МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Интеллектуальные информационные технологии»

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ В ЭВМ И СИНТЕЗ ЦА ДЛЯ ИХ МИКРОПРОГРАММНОГО ВЫПОЛНЕНИЯ

Методические указания к выполнению практических занятий по дисциплине
«Организация и функционирование традиционных и интеллектуальных компьютеров»
для студентов специальности
1-40 03 01 «Искусственный интеллект»

Брест 2007

УДК 681.3

Методические указания предназначены для выполнения практических работ по дисциплине «Организация и функционирование традиционных и интеллектуальных компьютеров», а также как вспомогательный теоретический и прикладной материал для выполнения курсовых работ по этому же предмету. Восемь заданий и необходимый теоретический материал к ним учитывают ключевые элементы курса, необходимые для более полного овладения подходами к созданию современных архитектур ЭВМ на основе микропрограммного описания автоматов. Методические указания предназначены для использования студентами специальности «Искусственный интеллект».

Составитель: Матюшков Л.П., доцент, к.т.н.

Рецензент: Садыхов Р.Х., зав. кафедрой ЭВМ УО «БГУИР», д.т.н., проф.,
лауреат Государственной премии Республики Беларусь

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В соответствии с типовой учебной программой «Организация и функционирование традиционных и интеллектуальных компьютеров» (утв. Министерством Образования Республики Беларусь 24.06.2001 г. с регистрационным номером № ТД-182/тип) главной задачей практических занятий является углубление теоретической и практической подготовки студентов в области автоматизации процессов обработки информации, усвоение принципов построения и функционирования ЭВМ, ознакомление с особенностями выполнения арифметических и логических операций, методами синтеза и минимизации логических схем и схем с памятью, представлением информации в различных формах, тенденциями в развитии архитектур ЭВМ и принципах микропрограммного описания автоматов и методами синтеза их элементов, контроля выполнения арифметических, информационных и логических операций.

Пособие строится таким образом, что в рамках темы кратко приводятся теоретические положения с описанием основных методов решения задач и примерами.

Отчет о выполнении задания студентами должен отражать основные этапы решения задачи. Выбор конкретного задания и задач определяется с помощью ключа как функции от качественного и количественного состава букв из сплошного состава текста строки из фамилии, имени и отчества студента, а также его № в списке группы.

Теоретический материал и практические примеры могут использоваться как элементы для выполнения курсовой работы.

РАЗДЕЛ 1. СИСТЕМЫ СЧИСЛЕНИЯ, ПРЕДСТАВЛЕНИЕ ЧИСЕЛ В НОРМАЛЬНОЙ ФОРМЕ (ЗАДАНИЕ №1)

Выбор алфавита для описания ЦА влияет на сложность его технической реализации. Для представления чисел существует много систем счисления. Системой счисления называется совокупность цифровых знаков и правил их записи для однозначного изображения чисел. Среди них выделяют два крупных класса: позиционные и непозиционные системы.

Непозиционными называются системы, в которых применяется неограниченное количество цифр, и значение каждой цифры не зависит от ее позиции в числе. Таким примером является римская система счисления: $I(1), V(5), X(10), L(50), C(100)$.

Эта система для реализации в ЦА неудобна, так как количество цифр произвольно и правила выполнения операций над ними сложны.

В позиционных системах применяется конечный набор цифр, причем значение каждой из них зависит от позиции, занимаемой в числе. Количество различных цифр, применяемых в системе счисления, называется ее основанием.

Известная нам десятичная система счисления берет свое название от основания, содержащего десять цифр: 0, 1, 2,..., 9.

Произвольное десятичное число X можно представить в виде суммы попарных произведений:

$$X = \sum_{p=1}^{c} x_p \times 10^{k-p},$$

где c – число цифр в числе, k – число цифр в целой части числа, p – порядковый номер цифры слева направо в записи числа.

Например, $X = 118{,}375$, $C = 6$, $k = 3$. Тогда

$$X = 1 \times 10^2 + 1 \times 10^1 + 8 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2} + 5 \times 10^{-3}.$$

Каждая позиция числа называется разрядом и одна и та же значащая цифра имеет разный вес в зависимости от разряда, в котором она находится. Естественно, что нуль в любом разряде отражает нулевой вес.

В определении позиционной системы счисления нет ограничений на величину основания, поэтому могут существовать позиционные системы счисления с любым конечным числом цифр: двоичная, троичная и т.д. Предложенная форма записи для любого десятичного числа X может применяться в любой системе счисления, где X_p представляет одну из цифр системы счисления, а 10 будет означать код основания системы счисления p. Например, в двоичной системе счисления это будет два, в троичной три и т.д., а индексы p и др. представляются как соответствующие числа в данной системе счисления или же десятичной системы счисления для удобства чтения.

Использование систем с основанием более 10 представляет для нас некоторые неудобства, так как необходимо вводить новые знаки для цифр больших 9.

Чтобы отличить числа, записанные в других системах счисления, обычно после записи числа нижним индексом помечают основание системы.

Например, $101{,}1_2$ или $201{,}2_3$. Представим эти числа в соответствии с нашей общей записью для любого X:

$$X = 101{,}1_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} = 5{,}5_{10},$$
$$X = 201{,}2_3 = 2 \times 3^2 + 0 \times 3^1 + 1 \times 3^0 + 2 \times 3^{-1} = 19{,}7_{10}.$$

Если ограничиться конечной разрядной сеткой для представления чисел, то можно сделать следующие выводы из формулы для представления p-разрядного числа X в избранной системе счисления:

1. Минимальным числом, которое представляется при такой разрядной сетке, является $X = 10^{-p+1}$, при фиксации запятой после 1 разряда.

2. Максимальным числом при использовании всех p разрядов будет $X = 10^p - 1$, т.е. когда в каждом разряде записана максимальная цифра;

3. Общее количество целых чисел, которые можно записывать в p-разрядную сетку равно $K = 10^p$.

Конечно, если известно некоторое число M, то легко вычисляется и количество разрядов p, обеспечивающее запись любого числа из множества, например, целых чисел от 1 до M. Оно вычисляется как логарифм по основанию системы счисления от числа M, значение которого округляется до большего целого числа при наличии дробной части логарифма. Например, десятичный логарифм от числа 95 приблизительно равен 1,9. Это означает, что в десятичной системе счисления для представления чисел от 0 до 95 достаточно два разряда. В двоичной системе счисления для решения этой же задачи уже потребуется 7 разрядов, т.к. двоичный логарифм от числа 95 более 6, но менее 7. Без округления p выбирается лишь при целом значении логарифма.

Выбор системы счисления для цифрового автомата (ЭЦВМ) осуществляется с учетом следующих соображений: простота технической реализации запоминающего элемента, который мог бы хранить любую из цифр системы счисления с заданным основанием; помехоустойчивость кодирования цифр на носителях информации; минимум затрат оборудования при построении узлов и блоков ЭЦВМ; простоту арифметических действий и тем самым легкость реализации управления; наибольшее быстродействие в выполнении операций в секунду; возможность использования формального аппарата для анализа и синтеза ЭЦВМ; удобство работы человека и т.д.

Рассмотрим с этих позиций различные системы счисления. По критерию простоты технической реализации преимущество имеет двоичная система, так как две позиции легче удерживать и существует много простых технических реализаций для двухпозиционного элемента: электромеханическое реле (если контакт замкнут, то оно хранит единицу, а в противном случае – 0); транзистор (открыт – 0, закрыт – 1); магнитный материал (намагничен – 1, размагничен – 0) и т.д.

В современной электронной технике широко используется элемент, который называется триггером, для устойчивого хранения поступившего на его вход сигнала 0 или 1.

Для хранения -разрядного двоичного числа требуется p триггеров, из которых формируется так называемый регистр.

В двоичной системе счисления очень легко реализуются арифметические операции:

0 + 0 = 0	0 – 0 = 0	0 x 0 = 0	
0 + 1 = 1	1 – 1 = 0	0 x 1 = 0	0 : 1 = 0
1 + 0 = 1	1 – 0 = 1	1 x 0 = 0	1 : 1 = 1
1 + 1 = 10	10 – 1 = 1	1 x 1 = 1	

Практически их все можно свести к сложению, как будет показано далее. Например, при умножении числа сдвигаются, а затем складываются. Двоичная система счисления имеет и соответствующий формальный аппарат для анализа и синтеза вычислительных устройств на основе алгебры логики. Поэтому основной системой счисления для большинства ЭЦВМ стала двоичная, хотя были редкие попытки использовать и другие системы счисления, в частности, троичную и десятичную.

Методы перевода из одной системы счисления в другую позволяют обеспечить как представление информации в машине, так и отображение ее для человека.

Простейший способ перевода чисел из одной системы счисления в другую может выполняться на основе таблиц соответствия (смотрите таблицу1.1).

Таблица 1.1

Десятичная система	Двоичная система	Троичная система	Восьмеричная система
0	0	0	0
1	1	1	1
2	10	2	2
3	11	10	3
4	100	11	4
5	101	12	5
6	110	20	6
7	111	21	7
8	1000	22	10
9	1001	100	11
10	1010	101	12

Табличный способ перевода чисел из одной системы счисления в другую довольно громоздок и практически реализуется при синтезе устройств ввода и вывода цифр чисел. В частности десятичные цифры при вводе переводятся в свои двоичные эквиваленты, а при выводе их двоичные эквиваленты снова переводятся в десятичные цифры.

Формальные методы перевода чисел из одной позиционной системы счисления в другую разработаны отдельно для их целой и дробной частей.

Пусть целое число X_a уже переведено в новую систему в виде числа $X_c = X_1 c^{p-1} + X_2 c^{p-2} + \cdots + X_{p-1} c + X_p$, где X – цифры новой системы счисления c, причем $X_i < c$. Тогда разделив данное число на c, мы получим в остатке X_p – младшую цифру числа X_c. Поступив с результатом деления также, получим X_{p-1} и т.д. до X_1 включительно. На основании изложенного легко сформулировать правило перевода числа X_a в число X_c, представив «c» в системе счисления «a» и выполняя последовательно операции также в системе «a».

Для перевода целого числа в новую систему его надо последовательно делить на основание новой системы до тех пор, пока не получится частное, у которого целая часть равна 0. Число в новой системе счисления записывается из остатков от последовательного деления, причем последний остаток будет старшей цифрой нового числа.

Примеры. Перевести 59_{10} в двоичную и восьмеричную системы счисления:

```
59 |2
58 29 |2
 1 28 14 |2
    1 14  7 |2
      0  6  3 |2
         1  2  1 |2
            1  0  0
               1
```

```
59 |8
56  7 |8
 3  0  0
    7
```

Перевод из любой системы счисления в привычную нам десятичную систему счисления можно выполнять путем представления всех цифр в записи числа X_a и степеней 10 в виде их десятичных эквивалентов и выполнить операции умножения, и затем все числа сложить в десятичной системе счисления.

Например, переведем 111011_2 и 73_8 в десятичное представление:

$$111011_2 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 59,$$
$$73_8 = 7 \times 8^1 + 3 \times 8^0 = 59.$$

Этим путем можно сделать проверку правильности перевода чисел из десятичной системы в двоичную как целых чисел, так и дробей.

Перевод правильных дробей. Пусть правильная дробь X_c уже переведена и представлена в системе счисления c:

$$X_c = X_1 c^{-1} + X_2 c^{-2} + \cdots + X_p c^{-p}.$$

Последовательно умножая X_c и производные от него выражения на «c» и, отбрасывая по мере получения X, содержащие «c» в нулевой степени, получим:

$X_1 + X_2 c^{-1} + \cdots + X_p c^{-p+1}$ отбрасываем X_1,

$X_2 + X_3 c^{-1} + \cdots + X_p c^{-p+2}$ отбрасываем X_2,

X_p отбрасываем X_p.

Исходя из этой схемы, можно записать правило перевода правильных дробей. Для перевода правильной дроби из одной позиционной системы (a) в другую (c) ее надо последовательно умножать на основание «c» системы счисления, пока, в новой дроби не будет нужного количества цифр, соответствующего заданной точности представляемой дроби. Правильная дробь в новой системе записывается из целых частей произведений, полученных при последовательном умножении, причем первая целая часть будет старшей цифрой новой дроби. (Если получается на некотором шаге нулевая дробная часть, то процесс обрывается). Все вычисления делаются в исходной системе счисления.

