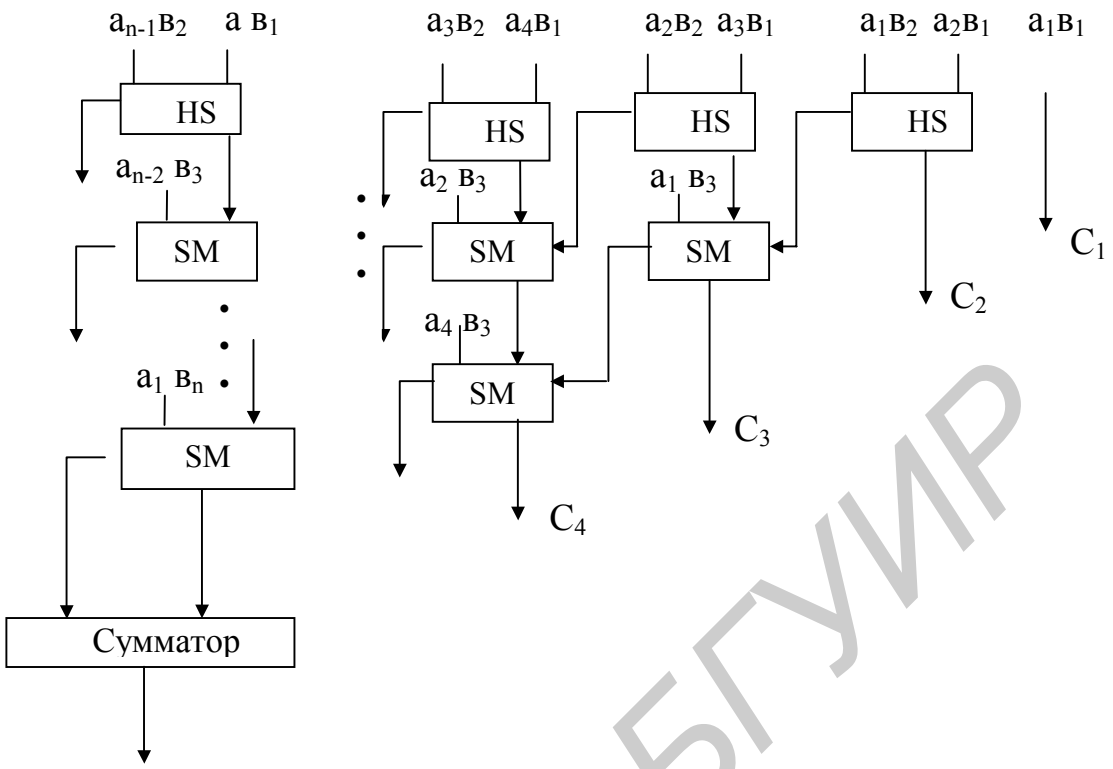


Рис. 10. Структурная схема матричного умножения

Независимо от метода деления после каждого вычитания делителя формируется цифра частного. Операция деления является операцией, дающей не всегда точный результат. Поэтому признаком окончания операции деления может быть достижение заданной точности (по числу сдвиговых сигналов). Если при выполнении деления получен нулевой i-й остаток, то деление прекращается и в оставшиеся разряды частного записываются нули. Первым шагом деления двух чисел машиной является пробное вычитание, выявляющее соотношение между делимым и делителем. При делении в случае переполнения следует: для чисел с фиксированной запятой процесс остановить, с плавающей запятой продолжить до конца, а потом, после получения последней n-й цифры частного, число сдвигается вправо на один разряд с добавлением единицы к порядку, равному разности порядков делимого и делителя.

Деление чисел в прямых кодах

Алгоритм деления с восстановлением остатка состоит в следующем.

1. Выполняется пробное вычитание с формированием первого остатка $A_1=[Дм]_{доп}+[-Дм]_{доп}$. Далее, если $A_1 < 0$, то в первый разряд, расположенный слева от запятой, заносится ноль (0), иначе единица (1) – переполнение и переход к пункту 5.

2. Если $A_i < 0$, то восстанавливаем предыдущий остаток $A_i=A_i+[Дм]_{доп}$.

3. Формирование очередного остатка. Если $A_{i+1}=A_i\cdot 2+[-Дм]_{доп}$, то в очередной разряд частного справа от запятой записывается ноль (Чт(n)=0), иначе записывается единица (Чт(n)=1).

4. Если достигнута заданная точность частного или получен нулевой ос-

таток A_{i+1}, то процесс деления окончен и осуществляется переход к пункту 5, иначе переходим к пункту 2 алгоритма.

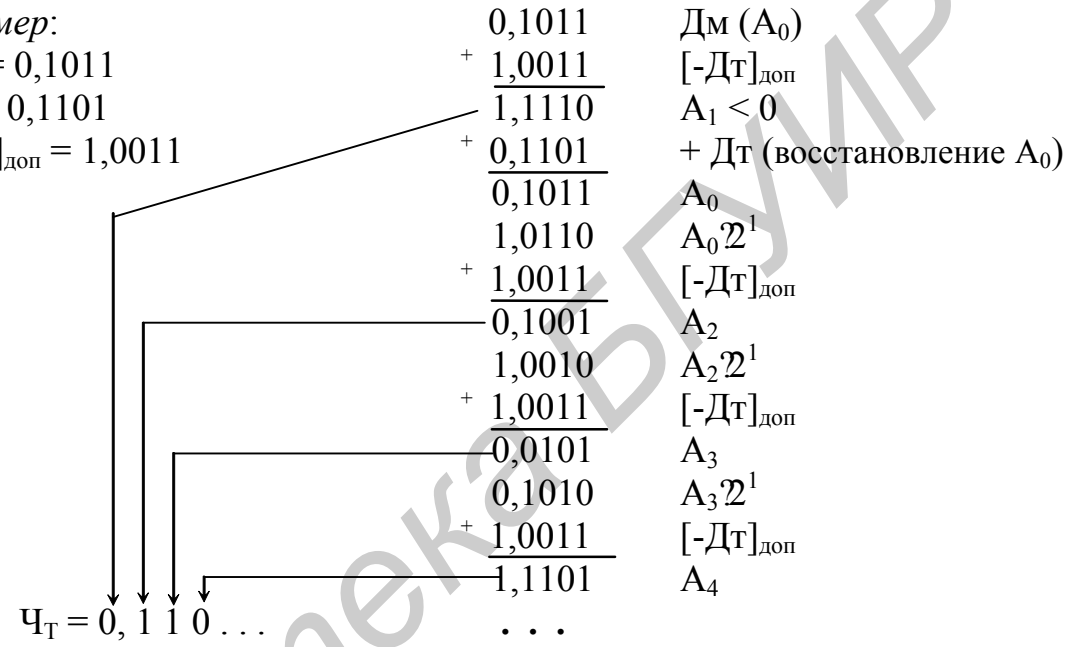
5. Окончание алгоритма.

Из рассмотренного алгоритма видно следующее:

1) необходимо затрачивать время на восстановление остатка;

2) процесс деления нерегулярный, в зависимости от делимого и делителя частное будет содержать нулей больше или меньше, и чем больше нулей, тем больше требуется времени на восстановление остатков.

Рассмотрим пример деления чисел.



Как видно из примера, для получения остатка A_{i+2} необходимо выполнить:

$$A_{i+2} = (A_{i+1} + Д_Т) \cdot 2^1 - Д_Т = A_{i+1} \cdot 2^1 + 2Д_Т - Д_Т = A_{i+1} \cdot 2^1 + Д_Т$$

Из этого следует, что восстанавливать остаток необязательно. Достаточно сдвинуть полученный отрицательный остаток влево на один разряд и добавить делитель. Это является основой алгоритма для выполнения деления без восстановления остатка.

Алгоритм деления без восстановления остатка:

1. Выполняется пробное вычитание с формированием первого остатка $A_1=[Дм]_{доп}+[-Дм]_{доп}$. Далее, если $A_1 < 0$, то в первый разряд, расположенный слева от запятой, заносится ноль (0), иначе единица (1) – что является признаком переполнение и ос4уществляется переход к пункту 5.

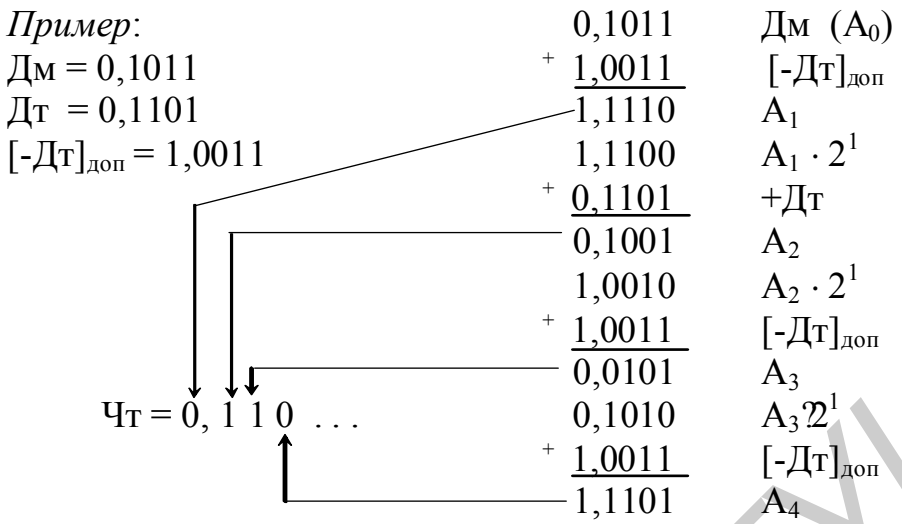
2. Формирование очередного остатка. Если $A_i < 0$, то $A_{i+1}=A_i \cdot 2+[Дм]_{доп}$, иначе $A_{i+1}=A_i \cdot 2+[-Дм]_{доп}$.

3. Если $A_{i+1} < 0$, то в очередной разряд частного справа от запятой записывается ноль (Чт(n)=0), иначе записывается единица (Чт(n)=1).

4. Если достигнута заданная точность частого или получен нулевой остаток A_{i+1}, то процесс деления окончен и осуществляется переход к пункту 5,

иначе переходим к пункту 2 алгоритма.

5. Окончание алгоритма.



Деление чисел в дополнительных кодах

При делении чисел знаковая и значащая части частного формируются раздельно. Знак частного формируется согласно формуле

Знак Чт = Знак Дм ⊕ Знак Дт.

Основой деления чисел в дополнительных кодах является деление без восстановления остатка. В отличие от деления в прямых кодах, здесь как для определения цифры частного, так и для определения действия сравнивается знак делимого (остатка) со знаком делителя.

Ниже приведен алгоритм деления чисел в дополнительных кодах.

1. Выполняется пробное вычитание: если знак Дм ≠ знаку Дт, то первый остаток $A_1=[Дм]_{доп}+[Дм]_{доп}$, иначе $A_1=[Дм]_{доп}+[-Дм]_{доп}$. Далее формируется первый разряд, расположенный слева от запятой - ноль (0), если знак A_1 ≠ знаку Дт, иначе единица (1).

2. Формирование очередного остатка. Если знак A_i ≠ знаку Дт, то $A_{i+1}=A_i\cdot 2+[Дм]_{доп}$, иначе $A_{i+1}=A_i\cdot 2+[-Дм]_{доп}$.

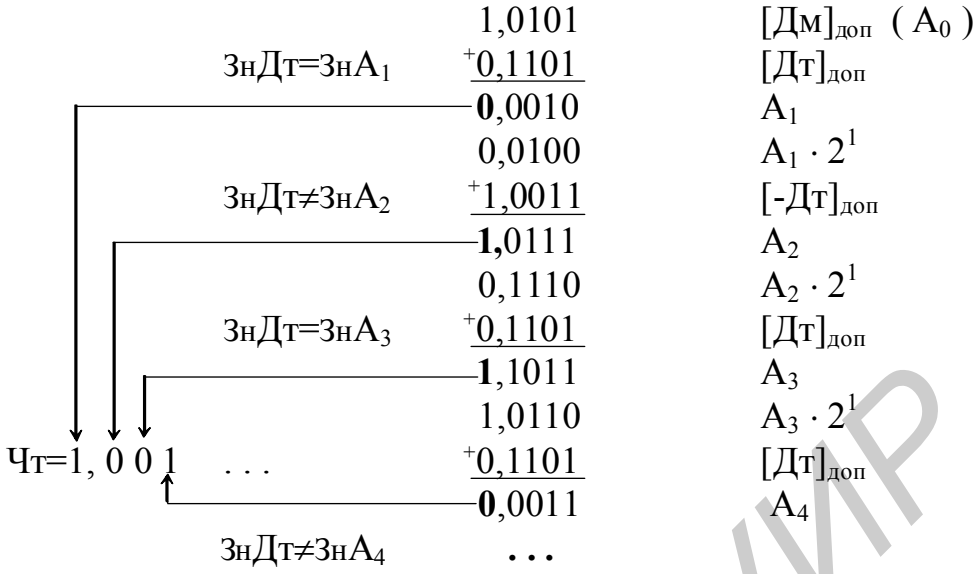
3. Если знак A_{i+1} ≠ знаку Дт, то в очередной разряд частного справа от запятой заносится ноль (Чт(n)=0), иначе - единица (Чт(n)=1).

4. Если достигнута заданная точность частого или получен нулевой остаток A_{i+1}, то процесс деления окончен, иначе переходим к пункту 2 алгоритма.

Пример: Дм = - 0,1011 $[\ Дм\ ]_{доп}$ = 1,0101

Дт = 0,1101 $[\ Дт\ ]_{доп}$ = 0,1101 $[-Дт\ ]_{доп}$ = 1,0011

На деление Дм и Дт поступают в дополнительном коде



Методы ускорения деления

Логические методы основываются на анализе остатка, по виду которого можно сформировать несколько цифр частного в пределах одного такта.

При этом Дт выбирается (формируется) таким образом, чтобы после запятой шла единица, то есть он нормализован. Если очередной остаток получился настолько мал, что после запятой следует r+1 нулей, то в частное может быть записано r ”0” или ”1”, а остаток может быть сдвинут на r разрядов влево.

Итак, для осуществления названных методов ускорения кроме устройства управления делением требуется логическая схема, осуществляющая две функции:

1) сдвиг модулей делителя и делимого до тех пор, пока у модуля делителя после запятой не останется ни одного нуля;

2) выявление остатков вида 0,0…01 или 1,1…10.

Степень сложности логической схемы определяется количеством разрядов, участвующих в косвенном сравнении модулей делителя и делимого (остатков).

Обычно реализация логических методов ускорения при делении несколько сложнее, чем при умножении. При делении необходимо либо каждый остаток переводить в прямой код, либо, если остаток оставить в дополнительном коде, анализировать два разряда одновременно 0,0… или 1,1… .

Двоично-десятичные коды

Пусть число А представлено в системе счисления с основанием r:

$$A_r = \sum_{i=1}^{n} a_i \cdot r^{p-i} \qquad (r \neq 2^k,\ k = 1,2,3,\ldots).$$

Цифры a_i будем представлять двоичными разрядами $d_1,d_2,\ldots,d_m$. Каждому двоичному разряду припишем веса $p_1,p_2,\ldots,p_m$. Тогда каждый разряд a_i числа А

будет иметь вид $a_i = \sum_{l=1}^{m} d_l \cdot p_l$, а все число

$$A_r = \sum_{i=1}^{n} \left(\sum_{l=1}^{m} d_l \cdot p_l \right) \cdot r^{p-i}, \qquad (4)$$

где n и m определяют общее число двоичных разрядов.

Если каждый разряд числа имеет вес и при $r \neq 2^k$ не выполняется равенство $p_k = r \cdot p_{k-1}$, то системы принято называть *взвешенными*. Количество разрядов m должно удовлетворять выражению $m \geq \log_2 r$. Если десятичное число записано в виде (4), то будем говорить, что число представлено в двоично-десятичном коде. Наибольшее распространение из них получили коды, в которых десятичная цифра представляется двоичной тетрадой (BCD-коды). Существует множество способов кодирования десятичных цифр. Существенным при этом является простота представления инверсных кодов и простота выделения (формирования) сигнала переноса из цифры в цифру.

Сформулируем набор требований, позволяющих упростить выполнение арифметических операций и операций перевода чисел.

- Четность, состоит в том, что четным десятичным цифрам соответствуют только четные двоичные коды и наоборот. Это обеспечивает эффективность операций округления, умножения и деления чисел в BCD-кодах.
- Дополняемость, заключается в том, что сумма двоичного кода и инверсного ему кода любой десятичной цифры должна быть равна 9. Это обеспечивает эффективность операции алгебраического сложения в BCD-кодах.
- Упорядоченность, то есть большей десятичной цифре соответствует большая тетрада и наоборот.
- Единственность представления десятичной цифры двоичной тетрадой.
- Взвешенность, то есть каждому разряду двоичного представления десятичной цифры поставлен в соответствие вес. Это обеспечивает эффективность всех арифметических и логических операций в BCD-кодах.

Если каждая десятичная цифра кодируется соответствующим двоичным эквивалентом, то такое кодирование называется *кодом прямого замещения*.

BCD-код – код, взаимодополняемый до 15. Это создает некоторые неудобства при суммировании чисел - ввод поправки в некоторых случаях. В то же время этот код имеет одно существенное достоинство – аддитивность: *сумма кодов равна коду суммы*.

0011 код 3
0101 код 5
1000 код 8

Основной недостаток этого кода заключается в том, что инверсия какой-либо цифры оказывается цифрой, дополняющей данную цифру до 15, а не до 9.

$a = 0100 \rightarrow$
$\bar{a} = 1011 \rightarrow 11$, то $a + \bar{a} = 1111 = 15$

В табл. 4 показаны различные способы кодирования десятичных цифр.

Таблица 4

Десятичные цифры	Эквивалент в коде						
	8421	2421	7421	5211	8421+3	3а+2	2 из 5
0	0000	0000	0000	0000	0011	00010	11000
1	0001	0001	0001	0001	0100	00101	00011
2	0010	0010	0010	0011	0101	01000	00101
3	0011	0011	0011	0101	0110	01011	00110
4	0100	0100	0100	0111	0111	01110	01001
5	0101	1011	0101	1000	1000	10001	01010
6	0110	1100	0110	1010	1001	10100	01100
7	0111	1101	1000	1100	1010	10111	10001
8	1000	1110	1001	1110	1011	11010	10010
9	1001	1111	1010	1111	1100	11101	10100

Суммирование чисел с одинаковыми знаками в BCD-коде

При выполнении операций над отмеченными кодами возможны следующие особенности:

- наличие разрешенных и запрещенных комбинаций, свидетельствующих о правильности результата или необходимости его коррекции;
- при сложении тетрад возможен потетрадный (16 единиц), а не поразрядный (10 единиц) перенос, что также требует корректировки результата.

При сложении чисел в BCD-коде возможны три случая:

1) $(a + b) \le 9$. В этом случае если действия выполняются по правилам двоичной арифметики, то величина получаемой суммы не превышает девяти и коррекция результата не требуется.

```
  5     0101
+ 3     0011
  8     1000
```

2) $10 \le (a + b) \le 15$. Если результат сложения двух чисел попадает в данный диапазон чисел, то возможны два случая результирующей тетрады.

```
  5     0101          9     1001
+ 6     0110        + 4     0100
 11     1011         13     1101
```

В этом случае в тетраде накопилось более девяти единиц и должен быть выполнен десятичный перенос. Перенос единицы в старший разряд выполняется принудительно логической схемой. Условием для формирования единицы переноса является возникновение запрещенной комбинации (наличие единицы в разрядах с весом 8 и 4 или 8 и 2). Однако тетраду надо освободить от десяти избыточных единиц. Это тоже делается принудительно добавлением 0110 (шестерки), что приводит к возникновению шестнадцатеричного переноса. Этот перенос игнорируется. Схема формирования принудительного переноса приведена на рис.11.

3) (a + b) ≥ 16. Здесь в процессе суммирования возникает шестнадцатеричный перенос, в результате которого тетраду покидают вместе с десятком и те шесть единиц, которые принадлежат тетраде. Чтобы восстановить верное значение этой тетрады, необходимо к ней добавить 0110 (шесть).

```
  8     1000
+ 9     1001
 17   1 0001
      0110  Коррекция(+ 6)
      0111
```

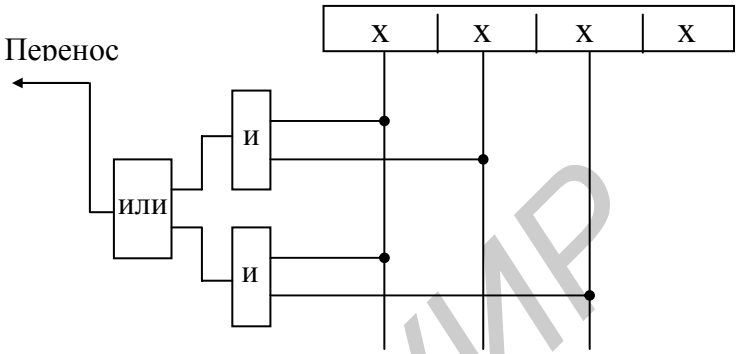


Рис. 11. Схема определения запрещенной комбинации

Таким образом, из сказанного выше можно сформулировать следующие правила потетрадного сложения чисел в BCD-кодах.

- Если при потетрадном сложении перенос в соседнюю старшую тетраду не возникает, то результат суммирования не требует коррекции.
- Коррекция результата потетрадного сложения путем добавления поправки 0110 требуется в случае, если возникает:

а) потетрадный перенос в старшую тетраду;

б) запрещенная комбинация.

Устройство, которое работает по сформулированным выше правилам, называется ***одноразрядным двоично-десятичным сумматором*** (рис.12).

Пример: сложить числа A=169 и B=378 в BCD-коде

```
  A = 169          A = 0.0001 0110 1001
+ B = 378        + B = 0.0011 0111 1000
A + B = 547      A + B = 0.0101 1110 0001
                         0110 0110
                 ---------------------
                 0.0101 0100 0111
                 |
                 перенос игнорируется
```

Суммирование чисел с разными знаками в BCD-коде

Отрицательные BCD-коды должны представляться в прямом, обратном или дополнительном кодах. Особенностью BCD-кодов является то, что инверсия тетрады означает дополнение до 15, а для соответствующей десятичной цифры до 9. Следовательно, необходимо убрать разницу. Один из приемов формирования обратного BCD-кода состоит в добавлении во все тетрады отрицательного числа 0110, затем их инверсии.

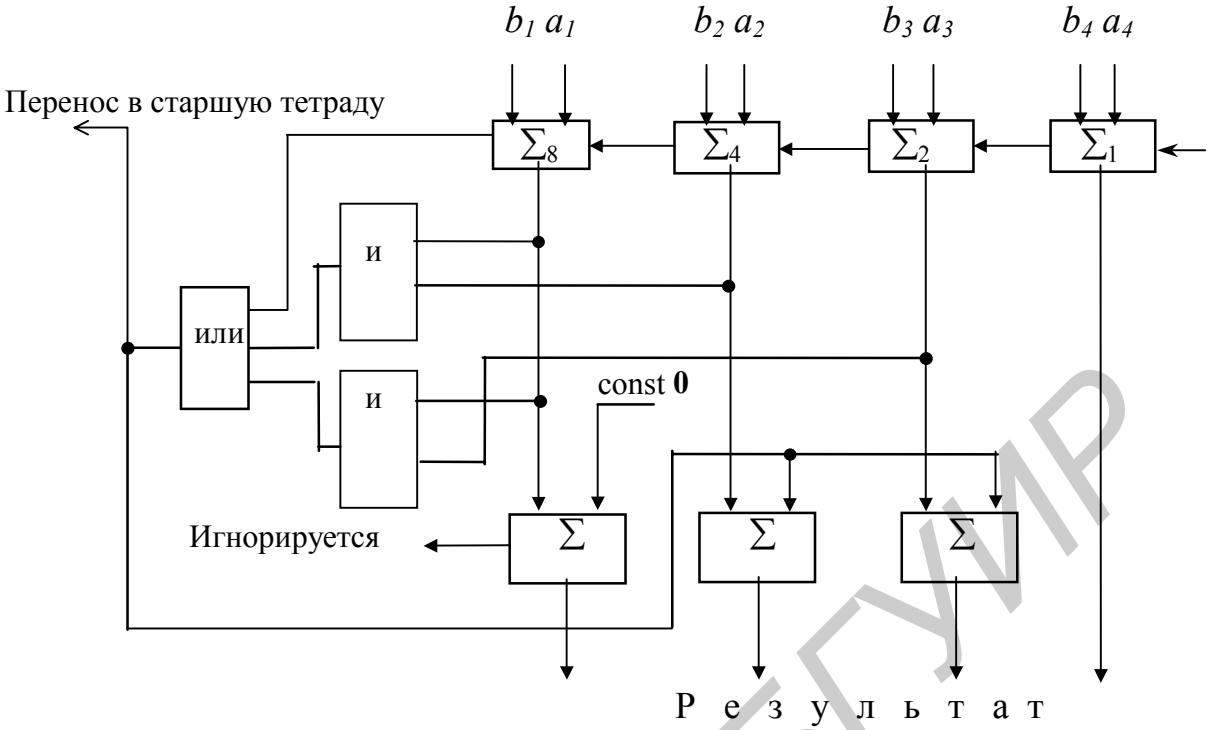


Рис. 12. Структурная схема одноразрядного двоично-десятичного сумматора

При сложении чисел с разными знаками возможны следующие случаи.

1) a - b ≥ 0

a =	7	0 . 0111	$[\ a\ ]_{обр}$
b=	-3	1 . 1100	$[\ b\ ]_{обр}$
	4	10 . 0011	
		⌊⟶ 1	
		0 . 0100	

При образовании инверсии отрицательной тетрады в нее добавляются 15 единиц. Эти 15 единиц находятся и в сумме. Но благодаря шестнадцатеричному переносу из тетрады уходит 16 единиц (15+1 − эта единица восстанавливается добавлением по цепи циклического переноса).

2) a - b < 0

a =	3	0 . 0011	$[\ a\ ]_{обр}$
b =	-7	1 . 1000	$[\ b\ ]_{обр}$
	-4	1 . 1011	
		0 . 0100	

Здесь, как и в предыдущем примере, в тетраде суммы пятнадцать лишних единиц. Но при переходе от инверсной формы к прямой лишние единицы уничтожаются сами собой. Это то же самое, что от значащей части суммы вычесть пятнадцать: 1011 - 1111 = 0100. Рассмотрим несколько примеров.

A = 378	0. 0011 0111 1000
– B = 169	1. 1110 1001 0110
A – B = 209	10. 0010 0000 1110
	⌊циклический перенос 1

0. 0010 0000 1111

Из последней тетрады нет переноса, таким образом, это соответствует заему в нее 16 единиц (вместо необходимых 10). Следовательно, из нее необходимо удалить лишние шесть единиц. Для этого в тетраду добавляется 10 - дополнение шести до шестнадцати:

0. 0010 0000 1111
1010
0. 0010 0000 1001
\+ 2 0 9

– A = 169	0. 0001 0110 1001
B = 378	1. 1100 1000 0111
A–B= - 209	1. 1101 1111 0000
	0110
	1. 1101 1111 0110
	- 0010 0000 1001
	- 2 0 9

Таким образом, в тетраду производится заем, если результат:

- положительный и из тетрады нет переноса;
- отрицательный и из тетрады есть перенос.

BCD-коды с избытком 3

Иначе говоря, это коды чисел из системы (BCD + 3). В этом коде каждая десятичная цифра a_i представляется в виде двоичного эквивалента суммы a_i+3. В отличие от BCD-кода код BCD+3 – самодополняющийся, но не имеющий свойства взвешенности. Применяется наиболее часто в десятичной арифметике, так как при выполнении двоичного суммирования легко выделить десятичный перенос.

Возможны следующие два случая сложения чисел в BCD-коде +3:

1) $a + b \le 9$; $[(a + 3) + (b + 3)] \le 15$.

И, следовательно, в тетраде суммы будут лишние 6 единиц. Чтобы тетрада суммы осталась тоже с избытком 3, нужно вычесть 3.

2) $a + b \ge 10$; $[(a + 3) + (b + 3)] \ge 16$.