Примеры. Перевести в двоичную систему счисления дроби 0,375 и 0,57 с точностью до 0,04.

0,375		0,57		
2		2		
0,750		1,14		
2	$0{,}375_{10} = 0{,}011$,	2	$0{,}57_{10} = 0{,}10010$,	
1,500		0,28		
2		2		т.к. $2^{-6} < 0{,}04$
1,0		0,56		
		2		
		1,12		
		2		
		0,24		

Ошибка перевода первой дроби равна 0. Во втором случае не превышает значения последнего разряда полученной двоичной дроби, т.е. 0,03125.

В том случае, когда при переводе десятичной дроби в двоичную форму представления процесс перевода будет бесконечным, его прекращают по двум причинам:

1. Получают заданное число разрядов.
2. Получают дробь с заданной точностью.

Неправильные дроби, имеющие целую и дробную части переводятся по частям. А затем записывается общий результат в новой системе счисления.

В выбранной системе счисления в ЦА используются различные формы представления чисел.

Естественной формой представления числа называется его запись в порядке следования цифр с отделением запятой целой части числа от дробной $X = X_1 X_2 \ldots X_c, X_{c+1} \ldots X_p$.

В ряде случаев удобно пользоваться нормальной формой представления числа, когда оно записывается как правильная дробь с фиксацией запятой перед первым значащим

разрядом и умножается на соответствующую степень основания, чтобы получить его истинное значение, т.е.:

$$X = 0,X_1X_2 \ldots X_p \times 10^a,$$

где $a = e$, e – число цифр числа в целой части или же $a = e$, где e – число нулей перед значащей цифрой дробной части числа, если целой части нет.

Показатель степени a называется порядком нормального числа, а цифровая часть $X_1 \ldots X_p$ – мантиссой. При записи порядок и мантисса представляются в естественной форме.

Приведем примеры нормальной записи чисел. Чтобы отличать порядок от мантиссы, будем записывать его вместе со знаком в круглых скобках после мантиссы.

Пусть X = 118,375 – десятичное число, Y = 1110110, 011 – двоичное число и T = 0, 0175 – правильная десятичная дробь, тогда они могут быть представлены в нормальной форме следующим образом:

$$X = 118{,}375 = 0{,}118375 \times 10^3 = 118375\ (+3),$$

$$Y = 1110110{,}011 = 0{,}111011011 \times 10^{111} = 1110110011\ (+111),$$

$$T = 0{,}0175 = 0{,}175 \times 10^{-1} = 0175\ (-1).$$

Так как в естественной форме положение запятой строго фиксируется между целой и дробной частями, то при машинной записи таких чисел при фиксированной разрядной сетке диапазон представления чисел будет колебаться от самого большого числа, которое будет играть роль «бесконечности» в машинном представлении до самого маленького, начиная с которого последующие числа будут восприниматься как машинный нуль.

Например, при выделении в восьмиразрядном двоичном числе одного разряда под знак + (0) или – (1) и пяти разрядов под целую часть самым большим числом по модулю будет $+11111{,}11 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} = 16 + 8 + 4 + 2 + 1 + 0 + 0{,}5 + 0{,}25 = 31{,}75$

а самым маленьким 0 и числа меньше $|0{,}01|$,которые будут также восприниматься при вводе и выполнении машинных операций как нуль, а числа большие $|31{,}75|$ будут давать переполнение разрядной сетки. Машины, в которых фиксируется запятая после некоторого разряда при представлении чисел, называются машинами с фиксированной запятой. Их недостатки мы уже частично увидели из рассмотренных примеров.

При представлении чисел в нормальной форме положение запятой определяется каждый раз величиной порядка и его знаком. Поэтому запятая как бы перемещается на заданную позицию (плавает). Машины, использующие такой принцип представления чисел, называются машинами с плавающей запятой. Например, если для тех же 8 разрядов отвести 4 разряда, включая знак под мантиссу, и 4 разряда, включая знак под порядок, получим следующий диапазон представления чисел $0{,}111 \times 10^{111} < |X| < 0$, т.е. $111000_2 = 56{,}0$ и числа, воспринимаемые как нуль, будут меньше $0{,}1 \times 10^{-111} = 0{,}0001_2 = 0{,}0625$.

В современных ЭЦВМ используются обе формы представления чисел в зависимости от характера решаемых задач.

Очевидно, что при выполнении арифметических операций над числами с плавающей запятой могут получиться результаты в ненормализованной форме. В этом случае обеспечивается их автоматическая нормализация. При сложении или вычитании двух чисел одно из них денормализуется, чтобы уравнять порядки чисел, иначе над ними нельзя выполнять эти операции. Очевидно, что в этом случае маленькое число может быть вообще потеряно, так как его порядок будет очень мал по сравнению с большим по модулю числом.

Мы видим, что обе системы представления чисел в ЭВМ дают погрешность. Поэтому при вычислительных процессах с большим количеством арифметических операций ошибка может быть существенной и причиной неверного решения задачи, может стать недостаточность выбранной разрядной сетки для их решения.

Особые трудности в оценке погрешностей вычислений представляет операция вычитания близких по значению чисел. Дело в том, что эти числа при вычислениях или вводе могут оказаться практически одинаковыми и выполнение операции вычитания может дать машинный нуль:

Задание №1

В этом задании требуется перевести целые, дробные и смешанные числа A_{10}, B_{10} и C_{10} из десятичной системы счисления в двоичную и p-ичную, а также записать эти же числа в нормальной форме.

Число M принимается равным количеству букв в фамилии студента. $K = M$, если $M < 10$, иначе $K = 10 - Г$ ($Г$ – количество гласных в первых четырех буквах фамилии студента). При переводе дробной части получать четыре знака числа (№ - соответствует номеру студента в списке группы).

$A_{10} = 30 + (M + №)$, $B_{10} = 0{,}3\,(M + №)$, $C_{10} = 15,(M + №)$.

1. Пример 5. Иванов. $M = 6, K = 6, № = 5$. $A_{10} = 30 + 6 + 5 = 41$, $B_{10} = 0{,}311$, $C_{10} = 15{,}11$.

2. Пример 3. Вознесенский. $A_{10} = 30 + 12 + 3 = 45$, $B_{10} = 0{,}315$, $C_{10} = 15{,}15$, $K = 10 - 1 = 9$.

РАЗДЕЛ 2. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ НАД ЧИСЛАМИ (ЗАДАНИЕ №2)

Сложение, умножение и вычитание на большинстве ЦА выполняется в двоичной системе, опираясь на правила, отраженные в следующих таблицах:

а) Сложение	б) Вычитание	в) Умножение
0 + 0 = 0	0 – 0 = 0	0 x 0 = 0
0 + 1 = 1	1 – 0 = 1	0 x 1 = 0
1 + 0 = 1	1 – 1 = 0	1 x 0 = 0
1 + 1 = (1) 0	0 – 1 = (1) 1	1 x 1 = 1
перенос в ст. разряд	заем из старшего разряда	

В основу арифметическо-логического устройства (АЛУ) любой ЭВМ может быть положен сумматор. В арифметических операциях чаще всего участвуют два числа A и B, и в результате получается новое число C. Любое арифметическое действие записывается так:

$C = A \nabla B$,

где ∇- знак арифметического действия (сложение, умножение, вычитание, деление).

Операндом является число, участвующее в арифметической операции, выполняемой ЦА. Так как ЦА оперирует только автоматными изображениями чисел, то последние выступают в качестве операндов. Поэтому для машинных операций более правильно писать *[C] = [A] ∇ [B]*, где *[X]* – автоматное изображение операнда.

Обозначим двоичные разряды операндов соответственно $A(i)$, $B(i)$, $C(i)$, перенос из данного разряда i в $(i+1)$ через $П(i)$, а символом «□» поразрядное сложение по модулю 2, и с учетом этих обозначений опишем работу основных элементов АЛУ.

Двоичным полусумматором называется устройство, выполняющее арифметические действия по правилам, указанным в таблице 2.1 (его условное обозначение смотрите на рисунке 2.1.а) для входов A_1 и B_1 и выходов C_1 и $П_1$.

Таблица 2.1

$A(i)$	$B(i)$	$C(i)$	$П(i)$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Появление единицы переноса несколько усложняет правила сложения двоичных цифр. Поэтому правила поразрядных действий при сложении операндов A и B можно записать так $A(i) + B(i) + П(i-1) = C(i) + П(i)$, где $П(i-1)$ – перенос из разряда $(i-1)$, $П(i-1)$ и $П(i)$ принимают значения 0 или 1.

Двоичным сумматором называется устройство, выполняющее арифметические действия по правилам, указанным в таблице 2.2 (его условное обозначение смотрите на рис. 2.1.б) для входов A_p, B_p, $П_{p-1}$ и выходов C_p, $П_p$.

Выполнение операции вычитания в ЦА обычно сводят к замене операции вычитания алгебраическим сложением, т.е. $A - B = A + (-B)$. Для этой цели отрицательные числа в машинах представляют в виде специальных кодов.

Таблица 2.2

$A(i)$	$B(i)$	$П(i-1)$	$C(i)$	$П(i)$
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

В работе ЦА чаще всего используется прямой, обратный и дополнительный коды.

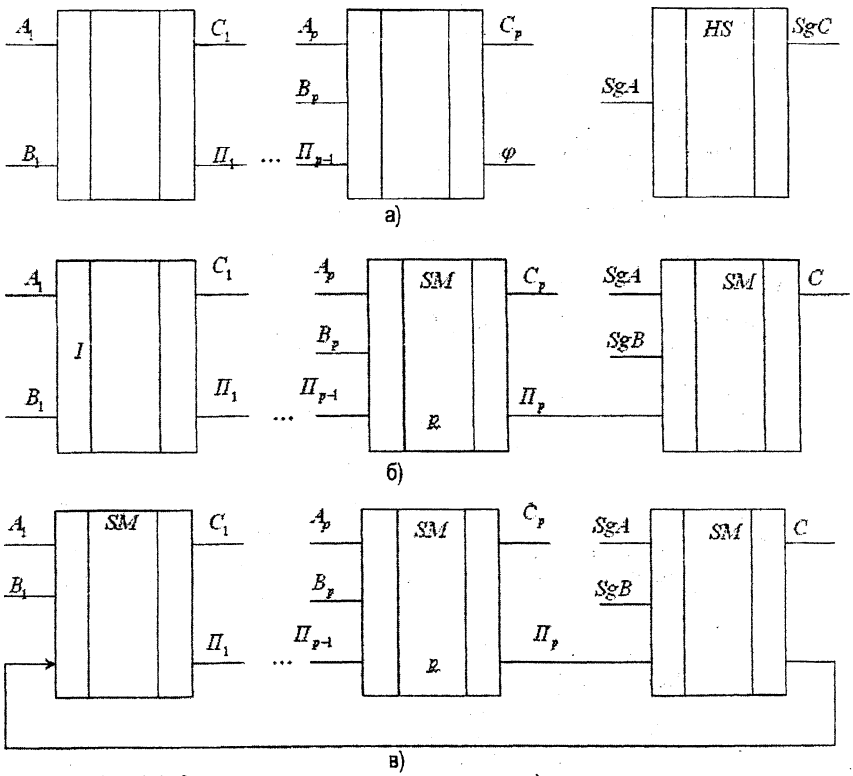


Рис. 2.1. Основные типы двоичных сумматоров: а) прямого кода; б) дополнительного кода; в) обратного кода

Если число $A = -0, A_1, A_2, \ldots, A_p$ задано в прямом коде, то оно в машинном представлении будет иметь следующий вид:

$[A]_{ПР} = 1, A_1, A_2, \ldots, A_p$.

Из определения следует, что в прямом коде все цифровые разряды отрицательного числа остаются неизменными, а в знаковой части записывается 1, т.е. если $A = -0{,}101$, то $[A]_{ПР} = 1{,}101$, а при $A \geq 0$ все разряды неизменны.

Дополнительный код положительного числа $A = 0, A_1, A_2, \ldots, A_p$ совпадает с его двоичной записью, а для отрицательного $[A_Д] = 1, \overline{A_1}, \overline{A_2}, \ldots, \overline{A_i}, \ldots, \overline{A_{k-1}}, \overline{A_k}, A_{k+1}, \ldots, A_p$, где $\overline{A_i} = 0$ при $A_i = 1$ и $\overline{A_i} = 1$ при $A_i = 0$, за исключением последнего значащего разряда k, для которого $\overline{A_k} = A_k = 1$ и далее $A_{k+1} = \cdots = A_p = 0$. Например. Для A=-0,101110 его представление в дополнительном коде будет: $[A_Д] = 1{,}010010$. Дополнительный код является математическим дополнением основанию системы счисления: $|A| + [A]_Д = 10_2$, где $|A|$ - абсолютное значение числа A $(0{,}101110 + 1{,}010010 = 10_2)$.

Иными словами $[A]_Д = \begin{cases} A, A \geq 0, \\ 10 - A, A < 0. \end{cases}$

В обратном коде положительные числа и нуль представляются в своей обычной записи, а коды отрицательных чисел поразрядно инвертируется, т.е. $A_i^{об} = \overline{A_i}$. Следует учитывать наличие двух нулей в обратном коде +0 и -0 (1,11...1).

Остановимся теперь детальнее на сложении чисел, представленных с фиксированной запятой на двоичных сумматорах. Рассмотрим насколько основных видов двоичных сумматоров.

Двоичный сумматор прямого кода (ДСПК) на p разрядов – сумматор, в котором отсутствует цепь поразрядного переноса между старшим цифровым и знаковым разрядами (смотрите рисунок 2.1).

На ДСПК можно складывать только числа, имеющие одинаковые знаки, т.е. на таком сумматоре не выполняется операция алгебраического сложения.

В самом деле, пусть заданы операнды $[A]_{пр} = SgA, A_1, A_2, \ldots, A_p$, $[B]_{пр} = SgB, B_1, B_2, \ldots, B_p$, где SgA, SgB - содержимое знаковых разрядов для изображений A и B. (*sign* - знак), A_k, B_ϕ – цифровые разряды изображений A и B.

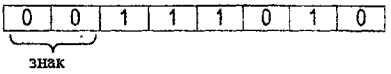
Если знаки А и В одинаковы, то сумма чисел будет иметь знак любого из них, а цифровая часть получится после сложения цифровых частей операндов:

1. $[A]_{пр}$=0,1011 $Sg\ A$=0
   $[B]_{пр}$=0,0100 $Sg\ B$=0
   $[C]_{пр}$=0,1111 $Sg\ C$=0
2. A=-0,0101 B=-0,1001
   $[A]_{пр}$=1,0101 $Sg\ A$=1 , ,0101
   $[B]_{пр}$=1,1001 $Sg\ B$=1 ⁺,1001
   $[C]_{пр}$=1,1110 $Sg\ C$=1 , ,1001

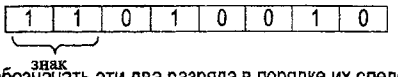
При сложении чисел на ДСПК возможен случай, когда абсолютное значение суммы операндов превышает единицу. Тогда имеет место переполнение разрядной сетки ЦА.

Признак переполнения – наличие единицы переноса из старшего разряда цифровой части сумматора. В этом случае должен вырабатываться сигнал переполнения φ=1, по которому происходит автоматический останов машины. Для других видов сумматоров вводится вспомогательный разряд в знаковую часть изображения числа, который называется разрядом переполнения. Такое представление числа (код) называется модифицированным. Схематически для правильной шестиразрядной двоичной дроби 0,111010 с положительным (а) и отрицательным (б) знаками это выглядит так:

1.



2.



Будем обозначать эти два разряда в порядке их следования слева неправо Sg_1 и Sg_2. Если при модифицированном представлении чисел при сложении на сумматорах дополнительно и обратного кода наступает переполнение, то $\varphi = 1$, когда $Sg_1C \wedge Sg_2C = 1$ или $\overline{Sg_1C} \wedge Sg_2C = 1$.

Двоичный сумматор дополнительного кода (ДСДК) – сумматор, оперирующий изображениями чисел в дополнительном коде. Характерная особенность ДСДК – наличие цепи поразрядного переноса из старшего разряда цифровой части в знаковый разряд (смотрите рисунок 2.1.6).

Признаком переполнения разрядной сетки сумматора дополнительного кода при сложении положительных чисел – отрицательный знак результата, а при сложении отрицательных чисел – положительный знак результата:

1. A=0,1011 B=0,1010
   $[A]_д$=0,1011
   $[B]_д$=0,1010
   $[C]_д$≠1,0101
2. A=-0,1010 B=-0,1001
   $[A]_д$=1,0101
   $[B]_д$=1,0111
   $[C]_д$≠0,1100

В этом случае использование модифицированного представления чисел в дополнительном коде с двумя знаковыми разрядами даст:

1. $[A]_д^м = 00{,}1011$
   $[B]_д^м = 00{,}1010$
   $\overline{[C]_д^м = 01{,}1010}$
2. $[A]_д^м = 11{,}0110$
   $[B]_д^м = 11{,}0111$
   $\overline{[C]_д^м = \underline{1}10{,}1100}$

Во втором примере подчеркнутая единица теряется.

Двоичный сумматор обратного кода (ДСОК) – сумматор, оперирующий изображениями чисел в обратном коде. Характерная особенность ДСОК – наличие цепи кругового или циклического переноса из знакового разряда в младший разряд цифровой части (смотрите рисунок 2.1.в).

Признаком переполнения разрядной сетки сумматора обратного кода является знак, противоположный знакам операндов.

1. A=0,0111 B=0,1101

$$[A]_{ОБ} = 0,0111$$
$$\underline{[B]_{ОБ} = 0,1101}$$
$$[C]^{М}_{Д} \neq 1,0100$$

2. A=-0,0110 B=-0,1101

$$[A]_{ОБ} = 1,1001$$
$$\underline{[B]_{ОБ} = 1,0010}$$
$$[C]^{М}_{Д} \neq 10,1100$$

Рассмотрим для этих случаев представление операндов в модифицированном коде.

1. $[A]^{М}_{ОБ} = 00,0111$

$$+$$
$$[B]^{М}_{ОБ} = 00,1101$$
$$\overline{\qquad\qquad}$$
$$[C]^{М}_{ОБ} = 01,0100$$

2. $[A]^{М}_{ОБ} = 11,1001$

$$+$$
$$[B]^{М}_{ОБ} = 11,0010$$
$$\overline{\qquad\qquad}$$
$$[C]^{М}_{ОБ} = 10,1100,$$ 01 и 10 – признаки переполнения.

Особенности сложения чисел, представленных в форме с плавающей запятой

При работе с ЦА с представлением чисел в форме с плавающей запятой должны быть два раздельных устройства для обработки мантисс и обработки порядков. После завершения операции сложения результат должен быть приведен в нормальную форму, а при сложении двух чисел с разными порядками необходимо одно из них денормализовать с целью уравнивания порядков. Уравнивание порядков приводит к необходимости поразрядного сдвига мантиссы вправо или влево.

Модифицированный сдвиг – операция, выполняемая следующим образом:

Исходная комбинация A	Сдвиг A влево	Сдвиг A вправо
$00,A_1A_2...A_p$	$0A_1A_2...A_p0$	$00,0A_1A_2...A_{p-1}$
$01,A_1A_2...A_p$	$1A_1A_2...A_p0$	$00,1A_1A_2...A_{p-1}$
$10,A_1A_2...A_p$	$0A_1A_2...A_p^*$	$11,0A_1A_2...A_{p-1}$
$11,A_1A_2...A_p$	$1A_1A_2...A_p^*$	$11,1A_1A_2...A_{p-1}$

Примечание: A_p^*=0 для дополнительного кода, A_p^*=1 для обратного кода.

Нарушение нормализации числа может быть справа и слева. Признак нарушения нормализации числа справа γ (величина равна или превышает единицу) – наличие разноименных комбинаций в знаковых разрядах сумматора. При $\gamma = \overline{Sg_1} \wedge Sg_2 = 1$ или $\gamma = Sg_1 \wedge \overline{Sg_2} = 1$ число сдвигается на один разряд вправо.

Признак нарушения нормализации числа слева β (результат меньше 10^{-1}) – наличие одинаковых комбинаций в разряде переполнения и старшем разряде цифровой части сумматора мантисс (A_1): $\beta = 1$, если $Sg_2 \wedge A_1 = 1$ или $\overline{Sg_2} \wedge \overline{A_1} = 1$ и тогда мантисса сдвигается влево.

Рассмотрим примеры. Пусть $A = M_A P_A$ и $B = M_B P_B$, где B - мантисса, P - порядок.

Случаи 1.P_A=P_B, т.е. обе мантиссы удовлетворяют условию возможности выполнения операции сложения.

Сложение мантисс осуществляется по правилам, изложенным ранее для чисел, представленных в виде двоичной записи с фиксированной запятой. Если после сложения мантисса результата удовлетворяет условию нормализации ($\beta = 0$ и $\gamma = 0$), то к результату приписывается порядок любого из операндов. Мантиссы и порядок в рассматриваемом примере будем обрабатывать на сумматорах дополнительного кода (шесть разрядов для мантиссы и четыре для порядка).

Пусть *A=0.1000*10$_2^{-11}$*, *B=-0,1011*10$_2^{-1}$*.

$[M_A]_Д^M = 00{,}1000$ $\qquad$ $[P_A]_Д = 1{,}101$

$[M_B]_Д^M = 11{,}0101$ $\qquad$ $[P_B]_Д = 1{,}101$

00,1000

11,0101

$[M_C]_Д^M$ 11,1101 $\gamma = 0$ $Sg_2 \wedge C_1 = 1$ т.е. $\beta = 1$

необходим сдвиг мантиссы $[M_C]_Д^M$ влево на 1 разряд ($[\overline{M_C}]_Д^M$):

$[M_C^1]_Д^M = 11{,}1010$ с одновременной коррекцией порядка (уменьшение c и e его на единицу), что достигается прибавлением кода порядка из всех единиц (1,111) Тогда $[P_C^1]_Д = 1{,}101 + 1{,}111 = 1{,}100$.

Для $[M_C^1]_Д^M$ снова $\beta = 1$ и аналогично делается следующий сдвиг и уменьшение порядка:

$[M_C^2]_Д^M = 11{,}0100$ $\qquad$ $[P_C^2]_Д = 1{,}011$

$\gamma = 0$, $\beta = 1 \wedge 0 = 0$. Процесс закончен и записываем результат перейдя из дополнительного кода в естественную двоичную форму записи:

M_C = -0,1100 $\qquad$ *P_C = -0,101* и тогда

*C = -0,1100*10$_2^{-101}$*

Выполним аналогичную операцию на сумматоре обратного кода для чисел *A = -0,1100 *10^{100}* и *B = - 0,1000 *10^{100}*

$[M_A]_{ОБ}^M = 11{,}0011$ $\qquad$ $[P_A]_{ОБ} = 0{,}100$

$[M_B]^M_{ОБ} = 11{,}0111$ $\qquad [P_B]_{ОБ} = 0{,}100$

$[M]^M_{ОБ} = 10{,}101\underline{1}$ $\qquad \beta = 0 \;\; Sg_2 \wedge C_1 = 0 \;\; \gamma = 1$

(Подчеркнутая единица получена в результате переноса из знакового разряда).

Здесь требуется модифицированный сдвиг мантиссы на один разряд вправо и одновременно коррекция порядка на величину 0,001;

$[M^1_C]^M_{ОБ} = 11{,}0101$ $\qquad [P^1_C]_{ОБ} = 0{,}100 + 0{,}100 = 0{,}101$

$\beta = 1, \; \gamma = 0$; $C = 0{,}1010 * 10_2^{101}$

Рассмотрим более общий случай, когда порядки чисел не равны, т.е. $P_A \neq P_B$. При операции сложения необходимо, чтобы веса разрядов совпадали, т.е. нужно на период выполнения операции уравнять порядки и нарушить нормализацию одного из чисел. Практически это означает, что порядок меньшего числа надо увеличить на величину $\Delta P = P_A - P_B$ и соответственно сдвинуть мантиссу меньшего числа вправо на ΔP, разрядов. Какой из операндов сдвигать укажет знак разности (P_A-P_B) при $P_A \geq P_B$ и - при $P_A < P_B$.