Здесь во всех случаях возникает шестнадцатеричный перенос, вместе с которым тетраду суммы покинут и шесть избыточных единиц; чтобы тетрада суммы осталась с избытком 3, надо добавить 3.

Если складываются числа с разными знаками, то избыток в тетраде суммы будет равен нулю и суммирование, таким образом, сводится к правилам суммирования в BCD-коде.

Пример. Выполнить сложение чисел 169 и 378 в BCD-коде +3.

	0.0100 1001 1100
A = 169	0.0110 1010 1011
B = 378	0.1011 0100 0111
A + B = 547	0011 0011 0011

0.1000 0111 1010
8 7 10

Пример . Выполнить вычитание из числа 378 числа 169 в BCD-коде +3.

A = 378 0.0110 1010 1011
B = 169 1.1011 0110 0011
A - B = 209 1 0.0010 0000 1110

циклический перенос 1

0.0010 0000 1111
+ 0011 -0011 -0011
0.0101 0011 1100
5 3 12

Пример. Выполнить вычитание из числа 169 числа 378 в BCD-коде +3.

A = 169 0.0100 1001 1100
B = 378 1.1001 0101 0100
A - B = -209 1.1101 1111 0000
-0011 -0011 +1100
1.1010 1100 0011
- 0101 0011 1100
5 3 12

Правило. Если из тетрады был перенос, надо добавить +0011, если переноса не было – -0011 (добавить 1100), независимо от знака слагаемых и знака суммы.

BCD-код с избытком 6 для одного из слагаемых

При сложении чисел с одинаковыми знаками неважно, к какому из слагаемых добавить 0110. Причем это равносильно добавлению 0011 к каждому слагаемому.

При суммировании чисел в коде с избытком 6 коррекция может понадобиться только в случае, когда сумма меньше 16. В остальных случаях (сумма больше 16) возникает перенос, удаляющий из тетрады 6 лишних единиц, и коррекции результата не требуется.

Результат должен быть без избытка.

Пример:

A = 169 0.0111 1100 1111
B = 378 0.0011 0111 1000
A + B = 547 0.1001 0100 0111
0110 .
0.0101 0100 0111

Суммирование чисел с разными знаками производится в BCD-коде, но в сущности тоже с избытком 6.

Система счисления в остаточных классах (СОК)

Поиски новых путей повышения эффективности выполнения арифметических операций привели к заключению, что в рамках обычной позиционной системы счисления ускорения операций добиться сложно. Следует отметить, что позиционные системы счисления обладают существенным недостатком – наличием межразрядных связей, влияющих на способы реализации арифметических операций. В конечном итоге это приводит к усложнению аппаратуры и снижению быстродействия. Оказалось, что арифметика, в которой межразрядные связи отсутствовали бы, может быть построена на основе непозиционной системы счисления, в частности системы счисления в остаточных классах. В системе остаточных классов числа представляются остатками от деления на выбранную систему оснований и все рациональные операции могут выполняться параллельно над цифрами каждого разряда в отдельности. В то же время системе остаточных классов присущи некоторые недостатки: ограниченность действий этой системы полем целых положительных чисел, трудность определения соотношений чисел по величине, определения выхода результата операции из диапазона и некоторые другие.

Определение. Если задан ряд положительных целых чисел $p_1, p_2, ..., p_n$, называемых в дальнейшем основаниями системы, то под системой счисления в остаточных классах будем понимать такую систему, в которой целое положительное число представляется в виде набора остатков (вычетов) по выбранным основаниям

$N = (\alpha_1, \alpha_2, ..., \alpha_n)$,

причем образование цифр α_i осуществляется следующим образом:

$$\alpha_i = N - \left[\frac{N}{p_i}\right] p_i, \text{ для } i = 1, 2, ..., n, \qquad (5)$$

где [] означает целую часть, то есть цифра i-го разряда α_i числа N есть наименьший положительный остаток от деления N на p_i.

Таким образом, в СОК в отличие от обобщенной позиционной системы счисления образование цифры каждого разряда производится независимо друг от друга. Очевидно, что $\alpha_i < p_i$.

В теории чисел доказано, что если числа p_i взаимно простые между собой, то описанное цифрами $\alpha_1, \alpha_2, ..., \alpha_n$ представление числа N -- единственно.

Диапазон представимых чисел в СОК равен:

$$\rho = p_1, p_2, ..., p_n.$$

Рассмотрим правила выполнения операций сложения и умножения в СОК в случае, если числа и результат находятся в диапазоне $[O, \rho]$.

Пусть операнды A и B представлены соответственно остатками α_i и β_i по основаниям p_i, i=1,2, ... ,n. Результаты операций A+B и A·B представлены остатками γ_i и δ_i по тем же основаниям p_i, то есть

$A=(\alpha_1\ \alpha_2\ ...\ ,\alpha_n)$,

$B=(\beta_1, \beta_2, \ldots, \beta_n)$,

$A+B=(\gamma_1, \gamma_2, \ldots, \gamma_n)$,

$A \cdot B=(\delta_1, \delta_2, \ldots, \delta_n)$,

при этом $A < \rho$, $B < \rho$, $A+B < \rho$, $A \cdot B < \rho$,

$\gamma_i \equiv \alpha_i + \beta_i \pmod{p_i}$,

то есть γ_i сравнимо с $\alpha_i + \beta_i$ по модулю p_i;

$\delta_i \equiv \alpha_i \beta_i \pmod{p_i}$.

При этом

$$\gamma_i \equiv \alpha_i + \beta_i - \left[\frac{\alpha_i + \beta_i}{p_i}\right] p_i,$$

$$\delta_i \equiv \alpha_i \beta_i - \left[\frac{\alpha_i \beta_i}{p_i}\right] p_i.$$

С учетом (5) отсюда следует:

$$\gamma_i = A+B - \left[\frac{A+B}{p_i}\right] p_i. \qquad (6)$$

Из представления A и B следует:

$A = k_i p_i + \alpha_i$,

$B = l_i p_i + \beta_i$,

где k_i и l_i - целые неотрицательные числа.

Тогда $A + B = (k_i + l_i) p_i + \alpha_i + \beta_i$,

$$\left[\frac{A+B}{p_i}\right] = k_i + l_i + \left[\frac{\alpha_i + \beta_i}{p_i}\right],$$

$$A + B = (k_i + l_i)p_i + \left[\frac{\alpha_i + \beta_i}{p_i}\right] p_i.$$

Подставляя полученные выражения в (6), получим:

$$\gamma_i \equiv \alpha_i + \beta_i - \left[\frac{\alpha_i + \beta_i}{p_i}\right] p_i. \qquad (7)$$

В случае умножения

$$\delta_i \equiv \alpha_i \beta_i - \left[\frac{\alpha_i \beta_i}{p_i}\right] p_i.$$

Аналогично сложению (A + B) получим:

$A \cdot B = k_i l_i p_i^2 + (\alpha_i l_i + \beta_i k_i)p_i + \alpha_i \beta_i$,

$$\left[\frac{A \cdot B}{p_i}\right] = k_i l_i p_i + \alpha_i l_i + \beta_i k_i + \left[\frac{\alpha_i \beta_i}{p_i}\right].$$

Таким образом,

$$\delta_i \equiv \alpha_i \beta_i - \left[\frac{\alpha_i \beta_i}{p_i}\right] p_i . \qquad (8)$$

Пример. Сложить числа A = 23 и B = 48 в СОК.

Пусть основаниями системы являются $p_1 = 3$, $p_2 = 5$, $p_3 = 7$.

По выбранным основаниям числа A и B в СОК примут вид:

A = 23 = (2, 3, 2),

B = 48 = (0, 3, 6),

получим A + B = (2, 1, 1).

Легко проверить, что 23 + 48 = 71, а число 71 = (2, 1, 1) в СОК.

Пример. Умножить число A=14 на B=8 в СОК.

A=14=(2, 4, 0),

B=8=(2, 3, 1).

В соответствии с (4) получим:

$$\gamma_1 = 4 - \left[\frac{4}{3}\right]3 = 4 - 3 = 1,$$

$$\gamma_2 = 12 - \left[\frac{12}{5}\right]5 = 12 - 2\cdot 5 = 2,$$

$$\gamma_3 = 1 - \left[\frac{1}{7}\right]7 = 1 - 0.7 = 1.$$

Таким образом, A·B = 122 = (1, 2, 1).

Охарактеризуем в общих чертах достоинства и недостатки введенной СОК.

Достоинства:

- независимость образования разрядов числа, в силу чего каждый разряд несет информацию обо всем исходном числе, а не о промежуточном числе, получившемся в результате формирования младших разрядов (позиционная система счисления). Отсюда вытекает независимость разрядов числа друг от друга и возможность их параллельной обработки, что в свою очередь в дальнейшем позволяет вводить принципиально новые методы арифметического контроля. При введении дополнительного контрольного основания остаток, взятый по этому основанию, при его избыточности позволяет контролировать и исправлять ошибки в цифрах по рабочим основаниям;
- малоразрядность остатков, представляющих число. Ввиду малого количества кодовых комбинаций открывается возможность построения табличной (не вычислительной) арифметики.

Недостатки:

- невозможность визуального сопоставления чисел, так как внешняя запись числа не дает представления о его величине;
- отсутствие простых признаков выхода результата операций за пределы

[0,ρ];

- ограниченность действий системы сферой целых положительных чисел;
- получение во всех случаях точного результата исключает возможность приближенных вычислений, округлений.

Представление отрицательных чисел в СОК

Рассмотрим правила выполнения операций вычитания в системе остаточных классов для чисел A и B, удовлетворяющих условию $A\text{-}B \subseteq [0, \rho]$.

$A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$,

$B = (\beta_1, \beta_2, \ldots, \beta_n)$,

$A\text{-}B = (\gamma_1, \gamma_2, \ldots, \gamma_n)$,

при этом $A<\rho$, $B<\rho$, $0<=A\text{-}B<\rho$.

Как и ранее, получаем для A-B:

$$\gamma_i = \alpha_i - \beta_i - \left[\frac{\alpha_i - \beta_i}{p_i}\right] p_i,$$

$\gamma_i \equiv \alpha_i - \beta_i \pmod{p_i}$.

Операция вычитания при положительном результате выполняется вычитанием соответствующих цифр разрядов, в результате приводится наименьший положительный остаток.

При отрицательной разности цифр берется ее дополнение к основанию. При этом знак результата в результате никак не отражен.

Возникает необходимость ввести специальным образом знак в представление числа и определить правила выполнения операций, обеспечивающих получение не только величин, но и знака результата.

Рассмотрим варианты введения отрицательных чисел.

Пусть $p_1, p_2, \ldots p_n$ - основания системы счисления.

$\rho = p_1, p_2, \ldots, p_n$ - диапазон представимых чисел.

Пусть $p_2 = 2$. Обозначим через B СОК $P=(1, 0, 0, \ldots, 0)$. Будем оперировать числами, лежащими в диапазоне $0 \leq |N| < P$.

Примем в качестве нуля число P и представим положительные числа $N = |N|$ как $N' = P + |N|$, отрицательные числа $N = -|N|$ как $N' = P - |N|$. Таким образом, $N' = P + N$ - искусственная форма (общая форма представления + и - чисел).

Следовательно, мы всегда будем иметь дело с положительными числами, так как $|N|<P$.

Но числа в интервале [0,P) в искусственной форме будут отображать отрицательные, а в интервале $[P, \rho)$ - положительные числа.

Операцию сложения и вычитания можно выполнять следующим образом:

$N'_1 = P + N_1$,

$N'_2 = P + N_2$,

$N'_1 + N'_2 = P + N_1 + P + N_2 = 2P + (N_1 + N_2)$.

Для суммы $N_1 + N_2$ искусственная форма:

$(N_1 + N_2)' = P + (N_1 + N_2)$,

$(N'_1 + N'_2) = N'_1 + N'_2 - P$

или, так как P = (1, 0, 0, ..., 0), то $(N_1 + N_2)' = N'_1 + N'_2 + P$. (9)

Пример. Пусть $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, $p_4 = 7$.

$P = 3 \cdot 5 \cdot 7 = 105$,

$N_1 = 17$, $N_2 = 41$,

$N'_1 = (1, 0, 0, 0) + (1, 2, 2, 3) = (0, 2, 2, 3)$,

$N'_2 = (1, 0, 0, 0) + (1, 2, 1, 6) = (0, 2, 1, 6)$.

Согласно (9) $(N_1 + N_2)' = (0, 2, 2, 3) + (0, 2, 1, 6) + (1, 0, 0, 0) = (1, 1, 3, 2)$ - искусственная форма $N_1 + N_2$, что можно проверить, перейдя к десятичной системе счисления.

$(17 + 41)' = (58)' = 105 + 58 = (1, 0, 0, 0) + (0, 1, 3, 2) = (1, 1, 3, 2)$.

Пример. $N_1 = 17$, $N_2 = 41$.

$N'_1 = (0, 2, 2, 3)$,

$N'_2 = (1, 0, 0, 0) - (1, 2, 1, 6) = (0, 1, 5, 1)$,

$(N_1 + N_2)' = (0, 2, 2, 3) + (0, 1, 5, 1) + (1, 0, 0, 0) = (1, 0, 1, 4)$,

$(17 - 41)' = (-24)' = 105 - 24 = (1, 0, 0, 0) - (0, 0, 4, 3) = (1, 0, 1, 4)$.

Переход из положительного числа в отрицательное и обратно, то есть образование его дополнительного кода, производится вычитанием данного числа из числа $(1, p_1, p_2, \ldots, p_n)$.

$+41 = (1, 2, 1, 6)$,

$- 41 = (1, 3, 5, 7) - (1, 2, 1, 6) = (0, 1, 4, 1) = 64$.

Следует отметить, что если вычитаемое уже было представлено в искусственной форме, то для получения дополнительного кода надо его вычитать из $(2, p_1, p_2, \ldots, p_n)$.

Контроль работы цифрового автомата

В процессе вычислений происходит постоянная передача и преобразование информации, находящейся в памяти ЭВМ, арифметическом или управляющем устройствах. Таким образом, при проектировании ЭВМ необходимо предусмотреть меры как выявления ошибок, так и их исправления. Эта функция возлагается на систему контроля. Система контроля – совокупность аппаратных и программных методов и средств, обеспечивающих определение правильности работы автомата в целом или его отдельных узлов, а также автоматическое исправление выявленных ошибок. Различают следующие виды ошибок вычислений, возникающих:

- из-за погрешностей в исходных данных;
- вследствие методических погрешностей;
- из-за неисправностей в работе машины.

Третий вид ошибок является объектом для системы контроля. Решение задачи контроля возможно только при наличии определенной избыточности информации (аппаратной или информационной). Аппаратная избыточность может заключаться, например, в дублировании арифметических устройств. Ло-

гические методы могут заключаться в выполнении ”двойного счета” и последующего сравнения результатов, в проверке соотношений вычисляемых функций ($\sin^2 x+\cos^2 x=1$), в контроле результата при изменении шага вычисления и так далее. Однако эти методы направлены на выявление факта появления ошибки в вычислении, но не определяют место, где она произошла.

Некоторые понятия теории кодирования

Систематический код - код, включающий кроме m информационных k контрольных разрядов. При этом возникает избыточность (абсолютная и относительная). Абсолютная избыточность равна k, а относительная определяется соотношением k/n, где n=m+k – общее количество разрядов в кодовом слове (m – число информационных разрядов).

Корректирующая способность кода определяется вероятностью обнаружения или исправления ошибки. Если вероятность искажения одного символа n-разрядного слова равна p, то вероятность искажения k символов по теореме умножения вероятностей будет $\omega=\rho^k(1-\rho)^{n-k}$.

Число кодовых комбинаций, каждая из которых содержит k искаженных элементов, равна числу сочетаний из n по k:

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

Тогда полная вероятность искажения информации

$$P_{\sum} = \sum_{ш=1}^{л} \frac{n!}{i!(n-i)!} p^i (1-p)^{n-i}.$$

Корректирующая способность кода связана также с понятием кодового расстояния.

Кодовое расстояние d(A,B) для кодовых комбинаций A и B определяется как вес такой третьей кодовой комбинации, которая является их суммой по модулю 2.

Вес кодовой комбинации V(A) – количество единиц, содержащихся в кодовой комбинации.

В теории кодирования [5] показано, что систематический код обладает способностью обнаружить ошибки только тогда, когда минимальное кодовое расстояние для него больше или равно 2t:

$d_{min} \geq 2t$,

где t – кратность обнаруживаемых ошибок.

Это означает, что между соседними разрешенными кодовыми словами должно существовать по крайней мере одно кодовое слово.

В случае если необходимо не только обнаруживать, но и исправлять ошибку (указать место ошибки), минимальное кодовое расстояние должно быть $d_{min} \geq 2t+1$.

Обнаружение и исправление одиночных ошибок путем использования дополнительных разрядов

Рассмотрим возможность использования дополнительных (контрольных) разрядов для обнаружения и исправления ошибок. Эта возможность заключается в том, что к n информационным разрядам добавляется один контрольный разряд. В него записывается 0 или 1 таким образом, чтобы для каждого из передаваемых чисел сумма разрядов по модулю 2 была бы равна 0 (кодирование по методу четности) или 1 (нечетности).Появление ошибки в числе обнаружится по нарушению четности или нечетности. При этом виде кодирования допускается возможность выявления только одиночной ошибки. Чтобы одна комбинация разрядов числа превратилась в другую без выявления ошибки, необходимо изменение четного (2, 4, 6 и так далее) числа разрядов одновременно. Пример реализации метода контроля по методу четности-нечетности приведен в табл. 5.

Рассмотренный способ контроля по методу четности-нечетности может быть видоизменен для локализации (выявления места) ошибки в числе. Длинное число разбивается на группы разрядов, каждая из которых содержит k разрядов.

Таблица 5

Число	Контрольный разряд	Проверка (нечетности)
11011011	1	0
01101101	1	1-ошибка
11010101	0	0
10101001	1	0
01010111	0	0

Контрольные разряды выделяются всем группам по строкам и по столбцам согласно следующей схеме:

a_1	a_2	a_3	a_4	a_5	k_1
a_6	a_7	a_8	a_9	a_{10}	k_2
a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	k_3
a_{16}	a_{17}	a_{18}	a_{19}	a_{20}	k_4
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	k_5
k_6	k_7	k_8	k_9	k_{10}	

Если ошибка произошла в разряде a_s (единица изменилась на ноль или наоборот), то при проверке на четность (нечетность) сумма по i-й строке и j-му столбцу (на пересечении которых находится элемент a_s) изменится. Следовательно, можно зафиксировать нарушение четности (нечетности) по этой строке и столбцу. Это не только позволит обнаружить ошибку, но и локализовать ее место. Изменив значение разряда a_s на противоположное, можно исправить возникшую ошибку.

Контроль по методу четности-нечетности используется для контроля записи и считывания информации, а также для выполнения арифметических операций.

Коды Хемминга

Американский ученый Р. Хемминг предложил способ кодирования информации, позволяющий не только обнаруживать, но и исправлять ошибки при передаче одиночного слова любой разрядности. Эти коды – систематические. Пусть разрядность слова равна m. Для контроля информации требуется k дополнительных разрядов. Число k выбирается согласно следующим правилам.

1. Контролирующее число k выбирается так, чтобы оно имело количество комбинаций, достаточное для распознавания одной из m+k позиций или для сигнализации отсутствия ошибки. Полученное таким образом число описывает n=m+k+1 событий. Следовательно, необходимо, чтобы выполнялось неравенство $2^k \geq (m+n+1)$.

2. (m+k)-разрядные позиции нумеруются от единицы до (m+k), начиная от младшей значащей. Контрольные разряды k обозначаются P_0, P_1, P_2, …,P_{k-1} и помещаются в разряды, имеющие номера 1,2,4,8, …,2^{k-1} (m+k)-разрядного числа. Остальные m разрядов могут быть размещены в любом порядке между контрольными разрядами.

3. Контрольные разряды P_0, P_1, P_2, …,P_{k-1} выбраны таким образом, чтобы для определенных разрядов слова служить в качестве контрольных избыточных разрядов.

Проверка	*Проверяемые разряды*												
1	1	3	5	7	9	11	13	15	…				
2	2	3	6	7	10	11	14	15	18	19	22	…	
3	4	5	6	7	12	13	14	15	20	21	22	23	…
4	8	9	10	11	12	13	14	15	24	25	26	27	…
…	.	.	.										

P_0 выбрано с таким расчетом, чтобы в позициях 1, 3, 5, 7, 9, 11 … число единиц каждого слова было четным, P_1 выбрано для того, чтобы выполнялось условие четности в разрядах 2, 3, 6, 7, 10, 11, 14, 15 …, аналогично P_2 контролирует позиции 4,5,6,7,12,13,14,15,20… и P_3 – для разрядов 8, 9,10,11,12,13, 14,15,24,25… .

На основании рассмотренных правил в табл. 6 показаны семиразрядные коды. Контрольные разряды обозначены P_0, P_1 и P_2 и помещены в позициях 1, 2 и 4.

Операция обнаружения и исправления ошибок выполняется путем нахождения k-разрядного контрольного числа. При этом младший значащий разряд контрольного числа находится посредством проведения контроля на четность над разрядами 1,3,5,7,9… . Если контроль показывает правильность передачи, то пишется нуль, иначе − единица. Следующий разряд контрольного числа определяется путем проверки на четность разрядов

определяется путем проверки на четность разрядов 2,3,7,10,11,14,15,… . Остальные разряды контрольного числа находятся аналогично.

Таблица 6

Число	Разряды						
	7	6	5	4	3	2	1
	A	B	C	P_2	D	P_1	P_0
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	1	1	0	0	1
3	0	0	1	1	1	1	0
4	0	1	0	1	0	1	0
5	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1
7	0	1	1	0	1	0	0
8	1	0	0	1	0	1	1
9	1	0	0	1	1	0	0
10	1	0	1	0	0	1	0
11	1	0	1	0	1	0	1
12	1	1	0	0	0	0	1
13	1	1	0	0	1	1	0
14	1	1	1	1	0	0	0
15	1	1	1	1	1	1	1

Если контрольное число равно нулю, то это означает, что при передаче информации ошибка не произошла. Если же контрольное число не равно нулю, то оно указывает на тот разряд числа, где зафиксирована ошибка, которую необходимо исправить.

Пусть, например, передается число шесть 0110011, а принимается в виде 0110111, то есть произошла ошибка в третьем разряде. Выполняя контроль на четность с помощью разрядов P_0, P_1 и P_2, находим:

			Контрольное число
P_0	(1, 3, 5, 7) = (1, 1, 1, 0)	нечетность	1
P_1	(2, 3, 6, 7) = (1, 1, 1, 0)	нечетность	1
P_2	(4, 5, 6, 7) = (0, 1, 1, 0)	четность	0

Полученное контрольное число равно 011, что соответствует ошибке в третьем разряде.

Таким образом, дополнительный разряд P_i выбран так, чтобы проверять четность той совокупности разрядных позиций, чьи контрольные числа содержат единицу в позиции 2^i.

Логические основы вычислительной техники

Двоичные переменные и булевы функции

Для формального описания устройств вычислительной техники при их анализе и синтезе используется аппарат *алгебры логики*. Алгебру логики назы-

вают также булевой алгеброй. Основными понятиями алгебры логики являются двоичные переменные и переключательные (булевы) функции.

Двоичные переменные могут принимать только два значения: 0 (ложь) и 1 (истина) − и обозначаются символами x_1, x_2, … , x_n. Двоичные (логические, булевы) переменные являются аргументами булевых (переключательных) функций.

Функция f, зависящая от n переменных x_1, x_2, ..., x_n, называется *булевой*, или переключательной, если функция f и любой из ее аргументов x_i, (i = 1...n) принимают значения только из множества {0, 1}.

Иначе говоря, булева функция – это функция, и аргументы и значение которой принадлежат множеству { 0, 1 }. Множество { 0, 1 } обозначим через *B*.

Булеву функцию от *n* аргументов можно рассматривать как *n*-местную алгебраическую операцию на множестве *B*. При этом алгебра $<B;\Omega>$, где Ω – множество всевозможных булевых функций, называется *алгеброй логики (булевой алгеброй)*.

Конечность области определения функции имеет существенное достоинство: такие функции можно задавать перечислением значений при различных значениях аргументов. Для того чтобы задать значение функции от *n* переменных, надо определить значения для каждого из 2^n возможных наборов. Эти значения записывают в таблицу истинности в порядке соответствующих двоичных чисел (рассмотрим позже).

x_1	x_2	...	x_{n-1}	x_n	f
0	0	...	0	0	f(0,0,...,0,0)
0	0	...	0	1	f(0,0,...,0,1)
0	0	...	1	0	f(0,0,...,1,0)
0	0	...	1	1	f(0,0,...,1,1)
...	...	...	...	...	...
1	1	...	0	1	f(1,1,...,0,1)
1	1	...	1	0	f(1,1,...,1,0)
1	1	...	1	1	f(1,1,...,1,1)

Для того чтобы задать функцию, достаточно выписать значения *f*(0,0,...,0,0), *f*(0,0,...,0,1), *f*(0,0,...,1,0), *f*(0,0,...,1,1),..., *f*(1,1,...,0,0), *f*(1,1,...,0,1), *f*(1,1,...,1,0), *f*(1,1,...,1,1). Этот набор называют *вектором значений функции*.

Таким образом, булевы функции на конечном множестве своих аргументов могут принимать значения 0 и 1 и обозначаются $f(x_1,x_2, \dots ,x_n)$. Булевы функции могут служить аргументами более сложных логических функций.

Способы задания булевых функций

Для задания произвольной булевой функции широко используются **табличный (матричный)** и **аналитический** способы. При табличном способе булева функция $f(x_1, ...,x_n)$ задается таблицей истинности (табл. 7), в левой части которой представлены все возможные двоичные наборы длиной n, а в правой указываются значения функции на этих наборах.

Таблица 7

№ набора	$x_1x_2x_3$	f
0	0 0 0	0
1	0 0 1	1
2	0 1 0	0
3	0 1 1	0
4	1 0 0	1
5	1 0 1	1
6	1 1 0	0
7	1 1 1	1

Под двоичным набором понимается совокупность значений аргументов $x_1,x_2, ...,x_n$ булевой функции f. Двоичный набор имеет длину n, если он представлен n цифрами из множества {0,1}. В табл. 7 представлены все двоичные наборы длиной 3. Иногда двоичные наборы из таблицы истинности булевой функции удобно представлять их номерами. Запишем аргументы $x_1,x_2, ...,x_n$ в порядке возрастания их индексов. Тогда любому двоичному набору можно поставить в соответствие целое десятичное число N, называемое номером набора. Например, двоичные наборы **011** и **101** имеют номера **3** и **5** соответственно. Булевы функции, зависящие от большого числа переменных, задавать таблицей истинности неудобно в силу ее громоздкости. Например, таблица истинности булевой функции 8 переменных будет содержать $2^8 = 256$ строк. Для задания функций многих переменных удобно использовать модификацию таблицы истинности. Рассмотрим способ построения такой таблицы истинности для функции n переменных. Множество из n переменных функции разбивается на два подмножества: $x_1, x_2, ..., x_{j-1}$ и $x_j, x_{j+1}, ..., x_n$. Переменными $x_1, x_2, ..., x_n$ отмечают строки таблицы истинности, задавая в каждой строке значение соответствующего двоичного набора длиной j-1. Переменными $x_j, x_{j+i}, ..., x_n$ отмечают ее столбцы, задавая в каждом столбце значения соответствующего двоичного набора длиной n-j+1. Значение функции записывается в клетке на пересечении соответствующей строки и столбца (табл. 8).

При аналитическом способе булева функция задается формулой, то есть аналитическим выражением, построенным из операций булевой алгебры. Аналитический способ задания булевых функций занимает особое место в проектировании цифровых автоматов. Фактически все преобразования над булевыми функциями, необходимые для построения цифровых автоматов, ведутся на аналитическом уровне.

Таблица 8

$x_1,x_2,...x_{j-1}$	$x_j, x_{j+1}, ...,x_n$			
	00...0	0...1	...	11...1
00...0	0	1		1

00...1	1	0		0
...				
11...1	1	0		1

Основные понятия алгебры логики

Существует не более чем $2^к$ (где $к=2^n$) различных булевых функций n переменных. К этому выводу легко прийти, пользуясь простыми комбинаторными рассуждениями, и вспомнив, что на каждом из 2^n наборов функции могут принимать два значения.