Сложим числа с разными порядками на сумматоре обратного кода (6 разрядов мантисса и 4 разряда порядок) $A = 0{,}1011 * 2^{-2}$ и $B = -0{,}1001 * 2^{-3}$.

$[M_A]^M_{ОБ} = 00{,}1011$ $\qquad [P_A]_{ОБ} = 1{,}101$

$[M_B]^M_{ОБ} = 11{,}0110$ $\qquad [P_B]_{ОБ} = 1{,}100$ $\qquad -[P_B]_{ОБ} = 0{,}011$

$P_A + (-P_B) = P_A - P_B$ $\qquad [P_A - P_B]_{ОБ}$ 1,101

+0,011

10,000 = 0,001

т.е. $P_A - P_B = 0{,}001$ $P_A > P_B$ и нужно сдвигать вправо мантиссу M_B на один разряд.

$[M^1_B]^M_{ОБ} = 11{,}1011$

$[M_A]^M_{ОБ} = 00{,}1011$ $\qquad \overline{Sg_2} = 1 \qquad \overline{P_1} = 1$

$[M_C]^M_{ОБ} = 00{,}0111$ $\qquad \beta = 1, \; \gamma = 0$

$[M^1_C]^M_{ОБ} = 00{,}1110$ $\qquad (\beta = 1, \; \gamma = 0)$

Чтобы выполнить операцию нормализация до конца, нужно соответственно уменьшить показатель порядка.

$[P^1_C]_{ОБ} = 1{,}101$

1,110

11,011 $= 1{,}100_{ОБ}$ $\qquad [P^1_C] = -0{,}110$

$C = 0{,}1110 * 10^{-11}$.

При реализации операций сложения чисел в форме с плавающей запятой может возникнуть переполнение разрядной сетки сумматора порядков. В этом случае также вводят дополнительный разряд, который предназначен для фиксации сигнала переполнения.

Методы умножения двоичных чисел

На практике используются два основных метода умножения двоичных чисел: младшими разрядами вперед и старшими разрядами вперед.

Метод умножения, начиная с младших разрядов множителя, иллюстрируется следующим примером:

```
   1101          – множимое
  ×1011          – множитель
   1101
  1101           – частные произведения
 +0000
 +1101
 10001111        – произведение
```

Метод умножения, начиная со старших разрядов множителя, иллюстрируется следующим примером:

```
   1101
  ×1011
  1101
   0000
 +  1101
    1101
 10001111        – произведение
```

Анализируя оба случая, замечаем, что операция умножения в двоичной системе сводится к сдвигам множимого и сложению частных произведений и этот процесс управляется в соответствии со значениями разрядов множителя: если в разряде множителя находится единица, то к сумме частных произведений добавляется множимое с соответствующим сдвигом, а если в разряде множителя 0, то операция сложения не выполняется. В связи с этим на сдвигах и сложениях реализуются операции умножения в ЦА.

Для реализации операции умножения необходимо иметь сумматор, регистры для хранения множителя и множимого и схему анализа разрядов множителя. Сумматор и регистры должны иметь цепи сдвига содержимого в соответствии с принятым способом умножения.

При последовательном умножении вид многократно повторяющегося по количеству разрядов цикла процесс умножения может быть описан итерационной формой: $T(i) = T(i-1) + A \cdot B(i)$, где $T(i)$ и $T(i-1)$ суммы частных произведений на -ом и $(i-1)$-ом шагах. При точном умножении двух чисел количество цифр в их произведении не превышает суммы количества разрядов сомножителей. При умножений нескольких чисел количество цифр может превысить $2p$, если -разрядная сетка для представления одного сомножителя. На практике ограничиваются для ЦА удвоенным количеством разрядов сумматоров и тем самым результаты вычислений при умножениях более двух чисел будут с возрастающей погрешностью. Поэтому существенное значение имеет округление результатов так, чтобы погрешность была знакопеременной, что позволит по возможности свести погрешность к половине значения младшего разряда.

При выполнении операции умножения чисел возможен выход за пределы разрядной сетки только со стороны младших разрядов.

Умножение чисел A и B, представленных в форме с фиксированной запятой, на двоичном сумматоре прямого кода.

$[A]_{ПР} = SgA, A_1, A_2, \ldots, A_p,$

$[B]_{ПР} = SgB, B_1, B_2, \ldots, B_p,$

$[C]_{ПР} = SgC, C_1, C_2, \ldots, C_p.$

где $SgC = SgA \oplus SgB$, $\oplus$ – знак сложения по модулю 2.

Умножим числа $[A]_{ПР} = 1{,}11010$ и $[B]_{ПР} = 0{,}1101$.

Для отображения процесса умножения воспользуемся следующими обозначениями:

: = - оператор присваивания (левая часть от знака присваивается правой);

$[\overrightarrow{PA}]$ - сдвиг содержимого A в регистре P на 1 разряд вправо;

[*СМ*] - содержимое сумматора *СМ*

И.П.- исходное положение

Знак произведения будем определять отдельно от цифровой части: $SgC = SgA \oplus SgB = 11$. Без учета знака покажем получение результата на 10 разрядном сумматоре и 5 разрядных регистрах (без учета знака) *РА* и *РВ*. Процесс умножения тогда можно представить в таблице.

Сумматор	Регистр *В*	Примечание
0000000000	1 1 0 0 1	
+11010	$B_1 B_2 B_3 B_4 B_5$	
1101000000		
0110100000 - сдвиг	-1100	
0011010000	–110	
0001101000	—11	
+11010		
1110101000		
0111010100	—1	
+11010		
10100010100*		
1010001010		

Если в процессе умножения возникает единица переноса (*) из старшего разряда, то ее сохраняют.

Чтобы процесс умножения происходил правильно, необходимо предусмотреть блокировку выработки сигнала переполнения так как возможно временное переполнение на каком-то шаге умножения (*) . Из примера видно, что необязательно иметь сумматор $2p$ разрядов, хранение «хвостов» произведения можно осуществлять в освобождающихся разрядах регистра множителя путем создания цепи передачи информации из младшего разряда сумматора в старший разряд регистра множителя.

Особенности умножения и деления чисел, представленных в форме с плавающей запятой

При операции умножения (или деления) действия, выполняемые над мантиссами и порядками различны: мантиссы перемножаются (делятся), а порядки складываются (вычитаются). Результат умножения может получится ненормализованным и тогда потребуется операция нормализации с одновременной коррекцией порядка результата аналогично как и при сложении чисел.

Задание №2

Сложить в дополнительном и обратном модифицированных кодах с плавающей запятой заданные числа A и B в следующих вариантах (найти сумму C): $C = A + B$, $C = -A + B$, $C = (-A) + (-B)$.

При выполнении этих заданий учесть особенности сумматоров дополнительного и обратного кодов. Найти в прямом коде на соответствующем сумматоре произведение $C = (-A) \times B$, когда число разрядов сумматора в два раза больше, чем разрядная сетка задаваемых чисел.

В качестве исходных взять числа $A = -0{,}1 - - - \times 10^{+10}$, $B = 0{,}1 - - - \times 10^{1}$, где в соответствии с набором букв фамилии и имени студента черточки заменяются нулями и единицами. Гласной букве ставится в соответствие «0», а согласной – «1». Например, Лукин Сергей: $A = -0{,}1101 \times 10^{+10}$, $B = 0{,}1011 \times 10^{1}$.

РАЗДЕЛ 3. СИНТЕЗ КОМБИНАЦИОННЫХ УСТРОЙСТВ (ЗАДАНИЯ №№3, 4)

Процесс логического проектирования включает синтез цифрового автомата на логическом уровне, т.е. на основе системы булевых функций в заданном базисе (функционально полный набор элементарных логических функций) строится логическая структура устройства.

Логическая структура включает способы соединения простейших схем, которые называются логическими элементами. Чаще всего логическая структура будет включать такие элементы как инвертор, конъюнктор и дизъюнктор, поскольку они образуют функционально полный набор.

Синтез логических устройств делят на четыре этапа:

1. Составление таблицы истинности синтезируемого узла согласно его назначению и описанию принципа работы.
2. Составление математической формулы для логической функции, описывающей работу синтезируемого узла (по таблице истинности).
3. Анализ полученной функции с целью построения различных вариантов ее математического выражения и нахождения лучшего из них в соответствии с тем или иным критерием.
4. Составление функциональной (логической схемы узла) из элементов НЕ (инвертор), И (конъюнктор), ИЛИ (дизъюнктор).

Нормальные формы логических функций

Таблица истинности является одним из удобных инструментов для описания работы комбинационного устройства (узла). Однако она может быть значительной по объему. Если в таблице расположить записи возможных значений аргументов в порядке возрастания двоичного числа, то можно каждую строку представлять условно одним знаком y_i^k, где k – определяет количество переменных, а i - номер строки в двоичной нумерации в таблице. Тогда запись y_4^3 будет раскрываться как $y(1,0,0)$. Конкретное значение функции для соответствующего выхода y_i^1, y_i^2, ..., y_i^e запишется так: $y_i^k = y_i^1$, $y_i^k = y_i^2$ ... и т.д.

Для того чтобы получить аналитическое выражение функции, заданной таблично, нужно составить сумму конституент единицы для тех наборов значений входных двоичных переменных, для которых $y_i^k = 1$, причем символ любой переменной в некотором конституенте берется со знаком отрицания, если конкретное значение переменной x_j в y_i^k имеет значение 0.

Зададим закон функционирования синтезируемой электронной схемы, имеющей три входа (x_3, x_2, x_1) и два выхода (Φ, M), в виде таблицы 3.1 и на ее основе запишем соответствующие аналитические выражения для каждого выхода Ф и М.

Тогда все строки таблицы для переменных x_3, x_2, x_1 можно записать так: $y_0^3 = y(0,0,0)$, $y_1^3 = y(0,0,1)$, ..., $y_6^3 = y(1,1,0)$, $y_7^3 = y(1,1,1)$. Функции Ф и М через конституенты единицы можно описать так: $\Phi(x_1,x_2,x_3) = y_3^3 + y_4^3 + y_5^3 + y_6^3 = \bar{x}_3 x_2 x_1 + x_3 \bar{x}_2 \bar{x}_1 + x_3 \bar{x}_2 x_1 + x_3 x_2 \bar{x}_1$, $\mathrm{M}(x_1,x_2,x_3) = y_2^3 + y_3^3 + y_4^3 + y_5^3 = \bar{x}_3 x_2 \bar{x}_1 + \bar{x}_3 x_2 x_1 + x_3 \bar{x}_2 \bar{x}_1 + x_3 \bar{x}_2 x_1$.

Говорят, что эти функции в совершенной дизъюнктивной нормальной форме (СДНФ). Нормальная форма называется потому, что все ее члены имеют вид элементарных конъюнкций. Из-за того, что все члены соединены в одну функцию знаком дизъюнкции

эта форма называется дизъюнктивной. Термин совершенная связан с тем, что все члены имеют высший ранг, являясь конституентами единицы.

Таблица 3.1

x_3	x_2	x_1	Ф	М
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	0	0

По аналогии за счет использования свойств симметричности формул алгебры логики можно ввести понятие совершенной конъюнктивной нормальной формы (СКНФ).

Аналитическое выражение функции, заданной таблично в СКНФ, получается следующим образом: нужно составить логическое произведение конституент нуля для тех наборов значений входных двоичных переменных, для которых реализация функций y_i^p равна 0, причем символ каждой переменной инвертируется по отношению к его табличному вхождению в набор y_i^k.

Пользуясь этим правилом запишем функции Ф и М для рассмотренного ранее примера:

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

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

Функциональная схема (рис. 3.1) может строиться прямым замещением элементарных произведений, сумм и отрицаний соответственно конъюнкторами, дизъюнкторами и инверторами. Покажем это для примера (таблица 3.1) на базе а) СДНФ и б) СКНФ для выхода Ф.