Функций от одной переменной четыре: это *константа* 0 (f_0), *константа* 1 (f_1), *тождественная функция* (f_2), то есть функция, значение которой совпадает с аргументом, и функция *отрицания* (f_3), значение которой противоположно значению аргумента. Отрицание будем обозначать $\bar{x}$.

x	f_0	f_1	f_2	f_3
0	0	0	1	1
1	0	1	0	1

Функции от некоторого числа переменных можно рассматривать как функции от большего числа переменных, при этом значения функции не меняются при изменении этих "добавочных" переменных. Такие переменные называются фиктивными, в отличие от остальных – существенных (действительных).

Переменная x_i называется *фиктивной* (несущественной) переменной функции $f(x_1,\cdots,x_n)$, если

$$f(x_1,\cdots,x_{i-1},0,x_{i+1},\cdots,x_n) = f(x_1,\cdots,x_{i-1},1,x_{i+1},\cdots,x_n)$$

для любых значений $x_1,\cdots,x_{i-1},x_{i+1},\cdots,x_n$. Иначе переменная x_i называется *существенной*.

Функции двух переменных представлены в табл. 9.

Таблица 9

x_1x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	F_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
00	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
01	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
10	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
11	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Отметим наиболее часто используемые функции из числа приведенных в таблице:

f_0 (x1, x2) = 0 - тождественный ноль (константа 0);

f_1 (x1, x2) = $x_1 \cdot x_2$ – конъюнкция (логическое произведение, И). Иногда употребляется знак & или /\;

f_3 (x1, x2) = x_1 − повторение x_1;

f_5 (x1, x2) = x_2 – повторение x_2;

f_6 (x1, x2) = $x_1 \oplus x_2$ − сложение по модулю 2 или mod 2;
f_7 (x1, x2) = $x_1 + x_2$ − дизъюнкция (логическое сложение, ИЛИ или V);
f_8 (x1, x2) = $x_1 \downarrow x_2$ − функция Вебба (стрелка Пирса, ИЛИ-НЕ);
f_9 (x1, x2) = $x_1 \sim x_2$ − эквивалентность;
f_{13}(x1, x2) = $x_1 \rightarrow x_2$ − импликация;
f_{14}(x1, x2) = $x_1 \setminus x_2$ − штрих Шеффера (И-НЕ);
f_{15}(x1, x2) = 1 − тождественная единица (константа 1).

Основными операциями булевой алгебры являются: отрицание, логическое сложение и логическое умножение. В булевой алгебре возведение в степень и извлечение корня являются вырожденными логическими операциями, поскольку значения, принимаемые аргументами при возведении в степень и извлечении корня, остаются неизменными, если принять справедливость равенств 1·1= 1 и 0·0= 0. Операции вычитания и деления не рассматриваются и не допускаются.

Логическое отрицание (**функция НЕ**). Логическим отрицанием высказывания x называется такое сложное высказывание f(x), которое истинно, когда x ложно, и наоборот. Функция НЕ записывается следующим образом: $f_1=\overline{x}$. Условное изображение элемента реализующего функцию НЕ приведено на рис. 13,а.

Логическое умножение (конъюнкция). Конъюнкция (функция И) двух переменных x_1 и x_2 − это сложное высказывание, которое истинно только тогда, когда истинны x_1 и x_2, и ложно для всех остальных наборов переменных. Логическая функция конъюнкции имеет вид $f=x_1 \cdot x_2$. Для обозначения операции конъюнкции используются также символы & и Λ. Функция логического умножения (И) от n переменных имеет вид $f_2=(x_1, x_2, \ldots, x_n)= x_1 \cdot x_2 \cdot \ldots \cdot x_n = \Lambda\, x_i$. Условное изображение элемента, реализующего операцию логического умножения, приведено на рис.13,б.

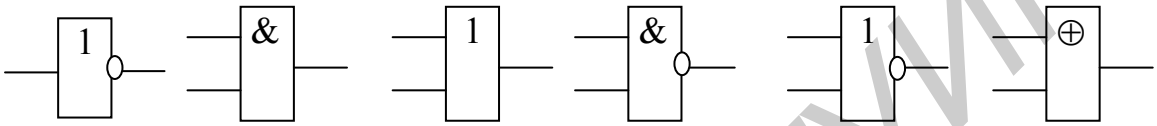
Логическое сложение (дизъюнкция). Дизъюнкция (функция ИЛИ) двух переменных x_1 и x_2 – это сложное высказывание, которое истинно тогда, когда истинна хотя бы одна из переменных x_1 и x_2, и ложно, когда они обе ложны. Логическая функция дизъюнкции имеет вид $f=x_1+x_2$. Для обозначения операции дизъюнкции используется также символ V. Функция логического сложения (ИЛИ) от n переменных имеет вид $f_2=(x_1, x_2, \ldots, x_n)= x_1+x_2+ \ldots +x_n = V\, x_i$. Условное изображение элемента, реализующего операцию логического сложения, изображено на рис.13,в.

Отрицание конъюнкции (операция Шеффера). Отрицание конъюнкции (функция И-НЕ) двух переменных x_1 и x_2 − сложное высказывание, ложное только при истинности обоих аргументов x_1 и x_2. Логическая функция И-НЕ имеет вид $f=\overline{x_1 \cdot x_2}$. Условное изображение элемента реализующего указанную операцию, изображено на рис. 13,г и называется элементом Шеффера или элементом И-НЕ.

Отрицание дизъюнкции (операция Пирса (Вебба)). Отрицание дизъюнкции (функция ИЛИ-НЕ) двух переменных x_1 и x_2 – сложное высказывание,

истинное только тогда, когда оба аргумента принимают ложное значение. Логическая функция ИЛИ-НЕ имеет вид $f=x_1+x_2$. Условное изображение элемента, реализующего указанную операцию, приведено на рис.13,д и называется элементом Пирса или элементом ИЛИ-НЕ.

Сложение по модулю 2 (исключающее ИЛИ). Сложение по модулю 2 − это сложное высказывание, которое истинно только тогда, когда истинна только одна из переменных x_1 и x_2. Логическая функция ”сумма по модулю 2” имеет вид $f=x_1 \oplus x_2$. Если число переменных $n>2$, то функция истинна на тех наборах, в которых число единиц нечетно. Условное изображение элемента, реализующего операцию сумма по модулю два, изображено на рис. 13,е.



а б в г д е

Рис. 13. Схемы логических элементов

Импликация. Это высказывание, принимающее ложное значение только в случае, если x_1 истинно, а x_2 ложно.

Простейшие булевы функции позволяют строить новые булевы функции с помощью обобщенной операции, называемой **операцией суперпозиции**.

Суперпозицией булевых функций f_0 и $f_1,...,f_n$ называется функция $f(x_1,...,x_m)= f_0(g_1(x_1,...,x_m),...,g_k(x_1,...,x_m))$, где каждая из функций $g_i(x_1, ...,x_m)$ либо совпадает с одной из переменных (тождественная функция), либо – с одной из функций $f_1,...,f_n$.

Функция $f(x,y) = (x \,\&\, y)$ является суперпозицией функций $\overline{}$ и &. Функция $g(x,y) = x \oplus (x \vee y)$ является суперпозицией функций $\oplus$ и $\vee$. Функция $h(x,y,z) = (x \,\&\, y) \oplus z$ − суперпозиция функций $\oplus$ и &.

Суперпозиция функций одного аргумента порождает функции одного аргумента. Суперпозиция функций двух аргументов дает возможность строить функции любого числа аргументов. Суперпозиция булевых функций представляется в виде логических формул. Однако следует отметить:

- одна и та же функция может быть представлена разными формулами;
- каждой формуле соответствует своя суперпозиция и, следовательно, своя схема соединений элементов;
- между формулами представления булевых функций и схемами, их реализующими, существует взаимно однозначное соответствие.

Очевидно, среди схем, реализующих данную функцию, есть наиболее простая. Поиск логической формулы, соответствующей этой схеме, представляет большой практический интерес. Преобразование формул булевых функций основано на использовании соотношений булевой алгебры.

Основные законы алгебры логики

Основные законы алгебры логики позволяют проводить эквивалентные преобразования логических функций, записанных с помощью операций И, ИЛИ, НЕ, приводить их к удобному для дальнейшего использования виду и упрощать запись.

При выполнении преобразований функций алгебры логики могут быть полезны следующие соотношения:

- всегда истинны высказывания: $x + 1=1$; $\bar{x} + x=1$;
- всегда ложны высказывания: $x \cdot 0=0$; $\bar{x} \cdot x=0$;
- правило двойного отрицания $\bar{\bar{x}}=x$;
- правило повторения $x + x + \ldots + x=x$; $x \cdot x \cdot \ldots \cdot x =x$.

Переместительный закон:

- для дизъюнкции $x_1+x_2 = x_2+x_1$;
- для конъюнкции $x_1 \cdot x_2 = x_2 \cdot x_1$;
- для суммы по модулю два $x_1 \oplus x_2 = x_2 \oplus x_1$.

Сочетательный закон:

- для дизъюнкции $x_1+(x_2+x_3)=(x_1+x_2)+x_3$;
- для конъюнкции $x_1 \cdot (x_2 \cdot x_3)= (x_1 \cdot x_2) \cdot x_3$;
- для суммы по модулю два $x_1 \oplus (x_2 \oplus x_3) = (x_1 \oplus x_2) \oplus x_3$,

то есть группирование переменных внутри дизъюнкции (конъюнкции) не изменяет значений функции.

Распределительный закон:

- для дизъюнкции $x_1+x_2 \cdot x_3=(x_1+x_2)(x_1+x_3)$,

то есть дизъюнкция переменной и конъюнкции эквивалентна конъюнкции дизъюнкций этой переменной с сомножителями;

- для конъюнкции $x_1 \cdot (x_2+x_3)= x_1 \cdot x_2+x_1 \cdot x_3$,

то есть конъюнкция переменной и дизъюнкции равносильна дизъюнкции конъюнкций этой переменной со слагаемыми.

Закон инверсии (*правило де Моргана*):

- для дизъюнкции $\overline{x_1+x_2}=\bar{x}_2 \cdot \bar{x}_1$;
- для конъюнкции $\overline{x_1 \cdot x_2}=\bar{x}_2+\bar{x}_1$,

то есть отрицание дизъюнкции (конъюнкции) переменных равно конъюнкции (дизъюнкции) отрицаний этих переменных.

Правило де Моргана справедливо для любого числа переменных:

$\overline{x_1+x_2+\ldots+x_n}= \bar{x}_1 \cdot \bar{x}_2 \cdot \ldots \cdot \bar{x}_n$,

$x_1 \cdot x_2 \cdot \ldots \cdot x_n= \overline{x_1 + x_2 + \ldots \cdot x_n}$.

Переместительный и сочетательный законы для дизъюнкции и конъюнкции, а также распределительный закон для конъюнкции совпадают с законами обычной алгебры. Но в обычной алгебре нет законов, аналогичных распределительному для дизъюнкции и законам инверсии. Их справедливость доказывает-

ся посредством составления таблиц истинности для левой и правой частей формулы.

Правило склеивания $x_1 \cdot x_2 + x_1 \cdot \overline{x_2} = x_1$.

Следующие соотношения могут быть выведены из рассмотренных выше:

$x_1 + x_1 \cdot x_2 = x_1$;

$x_1 + x_1 \cdot x_2 = x_1 \cdot 1 + x_1 \cdot x_2 = x_1 \cdot (1 + x_2) = x_1 \cdot 1 = x_1$;

$x_1 \cdot (x_1 + x_2) = x_1$.

Формы представления функций алгебры логики

Основными понятиями, лежащими в основе представления булевых функций в различных формах, являются понятия элементарной конъюнкции и элементарной дизъюнкции.

Элементарной конъюнкцией называется логическое произведение любого конечного числа различных между собой булевых переменных, взятых со знаком инверсии или без него.

Например, логические выражения вида $x_1\overline{x_2}\overline{x_3}$, $\overline{x_1}x_4$, $x_1x_2x_4$ являются элементарными конъюнкциями, а выражения вида $\overline{x_1x_2x_3}$, $\overline{\overline{x_1}x_4}$, $\overline{x_1x_2x_4}$ не являются элементарными конъюнкциями.

Элементарной дизъюнкцией называется логическая сумма любого конечного числа различных между собой булевых переменных, взятых со знаком инверсии или без него

Примером логического выражения, являющегося элементарной дизъюнкцией, могут служить $x_1 + x_2 + \overline{x_3}$, $\overline{x_1} + x_4$, $x_1 + \overline{x_2} + x_4$, а выражения вида $x_1 + x_2 + x_3$, $\overline{x_1 + x_4}$, $\overline{x_1 + x_2 + x_4}$ не являются элементарными дизъюнкциями.

Дизъюнктивной нормальной формой (ДНФ) булевой функции называется дизъюнкция конечного числа элементарных конъюнкций.

$f_{ДНФ} = x_1\overline{x_2}\overline{x_3} + x_1x_4 + x_2\overline{x_3}x_4 + x_1\overline{x_2}x_3$.

Число переменных, входящих в элементарную конъюнкцию, определяет *ранг* этой конъюнкции.

Совершенной ДНФ (СДНФ) логической функции от n аргументов называется такая ДНФ, в которой все конъюнкции имеют ранг n. СДНФ записывается по таблице истинности согласно правилу: *для каждого набора переменных, на котором булева функция принимает единичное значение, записывается конъюнкция ранга n и все эти конъюнкции объединяются дизъюнктивно; переменная имеет знак инверсии, если на соответствующем наборе имеет нулевое значение.*

$f_{СДНФ} = x_1x_2\overline{x_3} + \overline{x_1}\overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3} + x_1x_2x_3$.

В общем виде это можно записать следующим образом:

$$f(x_1, x_2, ..., x_n) = \bigvee f(\sigma_1, \sigma_2, ..., \sigma_n) \,\&\, x_1^{\sigma_1} \,\&\, x_2^{\sigma_2} ... \,\&\, x_n^{\sigma_{1n}},$$

где

$$x^{\sigma} = \begin{cases} \overline{x}, \text{ если } \sigma=0, \\ x, \text{ если } \sigma=1. \end{cases}$$

Элементарные конъюнкции, образующие СДНФ, называют также *конституентами* (составляющими) *единицы* (минтерм), так как они соответствуют наборам, при которых функция принимает значение, равное единице. Построение СДНФ по таблице истинности называют составлением булевой функции по условиям истинности.

Конъюнктивной нормальной формой (КНФ) булевой функции называется конъюнкция конечного числа элементарных дизъюнкций.

$f_{КНФ} = (x_1 + \overline{x}_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_4)(x_2 + \overline{x}_3 + x_4)(x_1 + x_2 + x_3)$.

Совершенной КНФ (СКНФ) логической функции от n аргументов называется такая КНФ, в которой все дизъюнкции имеют ранг n. СКНФ записывается по таблице истинности согласно правилу: *для каждого набора переменных, на котором булева функция принимает нулевое значение, записывается дизъюнкция ранга n и все эти дизъюнкции объединяются конъюнктивно; переменная имеет знак инверсии, если на соответствующем наборе имеет единичное значение.*

$f_{СКНФ} = (\overline{x}_1 + \overline{x}_2 + \overline{x}_3)(x_1 + \overline{x}_2 + \overline{x}_3)(\overline{x}_1 + x_2 + x_3)(x_1 + x_2 + x_3)$.

Элементарные дизъюнкции, образующие СКНФ, называют *конституентами* (составляющими) *нуля* (макстерм), так как они соответствуют наборам, при которых функция принимает нулевое значение. Построение СКНФ по таблице истинности называют составлением булевой функции по условиям ложности.

Теорема. Разложение в дизъюнкцию. Любую функцию $f(x_1,...,x_m)$ для любого n ($1 \le n \le m$) можно представить в виде

$$f(x_1,...,x_m) = \bigvee_{(\sigma_1,...,\sigma_n)} x_1^{\sigma_1} \,\&\, ... \,\&\, x_n^{\sigma_n} \,\&\, f(\sigma_1,...,\sigma_n,x_{n+1},...,x_m).$$

Доказательство. Покажем, что для любого набора значений переменных $(x_1,...,x_n,x_{n+1},...,x_m)$ значения левой и правой частей совпадают. Возьмём фиксированный набор $(x_1,...,x_n,x_{n+1},...,x_m)$. Рассмотрим выражение $x_1^{\sigma_1} \,\&\, ... \,\&\, x_n^{\sigma_n}$. Если одно из значений $x_i^{\sigma_i}$ равно 0, то и всё выражение равно 0. Тогда и выражение $x_1^{\sigma_1} \,\&\, ... \,\&\, x_n^{\sigma_n} \,\&\, f(\sigma_1,...,\sigma_n,x_{n+1},...,x_m)$ равно 0. Единице же выражение $x_1^{\sigma_1} \,\&\, ... \,\&\, x_n^{\sigma_n}$ равно только в том случае, если $\sigma_1 = x_1$, ..., $\sigma_n = x_n$. При этом $f(\sigma_1,...,\sigma_n,x_{n+1},...,x_m) = f(x_1,...,x_n,x_{n+1},...,x_m)$. Таким образом, значение правой части всегда равно $f(x_1,...,x_m)$, то есть значению левой части.

Теорема. Разложение в конъюнкцию. Любую функцию $f(x_1,...,x_m)$ для любого n ($1 \le n \le m$) можно представить в виде

$$f(x_1,...,x_m) = \&_{(\sigma_1,...,\sigma_m)} x_1^{\overline{\sigma}_1} \vee ... \vee x_n^{\overline{\sigma}_n} \vee f(\sigma_1...,\sigma_n,x_{n+1},...,x_m).$$

Разложения по всем переменным дают суперпозицию конъюнкции, дизъюнкции и отрицания.

Следствие 1. Совершенная дизъюнктивная нормальная форма.

Любая функция f может быть представлена в следующей форме:

$$f(x_1,...,x_m) = \bigvee_{(\sigma_1,...,\sigma_n)} x_1^{\sigma_1} \,\&\, ... \,\&\, x_m^{\sigma_m} \,\&\, f(\sigma_1,...,\sigma_m) =$$

$$= \bigvee_{(\sigma_1,...,\sigma_m): f(\sigma_1,...,\sigma_m)=1} x_1^{\sigma_1} \,\&\, ... \,\&\, x_m^{\sigma_m}$$

Следствие 2. Совершенная конъюнктивная нормальная форма.
Любая функция f может быть представлена в следующей форме:

$$f(x_1,...,x_m) = \&_{(\sigma_1,...,\sigma_m): f(\sigma_1,...,\sigma_m)=0} \; \bar{x}_1^{\sigma_1} \vee ... \vee \bar{x}_m^{\sigma_m}.$$

Таким образом, любая булева функция может быть представлена суперпозицией конъюнкции, дизъюнкции и отрицания. Разложение по всем переменным в дизъюнкцию называется совершенной дизъюнктивной нормальной формой функции, а в конъюнкцию – совершенной конъюнктивной нормальной формой. Совершенная дизъюнктивная и конъюнктивная нормальная формы дают способ представления булевой функции через суперпозицию конъюнкции, дизъюнкции и отрицания.

Чтобы получить совершенную дизъюнктивную нормальную форму, надо взять все наборы, на которых значение функции равно 1, и записать для каждого из них конъюнкцию переменных и их отрицаний. Если в наборе значение переменной равно 0, то переменную надо взять с отрицанием, если 1 – без отрицания. Из получившихся конъюнкций надо построить дизъюнкцию.

Чтобы получить совершенную конъюнктивную нормальную форму, надо взять все наборы, на которых значение функции равно 0, и записать для каждого из них дизъюнкцию переменных и их отрицаний. Если в наборе значение переменной равно 0, то переменную надо взять без отрицания, если 1 – с отрицанием. Из получившихся дизъюнкций надо построить конъюнкцию.

Системы функций алгебры логики

Любая булева функция может быть представлена аналитически одной из вышерассмотренных нормальных форм, которые используют ограниченное число элементарных булевых функций. Например, для СДНФ такими функциями являются "конъюнкция", "дизъюнкция" и "отрицание". Следовательно, существуют системы булевых функций, с помощью которых можно аналитически представить любую сколь угодно сложную булеву функцию. Проектирование цифровых автоматов основано на знании таких систем булевых функций. Последнее особенно важно для определения набора элементарных логических схем, из которых можно построить произвольный цифровой автомат. Проблема функциональной полноты является центральной проблемой функциональных построений в алгебре логики.

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.

Это обусловливает целесообразность постановки задачи определения свойств, которыми должны обладать функции, составляющие ФПСБФ.

Решение этой задачи основано на понятии замкнутого относительно операции суперпозиции класса функций. Класс булевых функций, функционально замкнутый по операции суперпозиции, есть множество функций, любая суперпозиция которых дает функцию, также принадлежащую этому множеству. Среди функционально замкнутых классов выделяют классы обычного типа, называемые предполными, которые обладают следующими свойствами. Предполный класс S не совпадает с множеством P всех возможных булевых функций, однако если в него включить любую не входящую в S булеву функцию, то новый функционально замкнутый класс будет совпадать с множеством P. Проведенные исследования показали, что предполных классов пять, а для построения ФПСБФ необходимо и достаточно, чтобы ее функции не содержались полностью ни в одном из пяти предполных классов.

Наряду с нормальными формами представления функций алгебры логики в вычислительной технике широко используются логические полиномиальные формы. Преобразования над формулами булевых функций иногда удобно выполнять в алгебре **Жегалкина**. Алгебра Жегалкина включает две двухместные операции: конъюнкцию и сложение по модулю 2, а также константу 1.

Теорема Жегалкина. Любая функция алгебры логики может быть представлена многочленом вида

$$f(x_1 x_2 \ldots x_n) = k_0 \oplus k_1 x_1 \oplus k_2 x_2 \oplus \ldots \oplus k_{n+1} x_1 x_2 \oplus k_{n+2} x_1 x_3 \oplus \ldots \oplus k_{n+m} x_1 x_2 x_n,$$

где k_i – коэффициент, принимающий значения 0 или 1.

Теорема позволяет представить любую ФАЛ в виде полиномов различной степени.

Задача построения полинома Жегалкина сводится к нахождению коэффициентов k_i. Для любых конституент единицы k_1 и k_2 имеет место следующее соотношение: $k_1 + k_2 = k_1 \oplus k_2$. Оно позволяет выполнить переход от СДНФ к полиному Жегалкина. Для этого достаточно заменить в СДНФ символ + (дизъюнкции) на символ $\oplus$ и выполнить подстановку вида $x=x \oplus 1$ с последующими преобразованиями в алгебре Жегалкина.

Перечислим предполные классы булевых функций.

Класс линейных функций. Функция алгебры логики называется линейной, если ее можно представить полиномом первой степени:

$$f(x_1 x_2 \ldots x_n) = k_0 \oplus k_1 x_1 \oplus k_2 x_2 \oplus \ldots \oplus k_n x_n .$$

Примем без доказательства утверждение, что при суперпозиции линейных функций получаем линейную функцию. Существует восемь линейных функций от двух переменных (табл.10). Следовательно, в функционально полном базисе должна содержаться хотя бы одна нелинейная логическая функция.

Класс функций, сохраняющих ноль. К булевым функциям, сохраняющим константу 0, относят такие булевы функции $f(x_1,\ldots,x_n)$, для которых справедливо соотношение $f(0,\ldots,0)=0$.

Класс функций, сохраняющих единицу. К булевым функциям, сохраняющим константу 1, относят такие булевы функции $f(x_1,...,x_n)$, для которых справедливо соотношение $f(1,...,1)=1$.

Таблица 10

$k_2\ k_1\ k_0$	$f(x_1x_2\ldots x_n)=k_0\oplus k_1x_1\oplus k_2x_2$
0 0 0	0
0 0 1	1
0 1 0	x_1
0 1 1	$x_1 \oplus 1$
1 0 0	x_2
1 0 1	$x_1 \oplus 1$
1 1 0	$x_1 \oplus x_2$
1 1 1	$1 \oplus x_1 \oplus x_2$

Класс монотонных функций. Функция алгебры логики называется монотонной, если при любом возрастании набора аргументов значения этой функции не убывают. Двоичный набор $A=<a_1,a_2,...,a_n>$ не меньше двоичного набора $B=<b_1,b_2,...,b_n>$, если для каждой пары (a_i,b_i) $i = 1...n$ справедливо соотношение $a_i \geq b_i$.

Если у двух наборов есть и большие и меньшие аргументы $f(0,1)$ и $f(1,0)$, то наборы считаются несравнимыми. Таким образом, если $f(0,0)\geq f(0,1) \geq f(1,1)$ или $f(0,0) \geq f(1,0) \geq f(1,1)$, то функция f является монотонной.

Класс самодвойственных функций. Булевы функции $f_1(x_1,...,x_n)$ и $f_2(x_1,...,x_n)$ называются двойственными друг другу, если выполняется соотношение $f_1(x_1, x_2,..., x_n) = \overline{f_2}(\overline{x}_1, \overline{x}_2,..., \overline{x}_n)$.

К самодвойственным булевым функциям относят такие булевы функции, которые двойственны по отношению к самим себе, то есть булева функция называется самодвойственной, если на любых двух противоположных наборах $x_1,x_2,\ldots,x_n$ и $\overline{x}_1, \overline{x}_2,..., \overline{x}_n$ она принимает противоположные значения $f(x_1, x_2,..., x_n) = \overline{f}(\overline{x}_1, \overline{x}_2,..., \overline{x}_n)$.

Любая ФАЛ, полученная с помощью операции суперпозиции и подстановки из функций одного класса, принадлежит этому же классу.

Базисом называется полная система ФАЛ, с помощью которой любая ФАЛ может быть представлена суперпозиций исходных функций.

Теорема Поста−Яблонского. Для того чтобы система ФАЛ была полной, необходимо и достаточно, чтобы она содержала хотя бы одну функцию:

не являющуюся линейной;

не сохраняющую ноль;

не сохраняющую единицу;

не являющуюся монотонной;

не являющуюся самодвойственной.

Иначе говоря, система булевых функций является функционально полной тогда и только тогда, когда она целиком не содержится ни в одном из предполных классов.

Рассмотрим примеры ФПСБФ. Для удобства изложения материала сведем элементарные булевы функции двух переменных и некоторые функции одной переменной в таблицу (табл.11), классифицируя каждую из них по признакам принадлежности к предполным классам.

Таблица 11

Функции	Вид функции				
	Немонотонные	Нелинейные	Несамодвойственные	Не сохраняющие 0	Не сохраняющие 1
$F_0 \equiv 0$	-	-	+	-	+
$F_1 = x_1 \bullet x_2$	-	+	+	-	-
$F_3 = x_1$	-	-	-	-	-
$F_5 = x_2$	-	-	-	-	-
$F_6 = x_1 \oplus x_2$	+	-	+	-	+
$F_7 = x_1 \vee x_2$	-	+	+	-	-
$F_8 = x_1 \downarrow x_2$	+	+	+	+	+
$F_9 = x_1 \sim x_2$	+	-	+	+	-
$F_{13} = x_1 \rightarrow x_2$	+	+	+	-	+
$F_{14} = x_1/x_2$	+	+	+	+	+
$F_{15} \equiv 1$	-	-	+	+	-
$F_2(x) = x$	+	-	-	+	+

Из таблицы видно, что каждая из функций F_8 и F_{14} является ФПСБФ. Иными словами, используя, например, только булеву функцию F_{14} - "штрих Шеффера", можно записать в виде формулы любую булеву функцию. Признаком функциональной полноты, очевидно, является наличие плюса в каждом столбце таблицы, хотя бы для одной из составляющих систему булевых функций. К таким ФСПБФ, наиболее распространенным в практике построения цифровых автоматов, следует отнести:

{Λ,V,не}, { Λ,⊕,не}, { Λ,⊕,1}, { Λ,не}, {V,не}.

Иногда удобно строить ФПСБФ при наличии констант, то есть булевых функций "константа 0", "константа 1". Как следует из таблицы, функция "константа 0" несамодвойственна и не сохраняет 1; функция "константа 1" несамодвойственна и не сохраняет 0. Вместе с тем константы являются линейными и монотонными функциями. Отсюда непосредственно (на основании теоремы о функциональной полноте) вытекает следующее: система булевых функций является ослабленно функционально полной, если она содержит хотя бы одну нелинейную и хотя бы одну немонотонную булеву функцию. Примерами ослабленных ФПСБФ могут служить следующие системы:

{ Λ,⊕}, { Λ,~}, { V, ⊕}, {V,~}, {→}

Логические элементы, реализующие логические функции функционально полного набора, образуют *функционально полный набор логических элементов,* с помощью которых можно построить любую логическую схему.

Базис может быть избыточным и минимальным.

Минимальный базис – такой, что удаление одной (любой) функции превращает систему ФАЛ в неполную. Иначе говоря, функционально полный базис – это набор операций алгебры логики (и соответствующих им элементов), позволяющих построить любую функцию алгебры логики.

Функционально полным базисом является базис И, ИЛИ, НЕ. В то же время он функционально избыточен. Удаление из него элемента И или ИЛИ превращает его в минимальный базис.

Для примера рассмотрим базис, образованный, например, элементами И и НЕ. В этом базисе реализуем функцию ИЛИ, тем самым докажем функциональную полноту выбранного базиса (рис.14).

$$x_1 + x_2 = \overline{\overline{x_1 + x_2}} = \overline{\overline{x_1} \cdot \overline{x_2}}\,.$$

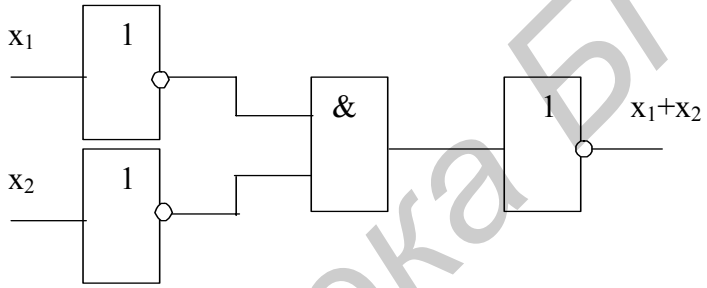


Рис. 14. Реализация операции ИЛИ в базисе И, НЕ

Минимизация ФАЛ

При проектировании цифровых автоматов широко используются методы минимизации булевых функций, позволяющие получать экономичные схемы. Общая задача минимизации булевых функций может быть сформулирована следующим образом: найти аналитическое выражение заданной булевой функции в форме, содержащей минимально возможное число букв. Следует отметить, что в общей постановке данная задача пока не решена, однако достаточно хорошо исследована в классе дизъюнктивно-конъюнктивных форм.

Минимальной дизъюнктивной нормальной формой (МДНФ) булевой функции называется ДНФ, содержащая наименьшее число букв (по отношению ко всем другим ДНФ, представляющим заданную булеву функцию).

Булева функция $g(x_1,...,x_n)$ (табл.12) называется *импликантой* булевой функции $f(x_1,...,x_n)$, если для любого набора переменных, на котором g=1, справедливо f=1.

Таблица 12

$x_3x_2x_1$	f	g_1	g_2	g_3	g_4	g_5	g_6	g_7
000	0	0	0	0	0	0	0	0
001	0	0	0	0	0	0	0	0
010	0	0	0	0	0	0	0	0

011	1	0	0	0	1	1	1	1
100	0	0	0	0	0	0	0	0
101	0	0	0	0	0	0	0	0
110	1	0	1	1	0	0	1	1
111	1	1	0	1	0	1	0	1

$f = \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + x_1x_2x_3 = g_7;$

$g_1 = x_1x_2\bar{x}_3;$

$g_2 = x_1\bar{x}_2x_3;$

$g_3 = \bar{x}_1x_2x_3 + x_1x_2\bar{x}_3 = x_1x_2\,(x_3 + \bar{x}_3) = x_1x_2;$

$g_4 = \bar{x}_1x_2x_3;$

$g_5 = \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 = x_2x_3;$

$g_6 = x_1x_2x_3 + x_1x_2x_3.$

Импликанта g булевой функции f, являющаяся элементарной конъюнкцией, называется *простой*, если никакая часть импликанты g не является импликантой функции f.

Из примера видно, что импликанты $g_3 = x_1x_2$ и $g_5 = x_2x_3$ являются простыми импликантами функции f. Импликанты g_1, g_2, g_4, g_6 не являются простыми, так как их части являются импликантами функции f, например g_1 является частью g_3. Приведем без доказательства два утверждения, полезные при получении минимальной ДНФ.

1. Дизъюнкция любого числа импликант булевой функции f также является импликантой этой функции.

2. Любая булева функция f эквивалентна дизъюнкции всех своих простых импликант. Такая форма представления булевой функции называется *сокращенной* ДНФ.

Перебор всех возможных импликант для булевой функции f из рассмотренного примера дает возможность убедиться, что простых импликант всего две: g_3 и g_5. Следовательно, сокращенная ДНФ функции f имеет вид

$f = g_3 + g_5 = x_1x_2 + x_2x_3.$

Как видно из табл. 12, импликанты g_3, g_5 в совокупности покрывают своими единицами все единицы функции f. Получение сокращенных ДНФ является первым этапом отыскания минимальных форм булевых функций. Как уже отмечалось, в сокращенную ДНФ входят все простые импликанты булевой функции. Иногда из сокращенной ДНФ можно убрать одну или несколько простых импликант, удаление которых не приводит к изменению значений функции на всевозможных значениях ее переменных. Такие простые импликанты назовем *лишними*.

Сокращенная ДНФ булевой функции называется *тупиковой*, если в ней отсутствуют лишние простые импликанты.

Исключение лишних простых импликант из сокращенной ДНФ булевой функции не является однозначным процессом, то есть булева функция может иметь несколько тупиковых ДНФ.

Тупиковые ДНФ булевой функции f, содержащие минимальное число букв, являются *минимальными* (МДНФ). МДНФ тоже может быть несколько.

Минимизировать функции, то есть находить наиболее простое выражение для исходной функции, можно различными методами. Все они практически различаются лишь на первом этапе - этапе получения сокращенной ДНФ. Следует отметить, что, к сожалению, поиск МДНФ всегда связан с некоторым перебором решений. Рассмотрим некоторые из них.

Метод Квайна

Теорема Квайна. Для получения минимальной формы логической функции необходимо в СДНФ произвести все возможные склеивания и поглощения, так чтобы в результате была получена сокращенная ДНФ. Сокращенная ДНФ в общем случае может содержать лишние простые импликанты, которые необходимо выявить на втором этапе минимизации.

На первом этапе выполняется переход от функции, заданной в форме СДНФ, к сокращенной ДНФ. Это основано на использовании следующих соотношений:

1) *соотношение неполного склеивания*

$Fx + F\bar{x} = Fx + F\bar{x} + F$, где Fx и $F\bar{x}$ - две конъюнкции, а F - любое элементарное произведение;

2) *соотношение поглощения*

$F \vee F\tilde{x} = F$.

Справедливость обоих соотношений легко проверяется. Суть метода заключается в последовательном выполнении всех возможных склеиваний и затем всех поглощений, что приводит к сокращенной ДНФ. *Метод применим к совершенной ДНФ.* Из соотношения поглощения следует, что произвольное элементарное произведение поглощается любой его частью. Для доказательства достаточно показать, что произвольная простая импликанта $p = x_{i1}x_{i2} \ldots x_{in}$ может быть получена. В самом деле, применяя к p операцию развертывания (обратную операции склеивания)

$$F = F(x \vee \bar{x}) = Fx + F\bar{x}$$

по всем недостающим переменным $x_{ik}, \ldots, x_{im}$ исходной функции f, получаем совокупность S конституент единицы. При склеивании всех конституент из S получим импликанту p. Последнее очевидно, поскольку операция склеивания обратна операции развертывания. Множество S конституент обязательно присутствует в совершенной ДНФ функции f, поскольку p - ее импликанта.

В результате выполнения склеивания получается конъюнкция n-1 ранга, а конъюнкции Fx и $F\bar{x}$ остаются в исходном выражении и участвуют в сравнении с другими членами СДНФ. Таким образом удается снизить ранг термов.

Склеивание и поглощение выполняются до тех пор, пока имеются члены, не участвовавшие в попарном сравнении. Термы, подвергшиеся операции склеивания, отмечаются. Неотмеченные термы представляют собой простые импликанты и включаются в сокращенную ДНФ. Все отмеченные конъюнкции ранга n-1 подвергаются вновь операции склеивания до получения термов n-2 ранга и так далее до тех пор, пока количество неотмеченных конъюнкций больше 2. В результате выполнения первого этапа получена сокращенная ДНФ.

Полученное логическое выражение не всегда оказывается минимальным. На втором этапе переходят от сокращенной ДНФ к тупиковым ДНФ и среди них выбирают МДНФ.

Для формирования тупиковых ДНФ строится *импликантная таблица* (*матрица*), строки которой отмечаются простыми импликантами сокращенной ДНФ, а столбцы − конституентами единицы исходной СДНФ. В строке напротив каждой простой импликанты ставится метка под теми наборами (конституентами единицы), на которых она принимает значение 1. Соответствующие конституенты поглощаются (покрываются) данной простой импликантой.

Из общего числа простых импликант необходимо отобрать их минимальное число, исключив лишние. Формирование тупиковых форм и выбор минимального покрытия начинается с выявления обязательных простых импликант, то есть таких, которые (и только они) покрывают некоторый исходный набор. Рассмотрим пример минимизации методом Квайна логической функции:

$$f_{СДНФ}= V(1,2,5,6,7)=\underset{1}{\bar{x}_1\bar{x}_2x_3}+\underset{2}{\bar{x}_1x_2\bar{x}_3}+\underset{3}{x_1\bar{x}_2x_3}+\underset{4}{x_1x_2\bar{x}_3}+\underset{5}{x_1x_2x_3}.$$

Выполним операцию склеивания:

1 – 3	(x_1)	$\bar{x}_2x_3$ 1
2 – 4	(x_1)	$x_2\bar{x}_3$ 2
3 – 5	(x_2)	x_1x_3 3
4 – 5	(x_3)	x_1x_2 4

В результате выполнения первого шага склеивания получаем четыре новые конъюнкции, простых импликант не выявлено. Полученные конъюнкции более не склеиваются и образуют сокращенную ДНФ.

$$f_{сокр\ СДНФ}=\bar{x}_2x_3+ x_2\bar{x}_3+ x_1x_3+ x_1x_2 .$$

Для выявления обязательных простых импликант и формирования на их основе минимального покрытия строится импликантная таблица (табл. 13). В строках импликантной таблицы записываются простые импликанты, а в столбцах − конституенты единицы. Звездочка ставится, если простая импликанта покрывает конституенту.

Таблица 13

	$\bar{x}_1\bar{x}_2x_3$	$\bar{x}_1x_2\bar{x}_3$	$x_1\bar{x}_2x_3$	$x_1x_2\bar{x}_3$	$x_1x_2x_3$
$\bar{x}_2x_3$	*		*		
$x_2\bar{x}_3$		*		*	
x_1x_3			*		*
x_1x_2				*	*

Простые импликанты $\bar{x}_2x_3 + x_2\bar{x}_3$ являются обязательными, так как только они покрывают конституенты $\bar{x}_1\bar{x}_2x_3$ и $\bar{x}_1x_2\bar{x}_3$ и включаются в минимальное покрытие. Остается одна непокрытая конституента $x_1x_2x_3$, которая может быть покрыта одной из двух оставшихся простых импликант. Это приводит к получению двух тупиковых форм:

$f_{МДНФ}=\bar{x}_2x_3 + x_2\bar{x}_3 + x_1x_3$ - первая тупиковая форма;

$f_{МДНФ}=\bar{x}_2x_3 + x_2\bar{x}_3 + x_1x_2$ - вторая тупиковая форма.

Метод Блейка - Порецкого

Метод позволяет получать сокращенную ДНФ булевой функции f из ее произвольной ДНФ. Базируется на применении формулы обобщенного склеивания:

$$Ax + B\bar{x} = Ax + B\bar{x} + AB,$$

справедливость которой легко доказать. Действительно,

$$Ax = Ax + ABx, \qquad B\bar{x} = B\bar{x} + AB\bar{x}.$$

Следовательно,

$$Ax + B\bar{x} = Ax + ABx + B\bar{x} + AB\bar{x} = Ax + B\bar{x} + AB.$$

В основу метода положено следующее утверждение: если в произвольной ДНФ булевой функции f произвести все возможные обобщенные склеивания, а затем выполнить все поглощения, то в результате получится сокращенная ДНФ функции f.

Рассмотрим пример. Пусть булева функция f задана произвольной ДНФ.

$f_{днф}= x_1x_2 + x_1x_2x_3 + x_1x_2.$

Необходимо, используя метод Блейка – Порецкого, получить сокращенную ДНФ функции f. Проводим обобщенные склеивания. Легко видеть, что первый и второй элемент исходной ДНФ допускают обобщенное склеивание по переменной x_1. В результате склеивания получим:

$x_1x_2 + x_1x_2x_3 = x_1x_2 + x_1x_2x_3 + x_1x_2.$

Первый и третий элемент исходной ДНФ допускают обобщенное склеивание как по переменной x_1, так и по x_2. После склеивания по x_1 имеем:

$x_1x_2 + x_1x_2 = x_1x_2 + x_1x_2 + x_2x_2.$

После склеивания по x_2 имеем:

$x_1x_2 + x_1x_2 = x_1x_2 + x_1x_2 + x_1x_1.$

Второй и третий элемент ДНФ допускают обобщенное склеивание по переменной x_2. После склеивания получаем:

$x_1x_2x_3 + x_1x_2 = x_1x_2x_3 + x_1x_2 + x_1x_3.$

Выполнив последнее обобщенное склеивание, приходим к ДНФ:

$f=x_1x_2 + x_1x_2x_3 + x_2x_3 + x_1x_2 + x_1x_3.$

После выполнения поглощений получаем:

$f=x_1x_2 + x_2x_3 + x_1x_2 + x_1x_3.$

Попытки дальнейшего применения операции обобщенного склеивания и поглощения не дают результата. Следовательно, получена сокращенная ДНФ

функции f. Далее задача поиска минимальной ДНФ решается с помощью импликантной матрицы точно так же, как в методе Квайна.

Метод минимизирующих карт Карно (Вейча)

При минимизации логической функции от небольшого числа переменных удобным является графический метод представления функции с помощью диаграмм (карт) Вейча и их разновидности - Карно. Карта Вейча представляет собой развертку n-мерного куба на плоскости. При этом вершины куба представляются клетками карты, каждой из которых поставлена в соответствие конститутиента единицы или нуля. Переменные, обозначающие клетки диаграммы, расставляются таким образом, чтобы наборы, записанные в двух смежных клетках, имели кодовое расстояние, равное единице. Поскольку такие наборы располагаются в смежных клетках, они получили название *соседних наборов*. В клетку карты, соответствующую конституенте единицы, заносится 1, иначе − 0. Таким образом, для минимизации функции она должна быть представлена в форме СДНФ. Минимизация булевой функции с использованием карт в дизъюнктивной (конъюнктивной) форме заключается в объединении единичных (нулевых) клеток в контуры, каждому такому контуру соответствует простая импликанта.

Можно сформулировать следующие правила минимизации:

- количество клеток карты в одном контуре должно быть равно 2^n;
- для контура, содержащего 2^n клеток, должно быть **n** осей симметрии;
- количество контуров должно быть минимально;
- число единиц в контуре должно быть максимально;
- контуры могут пересекаться, то есть некоторая клетка может входить в несколько контуров.

Пример: $f_{сднф}= x_1\bar{x}_2\bar{x}_3+ \bar{x}_1\bar{x}_2x_3+ x_1x_2x_3+ x_1\bar{x}_2x_3+ \bar{x}_1x_2x_3$.

На рис. 15 показана заполненная карта Вейча, соответствующая функции $f_{СДНФ}$. На карте обозначены четыре контура, каждый из которых содержит по две клетки. Контур 2 можно считать лишлишним, так как он покрывает клетки, уже покрытые двумя другими контурами (1 и 3). Аналогично можно считать лишним контур 3 (покрывается контурами 2 и 4). Здесь возможны несколько тупиковых форм ФАЛ. Таким образом, по данной карте может быть получена одна из тупиковых форм:

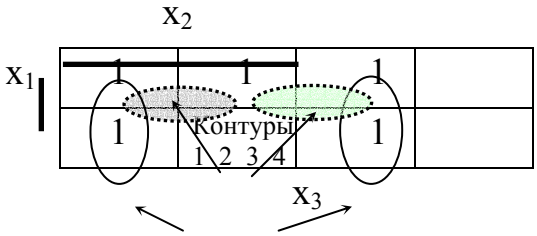


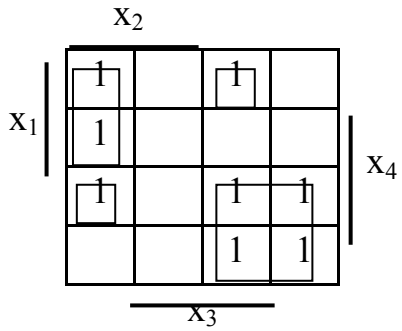
Рис. 15. Карта Вейча для $f_{СДНФ}$

$f_{\min ДНФ}= \bar{x}_2x_3+x_2\bar{x}_3+x_1x_2$ - первая тупиковая форма;

$f_{\min ДНФ}= \bar{x}_2x_3+x_2\bar{x}_3+x_1x_3$ - вторая тупиковая форма.

Если функция задана в форме ДНФ, то необязательно ее приводить к форме СДНФ, что является одним из преимуществ карты Вейча. Для этого рас-

сматривается каждый дизъюнктивный член функции в отдельности, и в соответствующие ему клетки карты заносятся единицы.



Рассмотрим сказанное на примере функции
$f_{ДНФ}=\bar{x}_1\bar{x}_2+ x_1\bar{x}_2\bar{x}_3+ x_1\bar{x}_2x_3\bar{x}_4+ \bar{x}_1\bar{x}_2x_3x_4$.
Первому члену ДНФ поставлены в соответствие четыре клетки карты, второму – две клетки, третьему и четвертому − по одной клетке соответственно (рис.16). Далее объединение единиц в контуры и выбор их минимального числа осуществляются рассмотренным выше методом.

Рис. 16. Карта Вейча для $f_{ДНФ}$

Если выбраны самые большие контуры и использовано по возможности меньшее их число, то будет получена самая простая дизъюнктивная нормальная форма. Дальнейшее упрощение можно получить за счет выполнения скобочных преобразований. Выражению с меньшим числом вхождений букв соответствует схема, имеющая меньшее число входов элементов, так что упрощение функций ведет к упрощению реализующих их схем.

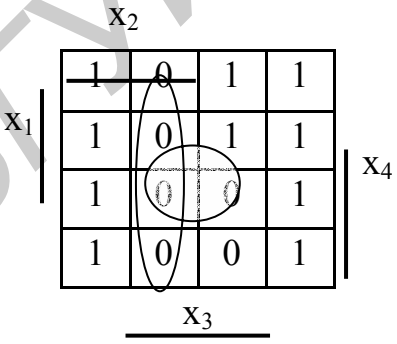


Рис.17. Карта Вейча для $f_{кнф}$

Принимая во внимание клетки карт, не содержащие единиц, и поступая с ними так же, как мы поступали с клетками, содержащими единицы, можно получать конъюнктивные нормальные формы (рис. 17).

$f_{кнф}=(x_2+x_3)(\bar{x}_1+\bar{x}_3)$.

Если логическая функция задана таблицей истинности, то более удобной для графического представления функции является карта Карно. В отличие от карты Вейча в карте Карно строки и столбцы закодированы r-разрядным кодом Грея. Код Грея – двоичный код, в котором рядом стоящие коды – соседние (их кодовое расстояние равно единице). В карте Карно каждой клетке соответствует код, состоящий из кода строки и кода столбца (рис. 18).

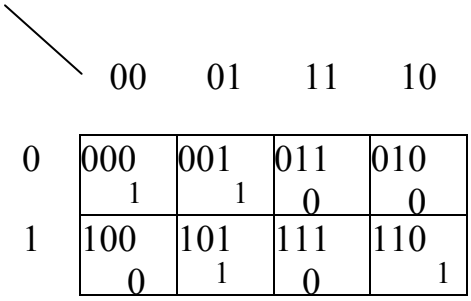
	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Рис.18. Структура карты Карно

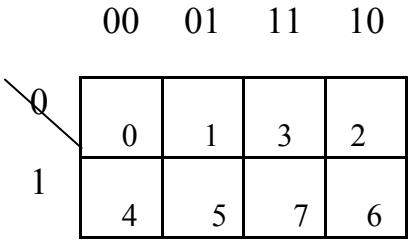
На рис. 19 показано соответствие клеток карты Карно и строк таблицы истинности. При этом в карте рис. 19,б показаны координаты единичных и нулевых значений функции, а в карте рис. 19,в показано соответствие строк таблицы истинности и ячеек карты.

	x_1	x_2	x_3	f
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

а



б



в

Рис. 19. Таблица истинности и карта Карно

Минимизация конъюнктивных нормальных форм

Минимизация КНФ производится аналогично рассмотренным методам минимизации ДНФ булевых функций, поэтому остановимся лишь на основных положениях.

Напомним, что конституентой нуля называется функция, принимающая значение 0 на одном наборе. Она выражается дизъюнкцией всех переменных функций. Например, набору 0110 соответствует конституента нуля $x_1+\bar{x}_2+\bar{x}_3+x_4$.

Имплицентой g булевой функции f называется функция, принимающая значение 0 на подмножестве нулевых наборов функции f.

Простой имплицентой функции f называется элементарная дизъюнкция, являющаяся имплицентой функции f, причем никакая ее собственная часть имплицентой функции f не является.

Задачей минимизации КНФ является определение минимальной КНФ. Эта задача также решается в два этапа: поиск сокращенной КНФ (конъюнкция всех простых имплицент) и затем нахождение минимальной КНФ. Второй этап минимизации выполняется с помощью таблицы Квайна точно так же, как и при поиске минимальной ДНФ, так как возможны только два варианта: либо данная простая имплицента поглощает данную конституенту нуля, либо нет в соответствии с соотношением поглощения:

$(A \vee x)A = A$.

Что касается первого этапа - поиска всех простых имплицент, то практически все методы минимизации ДНФ имеют свои аналоги для КНФ. Рассмотрим это подробнее.

Соотношение склеивания по Квайну:

$(A \vee x)(A \vee \bar{x}) = (A \vee x)(A \vee \bar{x})A$.

Соотношение склеивания по Блейку:

$(A \vee x)(B \vee \bar{x}) = (A \vee x)(B \vee \bar{x})(A \vee B)$.

Метод Нельсона в применении к задаче минимизации КНФ: раскрытие скобок в произвольной ДНФ функции и выполнение поглощений приводит к сокращенной КНФ. Предполагаются скобки в начале и конце каждого элемен-

тарного произведения исходной ДНФ $x_1x_2+x_1x_2$ и использование второго дистрибутивного закона. Например, функция, заданная минимальной ДНФ дает возможность определить ее сокращенную КНФ:

$(x_1 + \overline{x_1})(x_1 + x_2)\ (x_1 + \overline{x_2})(x_2 + \overline{x_2}) = (x_1 + x_2)(x_1 + \overline{x_2})$.

По диаграмме Вейча поиск минимальной КНФ осуществляется так же просто, как и в случае ДНФ. Отличие состоит лишь в том, что анализируются нулевые наборы и переменные выписываются с инверсиями. Например, для функции, заданной диаграммой (рис. 20), минимальной КНФ является

$f_{\text{min КНФ}} = (\overline{x_1} + x_3)(x_2 + x_3)(\overline{x_1} + \overline{x_2} + \overline{x_4})$.

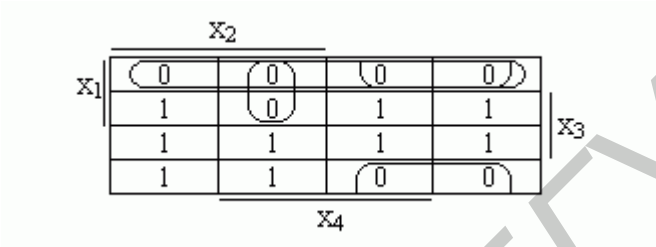


Рис. 20. Карта Вейча для поиска $f_{\text{МКНФ}}$

Для сравнения найдем минимальную ДНФ:

$f_{\text{min ДНФ}} = x_1x_2+x_2x_3+ x_3\ x_4$.

В данном случае ДНФ оказалась проще. В общем случае о сравнительной сложности минимальных ДНФ и КНФ нельзя говорить заранее.

Минимизация не полностью определенных ФАЛ

Если при синтезе логической схемы, реализующей некоторую ФАЛ n переменных, окажется, что некоторые наборы из общего числа 2^n никогда не смогут появиться на входах схемы, то данная логическая функция не определена на этих наборах. Тогда 2^n наборов переменных можно подразделить на три группы: множество наборов L, на которых функция принимает единичное значение, множество наборов D, на которых функция принимает нулевое значение, и множество наборов N, на которых функция не определена (неопределенные наборы). ФАЛ, содержащая неопределенные наборы, называется не полностью или частично определенной. Неопределенные наборы могут быть использованы для улучшения качества минимизации. При этом неопределенные наборы (при минимизации, например, картами Вейча, Карно) могут участвовать в образовании контуров как с единичными, так и с нулевыми наборами. Это приводит к формированию более простой минимизированной логической функции.

Рассмотрим пример. Пусть задана $f_{\text{СДНФ}} = \overline{x_1}x_2x_3 \vee x_1\overline{x}_2x_3 \vee x_1x_2\overline{x_3}$.

Звездочками на карте (рис. 21) отмечены наборы, на которых функция f не определена. Если не учитывать неопределенные наборы, то минимальная форма будет иметь вид: $f_{МДНФ} = x_2\bar{x}_3 + x_1\bar{x}_2x_3$. В случае если неопределенные наборы участвуют в образовании контуров, а следовательно, и $f_{МДНФ}$, то функция примет следующий вид: $f_{МДНФ} = x_2\bar{x}_3 + x_1\bar{x}_2$. Таким образом, схемная реализация полученной $f_{МДНФ}$ будет "дешевле".

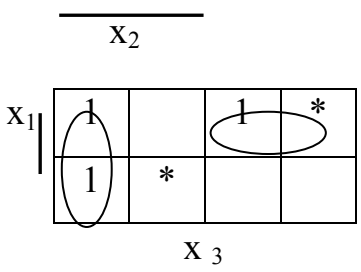


Рис.21. Карта Вейча

Приведем примеры минимизации частичных булевых функций (рис.22).