Нами пока специально опущен третий этап – минимизация логических функций, к рассмотрению которого мы сейчас переходим. Инженер-разработчик может предопределить параметры проектируемого комбинационного устройства путем целенаправленного подбора в соответствии с выбранными критериями.

Критерии могут порождать противоречивые ситуации, например, быстродействие обеспечивается за счет организации параллельной работы элементов устройства, но это ведет к увеличению оборудования, а значит возрастает и стоимость.

Поэтому какой-то из частных критериев считается более главным и по нему ведется оптимизация. Чаще всего основным критерием является минимум оборудования.

Можно заметить, что в рамках нормальных форм минимальной будет такая разновидность функции, которая состоит из наименьшего количества членов при наименьшем, по возможности, общем числе символов переменных, а также с возможно меньшим числом отрицаний переменных.

Минимизация функции обычно проводится в три этапа:

1. Выполнение перехода от СДНФ (или СКНФ) к сокращенной СДНФ (СКНФ) путем производства всевозможных склеиваний друг с другом сначала конституент, а затем всех производных членов более низкого ранга, чтобы получить сокращенную нормальную дизъюнктивную (конъюнктивную) форму, членами которой служат несклеивающиеся элементы. Они называются простыми импликантами. Не исключается ситуация, когда исходная СДНФ сразу содержит простые импликанты.

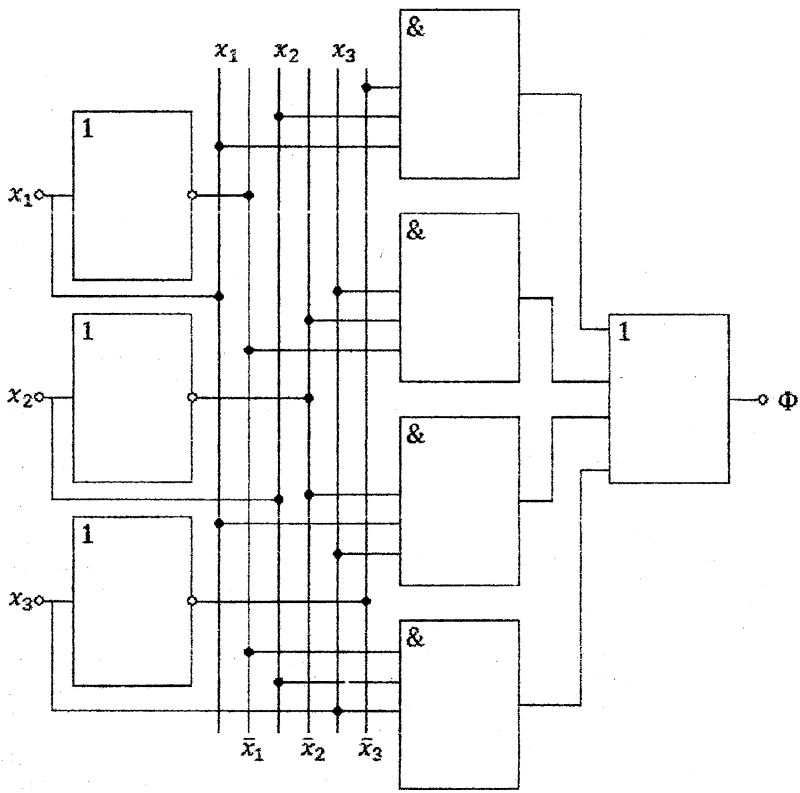


Рис. 3.1 Функциональная схема для выхода Ф на базе СДНФ

2. Переход от сокращенной нормальной формы к тупиковой нормальной форме (ТНФ). ТНФ – называется такая ДНФ, членами которой являются простые импликанты, среди которых нет ни одной лишней. Лишней называется импликанта, удаление которой не влияет на истинность значения функции. ДНФ в частном случае может быть сразу ТНФ.

3. Переход от ТНФ к ее минимальной форме. Этот этап называется факторизацией. Он не является полностью формальным и требует определенной интуиции и опыта. В

первую очередь здесь используется метод проб и испытаний. Для уменьшения числа операций отрицания применяют инверсии, а для уменьшения числа конъюнкций и дизъюнкций – распределительные законы. Здесь же учитывается возможность реализации схемы на реальных элементах с учетом их ограничений по количеству входов, величине допустимой нагрузки и т.п. Из-за сложностей третьего этапа при минимизации иногда ограничиваются первыми двумя.

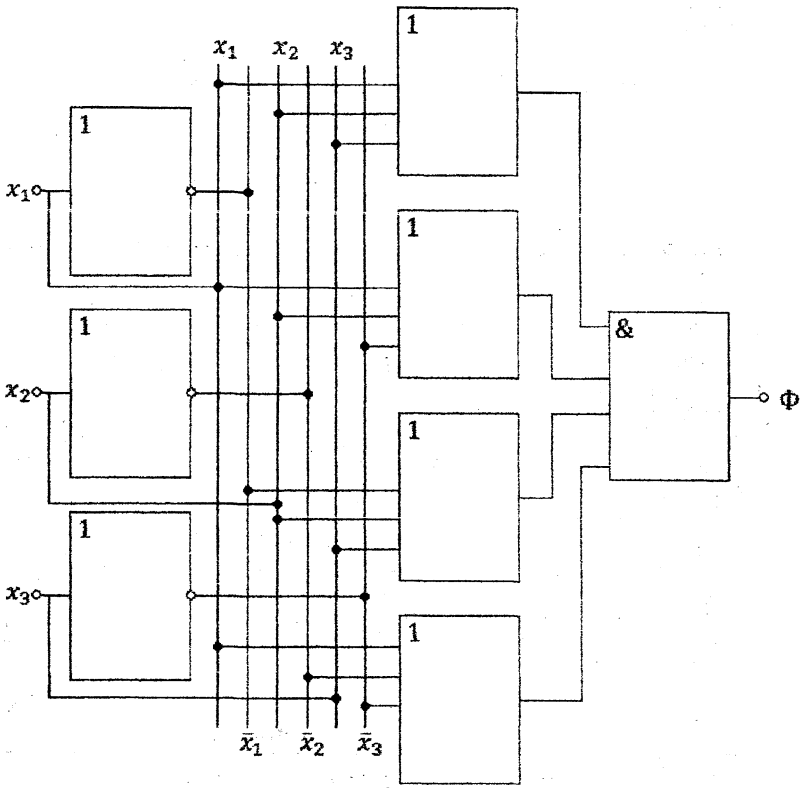


Рис. 3.2 Функциональная схема для выхода Ф на базе СКНФ

Покажем, как выполняется процесс минимизации различными методами на примере функции Ф (таблица 3.1 и записи СДНФ и СКНФ).

Проведем по описанным выше шагам минимизацию функции Ф.

1. $\Phi_{\text{СДНФ}} = x_3x_2x_1 + \bar{x}_1\bar{x}_2x_3 + x_3\bar{x}_2x_1 + x_3x_2\bar{x}_1,$

   $\Phi_{\text{ДНФ}} = \bar{x}_1x_3 + \bar{x}_2x_3 + x_1x_2\bar{x}_3.$

Все члены этой формы изолированы и поэтому выполняется 2-й этап.

2. Любая импликанта становится равной 1 только на одном наборе аргументов, но если на этом наборе равны 1 и другие импликанты, то это будет означать, что она не влияет на значение истинности функции.

Применим это правило для $\Phi_{\text{ДНФ}}$: $x_1x_2\bar{x}_3 = 1$ при $x_1 = 1$, $x_2 = 1$ и $x_3 = 0$, а на этом же наборе $\bar{x}_2x_3 = 0$, $0 = 0$ и $x_1x_3 = 0$, $0 = 0$, т.е. дальнейшие упрощения $\Phi_{\text{ДНФ}}$ за счет $x_1x_2\bar{x}_3$ невозможны. Аналогичные выводы получим относительно $\bar{x}_2x_3$, т.е. $\Phi_{\text{ДНФ}} = \Phi_{\text{ТНФ}}$ (тупиковая форма).

Проведем аналогичные операции с $\Phi_{\text{СКНФ}}$.

1. $\Phi_{\text{СКНФ}} = (x_1 + x_2 + x_3)(\bar{x}_1 + x_2 + x_3)(x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + \bar{x}_2 + \bar{x}_3)$,
   $\Phi_{\text{КНФ}} = (x_2 + x_3)(x_1 + x_3)(\bar{x}_1 + \bar{x}_2 + \bar{x}_3) = \bar{x}_1x_3 + \bar{x}_2x_3 + x_1x_2\bar{x}_3$.
2. После аналогичных действий как и для ДНФ получим, что:
   $\Phi_{\text{КНФ}} = \Phi_{\text{ТНФ}} = \bar{x}_1x_3 + \bar{x}_2x_3 + x_1x_2\bar{x}_3$.

На 3-м этапе упрощаем ТНФ функции Ф с точки зрения ее схемной реализации.

Расчетно-табличный метод минимизации (Квайна-Мак-Класки)

Метод был предложен Квайном и в дальнейшем развивался Мак-Класки с позиций его автоматизации на ЭВМ.

Первый и третий этапы в этом случае идентичны с расчетным методом, а нахождение тупиковой формы (второй этап) проводится с помощью специальной таблицы.

На примере проиллюстрируем особенности этого метода для функции $У_{\text{ДНФ}} = \bar{x}_1x_3 + \bar{x}_2x_3 + \bar{x}_1x_2$.

На втором этапе строится таблица (3.2) покрытий (Квайна). Она имеет число строк, равное количеству импликант в сокращенной ДНФ, а количество столбцов берется равным количеству конституент в СДНФ с наименованиями столбцов в порядке следования конституент в СДНФ.

Крестиками отмечаются клетки на пересечении импликанта и конституенты, если импликант полностью содержится в данной конституенте. Далее анализируется состав крестиков по столбцам. Если в данном столбце находится только один крестик, то это означает обязательное вхождение соответствующей импликанты в $У_{\text{ТДНФ}}$. Таковыми импликантами являются $\bar{x}_2x_3$ и $\bar{x}_1x_2$, но они вместе покрывают еще и конституенты $\bar{x}_1\bar{x}_2x_3$ и $\bar{x}_1x_2x_3$, что позволяет исключить импликант $\bar{x}_1x_3$ из $У_{\text{СДНФ}}$ и получить $У_{\text{ТДНФ}} = \bar{x}_1x_2 + \bar{x}_2x_3$.

Таблица 3.2

Импликанты	Конституенты			
	$\bar{x}_1\bar{x}_2x_3$	$\bar{x}_1x_2\bar{x}_3$	$\bar{x}_1x_2x_3$	$x_1\bar{x}_2x_3$
$\bar{x}_1x_3$	X		X	
$\bar{x}_2x_3$	X			X
$\bar{x}_1x_2$		X	X	

В более сложном случае может возникнуть задача выбора из нескольких вариантов покрытий, тогда можно сохранять импликанты, например, в порядке следования их в столбце или еще по какому-нибудь принципу.

О тех импликантах, которые нельзя вычеркнуть (осуществляют единственное покрытие) говорят, что они входят в ядро функции.

Табличный метод минимизации

В этом методе два первых этапа выполняются на основе специальной таблицы - диаграммы Вейча-Карно, а третий этап аналогичен расчетному методу.

Диаграмма Вейча-Карно является разновидностью табличной записи некоторой функции, заданной в СДНФ или СКНФ. Она имеет вид прямоугольника, разбитого на 2^p малых квадратов. Каждому квадрату ставится в соответствие одна из конституент в разложении p –аргументной функции (для СДНФ или СКНФ), а на осях x и y размещается по $p/2$ координат при четном p, и при нечетном p по $(p-1)/2$ и $(p+1)/2$ соответственно.

Чтобы задать функцию с помощью такой таблицы, нужно в каждую ее клетку записать одну конституенту (единицы в СДНФ или нуля в СКНФ) и значение соответствующей конкретной реализации функции. Использование таких таблиц для минимизации предполагает некоторые ограничения на записи: в соседних по горизонтали (или вертикали) клетках должны находиться соседние конституенты.

Координаты по осям x и y на диаграмме Вейча-Карно проставляются в двоичном циклическом коде Грея, что обеспечивает выполнение условия их соседства.