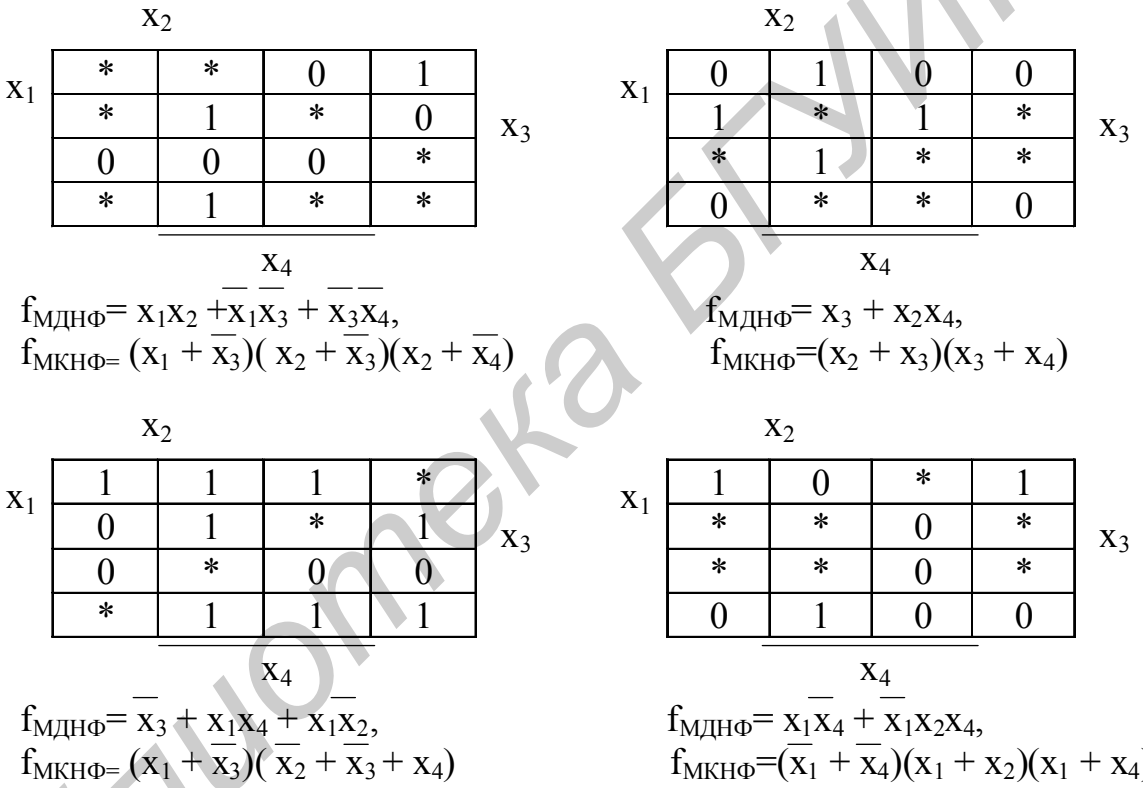


Рис. 22. Примеры минимизации функций

Кубическое задание функций алгебры логики

Как следует из рассмотренного выше, функция алгебры логики (булева функция) может быть задана:

- аналитически (системой булевых функций);
- словесным описанием;
- таблицей истинности;
- картами (диаграммами) Венна, Вейча, Карно;
- логической схемой.

Более компактной формой записи функций алгебры логики является форма, когда вместо полного перечисления всех конъюнкций (дизъюнкций) ис-

пользуют номера наборов, на которых функция принимает единичное значение. Так, например, форма записи $f(x_1x_2x_3)=V\,F(0,2,3)$ означает, что функция от трех переменных принимает единичное значение на 0, 2 и 3 наборах таблицы истинности. Такая форма записи называется *числовой*.

Некоторые методы минимизации ориентируются на задание булевой функции в виде кубического покрытия. При этом логическая функция удобно интерпретируется с использованием ее геометрического представления. Так, функцию двух переменных можно интерпретировать как плоскость, заданную в системе координат x_1x_2. Получится квадрат, вершины которого соответствуют комбинациям переменных. Для геометрической интерпретации функции трех переменных используется куб, заданный в системе координат $x_1x_2x_3$.

Новое представление булевой функции получается путем отображения булевой функции n переменных на n-мерный куб (n-куб).

Для отображения булевой функции n переменных на n-кубе устанавливается соответствие между членами СДНФ и вершинами n-куба. На n-кубе определим координатную систему с координатами $(e_1,.....,e_n)$, $e_i=0,1$.

Установим соответствие между членом СДНФ $x_1^{e1}\,x_2^{e2}...\,x_n^{en}$ и некоторой вершиной $e_1,e_2,\,....,e_n$ куба. При этом $x_i^{ei}=x_i$, если $e_i=1$, и $x_i^{ei}=\bar{x}_i$, если $e_i=0$.

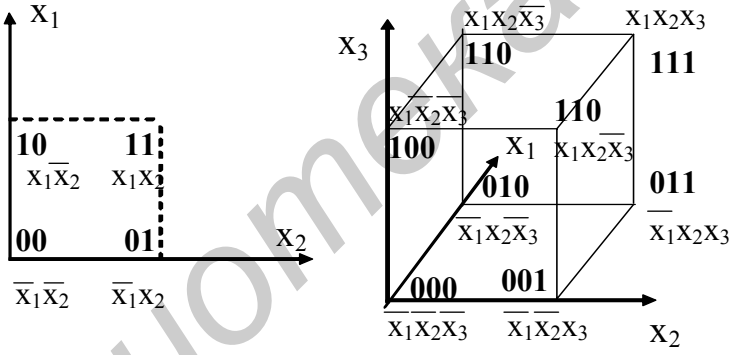


Рис.23. Геометрическое представление функции двух и трех переменных

Каждый набор при кубическом задании ФАЛ называется кубом.

$$f = \bar{x}_1x_2x_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1\bar{x}_2x_3 \vee x_1x_2\bar{x}_3 \vee x_1x_2x_3.$$

Как следует из таблицы истинности (табл. 14), функция f определена на трех группах наборов переменных: L={3,4,5,6,7}, D={0,2} и N={1}.

Конъюнкции максимального ранга (конституенты единицы) принято называть 0-кубами. Множество 0-кубов образуют кубический комплекс

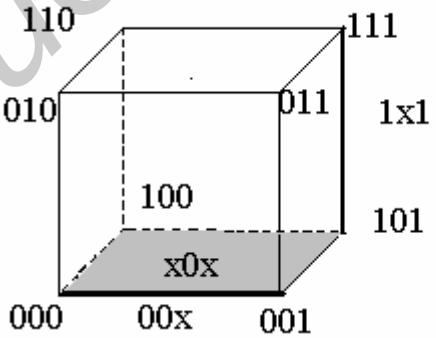
Таблица 14

	x_1	x_2	x_3	f
0	0	0	0	0
1	0	0	1	-
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

$$K^0 = \left\{ \begin{matrix} 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{matrix} \right\}.$$

Над 0-кубами, кодовое расстояние которых равно 1, выполняется операция склеивания, в результате которой образуются новые кубы, содержащие свободные координаты. *Свободная (независимая)* координата может принимать как нулевое, так и единичное значение, остальные компоненты называются *связанными*. Куб, содержащий свободные координаты, покрывает кубы, на которых он образован. Куб с одной независимой координатой x называется *кубом первой размерности* и в геометрическом представлении это ребро, покрывающее обе вершины. Кубы, образующиеся в результате последовательного выполнения операции склеивания, назовем r-кубами, где r – размерность полученного куба.

Геометрическая интерпретация сказанного приведена на рис. 24. В результате склеивания кубов 101 и 111 (0-кубы, вершины) образован 1-куб 1x1 (ребро), а 1-кубов 00x и 10x - 2-куб x0x (грань).



Ри

Кубическое еремя переменными 0,1,x вместо x_1, x_2

Количество его размерность r, чем i-го куба. больше r, тем больше свободных координат и тем меньше входов будет иметь реализующая его схема (логический элемент).

Цена схемы определяется количеством входов элементов, используемых для ее реализации:

$$C = \sum_{i=1}^{k} (n - r_i) + k,$$

где k − количество полученных кубов; $n\text{-}r_i$ − количество единичных и нулевых значений

Два r-куба могут образовать r+1-куб, если в этих r-кубах все координаты, в том числе и свободные, совпадают, за исключением лишь какой-либо одной, которая в этих кубах имеет противоположное значение.

На рис. 25 приведено изображение куба, соответствующего булевой функции от четырёх переменных (гиперкуб).

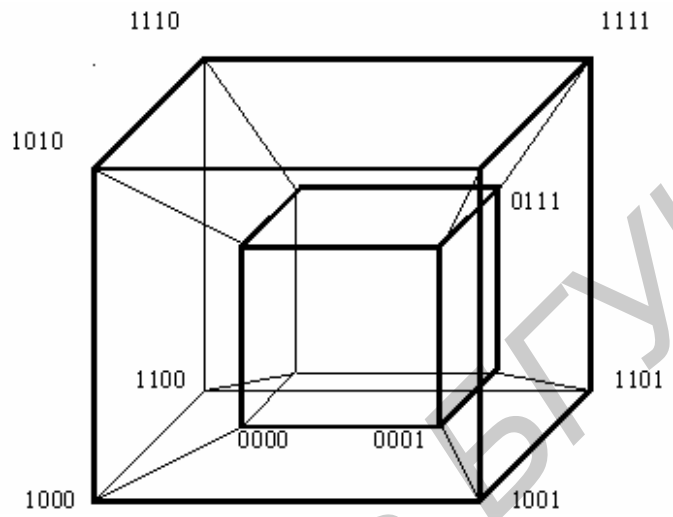


Рис. 25. Геометрическое представление функции четырех переменных

Как следует из рисунка, геометрическое представление 4-куба уже довольно сложное. Поэтому для функций, зависящих более чем от четырёх переменных, предпочтительным является аналитическое представление булевых функций.

Метод Квайна −Мак-Класки

Этот метод ориентирован на числовое задание ФАЛ в виде кубического комплекса, состоящего из 0-кубов. Метод представляет собой формализованный на этапе нахождения простых импликант метод Квайна. Основной недостаток метода Квайна – необходимость выполнения полного сравнения (склеивания) всех первичных импликант. В случае большого их количества сложность этого сравнения существенно возрастает. В рассматриваемом методе все исходные n-кубы разбиваются на непересекающиеся подгруппы по количеству единиц в кубе. Пусть, например, задана функция

$f_{СДНФ}(x_1x_2x_3x_4) = V(2, 3, 4, 6, 9, 10, 11, 12)$.

Сформируем кубический комплекс K, состоящий из 0-кубов:

K=(0010, 0011, 0100, 0110, 1001, 1010, 1011, 1100).

Выполнив разбиение комплекса K на группы, получим:

$$K_1^0 = \begin{Bmatrix} 0010 \\ 0100 \end{Bmatrix}, \qquad K_2^0 = \begin{Bmatrix} 0011 \\ 1100 \\ 0110 \\ 1001 \\ 1010 \end{Bmatrix}, \qquad K_3^0 = \{1011\}.$$

Попарное сравнение можно проводить только между соседними по номеру группами кубов, так как кубы, порождающие новые кубы, должны иметь кодовое расстояние, равное 1. В результате сравнения кубов получим:

$$K_1^0 \text{ и } K_2^0 \Rightarrow K_1^1 = \begin{Bmatrix} 001x \\ 0x10 \\ x010 \\ x100 \\ 01x0 \end{Bmatrix}, \qquad K_2^0 \text{ и } K_3^0 \Rightarrow K_2^1 = \begin{Bmatrix} x011 \\ 10x1 \\ 101x \end{Bmatrix}.$$

После выполнения первого шага алгоритма простых импликант не выявлено. Полученные 1-кубы разобьем на n групп кубов в зависимости от местоположения свободной координаты в кубе.

$$K_1^1 = \begin{Bmatrix} x011 \\ x100 \\ x010 \end{Bmatrix}, \qquad K_2^1 = \{0x10\}, \qquad K_3^1 = \begin{Bmatrix} 10x1 \\ 01x0 \end{Bmatrix}, \qquad K_4^1 = \begin{Bmatrix} 001x \\ 101x \end{Bmatrix}.$$

Далее выполняется сравнение кубов внутри каждой из групп. В результате могут быть получены 2-кубы, которые, в свою очередь, аналогично 1-кубам будут объединены в группы по совпадению свободных координат и вновь будет выполнено сравнение. На каждом шаге сравнения выявляются кубы, не участвовавшие в образовании новых кубов, и исключаются из дальнейшего упрощения. Для рассматриваемого примера сравнение в группах $K_1^1 \ldots K_4^1$ привело к образованию двух новых кубов x01x и x01x и кубов, не образовавших новых {x100, 0x10, 10x1, 01x0}.

$$K_1^2 = \{x01x\}, \qquad K_2^2 = \{x01x\}.$$

Дальнейшее сравнение не приводит к формированию новых кубов $K_1^2 = K_2^2 = \{x01x\}$. Таким образом, получено множество простых импликант:

$f_{\text{сокр.ДНФ}}$={x100, 0x10, 10x1, 01x0, x01x} .

Далее аналогично методу Квайна строится импликантная таблица (табл.15). Формирование минимального покрытия сводится к выявлению обязательных простых импликант и построению на их основе тупиковых форм.

Таблица 15

Простые импликанты	Конституенты единицы							
	0010	0100	0011	1100	0110	1001	1010	1011
X100		*		*				
0x10	*				*			
10x1						*		*
01x0		*			*			
X01x	*		*				*	*

Из таблицы следует, что простые импликанты x100, 10x1, x01x являются обязательными. Оставшиеся две простые импликанты не являются обязательными и образуют следующие две тупиковые формы.

$f_{МДНФ}$ = {x100, 10x1, x01x, 0x10} – 1-я тупиковая форма;

$f_{МДНФ}$ = {x100, 10x1, x01x, 01x0} – 2-я тупиковая форма.

Алгоритм извлечения (Рота)

Метод Рота ориентируется на задание логической функции в форме *произвольного кубического покрытия*, что позволяет упростить процесс подготовки выражения для минимизации. Достоинство алгоритма Рота – полная формализация действий на всех этапах минимизации функции.

Специальные логические операции алгоритма Рота: *, ∩ , #

Реализация алгоритма извлечения осуществляется на основе специальных логических операций, которые позволяют полностью формализовать процесс получения минимальной формы.

Операция умножения кубов (*). Операция умножения кубов $a=a_1a_2...a_n$ и $b=b_1b_2...b_n$ обозначается как $c=a*b$ и служит для образования r-куба, противоположные (r-1) грани которого содержатся в кубах a и b. Предварительные координаты куба c определяются в соответствии с таблицей, приведенной ниже.

a_i \ b_i	*	0	1	x
	0	0	Y	0
	1	y	1	1
	x	0	1	x

Окончательно координаты куба формируются:

$$a*b=\begin{cases} m(a_1*b_1)m(a_2*b_2)\ ...\ m(a_n*b_n), & \text{если } a_i*b_i=y \text{ только для одного i,} \\ \varnothing, & \text{если } a_i*b_i=y \text{ для } i \geq 2, \end{cases}$$

где $m(a_i*b_i)$ – окончательная i-я координата куба c; m(0)=0; m(1)=1; m(x)=x;

y – условное обозначение того, что координаты a_i и b_i противоположны.

Эта операция соответствует операции склеивания: образуется новый r-куб, если кодовое расстояние двух исходных кубов равно 1.

Рассмотрим некоторые примеры, графическая интерпретация которых приведена на рис. 26.

1) 011
*001
0y1 ⇒ 0x1 – ребро покрывает две вершины

2) 11x
*01x
y1x ⇒ x1x – грань

3) 0x1
*1x0
уху $\Rightarrow \varnothing$ – две координаты имеют значение у

4) x1x
*011
011

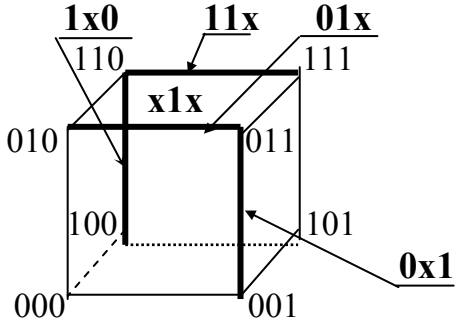


Рис. 26. Графическая интерпретация операции *

Операция пересечения кубов (∩). Операция пересечения кубов $a=a_1a_2...a_n$ и $b=b_1b_2...b_n$ обозначается как $c=a\cap b$ и служит для выделения куба $c=c_1c_2...c_n$, являющегося общей частью кубов a и b. Координаты $c_1c_2...c_n$ определяются согласно следующей таблице.

a_i \ b_i	0	1	x
0	0	∅	0
1	∅	1	1
x	0	1	x

$$a\cap b=\begin{cases} m(a_1\cap b_1)m(a_2\cap b_2)\ ...\ m(a_n\cap b_n,), \\ \varnothing, \text{ если существует такое i, для которого } a_i\cap b_i=\varnothing, \end{cases}$$

где $m(a_i*b_i)$ – окончательная i-я координата куба c; m(0)=0; m(1)=1; m(x)=x.

Рассмотрим примеры и их графическую интерпретацию (рис. 27).

1) ∩100
001
∅0∅ $\Rightarrow \varnothing$ общей части у кубов нет

2) ∩1x0
10x
100 $\Rightarrow$ 100 общая часть кубов (рёбер) - вершина

3) ∩x1x
0xx
01x $\Rightarrow$ 01x общая часть – ребро

4) ∩0xx
0x0
0x0 (совпадает с операцией *)

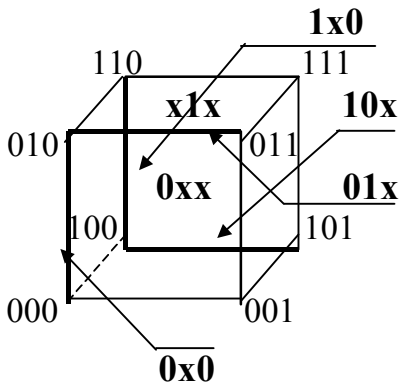


Рис. 27. Графическая интерпретация операции ∩

Операция вычитания кубов (#). Операция вычитания из куба $a=a_1a_2...a_n$ куба $b=b_1b_2...b_n$ обозначается как c=a#b и служит для удаления из куба a общей части кубов a и b.

Координаты куба c формируются согласно следующей таблице.

a_i \ b_i	#	0	1	x
0	z	y	z
1	y	z	z
x	1	0	z

$$c=a\#b=\begin{cases}\varnothing, & \text{если для любого i } a_i\#b_i=z,\\ a, & \text{если существует такое i, что } a_i\#b_i=y,\\ \bigcup\limits_{i=1} a_1a_2...a_{i-1}\,\alpha_i\,a_{i+1}...a_n, & \text{если } \alpha_i \text{ равно 0 или 1 для одного или нескольких i,}\end{cases}$$

где z означает, что координаты совпадают, а y, как для операции *, означает, что координаты a_i и b_i противоположны.

По этим α_i-координатам производится объединение (U) кубов, получаемых в результате замены в кубе a символа x на соответствующее значение (0,1) координаты α_i. Рассмотрим примеры выполнения операции # и их графическую интерпретацию (рис. 28).

1) $_{\#}$x1x
x11
zz0 ⇒ x10

2) $_{\#}$x1x
110
0z1 ⇒ {01x, x11}

3) $_{\#}$xx1
x10
z0y ⇒ xx1

4) $_{\#}$x11
xx1
zzz ⇒ ∅

5) # 0xxx
xx01
zz10 ⇒ { 0x1x, 0xx0 }

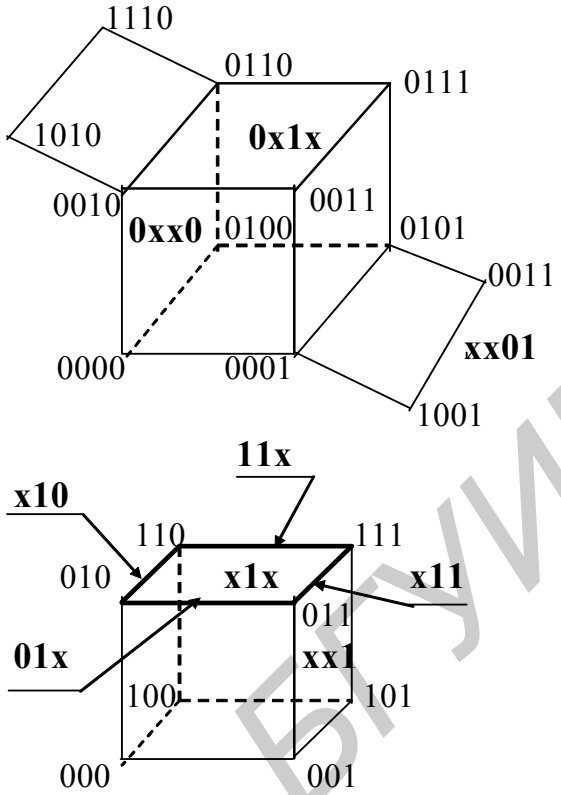
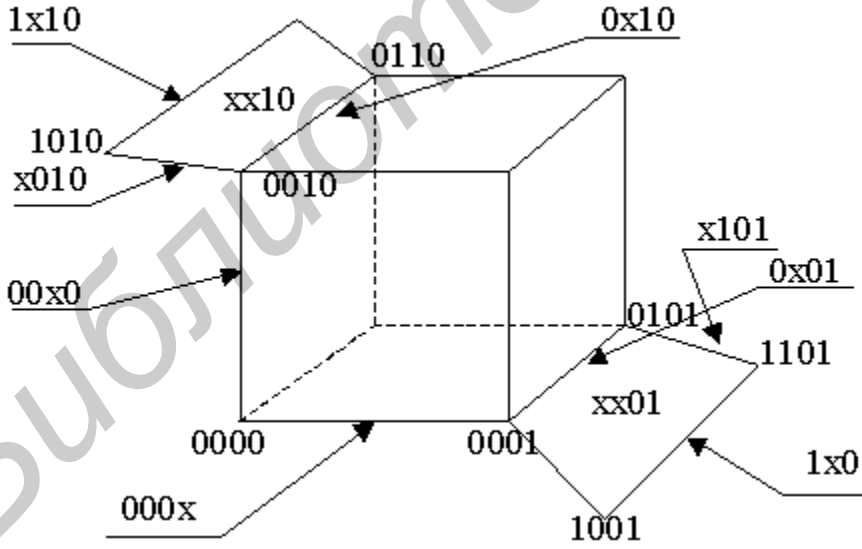


Рис. 28. Графическая интерпретация операции #

Далее рассмотрим алгоритм извлечения (Рота) на примере минимизации булевой функции, заданной покрытием C_0 (рис.29).



$$C_0 = \begin{cases} L = \begin{cases} x010 \\ 0x10 \\ 0000 \\ 0x01 \end{cases} \\ N = \begin{cases} 1110 \\ 1x01 \end{cases} \end{cases}$$

Рис. 29. Геометрическое задание исходного покрытия

Исходное покрытие C_0 задано объединением множеств кубов L и N. Кубы комплекса N − это наборы, на которых функция не определена и которые включаются в покрытие ради возможного дополнительного упрощения комплекса L в процессе минимизации. Минимальное покрытие комплексов L и N, C_{min} называется К-покрытием L.

Общий алгоритм построения минимального К-покрытия называется алгоритмом извлечения и состоит в следующем:

- нахождении множества Z простых импликант комплекса К;
- выделении L-экстремалей на множестве Z;
- применении алгоритма ветвления при отсутствии L-экстремалей;
- нахождении абсолютно минимального покрытия из некоторого множества избыточных покрытий.

1. Нахождение множества простых импликант

Преобразование исходного покрытия C_0 комплекса К в множество простых импликант Z осуществляется с помощью операции умножения кубов. В результате первого шага (C_0*C_0) (табл. 16) предусматривается выявление как новых кубов C_y (первой и более высокой размерности), так и кубов, которые не образуют новых кубов (включаются в множество Z_0). Из полученных новых кубов образуется множество A_1. Также формируется множество $B_1=C_0-Z_0$. Для следующего шага получения множества Z формируется множество $C_1=A_1 \cup B_1$. Для уменьшения мощности множества кубов C_1 выполним операцию поглощения (удаления) кубов, образующих множество C_1, кубами из множества A_1 ($A_1 \subseteq C_1$).

Таблица 16

C_0*C_0	x010	0x10	0000	0x01	1110	1x10
x010	-					
0x10	0010	-				
0000	00y0	00y0	-			
0x01	ø	ø	000y	-		
1110	1y10	y110	ø	ø	-	
1x01	ø	ø	ø	yx01	ø	-
A_1	00x0 1x10	x110	000x	xx01		

Для рассматриваемого примера получим:

$$A_1=\begin{Bmatrix}00x0\\1x10\\x110\\000x\\xx01\end{Bmatrix} \qquad C_1=\left\{\begin{matrix}\begin{Bmatrix}00x0\\1x10\\000x\\x110\\\underline{xx01}\end{Bmatrix} A_1\\ \begin{Bmatrix}x010\\0x10\\0000\\\underline{0x01}\\1110\\1x01\end{Bmatrix} B_1\end{matrix}\right. \quad \begin{matrix}\text{после выполнения}\\ \Rightarrow \text{ операции } \Rightarrow C_1=\\ \text{поглощения}\end{matrix} \quad \begin{Bmatrix}00x0\\1x10\\000x\\x110\\xx01\\x010\\0x10\end{Bmatrix}$$

$Z_0=\varnothing$

Среди кубов C_0, возможно, находятся такие кубы, которые с кубами множества A_1 могут дать новые кубы или оказаться простыми импликантами после

второго шага (C_1*C_1). При формировании таблицы для выполнения операции C_1*C_1 (табл. 17) следует учесть, что B_1*B_1 уже выполнялось на шаге C_0*C_0. Следовательно,

C_1*C_1=(A_1UB_1)*(A_1UB_1)=(A_1*A_1)U(A_1*B_1)U(B_1*A_1)U(B_1*B_1)=(A_1*A_1)U(A_1*B_1).

Таблица 17

C_1*C_1	00x0	1x10	000x	x110	xx01
00x0	-				
1x10	y010	-			
000x	0000	ø	-		
x110	0y10	1110	ø	-	
xx01	000y	ø	0001	ø	-
x010	0010	1010	00y0	xy10	Ø
0x10	0010	yx10	00y0	0110	Ø
A_2	ø	xx10	ø	xx10	Ø

В результате выполнения умножения C_1*C_1 получим:

A_2={xx10},

Z_1= $\left\{\begin{matrix}00x0\\000x\end{matrix}\right\}$.

Необходимо отметить, что куб xx01 не дал нового куба. Но это куб второй размерности и новые кубы может дать на третьем шаге (C_2*C_2) (табл.18). Поэтому его не следует включать в число кубов, образующих множество Z_1.

$$B_2=\left\{\begin{matrix}1x10\\x110\\xx01\\x010\\0x10\end{matrix}\right\},\quad C_2=A_2UB_2=\left\{\begin{matrix}xx10\\1x10\\x110\\x010\\0x10\\xx01\end{matrix}\right\}=\left\{\begin{matrix}xx10\\xx01\end{matrix}\right\}.$$

Таблица 18

C_2*C_2	xx10
xx10	-
xx01	Ø
A_3	Ø

Таким образом, получим A_3= Ø, следовательно, новых кубов нет.

Z_2= $\left\{\begin{matrix}xx10\\xx01\end{matrix}\right\}$.

B_3=C_2-Z_2= Ø; C_3=A_3UB_3= Ø.

На этом процесс выявления простых импликант окончен.

$$Z = \bigcup_{i=1}^{k} Z_i,$$

$Z=Z_0 U Z_1 U Z_2=\left\{\begin{matrix}00x0\\000x\\xx01\\xx10\end{matrix}\right\}$ - сформированное множество простых импликант.

Необходимо выяснить, не содержатся ли в этом множестве “лишние” простые импликанты.

2. Определение L-экстремалей

Множество Z может быть избыточным. Прежде всего необходимо выявить обязательные простые импликанты, называемые в алгоритме извлечения L-экстремалями. L-экстремаль – это куб, который (и только он) покрывает некоторую вершину из множества L, не покрываемую никаким другим кубом из множества Z.

Для определения L-экстремалей воспользуемся операциями вычитания (#) (табл. 19) и пересечения ($\cap$) кубов (табл.20). В табл. 19 $z \subseteq Z$ – некоторая простая импликанта, из которой вычитаются остальные Z-z.

Таблица 19

z#(Z-z)	00x0	000x	xx01	xx10
00x0	-	zzz1 0001	11zy xx01	11zz 1x10 x110
000x	zz1z 0010	-	11zz 1x01 x101	y1yz 1x10 1yyz x110
xx01	zzyy 0010	zzzz ø	-	
xx10	zzzz ø	ø	zzyy 1x01 zzyy x101	-
Остаток	ø	ø	1x01 x101	1x10 x110

Таким образом, из таблицы получено множество L-экстремалей.

$$E = \left\{\begin{matrix}xx01\\xx10\end{matrix}\right\}.$$

1. Если результат вычисления будет Ø хотя бы в одном, любом случае, то это значит, что среди простых импликант есть такие кубы, которые покрывают *уменьшаемый*, а следовательно, этот уменьшаемый не может быть L-экстремалью.

2. Если же полученный результат не Ø, то в противоположность преды-

дущему утверждению *уменьшаемый* куб оказывается кубом большей размерности по отношению к другим простым импликантам.

3. Что касается простых импликант, ”удаленных” от *уменьшаемой*, то они с ней дают координаты ”у” и, таким образом, остается *уменьшаемый* куб при вычитании этих ”*удаленных”* кубов.

После выявления L-экстремалей следует выяснить, не являются ли некоторые из них простыми импликантами, остатки которых покрывают только некоторое подмножество кубов комплекса N, которое нет необходимости покрывать, вводя в минимальное покрытие соответствующие наборы. Для этого необходимо выполнить операцию пересечения остатков, полученных при выполнении операции z#(Z-z) с кубами из комплекса L. Во множестве E необходимо оставить только те кубы, остатки от которых пересекаются с кубами из комплекса L.

Таблица 20

z#(Z-z)∩L	1x01	x101	1x10	x110
x010	ø	ø	1010	ø
0x10	ø	ø	1010	ø
0000	ø	ø	ø	0110
0x01	ø	1101	ø	ø

Из таблицы видно, что куб 1x01 не пересекается с кубами комплекса L. Однако куб x101 имеет с кубом 0x01 (из комплекса L) общую вершину 0101. Оба куба (1x01, x101) входят в куб более высокой размерности xx01 (L-экстремаль). Таким образом, куб 1x01, образованный на комплексе N, позволил уменьшить цену схемы. Выясним далее, какие из вершин комплекса L не покрываются L-экстремалями. Для этого из каждого куба комплекса L вычтем (#) элементы множества E (табл.21). В результате вычитания получим L_1=L#E.

Таблица 21

L#E	x010	0x10	0000	0x01
xx01	zzyy x010	zzyy 0x10	zzzy 0000	zzzz ø
xx10	zzzz ø	zzzz ø	zzyz 0000	ø
	ø	ø	0000	ø

Из таблицы видно, что L_1={0000}. Однако не покрытые L-экстремалями кубы должны быть покрыты другими импликантами из множества.

$$Z=Z\text{-}E=\begin{Bmatrix}00x0\\000x\\xx01\\xx10\end{Bmatrix}-\begin{Bmatrix}xx01\\xx10\end{Bmatrix}=\begin{Bmatrix}00x0\\000x\end{Bmatrix}.$$

Теперь из полученного множества Z надо выбрать минимальное число кубов с минимальной ценой (максимальной размерностью), чтобы покрыть непокрытые L-экстремалями элементы комплекса L. Выбор так называемого немаксимального куба осуществляется с помощью операции частичного упорядочивания кубов (табл. 22).

Куб a будет немаксимален по отношению к кубу b, если выполняются одновременно два условия:

1) $C^a \geq C^b$, где C^a – цена куба a;

2) $a \cap L_1 \subseteq b \cap L_1$, куб b покрывает не меньше кубов чем куб a.

$$Z = \begin{Bmatrix} 00x0 \\ 000x \end{Bmatrix} \begin{matrix} a \\ b \end{matrix}$$

Таблица 22

	∩	0000
a	00x0	0000
b	000x	0000

$C^a = C^b$

Следовательно, кубы a и b равноценны и для покрытия вершины 0000 можно выбрать любой из них в качестве экстремали второго порядка

$E'_2 = \{000x\}$ или $E''_2 = \{00x0\}$.

Следовательно, могут быть получены две тупиковые формы.

$$f^1{}_{МДНФ} = C_{min} = E_1 \cup E_2' = \begin{Bmatrix} xx10 \\ xx01 \\ 000x \end{Bmatrix}$$ - первая тупиковая форма (рис. 30).

$$f^2{}_{МДНФ} = C_{min} = E_1 \cup E_2'' = \begin{Bmatrix} xx10 \\ xx01 \\ 00x0 \end{Bmatrix}$$ - вторая тупиковая форма.

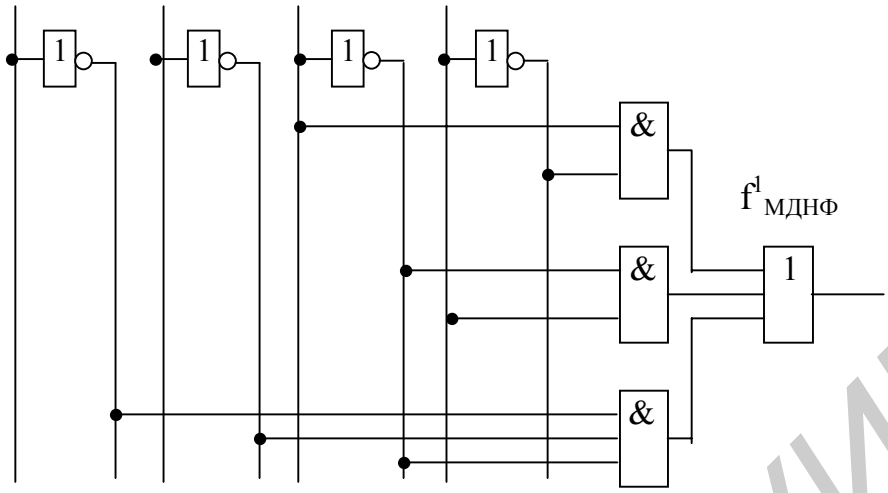


Рис. 30. Реализация $f^1_{МДНФ}$

Минимизация ФАЛ методом преобразования логических выражений

Рассмотрим подход к упрощению ФАЛ, заключающийся в применении к ней скобочных преобразований. Пусть имеется функция

$f=x_1x_3x_4x_6+x_2x_3x_4x_6 + x_5x_6 + x_7$.

Применим к ней скобочные преобразования, в результате чего получим функцию $f=((x_1+x_2)x_3x_4 + x_5)x_6 + x_7$.

Из выражений видно, что цена схемы до минимизации была равна 14, после стала равна 11. Таким образом, общая стоимость схемы сократилась, однако исходной функции соответствовала схема, имеющая два уровня элементов, а преобразованной − пять уровней. Таким образом, полученная схема будет работать примерно в 2,5 раза медленнее исходной.

Применение правил и законов алгебры логики к синтезу некоторых цифровых устройств

Синтез одноразрядного полного комбинационного сумматора

Пусть имеется два числа:

$A=a_1a_2 \ldots a_{i-1}a_ia_{i+1} \ldots a_n$,

$B=b_1b_2 \ldots b_{i-1}b_ib_{i+1} \ldots b_n$.

В зависимости от значений аргументов a_i, b_i, z_i (перенос в i-й разряд) формируется значение булевых функций C_i и $П_i$. Введем следующие обозначения.

$a_i \Rightarrow x$ $\quad C_i \Rightarrow C$, где C_i – значение суммы в разряде i

$b_i \Rightarrow y$ $\quad П_i \Rightarrow П$ $\quad П_i$ – значение переноса из разряда i

$z_i \Rightarrow z$

Таблица истинности, отражающая алгоритм работы сумматора, имеет следующий вид (табл. 23).

Таблица 23

x	Y	z	C	П
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\text{П} = \bar{x}\,yz + x\,\bar{y}\,z + x\,y\bar{z} + x\,y\,z =$$
$$= \bar{x}\,yz + x\,\bar{y}\,z + x\,y\bar{z} + x\,y\,z + x\,y\,z + x\,y\,z =$$
$$= yz\cdot(\bar{x} + x) + x\,z(\bar{y} + y) + x\,y(\bar{z} + z).$$
$$\text{П} = yz + x\,z + x\,y.$$
$$\text{C} = \bar{x}\,\bar{y}\,yz + \bar{x}\,y\,\bar{z} + x\,\bar{y}\,\bar{z} + x\,y\,z +$$
$$+ \bar{x}\,\bar{y}\,yx + \bar{x}\,x\,\bar{z} + y\,\bar{y}\,\bar{z} \quad \Leftarrow \text{ Логические нули}$$
$$+ \bar{x}\,\bar{y}\,y + \bar{x}\,z\,\bar{z} + z\,\bar{y}\,\bar{z} =$$
$$= (x + y + z)(\bar{x}\,\bar{y} + \bar{x}\,\bar{z} + \bar{y}\,\bar{z}) + x\,y\,z,$$
$$\text{но} \quad \bar{\text{П}} = \overline{yz + x\,z + x\,y} = (\bar{x} + \bar{y})(\bar{x} + \bar{z})(\bar{y} + \bar{z}) =$$
$$= \bar{x}\,\bar{y} + \bar{x}\,\bar{z} + \bar{y}\,\bar{z}.$$

Запись одной функции с участием другой носит название совместной минимизации. Следовательно, с учетом этого функция **C** будет иметь вид

$$\text{C} = (x + y + z)\cdot\bar{\text{П}} + x\,y\,z.$$

Таким образом, логическая схема синтезированного одноразрядного полного комбинационного сумматора имеет следующий вид (рис. 31).

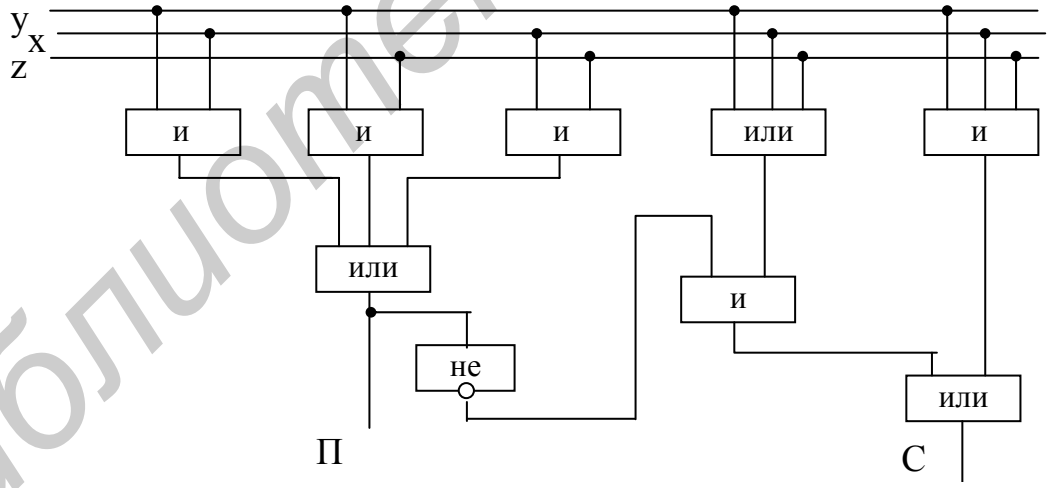


Рис. 31. Логическая схема полного комбинационного сумматора

Синтез одноразрядного комбинационного полусумматора

Одноразрядный полусумматор − это устройство для сложения разрядов двух чисел без учета переноса из предыдущего разряда, имеющее два входа (два суммируемые разряды) и два выхода (суммы и переноса).

Таблица истинности, отражающая алгоритм работы полусумматора, имеет следующий вид (табл. 24).

Таблица 24

x	y	C	П
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$П = x\,y$$

$$C = \bar{x}\,y + x\,\bar{y}$$

Логическая схема, соответствующая записанной системе булевых функций представлена на рис.32.

Данная схема может быть упрощена, если функция C будет записана на нулевых наборах и использована совместная минимизация.

$$\bar{C} = \bar{x}\,\bar{y} + x\,y$$

$$C = \bar{\bar{C}} = \overline{\bar{x}\,\bar{y} + x\,y} = \overline{\bar{x}\,\bar{y}}\,\overline{(x+y)} = (x+y)\cdot\bar{П}$$

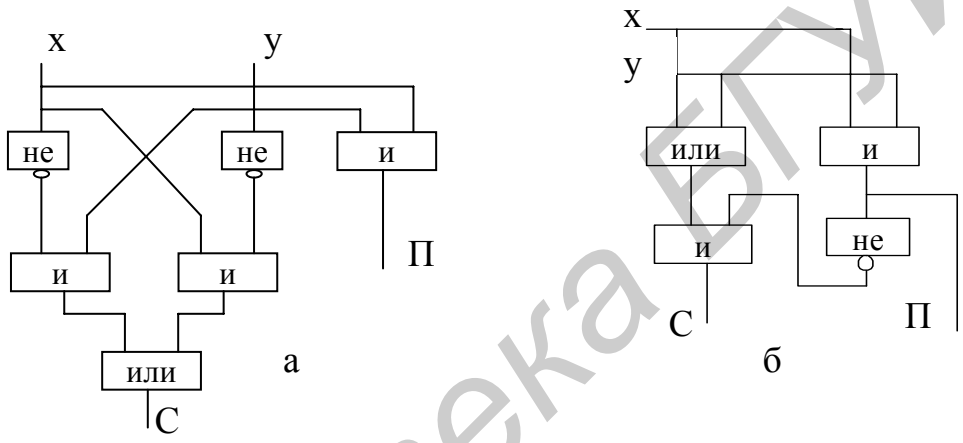


Рис. 32 Логические схемы полусумматора:
а - до упрощения; б – после упрощения

Синтез одноразрядного полного комбинационного сумматора на двух полусумматорах

Согласно рассмотренному выше материалу функция суммирования для полного комбинационного сумматора имеет вид

$$C = \bar{x}\,\bar{y}z + \bar{x}\,y\,\bar{z} + x\,\bar{y}\bar{z} + x\,y\,z = (\overline{x\,y} + x\,\bar{y})\,\bar{z} + (\bar{x}\,\bar{y} + x\,y)\,z.$$

Введем обозначение $M = x \oplus y$, тогда

$$C = M\bar{z} + \bar{M}\,z = M \oplus z = x \oplus y \oplus z.$$

Аналогично можно выполнить преобразование функции переноса П:

$$П = \bar{x}\,yz + x\,\bar{y}\,z + x\,y\bar{z} + x\,y\,z = (\bar{x}\,y + x\,\bar{y})\,z + x\,y(\bar{z} + z) = (x \oplus y)z + xy.$$

Таким образом, схема полного сумматора на двух полусумматорах будет иметь следующий вид (рис. 33).

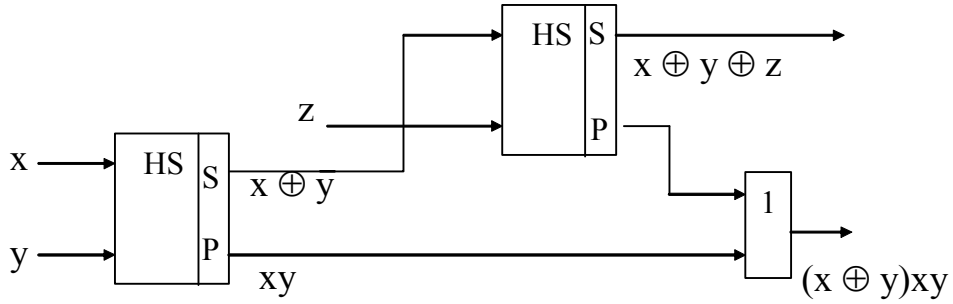


Рис. 33. Схема сумматора на двух полусумматорах

Синтез одноразрядного комбинационного вычитателя

Пусть имеется два числа:

$A=a_1a_2 \ldots a_{i-1}a_i a_{i+1} \ldots a_n$,

$B=b_1b_2 \ldots b_{i-1}b_i b_{i+1} \ldots b_n$.

В зависимости от значений аргументов a_i, b_i, z_i (заем из i-го разряда) формируется значение булевых функций P_i и $З_i$. Введем следующие обозначения.

$a_i \Rightarrow x$ $\quad P_i \Rightarrow P$, где P_i – значение разности в разряде i

$b_i \Rightarrow y$ $\quad З_i \Rightarrow З$ $\quad З_i$ – значение заема в разряд i

$z_i \Rightarrow z$ - заем из i-го разряда

Таблица истинности, соответствующая устройству, выполняющему операцию вычитания, имеет следующий вид (табл. 25).

Таблица 25

x	y	z	P	З
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$З = \bar{x}\,\bar{y}z + \bar{x}\,y\,\bar{z} + \bar{x}\,y\,z + x\,y\,z =$$

(Выполним склеивания 1 и 3, 2 и 3, 3 и 4 наборов)

$$= \bar{x}\,z + \bar{x}\,y + y\,z = \bar{x}(z + y) + y\,z.$$

$$П = yz + x\,z + x\,y = x\,(y + z) + yz.$$

$$P = \bar{x}\,\bar{y}z + \bar{x}\,y\,\bar{z} + x\,\bar{y}\bar{z} + x\,y\,z = C.$$

Как видно, функции разности P и суммы C совпадают (функция C была получена ранее).

Объединенная схема одноразрядного комбинационного сумматора-вычитателя

Для системы булевых функций C, P, П и З, полученных выше, может быть построена логическая схема (рис. 34) на элементах И, ИЛИ и НЕ.

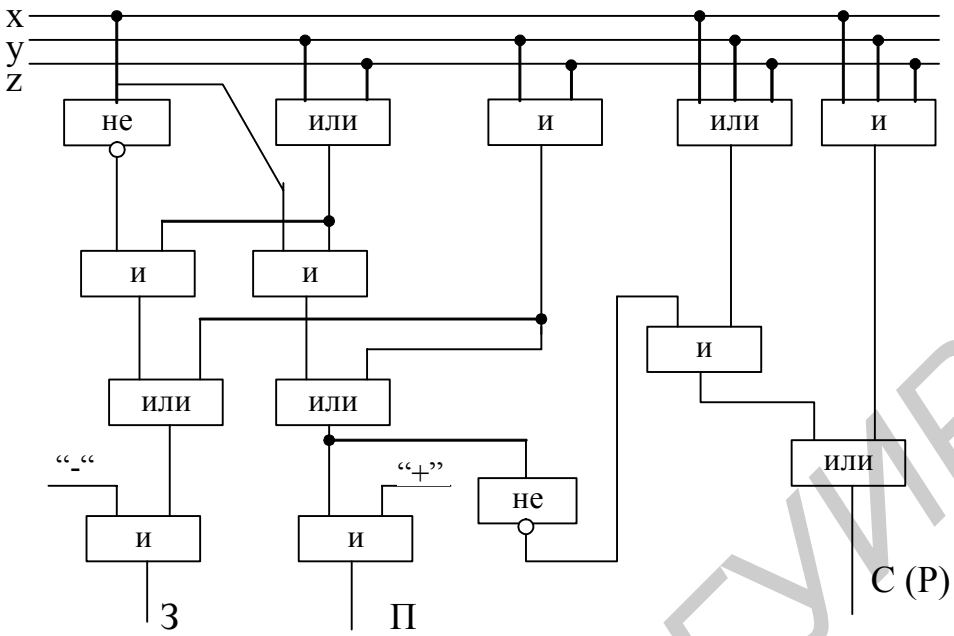


Рис.34. Схема сумматора-вычитателя

Триггер со счетным входом как полный одноразрядный сумматор

Триггером называется устройство, имеющее два устойчивых состояния и способное под действием входного сигнала скачком переходить из одного устойчивого состояния в другое. Триггер – это простейший цифровой автомат с памятью и способностью хранить 1 бит информации.

Триггер со счетным входом (Т-триггер) может быть рассмотрен как полный сумматор, работающий в три такта. В основе работы этого устройства лежит операция ”сумма по модулю 2”.

Первый такт.

1) $\tau(t-1)=0$ – триггер находится в исходном состоянии.

2) T=x первое слагаемое подается на вход триггера

$$\tau(t) = \overline{T}\,\tau(t-1) + T\,\overline{\tau}(t-1) = \overline{x}\cdot 0 + x\cdot 1 = x,$$

следовательно, после первого такта содержимое триггера будет соответствовать его входному сигналу.

Второй такт. Во втором такте на вход триггера подается второе слагаемое y.

$$\tau(t+1) = \overline{T}\,\tau(t) + T\,\overline{\tau}(t) = \overline{y}\cdot x + y\cdot\overline{x} = x \oplus y,$$

на выходе триггера формируется сумма по модулю 2 слагаемых x и y.

Третий такт. В третьем такте на вход триггера поступает значение, соответствующее третьему слагаемому – z.

$$\tau(t+2) = \overline{T}\,\tau(t+1) + T\,\overline{\tau}(t+1) = \overline{y}\cdot x + y\cdot\overline{x} = \overline{z}(x \oplus y) + z(\overline{x \oplus y}) =$$

$$= \overline{x}yz + x\overline{y}\,\overline{z} + xyz + \overline{x}\,\overline{y}z.$$

Из полученной функции видно, что на выходе Т-триггера получена полная сумма трех слагаемых и, следовательно, триггер со счетным входом является полным сумматором, работающим в три такта.

Введение в теорию конечных автоматов

Основные понятия теории автоматов

Все рассмотренные выше устройства относятся к классу комбинационных схем, то есть дискретных устройств без памяти. Наряду с ними в цифровой технике широкое распространение получили последовательностные автоматы, или, иначе, комбинационные схемы, объединенные с элементами памяти.

Под термином *автомат* можно понимать некоторое реально существующее устройство, функционирующее на основании как сигналов о состоянии внешней среды, так и внутренних сигналов о состоянии самого автомата. В этом плане ЭВМ может быть рассмотрена как цифровой автомат. Под цифровым автоматом понимается устройство, предназначенное для преобразования цифровой информации. С другой стороны, под термином *автомат* можно понимать математическую модель некоторого устройства. Общая теория автоматов подразделяется на две части: *абстрактную* и *структурную* теорию автоматов. Различие между ними состоит в том, что абстрактная теория абстрагируется от структуры как самого автомата, так и входных и выходных сигналов. В абстрактной теории анализируются переходы автомата под воздействием абстрактных входных слов и формируемые на этих переходах абстрактные выходные слова.

В структурной теории рассматривается прежде всего структура как самого автомата, так и его входных и выходных сигналов, способы построения автоматов из элементарных автоматов, способы кодирования входных и выходных сигналов, состояний автомата.

В соответствии с этим принято различать две модели автоматов: структурную и абстрактную. Абстрактная модель применяется при теоретическом рассмотрении автоматов. Структурная модель служит для построения схемы автомата из логических элементов и триггеров и предназначена для выполнения функции управления.

Абстрактный автомат – это математическая модель цифрового автомата, задаваемая шестикомпонентным вектором $S=(A,Z,W,\delta,\lambda,a_1)$, где $A=\{a_a,\ldots,a_m\}$ – множество внутренних состояний абстрактного автомата; $Z=[z_1,\ldots,z_k\}$ и $W=\{w_1,\ldots,w_l\}$ – соответственно множества входных и выходных абстрактных слов; δ - функция переходов; λ - функция выходов; a_1 – начальное состояние автомата. Абстрактный автомат может быть представлен как устройство с одним входом и одним выходом (рис. 35), на которые подаются абстрактные входные слова и формируются абстрактные выходные слова.

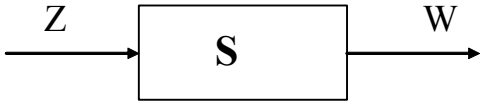


Рис.35. Абстрактный автомат

Понятие *состояние автомата* используется для описания систем, выходы которых зависят не только от входных сигналов, но и от предыстории, то есть информации о том, что происходило с автоматом в предыдущий интервал времени. Состояние автомата позволяет устранить время как явную переменную и выразить выходные сигналы как функцию состояний и входных сигналов.

По виду функции выходов все множество автоматов можно подразделить на два класса: автоматы Мили и автоматы Мура.

Автоматами Мура, или автоматами первого типа, называют автоматы, для которых выходной символ w(t) не зависит явно от входного символа z(t), а определяется только состоянием автомата в момент времени t. Закон функционирования автомата Мура может быть описан системой уравнений:

$$a(t+1) = \delta(a(t), z(t));$$

$$w(t) = \lambda(a(t)).$$

К автоматам второго типа, или *автоматам Мили*, относятся автоматы, поведение которых может быть описано системой уравнений:

$$a(t+1) = \delta(a(t), z(t));$$

$$w(t) = \lambda(a(t), z(t)).$$

Следовательно, в отличие от автомата Мура для автомата Мили выходной символ w(t) зависит не только от текущего состояния, но и от входного символа.

Между моделями автоматов Мили и Мура существует соответствие, позволяющее преобразовать закон функционирования одного из них в другой.

Совмещенная модель автомата (С-автомат). Абстрактный С-автомат – математическая модель дискретного устройства, определяемого вектором $S=(A,Z,W,U,\delta,\lambda_1,\lambda_2,a_1)$, где A, Z, δ и a_1 определены выше, а $W=\{w_1,\ldots,w_l\}$ и $U=\{u_1,\ldots,u_l\}$ – выходной абстрактный алфавит автомата Мили и Мура соответственно, λ_1 и λ_2 - функции выходов. Абстрактный С-автомат может быть представлен как устройство с одним входом, на который поступают слова из входного алфавита Z, и двумя выходами (рис. 36), на которых формируются абстрактные выходные слова из выходных алфавитов W и U.

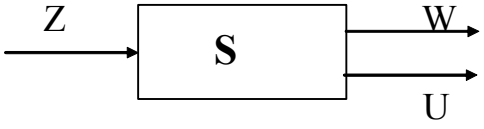


Рис. 36. Абстрактный С-автомат

Отличие С-автомата от моделей автоматов Мили и Мура заключается в том, что он одновременно реализует две функции выходов λ_1 и λ_2, каждая из которых характерна для одной из двух моделей.

$$\left\{\begin{array}{l} a(t+1)=\delta(a_m(t),z(t)); \\ w(t)=\lambda_1(a_m(t),z(t)); \\ u(t)=\lambda_2(a_m(t)). \end{array}\right\}$$

Автомат S называется *конечным*, если конечны множества A, Z и W, и *детерминированным*, если, находясь в некотором состоянии, он не может перейти более чем в одно состояние под действием одного и того же входного символа. Если в автомате выделено начальное состояние a_1, то он называется *инициальным*. Состояние a_s называется *устойчивым,* если для любого $z_k \in Z$, такого что $a_s=\delta(a_m,z_k)$, $a_s=\delta(a_s,z_k)$, то есть если автомат из состояния a_m в перешел в состояние a_s под действием входного слова z_k , то выйти из этого состояния он может только под действием другого входного слова. Автомат S является *асинхронным,* если каждое его состояние устойчиво, иначе *синхронным*. Автомат называется *полностью определенным*, если область определения функции δ совпадает с множеством пар (a_m,z_k), а функции λ для автомата Мили − с множеством пар (a_m,z_k), Мура − с множеством a_m. У *частичного* автомата функции δ и λ определены не для всех пар.

Способы задания автоматов

Закон функционирования автоматов может быть задан в виде систем уравнений, таблиц, матриц и графов. Под законом функционирования понимается совокупность правил, описывающих переходы автомата в новое состояние и формирование выходных символов в соответствии с последовательностью входных символов.

В зависимости от типа автомата при табличном задании закона функционирования автомата используются либо таблицы переходов и выходов (автомат Мили), либо совмещенная таблица переходов и выходов (автомат Мура). С помощью табл. 26 и 27 (таблицы переходов и таблицы выходов соответственно) задан закон функционирования абстрактного автомата Мили, для которого

$A=\{a_1,a_2,a_3,a_4\}$, $Z=\{z_1,z_2,z_3\}$, $W=\{w_1,w_2,w_3,w_4,w_5\}$.

Таблица 26

δ	a_1	a_2	a_3	a_4
z_1	a_2	a_2	-	a_4
z_2	a_4	-	a_1	a_3
z_3	a_3	a_3	a_4	-

$a(t+1)=\delta(a(t),z(t))$

Таблица 27

λ	a_1	a_2	a_3	a_4
z_1	-	w_2	-	w_3
z_2	w_2	-	w_4	w_5
z_3	w_3	w_1	w_3	-

$w(t)=\lambda(a(t),z(t))$

Строки таблиц отмечены входными символами (элементы множества Z),

а столбцы − состояниями (элементы множества A). Входные символы и состояния, которыми помечены строки и столбцы, относятся к моменту времени t. В табл. 26 (таблице переходов) на пересечении строки $z_i(t)$ и столбца $a_m(t)$ ставится состояние $a_s(t+1)=\delta(a_m(t),z_i(t))$. В табл. 27 (таблице выходов) на пересечении строки $z_i(t)$ и столбца $a_m(t)$ ставится выходной символ $w(t)=\lambda(a_m(t),z_i(t))$, соответствующий переходу из состояния a_m в состояние a_s. Таким образом, по таблицам переходов и выходов можно проследить последовательность работы автомата. Так, например, в начальный момент времени t=0 автомат, находясь в состоянии a_1 (первый столбец), под действием входного символа z_1 может перейти в состояние a_2, при этом выходной символ не формируется; под действием входного символа z_2 – в состояние a_4 с формированием выходного символа w_2; под действием символа z_3 − в состояние a_3 с формированием выходного символа w_3. Далее если на вход автомата, установленного в исходное состояние $a_m \subseteq A$, в моменты времени t=1,2,…,n подается некоторая последовательность букв входного алфавита (входных символов) $z_i \subseteq Z$, то на выходе автомата будут последовательно формироваться буквы выходного алфавита (выходные символы) $w_j \subseteq W$, при этом автомат будет переключаться в состояния $a_s \subseteq A$. Следовательно, с помощью таблиц переходов и выходов можно получить выходную реакцию автомата на любое входное слово.