Двоичные координаты отображаются в двоичный циклический код Грея по следующему правилу:

1. Самая старшая значащая цифра (единица) числа в коде Грея совпадает с самой старшей значащей цифрой этого же числа в двоичном коде.
2. Цифра в любом другом, более младшем разряде числа в коде Грея а) совпадает с соответствующей цифрой числа в двоичном коде, если слева от данной цифры в коде Грея имеется четное число единиц или же б) совпадает с отрицанием соответствующей цифры в двоичном коде, если слева от данной цифры в коде Грея имеется нечетное число единиц.

Например, построим код Грея У для числа $x = x_4x_3x_2x_1 = 1001_2$. $y_4 = x_4 = 1$, $y_3 = x_3 = \bar{0} = 1$, $y_2 = x_2 = 0$, $y_1 = x_1 = 1$, $y = 1101$.

Применение кодов Грея для нумерации по осям автоматически обеспечивает размещение в соседних клетках соседних членов СДНФ (СКНФ). В самих клетках ставятся лишь значения истинности реализации функции на данном наборе аргументов (ставятся только единицы для СДНФ или нули для СКНФ). Двоичные числа будем разбивать на две части: первую часть в порядке возрастания $(x_1\, x_2 \ldots x_{p/2})$ будем писать по оси y, а вторую $(x_{p/2+1} \ldots x_p)$ по оси x (при нечетном p будем по оси y брать последним $x_{(p-1)/2}$ и первым по оси x - $x_{(p+1)/2}$). Тогда, преобразовав каждое из чисел по оси в код Грея, получим нужную кодировку координат клеток в таблице так, чтобы выполнялось условие соседства.

Покажем на примере при $p = 3$ как выполнить нумерацию в кодах Грея при отсчетах, начиная с 0 (а) – обычная нумерация и (б) – нумерация в кодах Грея.

а)

x_1					
	1				
	0				
		00	01	10	11

x_2x_3

б)

x_1					
	1			$x_1x_2x_3$	
	0		$\bar{x}_1\bar{x}_2x_3$		
		00	01	11	10

x_2x_3

Таким образом, в соответствии с кодированием по Грею клетка с координатами 001 будет отражать конституент $\bar{x}_1\bar{x}_2x_3$, клетка 111 - $x_1x_2x_3$ и т.д.

Чтобы приступить к выполнению процедуры минимизации в тех клетках таблицы, где функция на данных конституентах принимает истинное значение записываются 1 в СДНФ (или же нули в СКНФ).

Процесс минимизации выполняется по следующим правилам: 2^i смежных клеток, расположенных в виде прямоугольника, соответствуют элементарной конъюнкции (дизъюнкции), ранг которой меньше ранга τ конституенты на i единиц.

Примечание. На диаграмме Вейча соседними клетками считаются также и клетки, находящиеся на противоположных концах любой строки и любого столбца. Это соответствует модели, когда диаграмма по горизонтали или вертикали свертывается в цилиндр. Переменные КНФ записываются в инверсной форме.

Получение результата минимизации может происходить со следующими особенностями:

1. Если функция на диаграмме не имеет смежных клеток с 1, то тупиковая нормальная форма тождественна ее СДНФ.
2. Если единицы заполняют две соседние по строке или столбцу клетки, то соответствующие этим клеткам две конституенты можно заменить одним элементарным членом, ранг которого на единицу меньше.
3. Если единицы заполняют половину всех клеток диаграммы и представляют из себя сплошной массив в виде прямоугольника или квадрата, то выражаемый ими член состоит всего из одной переменной со знаком отрицания или без него.

Таким образом, процесс табличной минимизации сводится к нахождению наиболее крупных групп из 2^i соседних (заполненных) клеток, причем надо стараться, чтобы каждая из заполненных клеток входила в какую-нибудь группу и, по возможности, только в одну. Импликанта, соответствующая избранной группе заполненных клеток, будет содержать в себе символы тех переменных, значения истинности которых совпадают у всех объединенных клеток.

Покажем на примере, как идет минимизация табличным методом для функции Ф (табл. 3.1).

$\Phi_{СДНФ} = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2x_3 + x_1x_2\bar{x}_3 + x_1\bar{x}_2x_3.$

Построим диаграмму Вейча-Карно для $\Phi_{СДНФ}$:

x_1	00	01	11	10	x_2x_3
1		1		1	
0		1	1		

$\Phi_{ТДФ} = \bar{x}_2x_3 + \bar{x}_1x_3 + x_1x_2\bar{x}_3.$

Построим диаграмму Вейча-Карно для функции Ф на базе СКНФ.

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

x_1	00	01	11	10	x_2x_3
1	0		0		
0	0			0	

Каждая из конституент нуля на диаграмме Вейча-Карно отражена в соответствии с их координатами в таблице 3.1. Поэтому в результатах, соответствующих тупиковым конъюнктивным формам (ТКНФ), каждая из координат при записи инвертируется.

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

Во всех методах минимизации можно эффективно использовать наборы переменных, на которых функция не определена путем введения конституентов 0 или 1, чтобы увеличить количество соседних членов для построения структуры функции с членами меньших рангов. Покажем это на примере диаграммы Вейча-Карно, где черточками показаны наборы, на которых функция не определена.

Отобразим функцию $У_{\text{СДНФ}}$ на диаграмме Вейча-Карно.

x_1x_2 \ x_3x_4	00	01	11	10
10	0	-	1	-
11	-	1	-	0
01	-	0	-	1
00	0	-	1	-

Воспользовавшись возможностью доопределить функцию, единицами вместо черточек (в прямоугольниках), получим в результате:

$У = x_1x_4 + \bar{x}_1x_2$.

Минимизация схем со многими выходами заключается в нахождении таких выражений для совокупности переключательных функций, в которых наиболее полно используются члены, общие для нескольких функций.

Задания №№ 3 и 4

Минимизация логических функций и синтез схем (без памяти).

В зависимости от последовательности букв в фамилии, имени и отчестве заменить букву n (каждую) по порядку на нуль (0), если буква в Ф.И.О. гласная, и на 1 в противном случае. Букву w разрешается заменить по своему усмотрению, исходя из возможности получить минимальное выражение для функции У.

Задачи.

1. Записать $У_{\text{СКНФ}}$ и $У_{\text{СДНФ}}$.
2. Записать функцию $У = У^4_{\text{Ф}} + У^4_{\text{И}} + У^4_{\text{О}}$.
3. Минимизировать функцию У алгебраическим методом с последующим отысканием тупиковой формы методом покрытий Квайна-Мак-Класки.
4. Применить для минимизации функции диаграмму Вейча-Карно с использованием w (черточка).
5. Начертить схему для полученной любым методом тупиковой формы.
6. Заменить w нулями и применить диаграмму Вейча-Карно для СКНФ и СДНФ, исходя из требуемого объема подготовительной работы для получения тупиковой формы.

x_4	x_3	x_2	x_1	y
0	0	0	0	0
0	0	0	1	w
0	0	1	0	n
0	0	1	1	1
0	1	0	0	1
0	1	0	1	n
0	1	1	0	w
0	1	1	1	n
1	0	0	0	1
1	0	0	1	n
1	0	1	0	w
1	0	1	1	n
1	1	0	0	0
1	1	0	1	n
1	1	1	0	1
1	1	1	1	w

РАЗДЕЛ 4. СИНТЕЗ ЦА С ПАМЯТЬЮ (ЗАДАНИЯ №№5, 6)

Построение любого ЦА возможно, если выбранный для синтеза набор элементов является функционально полным.

Поскольку в частном случае ЦА может обладать нулевой памятью, то необходимой составной частью такого набора является любая полная система логических (булевых) элементов. Другой ее составной частью должно быть некоторое элементарное устройство памяти. Под элементарным на практике обычно понимают такое устройство памяти, которое имеет несколько состояний и, следовательно, может запомнить любую букву (цифру) структурного алфавита и помимо входов обладает одним выходом для выдачи запомненной величины.

В теории ЦА доказано следующее утверждение: «Для того, чтобы набор некоторых элементов был функционально полным для структурного синтеза ЦА любой сложности, необходимо и достаточно, чтобы он содержал логические элементы, образующие функционально полный набор для синтеза комбинационных схем и запоминающие элементы, обладающие полной системой переходов и выходов».

Считают, что ЦА обладает полной системой переходов, если для каждой пары его внутренних состояний s_i и s_j найдется хотя бы один входной сигнал, который переводит автомат из состояния s_i в s_j.

Автомат Мура обладает полной системой выходов, если в каждом внутреннем состоянии он вырабатывает выходной сигнал, отличный от всех других выходных сигналов. Простейшим автоматом такого типа является триггер.

Триггер – логическая схема с двумя устойчивыми состояниями $s_1 = 0$ и $s_2 = 1$. Состояние триггера Т на выходе отображают двумя выходными сигналами $q(s)$ и $\bar{q}(s)$, причем $q(s_1) = 0$, $\bar{q}(s_1) = 1$, и наоборот $q(s_2) = 1$, $\bar{q}(s_2) = 0$. В этом случае говорят, что состояние триггера отображается в парафазном коде.

Можно показать, что триггер является простейшим автоматом Мура (запоминающим элементом) и при этом обладает полной системой переходов и выходов.

Поэтому любой функционально полный набор логических элементов, дополненный триггером, является функционально полным для синтеза ЦА с памятью.

Триггеры делятся на два класса: синхронные и асинхронные. Синхронный триггер имеет синхронизирующий вход С, разрешающий переключение триггера только в определенные моменты времени. При $C = 0$ триггер сохраняет свое состояние независимо от значений сигналов на его входах. При $C = 1$ триггер переключается в состояние, соответствующее входному сигналу и текущему его состоянию.

Асинхронный же триггер переключается лишь в момент, когда изменяется значение сигналов на его входе.

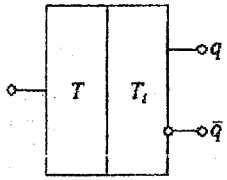
Триггер может иметь несколько входных сигналов:

1. R – установка триггера в 0 (сброс).
2. S – установка триггера в 1.
3. T – переключение триггера в противоположное состояние (0 в 1 или 1 в 0).
4. J – установка в 1 или противоположное состояние.
5. K – установка в 0 или противоположное состояние.
6. $Д$ – установка в состояние, соответствующее значению $Д$ (при $Д = 0$ $S = 0$, а при $Д = 1$ $S = 1$).
7. V – разрешение переключения в состояние, зависящее от значений других входных сигналов.

Соответственно триггеры могут быть оснащены различными наборами входов, в зависимости от которых выделяют следующие основные типы триггеров: T, RS, JK, $Д$ $ДV$.

Простейшим из них является асинхронный – триггер с одним счетным входом (от англ. to toggle – переключать), который меняет свое состояние на противоположное при поступлении на его вход 1 и сохраняет состояние неизменным при поступлении на его вход сигнала 0.

На схемах – триггер с порядковым номером i изображается так:



Канонический метод структурного синтеза ЦА.

Цель структурного синтеза ЦА – построение заданного автомата из элементов функционально полной системы.

Исходными данными для синтеза являются, словесное описание синтезируемого автомата, таблица переходов элементов памяти (элементарного автомата Мура) и некоторый логический базис из комбинационных элементов.

Метод основан на общем приеме разделения автомата на две части: комбинационную схему (КС) и память автомата (ПА).

Далее по количеству внутренних состояний автомата и основанию структурного алфавита определяется число элементов памяти. Структурный синтез автомата сводится к синтезу его комбинационной схемы.

Процесс структурного синтеза делят на 4 этапа:

1. Переход от словесного описания автомата к его формализованному описанию в виде таблиц переходов и выходов (или графа). Далее выбирается структурный алфавит и на его основе кодировка таблиц. Мы будем пользоваться двоичным структурным алфавитом и соответствующие таблицы сразу строить в кодированном виде.

Далее осуществляется переход от абстрактного автомата (рис. 4.1.а) к схеме структурного автомата (рис. 4.1.б), у которого определены множество входов $V = \{v_1, v_2, \dots, v_\tau\}$, множество элементов памяти $Q = \{q_1, q_2, \dots, q_n\}$ и множество выходов $Z = \{z_1, z_2, \dots, z_w\}$.