Как отмечалось выше, для автомата Мура выходной символ не зависит от входного, а определяется только текущим состоянием автомата. Это позволяет для автомата Мура объединить обе таблицы (переходов и выходов) в одну *совмещенную таблицу*. В совмещенной таблице переходов и выходов каждый столбец отмечается состоянием $a_m \subseteq A$ и выходным символом $w(t)=\lambda(a(t))$, соответствующим этому состоянию.

Другим, более наглядным способом описания закона функционирования автомата является представление его в виде графа. Граф автомата – ориентированный граф, вершины которого соответствуют состояниям, а дуги − переходам между ними. Дуга, направленная из вершины a_m в вершину a_s, соответствует переходу из состояния a_m в a_s. В начале дуги записывается входной символ z_i, влияющий на переход $a_s=\delta(a_m,z_i)$, а символ w_j записывается в конце дуги (автомат Мили) или рядом с вершиной (автомат Мура). На рис. 37 приведен граф автомата Мили, соответствующий закону функционирования, описанному выше (см. табл. 26 и 27).

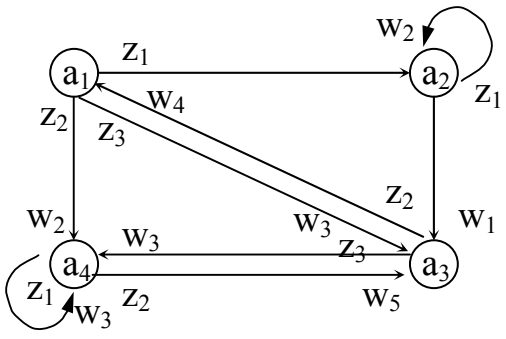


Рис.37. Граф автомата Мили

Структурный автомат

В отличие от абстрактного автомата структурный автомат имеет L входов и N выходов. На входы структурного автомата поступают наборы входных двоичных переменных из множества $X=\{x_1,x_2,\ldots,x_L\}$, а на выходах формируются выходные двоичные сигналы из множества $Y=\{y_1,y_2,\ldots,y_N\}$. Структурная модель автомата представляет собой две взаимосвязанные части: комбинационную схему и память. Комбинационная часть автомата кроме сигналов из множества Y формирует также двоичные сигналы, подаваемые на входы элементов памяти $D=\{d_1,d_2,\ldots,d_r\}$. Эти сигналы называются *функциями возбуждения элементов памяти* и представляют собой код состояния перехода. Сигналы, формируемые на выходах элементов памяти $T=\{\tau_1,\tau_2,\ldots,\tau_r\}$, подаются на входы комбинационной схемы наряду с входными переменными и называются *переменными обратной связи*. Переменные обратной связи являются кодом текущего состояния автомата. Структурная схема автомата изображена на рис. 38.

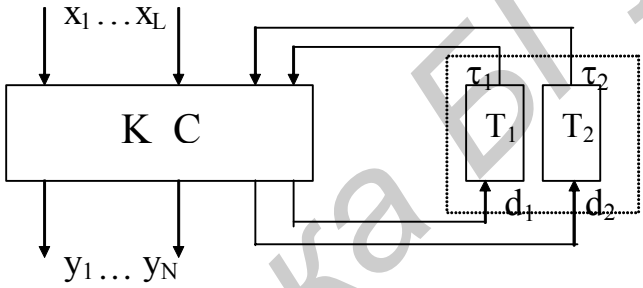


Рис. 38. Структурный автомат

Любому переходу в абстрактном автомате из состояния a_m в состояние a_s под действием входного слова z_i с формированием выходного слова w_j соответствует переход в стуктурном автомате из состояния a_m с кодом $\tau_1,\ldots,\tau_r$ в состояние a_s с кодом $d_1,\ldots,d_r$ под действием набора входных сигналов $x_1,\ldots,x_L$ с формированием выходного набора $y_1,\ldots,y_N$.

Память автомата

В качестве элементов памяти автомата используются простейшие схемы, предназначенные для приема, хранения и передачи одного бита информации − триггеры. Триггер имеет один или более входов и два выхода (прямой и инверсный). Выходные сигналы триггера зависят только от его состояния и изменяются только при смене состояния триггера. Таким образом, триггеры являются элементарными автоматами Мура (элементарными, так как они имеют только два устойчивых состояния). В основе любого триггера находится регенеративное кольцо из двух инверторов.

Триггеры можно классифицировать по следующим признакам:

1) по способу записи информации: несинхронизируемые (асинхронные) и синхронизируемые (синхронные) триггеры. У асинхронных триггеров запись

информации происходит под действием информационных сигналов, у синхронных кроме информационных на вход должны быть поданы разрешающие сигналы;

2) по способу синхронизации: синхронные триггеры со статическим управлением записью, синхронные двухступенчатые триггеры, синхронные триггеры с динамическим управлением записью;

3) по способу организации логических связей: триггеры с раздельной установкой состояния (RS-триггеры), триггеры со счетным входом (Т-триггеры), универсальные триггеры с раздельной установкой состояний (JK-триггеры), триггеры с приемом информации по одному входу (D-триггеры), комбинированные триггеры (RST-, JKRS-, DRS-триггеры и т.д.), триггеры со сложной входной логикой.

Приняты следующие изображения входов триггеров:

S − раздельный вход установки триггера в единичное состояние по прямому выходу;

R − раздельный вход сброса триггера в нулевое состояние по прямому выходу;

J и K − назначение аналогично входам S и R;

D − информационный вход. Используется для приема информации, записываемой в триггер;

T − счетный вход;

C − вход синхронизации.

D-триггер. Принцип работы синхронного D-триггера основан на том, что сигнал на выходе после переключения равен сигналу на входе D до переключения. На рис. 39 приведена схема одноступенчатого D-триггера на элементах И-НЕ и его условное изображение.

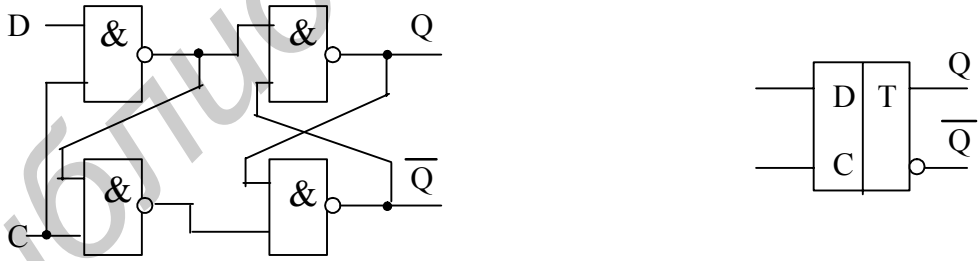


Рис. 39. D-триггер

Таблица 28

τ(t)	D	
	0	1
0	0	0
1	1	1

В табл. 28 приведена информация о работе D-триггера. Переключение состояний выполняется по формуле $\tau(t+1)= \tau(t)\overline{C}\ V\ DC$.

Т-триггер. Принцип работы Т-триггера основан

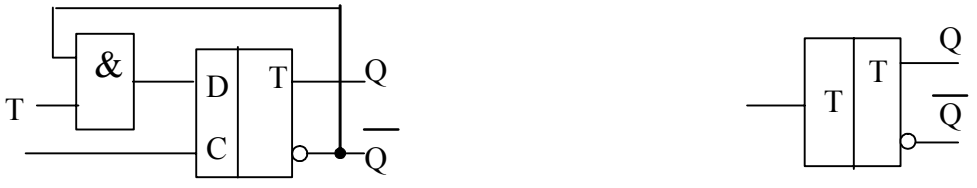


Рис. 40. Т-триггер на основе D-триггера

на том, что единичный сигнал на входе изменяет содержимое триггера на противоположное. На рис.40 приведена схема Т-триггера на элементах И-НЕ и его условное изображение.

В табл.29 приведена информация о работе Т-триггера. Переключение состояний выполняется по формуле $\tau(t+1)= \tau(t) \oplus T$.

Таблица 29

$\tau(t)$	T	
	0	1
0	0	1
1	1	0

RS-триггеры. Асинхронные RS-триггеры являются простейшими триггерами. Такие триггеры строятся на логических элементах: 2ИЛИ-НЕ – триггер с прямыми входами (рис. 41) или 2И-НЕ – триггер с инверсными входами. Выход каждого из логических элементов подключен к одному из входов другого элемента, что обеспечивает нахождение триггера в одном из двух устойчивых состояний.

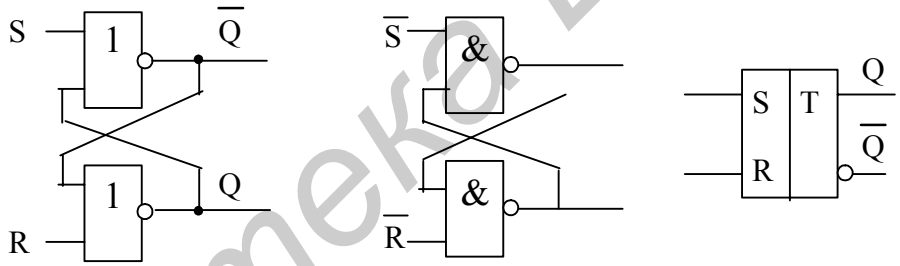


Рис. 41. RS-триггер

Табл.30 определяет переходы RS-триггера по формуле $\tau(t+1)=\tau(t)\overline{R}VS$.

Таблица 30

$\tau(t)$	R S			
	00	01	10	11
0	0	1	0	x
1	1	1	0	x

Возможны следующие режимы работы RS-триггера:
S=0, R=0 – режим хранения информации (значение триггера не изменяется);
S=0, R=1 – режим сброса (триггер всегда устанавливается в 0);
S=1, R=0 – режим записи логической единицы (триггер устанавливается в 1);
S=1, R=1 – запрещенная комбинация (значение триггера не неопределенное).

JK-триггеры. Асинхронный JK-триггер строится на базе RS-триггера. JK-триггер имеет два информационных входа. Простейший JK-триггер можно получить из RS-триггера, если ввести дополнительные обратные связи с выхо-

дов триггера на входы, которые позволяют устранить неопределенность в таблице состояний. Логическая схема и условное обозначение JK-триггера приведены на рис. 42.

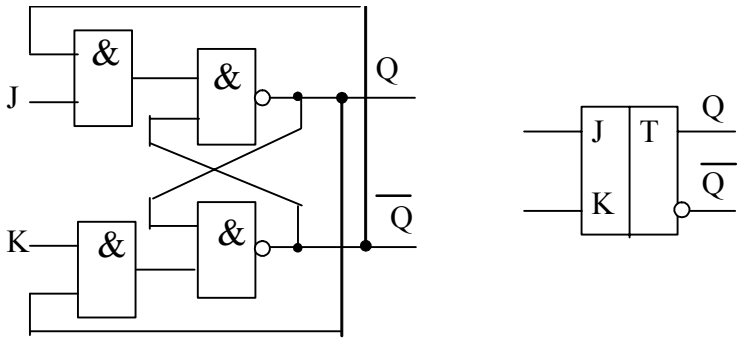


Рис. 42. JK-триггер

Табл. 31 определяет переходы JK-триггера согласно логической формуле $\tau(t+1)= \overline{\tau(t)}J \vee \tau(t) \overline{K}$.

Таблица 31

$\tau(t)$	J K			
	00	01	10	11
0	0	0	1	1
1	1	0	1	0

Возможны следующие режимы работы RS-триггера:
J=0, K=0 – режим хранения информации (значение триггера не изменяется);
J=0, K=1 – режим сброса (триггер всегда устанавливается в 0);
J=1, K=0 – режим записи логической единицы (триггер устанавливается в 1);
J=1, K=1 – режим инверсии содержимого триггера.

JK-триггер является универсальным триггером. Универсальность его состоит в том, что он может выполнять функции RS-, T- и D-триггеров. Для получения D-триггера K вход соединяется со входом J через инвертор. T-триггер получается из JK-триггера путем объединения входов J и K в один, называемый T-входом. Если JK-триггер предварительно установлен в 0 и на вход не подается комбинация 11, то он работает как RS-триггер.

Канонический метод структурного синтеза автоматов

Синтез цифровых устройств выполняется в два этапа:

- этап абстрактного синтеза;
- этап структурного синтеза.

Для перехода от абстрактного автомата к структурному требуется:

1) поставить в соответствие каждой букве входного алфавита $Z=x\{z_1,\ldots,z_k\}$ совокупность двоичных сигналов из множества $X=\{x_1,x_2,\ldots,x_L\}$, то *есть закодировать входные символы* абстрактного автомата. Значение L, определяющее количество двоичных переменных для кодирования абстрактных входных сигналов, вычисляется следующим образом: $L=]\log_2|Z|[$, где $|Z|$ - мощность множества Z (число различных элементов множества Z),]n[означает ближайшее целое число, большее либо равное n;

2) поставить в соответствие каждому символу выходного алфавита $W=\{w_1,\ldots,w_l\}$ совокупность двоичных выходных сигналов из множества $Y=\{y_1,y_2,\ldots,y_N\}$, то *есть закодировать выходные символы* абстрактного автомата. Значение N вычисляется следующим образом: $N=]\log_2|W|[$;

3) поставить в соответствие каждому состоянию абстрактного автомата $A=\{a_a,\ldots,a_m\}$ совокупность состояний элементов памяти $T=\{\tau_1,\tau_2,\ldots,\tau_r\}$, то есть *закодировать состояния* абстрактного автомата. Количество элементов памяти определяется условием $r=]\log_2|A|[$;

4) составить систему уравнений для функций $y_1,y_2,\ldots,y_N,d_1,d_2,\ldots,d_r$, в соответствии с которой будет построена комбинационная часть структурной схемы автомата.

Полученная таким образом система логических функций называется *канонической*.

Рассмотрим пример структурного синтеза автомата Мили, блок памяти которого будет построен на Т-триггерах. Исходные данные для выполнения синтеза структурной схемы заданы таблично. Это таблица переходов (табл.32)и таблица выходов (табл.33). Описание работы Т-триггера приведено в табл. 29.

Таблица 32

δ	a_1	a_2	a_3	a_4
z_1	a_2	a_2	-	a_4
z_2	a_4	-	a_2	a_2
z_3	a_3	a_3	a_4	a_3
z_4	-	a_4	-	a_1

$a(t+1)=\delta(a(t),z(t))$

Таблица 33

λ	a_1	a_2	a_3	a_4
z_1	w_1	w_2	-	w_5
z_2	w_2	-	w_4	w_6
z_3	w_3	w_1	w_3	w_5
z_4	-	w_2	-	w_1

$w(t)=\lambda(a(t),z(t))$

Определяем вначале общее количество входов, выходов и элементов памяти автомата:

$L=]\log_2|Z|[\ =\ ]\log_2 4[=2;$

$N=]\log_2|W|[=\ ]\log_2 6]=3;$

$r=]\log_2|A|[\ =\ ]\log_2 4[=2.$

Структурная схема автомата изображена на рис. 43.

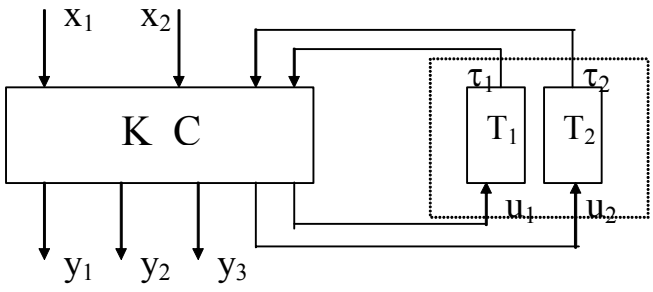


Рис. 43. Структурная схема автомата

На основании полученных значений построим таблицы и выполним кодирование входных, выходных символов и состояний исходного автомата

дирование входных, выходных символов и состояний исходного автомата (табл. 34-36).

Таблица 34

Z\X	x_1	x_2
z_1	0	0
z_2	0	1
z_3	1	0
z_4	1	1

Таблица 35

W\Y	y_1	y_2	y_3
w_1	0	0	0
w_2	0	0	1
w_3	0	1	0
w_4	0	1	1
w_5	1	0	0
w6	1	0	1

Таблица 36

A\T	τ_1	τ_2
a_1	0	0
a_2	0	1
a_3	1	0
a_4	1	1

По результатам кодирования строим таблицы переходов и выходов структурного автомата (табл. 37 и 38 соответственно). Они получаются путем занесения соответствующих значений из табл. 34-36 в исходные таблицы (см. табл. 32, 33).

Таблица 37

$\tau_1\tau_2$ / x_1x_2	00 a_1	01 a_2	10 a_3	11 a_4
00 z_1	01	01	-	11
01 z_2	11	-	01	01
10 z_3	10	10	11	10
11 z_4	-	11	-	00

Таблица 38

$\tau_1\tau_2$ / x_1x_2	00 a_1	01 a_2	10 a_3	11 a_4
00 z_1	000 w_1	001 w_2	-	100 w_5
01 z_2	001 w_2	-	011 w_4	101 w_6
10 z_3	010 w_3	000 w_1	010 w_3	100 w_5
11 z_4	-	001 w_2	-	000 w_1

$y_1y_2y_3$ $y_1y_2y_3$

На основании табл. 37, используя табл. 29, построим табл. 39 − таблицу функций возбуждения элементов памяти.

Таблица 39

$\tau_1\tau_2$ / x_1x_2	00 a_1	01 a_2	10 a_3	11 a_4
00 z_1	01	00	-	00
01 z_2	11	-	11	10
10 z_3	10	11	01	01
11 z_4	11	10	11	11

Если i-й триггер на некотором переходе переключается из состояния 0 в состояние 1 или наоборот, то u_i=1, в противном случае (то есть если i-й триггер не переключается) u_i=0. Например, рассмотрим переход из состояния 10 в состояние 11 (см. табл. 37, 4-й столбец, 3-я строка). Первый триггер (установленный в 1) не меняет своего

$$u_1 = \bar{\tau}_1\tau_2x_1\bar{x}_2 + \tau_1\tau_2\bar{x}_1\bar{x}_2 + \tau_1\bar{\tau}_2x_1\bar{x}_2 + \tau_1\tau_2x_1\bar{x}_2 + \tau_1\tau_2\bar{x}_1\bar{x}_2 + \tau_1\tau_2x_1\bar{x}_2$$

$$u_1 = \tau_1\tau_2x_1x_2 + \tau_1\tau_2x_1x_2 + \tau_1\tau_2x_1x_2 + \tau_1\tau_2x_1x_2 + \tau_1\tau_2x_1x_2 + \tau_1\tau_2x_1x_2$$

u_1u_2 $\bar{u}_1u_2$

$$y_1 = \tau_1\tau_2x_1x_2 + \tau_1\tau_2\bar{x}_1\bar{x}_2 + \tau_1\tau_2x_1\bar{x}_2$$

$$y_2 = \bar{\tau}_1\tau_2x_1\bar{x}_2 + \tau_1\bar{\tau}_2\bar{x}_1x_2 + \tau_1\bar{\tau}_2x_1\bar{x}_2$$

$$y_3 = \bar{\tau}_1\tau_2x_1x_2 + \tau_1\bar{\tau}_2x_1x_2 + \tau_1\bar{\tau}_2x_1x_2 + \tau_1\bar{\tau}_2x_1x_2 + \tau_1\tau_2\bar{x}_1x_2$$

значения, поэтому функция возбуждения элемента памяти для него u_1=0. Второй триггер изменяет свое значение с 0 на 1, следовательно, u_2=1. Остальные клетки табл. 39 заполняются аналогично. На основании табл. 38 и 39 запишем систему логических функций для построения комбинационной схемы автомата:

Для упрощения комбинационной схемы выполним минимизацию каждой из логических функций. Для этого используем метод минимизирующих карт Карно. На рис. 44 изображены пять карт Карно для минимизации каждой из пяти логических функций.

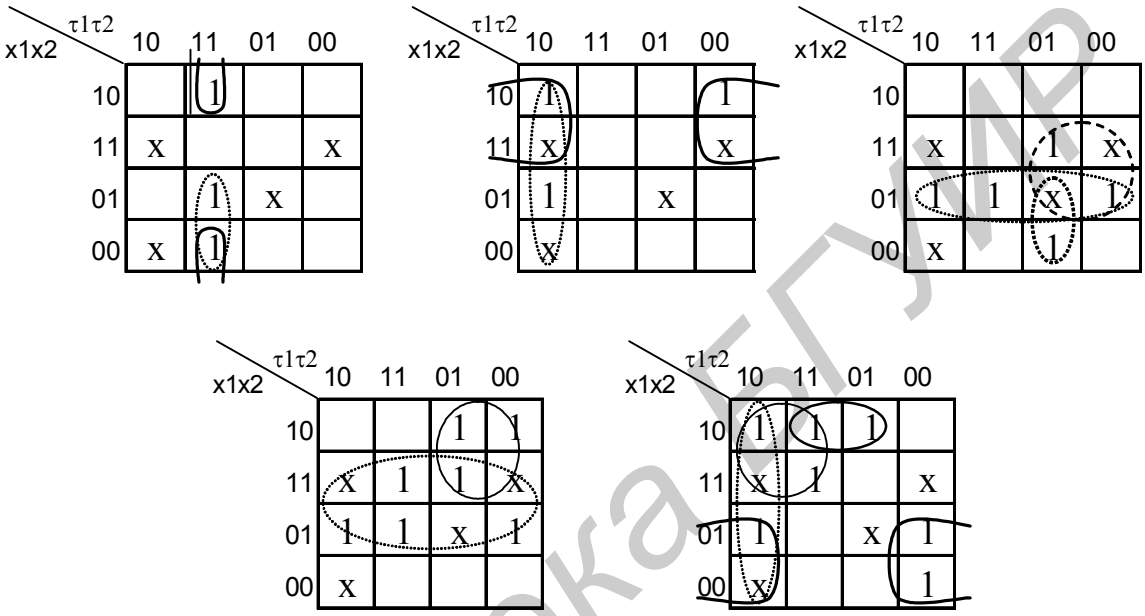


Рис. 44. Карты Карно для минимизации логических функций

По результатам минимизации запишем систему минимальных функций:

$$u_1 = x_2 + \bar{\tau}_1 x_1$$

$$u_2 = \bar{\tau}_2 \bar{x}_1 + \tau_1 x_1 + \tau_2 x_1 \bar{x}_2$$

$$y_1 = \tau_1 \tau_2 \bar{x}_1 + \tau_1 \tau_2 \bar{x}_2$$

$$y_2 = \tau_1 \bar{\tau}_2 + \bar{\tau}_2 x_1$$

$$y_3 = \bar{x}_1 x_2 + \bar{\tau}_1 x_2 + \bar{\tau}_1 \tau_2 \bar{x}_2$$

На рис. 45 изображена логическая схема, построенная на основании полученной системы булевых функций. При построении схемы использованы элементы ”И” и ”ИЛИ”.

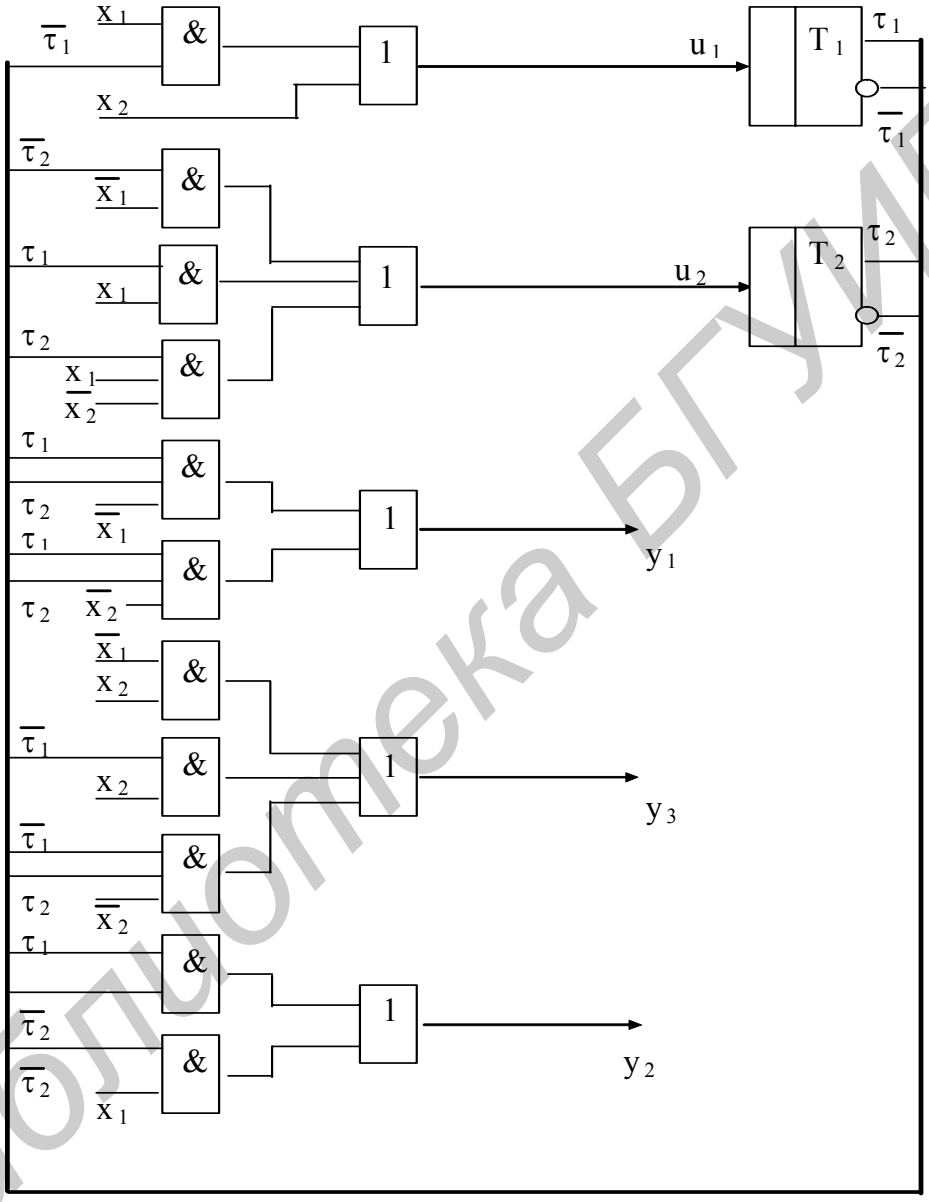


Рис. 45. Логическая схема автомата

Принцип микропрограммного управления

Общая структура устройства, выполняющего арифметические операции, например АЛУ, имеет структуру, представленную на рис. 46.

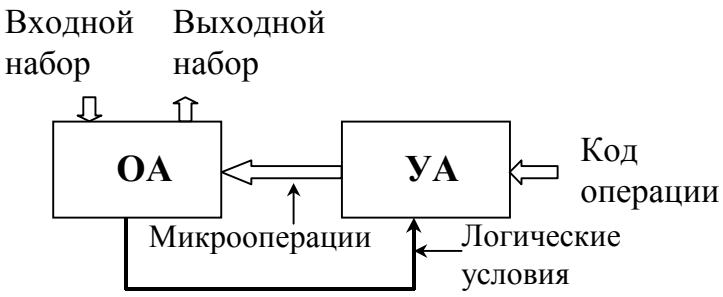


Рис. 46. Структура АЛУ

Элементарное действие, выполняемое за один такт автоматного времени (за один такт работы автомата), называется *микрооперацией*. Условия, влияющие на порядок выполнения автоматом микроопераций, называются *логическими условиями*. Проверка значений логических условий в каждом такте работы автомата позволяет определить группу выполняемых микроопераций. Совокупность операций, выполняемых за один такт, называется *микрокомандой*.

Принцип, согласно которому алгоритм работы некоторого устройства описывается в перечисленных выше терминах, называется принципом микропрограммного управления. Конечный автомат, алгоритм работы которого может быть описан на основе принципа микропрограммного управления, называется микропрограммным автоматом (МПА).

Граф-схема алгоритма

Для записи микропрограмм в компактной форме используются специализированные языки. Одним из способов графического представления микропрограммы является граф-схема алгоритма (ГСА). ГСА представляет собой ориентированный связный граф. ГСА может содержать вершины четырех типов: начальную, операторную, условную и конечную (рис. 47).

ГСА должна удовлетворять следующим основным требованиям:

- в ГСА имеются одна начальная и одна конечная вершины;
- входы и выходы вершин соединяются с помощью дуг;
- каждая вершина должна лежать на одном из путей следования из начальной вершины в конечную;
- один из выходов условной вершины может соединяться с ее входом;
- в каждой условной вершине записывается одно из логических условий xi (допускается запись одинаковых условий в различных вершинах);
- в каждой операторной вершине записывается микрокоманда (допускается пустая микрокроманда и повтор микрокоманды в различных вершинах).

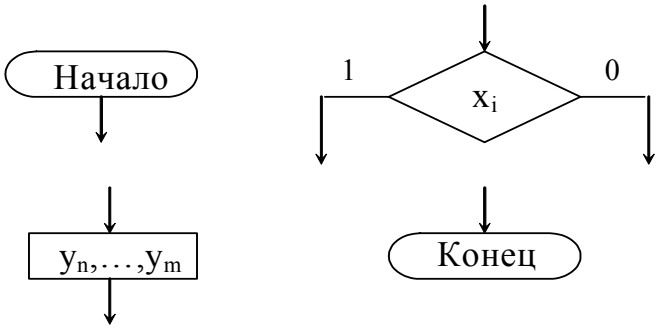


Рис. 47. Типы вершин ГСА

Пример синтеза МПА по ГСА

МПА может быть синтезирован по ГСА, описывающей микропрограмму работы проектируемого дискретного устройства.

Алгоритм синтеза МПА по ГСА состоит в следующем:

- разметка ГСА метками Мили (Мура);
- кодирование внутренних состояний;
- построение структурной таблицы по отмеченной ГСА;
- построение таблиц истинности или системы булевых функций;
- построение логической схемы автомата.

Как отмечалось выше, известны два класса автоматов: Мили и Мура. В качестве примера рассмотрим синтез микропрограммного автомата, управляющего операционным автоматом для выполнения операции деления чисел в дополнительных кодах. ГСА, соответствующая алгоритму деления, изображена на рис. 48. Описание алгоритма деления чисел в дополнительном коде приведено выше в соответствующем разделе.

После пробного вычитания Зн См может быть равен 0, это означает, что Дм больше Дт (произошло переполнение). В этот момент счетчик тактов Ст равен 0, деление прекращается (переход в конец по стрелке 2). В последующих тактах Зн См может быть равен нулю. Это означает, что остаток A_i > Дт, но Ст уже содержит ненулевое значение, и алгоритм выполняется по стрелке 4. Если Зн См равен 1, то остаток отрицательный и деление будет выполняться в направлении стрелки 3.

Синтез МПА Мили по ГСА

Для получения графа автомата Мили исходная ГСА отмечается метками Мили. Каждой метке на ГСА ставится во взаимно однозначное соответствие состояние автомата. Алгоритм отметки ГСА метками Мили состоит в следующем:

- выход начальной и вход конечной вершин отмечаются меткой a1;
- входы всех вершин, следующих за операторными отмечаются метками a2,…,am;
- одной меткой может быть отмечен только один вход.

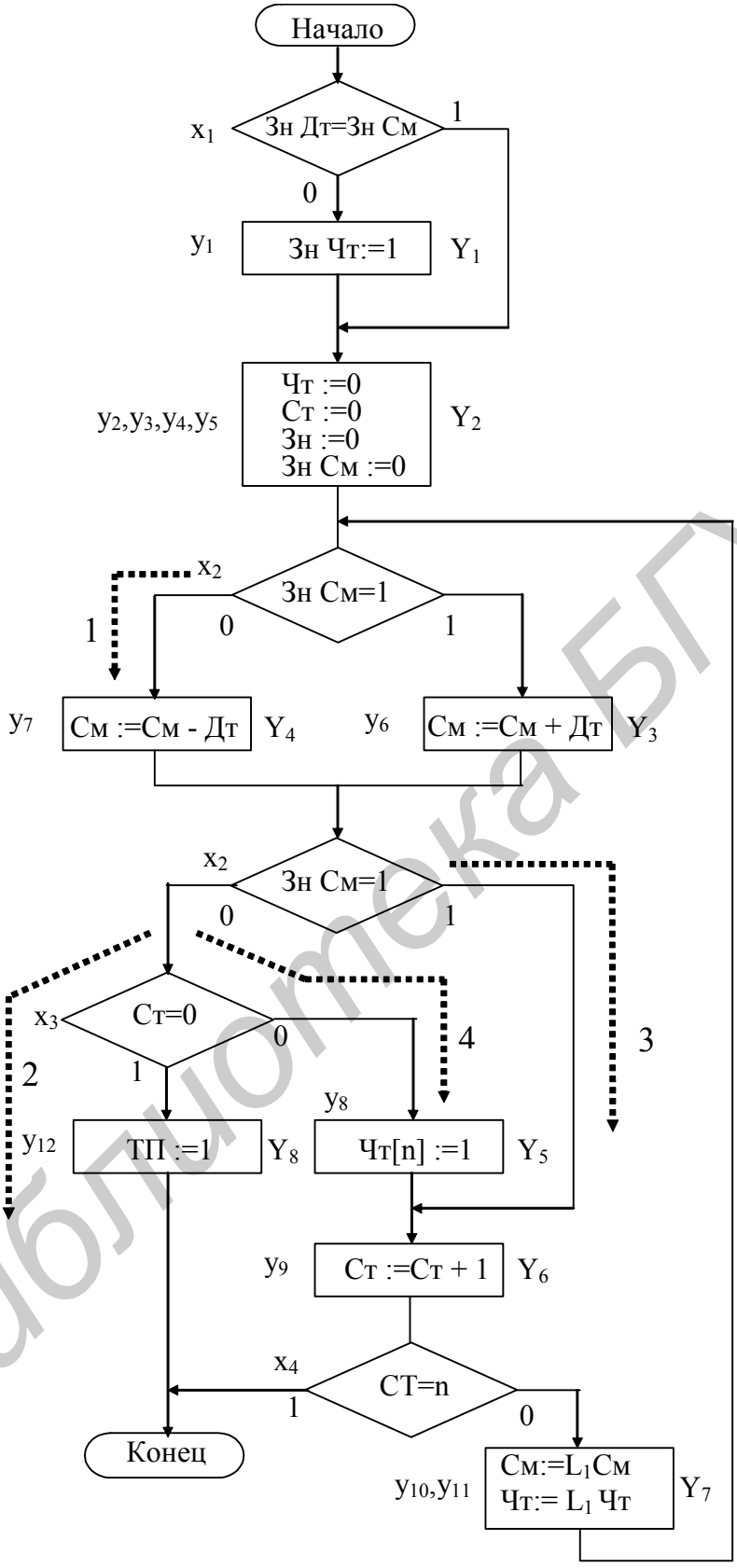


Рис. 48. ГСА выполнения операции деления чисел

На рис. 49 приведена ГСА, отмеченная метками Мили.

Кодирование состояний автомата может быть выполнено, как и ранее, если каждому состоянию поставить в соответствие двоичный эквивалент номера состояния. Для нахождения всевозможных переходов автомата на отмеченной ГСА отыскиваются все пути вида

$$a_m\, x_i,\ldots,x_j Y_k\, a_s\,, \quad \text{или} \quad a_m\, X(a_m a_s) Y_k\, a_s.$$

При достаточно большом числе состояний и переходов удобным является представление автомата структурной таблицей, содержащей всю необходимую для синтеза информацию. Структурная таблица может быть прямой или обратной. В *прямой* таблице (табл. 40) вначале записываются все переходы из состояния a_1, затем из состояния a_2 и т.д. В *обратной* таблице сначала записываются все переходы в состояние a_1, затем в a_2 и т.д..

Таблица 40

a_m	$K(a_m)$	a_s	$K(a_s)$	$X(a_m,a_s)$	$Y(a_m,a_s)$	$F(a_m,a_s)$
Исходн. состоян.	Код исх. состояния	Состоян. перехода	Код сост. перехода	Входной сигнал	Выходной сигнал	Функции возбуждения
a_1	000	a_2	001	$\overline{x_1}$	y_1	S_3
		a_3	010	x_1	$y_2y_3y_4y_5$	S_2
a_2	001	a_3	010	1	$y_2y_3y_4y_6$	S_2R_3
a_3	010	a_4	011	x_2	y_6	S_3
		a_4	011	$\overline{x_2}$	y_7	S_3
a_4	011	a_1	000	$\overline{x_2}x_3$	y_{12}	R_2R_3
		a_5	100	$\overline{x_2}\,\overline{x_3}$	y_8	$S_1R_2R_3$
		a_6	101	x_2	y_9	S_1R_2
a_5	100	a_6	101	1	y_9	S_3
a_6	101	a_1	000	x_4	—	R_1R_3
		a_3	010	$\overline{x_4}$	$y_{10}y_{11}$	$R_1S_2R_3$

Для реализации блока памяти МПА использованы RS-триггеры. В последнем столбце отмечены те функции возбуждения, которые приводят к изменению содержимого каждого из элементов памяти на соответствующем переходе. В таблице в столбце $F(a_m a_s)$ приведены функции переключения элементов памяти.

Для построения схемы, реализующей синтезируемый МПА, удобно результаты, приведенные в структурной таблице (см. табл. 40), представить в виде таблицы истинности (табл. 41).

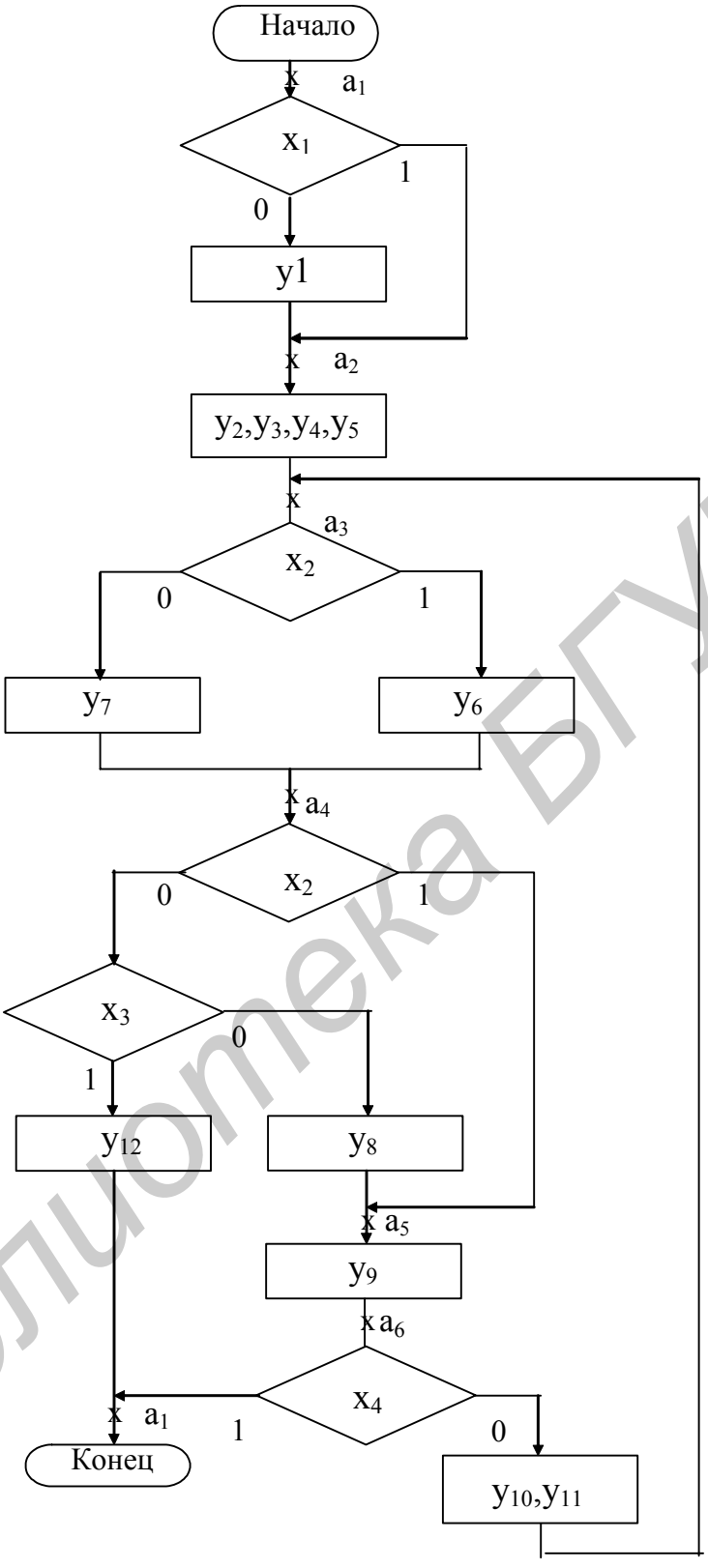


Рис. 49. ГСА, отмеченная метками Мили

Таблица 41

$x_1x_2x_3x_4\tau_1\tau_2\tau_3$	y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9 y_{10} y_{11} y_{12}	S_1 R_1 S_2 R_2 S_3 R_3
0 - - - 0 0 0	1 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 1 0
1 - - - 0 0 0	0 1 1 1 1 0 0 0 0 0 0 0	0 0 1 0 0 0
- - - - 0 0 1	0 1 1 1 0 1 0 0 0 0 0 0	0 0 1 0 0 1
- 1 - - 0 1 0	0 0 0 0 0 1 0 0 0 0 0 0	0 0 0 0 1 0
- 0 - - 0 1 0	0 0 0 0 0 0 1 0 0 0 0 0	0 0 0 0 1 0
- 0 1 - 0 1 1	0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 1 0 1
- 0 0 - 0 1 1	0 0 0 0 0 0 0 1 0 0 0 0	1 0 0 1 0 1
- 1 - - 0 1 1	0 0 0 0 0 0 0 0 1 0 0 0	1 0 0 1 0 0
- - - - 1 0 0	0 0 0 0 0 0 0 0 1 0 0 0	0 0 0 0 1 0
- - - 1 0 1 1	0 0 0 0 0 0 0 0 0 0 0 0	0 1 0 0 0 1
- - - 0 1 0 0	0 0 0 0 0 0 0 0 0 1 1 0	0 1 1 0 0 1

Для примера реализации логической схемы синтезируемого МПА рассмотрим реализацию функций y_1 y_2 и S_2 R_2 (рис. 50).

$y_1 = \bar{x}_1\bar{\tau}_1\bar{\tau}_2\bar{\tau}_3$

$y_2 = x_1\bar{\tau}_1\bar{\tau}_2\bar{\tau}_3 + \bar{\tau}_1\bar{\tau}_2\tau_3$

. . .

$S_2 = x_1\bar{\tau}_1\bar{\tau}_2\bar{\tau}_3 + \bar{\tau}_1\bar{\tau}_2\bar{\tau}_3 = \bar{\tau}_1\bar{\tau}_2\bar{\tau}_3$

$R_2 = \bar{x}_2x_3\bar{\tau}_1\tau_2\tau_3 + \bar{x}_2\bar{x}_3\bar{\tau}_1\tau_2\tau_3 + x_2\bar{\tau}_1\tau_2\tau_3 = \bar{\tau}_1\tau_2\tau_3$

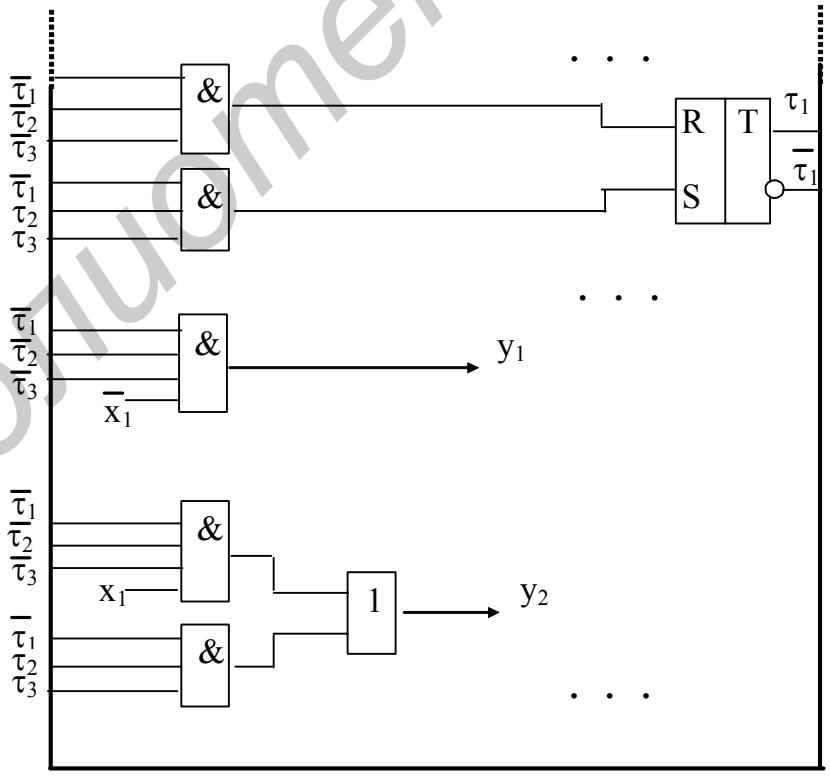


Рис. 50. Логическая схема автомата

Синхронизация автоматов

Нарушение функционирования автомата может быть вызвано явлениями, получившими название *гонки* и *риск сбоя*.

Гонки возникают из-за неодновременного срабатывания элементов памяти автомата вследствие разброса во времени переключения триггеров. Например, пусть под действием некоторого входного сигнала $X(a_m a_s)$ с кодом 00 автомат должен перейти из состояния a_m с кодом 101 в состояние a_s с кодом 110 (рис. 51). Если второй триггер изменит свое значение − переключится из 0 в 1 ранее, чем третий переключится из 1 в 0, то автомат перейдет в промежуточное состояние 111. Иначе, если третий триггер сработает ранее второго, − то в промежуточное состояние 100.

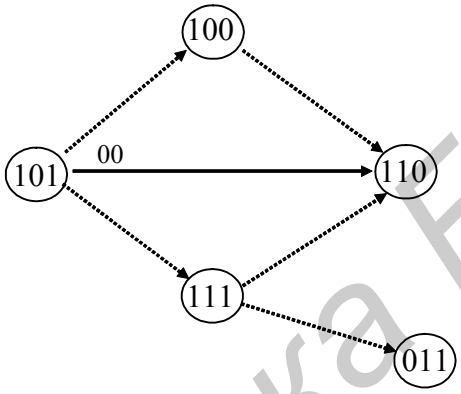


Рис. 51. Демонстрация явления состязаний

Таким образом, если на некотором переходе в автомате одновременно изменяют свое состояние несколько элементов памяти, то между ними возникает ”состязание”. Если из промежуточного состояния автомат в конечном счете, переходит в требуемое состояние a_s, то ”состязания” называются некритическими, если в ложное, например 011, то критическими или гонками.

Существует два основных подхода к устранению гонок: программный и аппаратный. Программный (алгоритмический) подход основан на соседнем кодировании состояний. При соседнем кодировании состояния автомата из множества $A=\{a_1,\ldots,a_m\}$ кодируются таким образом, что на любом переходе изменяет свое состояние не более чем один элемент памяти. Однако соседнее кодирование возможно выполнить не для всех автоматов.

Аппаратный подход основан на использовании двух ступеней памяти (рис. 52). Первая ступень памяти построена на триггерах $T_1',\ldots,T_r'$, вторая − на триггерах $T_1'',\ldots,T_r''$. Информация в триггеры первого уровня $T_1',\ldots,T_r'$ записывается по тактовому сигналу Ти, а в триггеры второго уровня $T_1'',\ldots,T_r''$ − по сигналу Ти, следующему непосредственно за Ти.

Если в течение первого полупериода (Ти) между триггерами первой ступени и возникают ”состязания”, то они все равно не изменяют состояния триггеров второй ступени, поскольку отсутствует синхросигнал Ти. Затем, с

приходом синхроимпульса $\overline{Т}$и, изменяют свое состояние триггеры второй ступени. Промежуточные коды, формируемые на их выходах, приводят к изменению (и, возможно, искажению) $\tau_1,\ldots,\tau_r$, а следовательно, и $D_1,\ldots,D_r$. Однако триггеры первой ступени не изменят своего состояния, поскольку отсутствует сигнал Ти. Таким образом, в итоге верный код состояния a_s с выходов памяти первой ступени переписывается в триггеры второго уровня, что соответствует переходу автомата в состояние a_s.

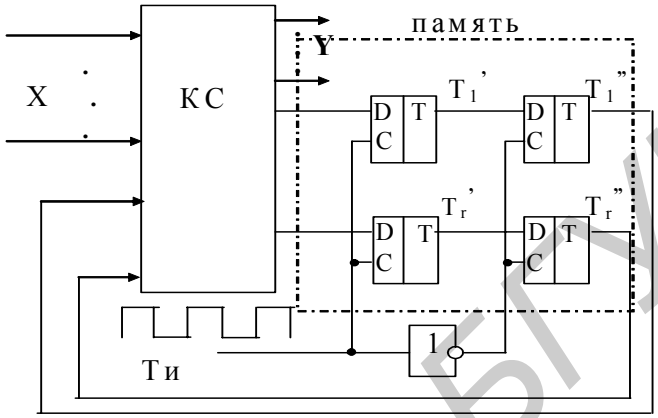


Рис. 52. Автомат с двухуровневой памятью

Если комбинационная схема автомата построена из синхронизируемых элементов, то гонки также устраняются путем разделения синхронизации памяти и комбинационной схемы (рис. 53). В этом случае двойная память не требуется.

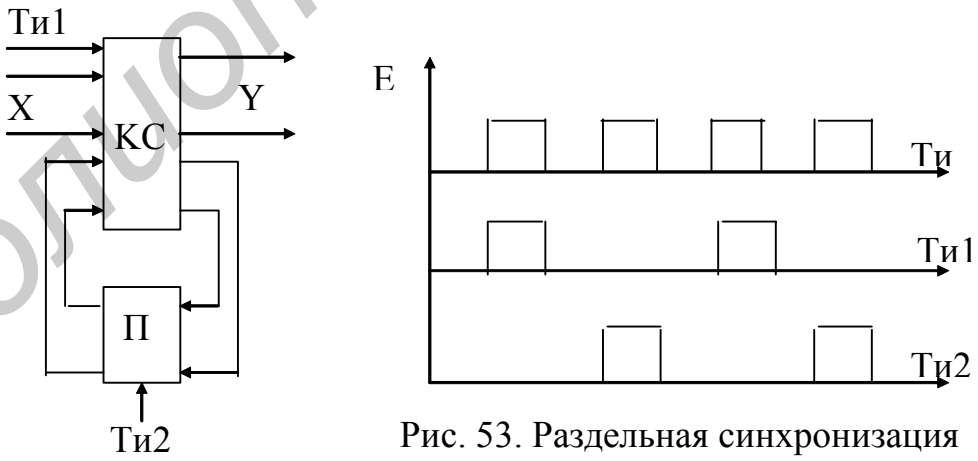


Рис. 53. Раздельная синхронизация

Риск сбоя. Наличие некритических ”состязаний” не нарушает правил перехода в автомате, но создает возможность возникновения риска сбоя. Риск сбоя заключается в том, что при переходе в некоторое промежуточное состояние (реально существующее в алгоритме) может быть выработан кратковременный ложный выходной сигнал. Например, в автомате Мура при переходе из

состояния 101 в состояние 110 появляется кратковременный сигнал y_k в промежуточном состоянии 100 (рис. 54).

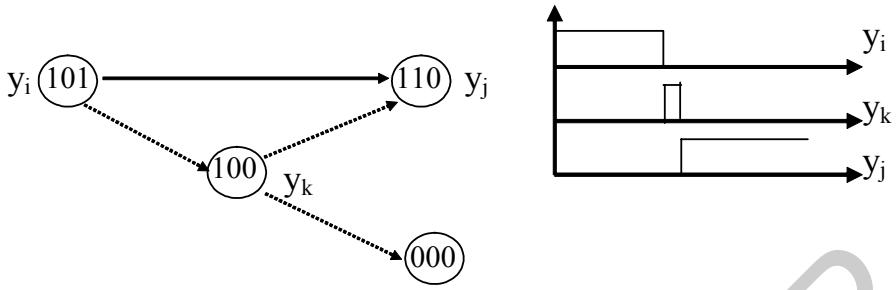


Рис. 54. Демонстрация явления риска сбоя

Хотя длительность ложного сигнала y_k достаточно мала, его возникновение может привести к непредсказуемым последствиям в работе устройства, которым управляет автомат. Для устранения риска сбоя можно воспользоваться следующими методами:

- выходы автомата, на которых может возникнуть риск сбоя, соединяются через конденсатор небольшой емкости с нулевым выходом источника питания (рис. 55);

- буферизация выходных сигналов;

- синхронизация выходных сигналов автомата. При этом комбинационная схема имеет дополнительный вход синхронизации C_s, поступающий с задержкой относительно основного синхросигнала. Сигналы на выходах автомата появляются только при наличии этого сигнала. Сигнал можно сформировать, как показано на рис. 56.

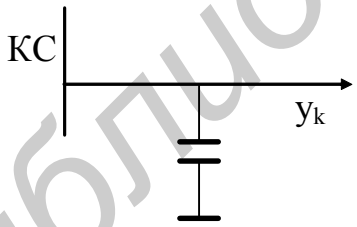


Рис. 55. Устранение риска сбоя с использованием емкости

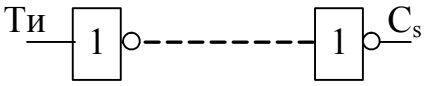


Рис. 56. Устранение риска сбоя с использованием линии задержки Ти

Схема может быть построена, например, из инверторов (их должно быть четное число). Данный подход позволяет устранить риск сбоя только для автомата Мура, так как синхронизация позволяет формировать выходные сигналы после окончания переходных процессов.

Литература

1. Савельев А.Я. Прикладная теория цифровых автоматов: Учеб. для вузов по спец. ЭВМ. –М.: Высш. шк., 1987.

2. Поснов Н.Н. Арифметика вычислительных машин в упражнениях и задачах: системы счисления, коды. –Мн.: Университетское, 1984.

3. Морозевич А.Н. Дмитриев А.Н. и др. МикроЭВМ, микропроцессоры и основы программирования. –Мн.: Выш. шк., 1990

4. Акушинский И.Я., Юдицкий Д.И. Машинная арифметика в остаточных классах. –М.: Сов. радио, 1968.

5. Питерсон У., Уэлдон Э. Коды, исправляющие ошибки: Пер. с англ. –М.: Мир, 1976.

Св. план 2004, поз.59

Учебное издание

Луцик Юрий Александрович,
Лукьянова Ирина Викторовна

АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Учебное пособие
для студентов специальности
”Вычислительные машины, системы и сети”
всех форм обучения

Редактор Т.А. Лейко
Корректор Е.Н. Батурчик
Компьютерная верстка

Подписано в печать . .2004.	Формат 60x84 1/16.	Бумага офсетная.
Гарнитур «Таймс».	Печать ризографическая.	Усл. печ. л.
Уч.- изд. л. 6,0.	Тираж 200 экз.	Заказ 561.

Издатель и полиграфическое исполнение:
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Лицензия ЛП № 156 от 30.12. 2002.
Лицензия ЛВ № 509 от 03.08. 2001.
220013, Минск, П.Бровки, 6