Для этого по количеству входных сигналов k, внутренних состояний C и выходных сигналов m (сведения выбираются из таблиц переходов и выходов) по основанию алфавита «b» (обычно $b = 2$) рассчитывается число входов (τ), элементов памяти (n) и выходов (w): $\tau \geq log_2 k$, $n \geq log_2 C$, $w \geq log_2 m$.

Выбираются целые числа с округлением в большую сторону (при наличии дробной части).

Для правильного функционирования схемы из двух частей должно выполняться следующее непременное условие: сигналы на входе памяти должны непосредственно участвовать в образовании ее выходных сигналов. Иначе выходные сигналы памяти через

КС практически сразу же, в том же автоматном такте, подавались бы снова на вход памяти, образуя цепь обратной связи и вызывая нежелательный результат (искажение информации, автоколебания системы и т.п.). Как следствие указанного условия – необходимость использования в качестве элементов памяти (ЭП) только автоматов Мура. Кроме того, количество сигналов на выходе памяти всегда должно равняться числу ее внутренних состояний. Поэтому условно внутренние состояния и выходные сигналы памяти можно отождествлять.

2. Определяется число входов у элементов памяти (число выходов ЭП обычно равно $\tau \geq log_2 k_{ЭП}$, где $k_{ЭП}$ – количество входных сигналов, необходимых для записи информации в ЭП). Далее подсчитывается количество связей, поступающих от КС на память автомата: $P = \tau_{ЭП} \cdot n$ (количество связей из памяти на КС = n). На этом же этапе происходит оформление КС как «черного ящика» и выявляется, что КС имеет общее число входов $l_{ОБЩ} = \tau + n$, а общее число выходов $w_{ОБЩ} = w + P$.

3. Строится таблица возбуждения памяти автомата на основе таблиц переходов автомата и ЭП.

По сути, таблица возбуждения памяти является частью таблицы истинности КС. Она задает соответствие между входными сигналами (словами) автомата и его памяти. Таблица возбуждения памяти автомата имеет те же входы, что и таблица переходов автомата, но в ее клетках вместо слов внутреннего состояния автомата проставляются слова из входных сигналов, приходящих на ЭП (сигналы возбуждения памяти) и обеспечивающих переход автомата в следующее состояние из предыдущего. Если обозначить множество сигналов возбуждения памяти через $H = \{h_1, h_2, \ldots, h_p\}$, то в общем виде можно записать $h_j = \varphi(v_1, v_2, \ldots, v_\tau)$, где $j = 1, 2, \ldots, p$, φ – символ некоторой булевой функции.

Пусть в одной из клеток таблицы переходов автомата указано внутреннее состояние $S_i = 101$, причем по выходным данным таблицы видно, что в это состояние автомат переходит из состояния $S_{i-1} = 011$. В этом случае память автомата состоит из 3-х двоичных ЭП, и, очевидно, что первый $ЭП_1$ (триггер младшего разряда) переходит из состояния 1 снова в состояние1, $ЭП_2$ из 1 в 0 и $ЭП_3$ из 0 в 1. Предположим, что в этом случае каждый ЭП имеет один вход. Тогда по таблице переходов ЭП находим те значения входного сигнала h_j, которые обеспечивают перевод $ЭП_1$ из 1 в 1 ($h_1 = 0$), $ЭП_2$ из 1 в 0 ($h_2 = 1$) и $ЭП_3$ из 0 в 1 ($h_3 = 1$). Найденные значения сигналов k (слово 110) записываются в соответствующие клетки таблицы возбуждения памяти.

4. На 4-ом этапе осуществляется комбинационный синтез автомата. Его можно выполнить, пользуясь двумя таблицами, описывающими работу КС: таблицей выходов автомата и таблицей возбуждения памяти. Эти две таблицы часто сводят в одну таблицу истинности КС.

Синтезируем схему двоичного суммирующего счетчика накапливающего типа, имеющего 8 внутренних состояний, в базисе НЕ, И, ИЛИ, Т–триггер по описанному каноническому методу.

1. Структурный алфавит (двоичный) в данном случае определяется прямо из задания, т.к. требуется синтезировать двоичный суммирующий счетчик. Т-триггер избран в качестве базового элемента, т.к. он имеет счетный вход.

Под суммирующим счетчиком понимают устройство, с выходов которого можно снять двоичную кодовую комбинацию, представляющую собой число импульсов, поступивших на вход счетчика после того, как он был приведен в исходное нулевое состояние (сброс). Счетчик является из постановки задачи автоматом Мура с одним входом и достаточно

составить только его таблицу переходов для чего нужно знать число элементов памяти $n \geq log_2 8 = 3$.

В таблице переходов автомата будут 4 аргумента: 1-ый – выходной сигнал T_3 третьего триггера (старший разряд) q_3^* в такте $(t-1)$; 2-ой – выходной сигнал T_2 второго триггера) q_2^* в такте $(t-1)$; 3-й – выходной сигнал T_1 первого триггера (младший разряд) q_1^* в такте $(t-1)$; 4-й аргумент – входной сигнал на единственном входе автомата v в такте $(t-1)$. Нетрудно заметить, что двоичное число $Q = q_3^* q_2^* q_1^*$ есть не что иное, как содержимое (внутреннее состояние) счетчика в такте $(t-1)$.

Количество функций в таблице будет равно трем: f_3 – выходной сигнал (состояние) третьего триггера (q_3 в T_3 старший разряд счетчика) в такте t, f_2 – выходной сигнал q_2 второго триггера T_2 в такте t, f_1 – выходной сигнал q_1 первого триггера T_1 в такте t, т.е. двоичное число $Q = q_3 q_2 q_1$ – содержимое счетчика на следующем такте t, причем $Q = Q^*$, если $v = 0$, и $Q = Q^* + 1$, если $v = 1$. Если в такте $(t-1)$ на счетчике было максимальное число $Q_{max} = 111_2$ и входной сигнал $v = 1$, то в такте t счетчик обнуляется. Кроме табл. 4.1 переходов на 1 этапе составляется упрощенная схема счетчика в виде структурного автомата (рис. 4.1.б), где $\tau = 1$ и $v_1 = v$.

Таблица 4.1

Состояние T_3 в $(t-1)$	q_3^*	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Состояние T_2 в $(t-1)$	q_2^*	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
Состояние T_1 в $(t-1)$	q_1^*	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Вход автомат. в $v(t-1)$	v	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
f_3 – сост. T_3 в t	q_3	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0
f_2 – сост. T_2 в t	q_2	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0
f_1 – сост. T_1 в t	q_1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0

2. На втором этапе определяем количество выходных связей комбинационной схемы, идущих на память автомата: $P = C_{\text{эп}} = 3$. Количество входных связей, идущих от памяти, $n = 3$. Следовательно, КС имеет общее число входов $1 + 3 = 4$, а общее число выходов $w_{\text{общ}} = w + p = 3$, поскольку счетчик является автоматом Мура ($w = 0$).

3. На третьем этапе строим табл. 4.2 возбуждения памяти счетчика. Ее аргументы и форма совпадают с табл. 4.1 и заполнять ее мы будем, пользуясь данными таблицы 4.1 и таблицы переходов элемента памяти T следующим образом. Рассмотрим для примера третий столбец табл. 4.1 (номер столбца определяется двоичным числом, состоящим из значений истинности аргументов по вертикали сверху вниз). Замечаем, что T_3 перешел из состояния $q_3^* = 0$ в состояние $q_3 = 0$. Далее выясняем, какой сигнал переводит T из нулевого состояния снова в нулевое.

Для этого на триггер T_3 нужно подать сигнал возбуждения $h_3 = 0$. Записываем 0 в третьем столбце таблицы возбуждения на строке, соответствующей функции h_3. Далее анализируем поведение триггера T_2. Замечаем, что он перешел из состояния $q_2^* = 0$ в состояние $q_2 = 1$. Для такого перехода на T–триггер требуется подать сигнал возбуждения $h_2 = 1$. Записываем единицу в третьем столбце таблицы возбуждения на строке соответствующей функции h_2. Аналогично для T_1 получаем, что $h_1 = 1$. По такой же методике заполняется значения истинности функций возбуждения h_3, h_2, h_1 во всех 16

столбцах таблицы 4.2. Чтобы облегчить эту процедуру можно заметить, что для перевода T–триггера в противоположное состояние (из $0 \to 1$ или из $1 \to 0$) требуется сигнал возбуждения равный 1 и при его неизменном состоянии сигнал – 0. Воспользовавшись тем, что при входном сигнале $v = 0$, слова $Q^* = q_3^* q_2^* q_1^*$ и $Q = q_3 q_2 q_1$ равны между собой, во всех столбцах таблицы возбуждения памяти, имеющих четные номера (включая нулевой столбец), проставляем нули ($H = h_3 h_2 h_1 = 000$).

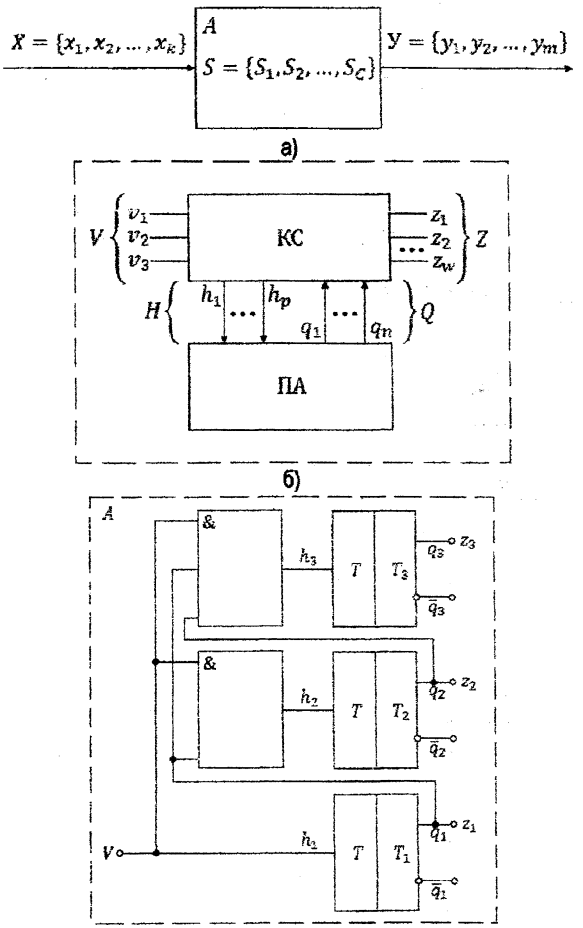


Рис. 4.1 Цифровой автомат (счетчик): а) абстрактный; б) структурный; в) функциональная схема

Остальные столбцы заполняем аналогично столбцу 3 (0011).

Таблица 4.2

T_3 в т. $(t-1)$	q_3^*	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
T_2 в т. $(t-1)$	q_2^*	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
T_1 в т. $(t-1)$	q_1^*	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Вх. в т. $(t-1)$	v	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Ф-я возб. T_3	h_3	0	0	0	0	0	0	0	*1*	0	0	0	0	0	0	0	*1*
Ф-я возб. T_2	h_2	0	0	0	***1***	0	0	0	***1***	0	0	0	***1***	0	0	0	***1***
Ф-я возб. T_1	h_1	0	**1**	0	**1**	0	**1**	0	**1**	0	**1**	0	**1**	0	**1**	0	**1**

4. На 4 этапе строится комбинационная схема синтезируемого счетчика, т.к. он является автоматом Мура и описывается только одной схемой возбуждения памяти. Следовательно, табл. 4.2 является таблицей истинности синтезируемой КС.

Построим для каждой из функций $h_3(q_3^*, q_2^*, q_1^*, v)$, $h_2(q_3^*, q_2^*, q_1^*, v)$ и $h_1(q_3^*, q_2^*, q_1^*, v)$ диаграмму Вейча-Карно.

h_3

$q_3^*q_2^*$ \ q_1^*v	00	01	11	10
10				
11			1	
01			1	
00				

h_2

$q_3^*q_2^*$ \ q_1^*v	00	01	11	10
10			1	
11			1	
01			1	
00			1	

h_1

$q_3^*q_2^*$ \ q_1^*v	00	01	11	10
10		1	1	
11		1	1	
01		1	1	
00		1	1	

$$\begin{cases} h_3 = q_2^* q_1^* v, \\ h_2 = q_1^* v, \\ h_1 = v. \end{cases}$$

В соответствии с системой этих выражений (КС) и информации о памяти построим функциональную схему суммирующего счетчика как ЦА (рис. 4.1.в).

Такой счетчик характеризуется высоким быстродействием. Время его срабатывания примерно равно сумме времен переходных процессов в одном конъюнкторе и в одном триггере, так как все операции выполняются параллельно, но зато требуется больше оборудования.

При уменьшении быстродействия можно сократить количество оборудования за счет перехода к схеме последовательного действия. Например, положив $h_3 = q_2^* h_2$, $h_2 = q_1^* h_1$, $h_1 = v$, получим соответствующею функциональную схему.

Аналогичным образом можно построить функциональную схему для любого конечного автомата A, заданного таблично. Чтобы избежать построения больших таблиц, можно перейти к графовой форме задания автомата и синтезировать схемы для функций возбуждения триггеров только для переходов состояний автоматов, задаваемых конструктором. Проиллюстрируем синтез автомата для этого случая.

Структурный синтез цифрового автомата с памятью, заданного графом

Автомат A задается в виде графа, на котором по стрелкам показаны двоичные входные сигналы в виде переменных x_i ($\bar{x}_i$) или их произведений, или безусловных переходов (1). Состоянием автомата соответствуют кружки с буквой a_j ($j = 0, 1, \ldots, (c-1)$). Если состояния A отражать через состояния триггеров, то на первом шаге синтеза каждому состоянию надо поставить в однозначное соответствие набор состояний триггеров. Минимальное количество триггеров $M \geq log_2 C$, причем при наличии дробной части у логарифма M получается как ближайшее целое число после округления с избытком (например, $M = 4$ при $log_2 C = 3.1$). Обычно также задается тип триггеров и полный набор логических элементов И, ИЛИ, НЕ (И-НЕ или ИЛИ-НЕ). После определения количества триггеров выполняется кодирование состояний автоматов. При целом $M = log_2 C$ все наборы триггеров будут задействованы, и каждому из наборов будет соответствовать только одно состояние. При округлении $log_2 C$ до целого числа могут появиться свободные комбинации, которые допускается использовать как эквивалентные для некоторых состояний. (Для упрощения изложения будем рассматривать случай, когда каждому состоянию соответствует только одна комбинация состояний триггеров).

Так как переход A в другое состояние обусловлен появлением какой-то комбинации входных сигналов или сигналом безусловного перехода, то для осуществления такого перехода надо переключить некоторые из триггеров T_j. Чтобы переключить конкретный триггер надо на него послать сигнал возбуждения h_j.

Синтез автомата предполагает построение функций h_j для каждого триггера T_j. Каждая функция h_j определяется как сумма логических произведений, из компонент соответствующих сигналов и предшествующих каждому переходу состояний все триггеров. Записав соответствующие логические функции, отражающие значения сигналов h_j, можно получить и схему для реализации автомата A.

Чтобы ее упростить, предварительно любыми известными методами выполняется минимизация, а также изучается вопрос о выделении общих элементов цепей для не-

скольких h_j, если это возможно. Покажем на примере заданного графом автомата A (рис. 4.2), как синтезировать его схему.

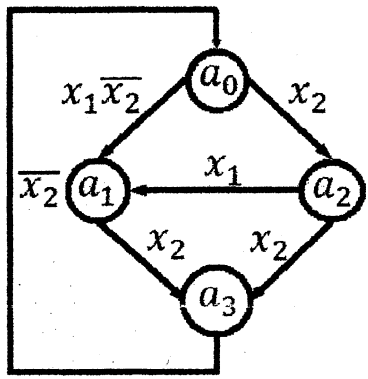


Рис.4.2. Граф автомата A

На первом шаге определяем необходимое минимальное количество триггеров T_i. Так как состояний четыре, то $M = log_2 4 = 2$ (округлений не требуется). На втором шаге в порядке следования номеров триггеров (T_2, T_1) проводим одно из возможных кодирований состояний: $a_0 \to 00$; $a_1 \to 01$; $a_2 \to 11$; $a_3 \to 10$. На третьем шаге, используя информацию о кодировках состояний, отраженных на графе, получаем все компоненты функций h_j ($j = \{1,2\}$, два триггера). Их удобно отражать в виде следующей таблицы 4.3.

Таблица 4.3. Компоненты функций h_j

$h \setminus a$	Переходы состояний					
	$a_0 \to a_1$ $00 \to 01$	$a_0 \to a_2$ $00 \to 11$	$a_1 \to a_3$ $01 \to 10$	$a_2 \to a_1$ $11 \to 01$	$a_2 \to a_3$ $11 \to 10$	$a_3 \to a_0$ $10 \to 00$
h_2	0	$x_2\overline{q}_1^*\overline{q}_2^*$	$x_2q_1^*\overline{q}_2^*$	$x_1q_1^*q_2^*$	0	$\bar{x}_2\overline{q}_1^*q_2^*$
h_1	$x_1\bar{x}_2\overline{q}_1^*\overline{q}_2^*$	$x_2\overline{q}_1^*\overline{q}_2^*$	$x_2q_1^*\overline{q}_2^*$	0	$x_2q_1^*q_2^*$	0

Если данный триггер в рассматриваемом переходе состояния a_k в a_j не нужно переключать, то компоненты функции h_j записаны как 0, а в остальных случаях записано логическое условие переключения соответствующего триггера в рассматриваемом переходе состояния a_k в a_j.

На четвертом шаге записываем все функции h_j и проводим минимизацию каждой из них отдельно.

$h_2 = \underline{x_2\overline{q}_1^*\overline{q}_2^*} + \underline{x_2q_1^*\overline{q}_2^*} + x_1q_1^*q_2^* + \bar{x}_2\overline{q}_1^*q_2^* = x_2\overline{q}_2^* + q_2^*(x_1q_1^* + \bar{x}_2\overline{q}_1^*)$,

$h_1 = x_1\bar{x}_2\overline{q}_1^*\overline{q}_2^* + x_2\overline{q}_1^*\overline{q}_2^* + \underline{x_2q_1^*\overline{q}_2^*} + \underline{x_2q_1^*q_2^*} = \overline{q}_1^*\overline{q}_2^*(x_1\bar{x}_2 + x_2) + x_2q_1^*$.

Общую часть для данных функций выделять нецелесообразно.

На пятом шаге строится схема автомата A. Ее рекомендуется располагать следующим образом: слева располагаются инверторы входных элементов, за ними шины входных элементов и шины для входов от триггеров, а крайними справа отображаются триггеры. Между шинами и триггерами отображаются реализации схем для функций h_j.

Изучив функции h_2 и h_1, видим, что инвертор будет один (для x_2), x_1 будет подаваться прямо на свою шину, а также будет четыре шины от триггеров. В итоге, после отображения функций h_2 и h_1, получим функциональную схему автомата A (рис. 4.3).

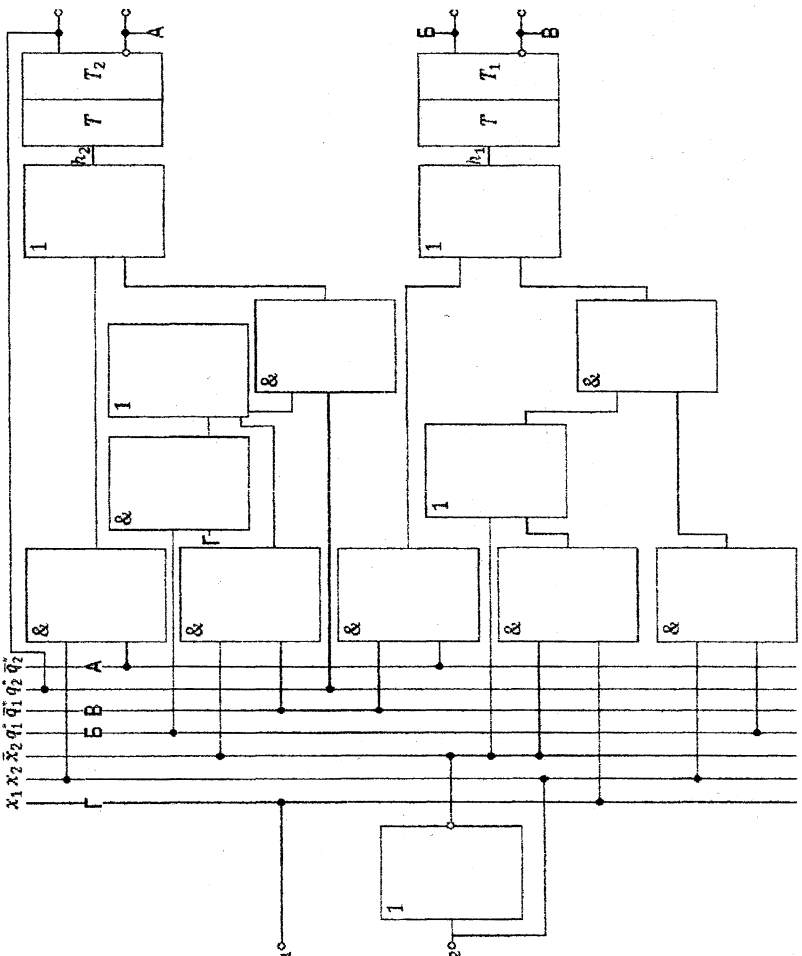


Рис. 4.3. Функциональная схема автомата A

Проверка правильности схемы выполняется по графу и схеме. Для этой цели выбирается состояние a_i (T_2^*, T_1^*) и проверяется, перейдет ли оно в состояние a_k (T_2, T_1) при заданных входах.

Задание (№№ 5, 6). Синтез автоматов с памятью

Оно включает тщательное изучение теоретических положений и правил реализации функциональных схем автомата. Исходная структура задания представляет из себя граф с неполностью определенными значениями входных сигналов (x_i) и представлена на рисунке 4.4.

Конкретный вариант задания получается, исходя из записи последовательности букв в фамилии, имени и отчестве студента. Для этой цели в порядке нарастания номеров исходящих состояний рассматриваются все стрелки из них в порядке номеров входящих состояний и по порядку номеров индексов i для x_i определяется над какими из x_i поставить отрицание. Отрицание ставится над теми x_i, где в последовательности букв в Ф.И.О. встречается гласная буква. Например, Иванов Иван Петрович: для a_0: $\bar{x}_0$(И), x_1(В) · $\bar{x}_2$(А); для a_1: x_1(Н) · $\bar{x}_3$(О) и т.д.

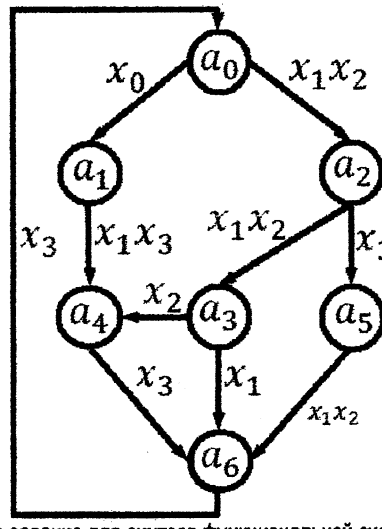


Рис. 4.4. Граф-задание для синтеза функциональной схемы автомата A

Результатом всей работы является функциональная схема автомата A с проверкой правильности переходов состояний (для любых 3-х по выбору студента).

РАЗДЕЛ 5. РАЗРАБОТКА МИКРОПРОГРАММ ДЛЯ ВЫПОЛНЕНИЯ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ (ЗАДАНИЕ №7)

Для обработки информации используются различные операционные устройства. Функцией операционного устройства является выполнение заданного множества операций $F = \{f_1, ..., f_g\}$ над входными словами $D = \{d_1, ..., d_k\}$ с целью вычисления слов $R = \{r_1, ..., r_Q\}$, представляющих результаты операций $R = f_g(D)$, $g = \{1, ..., G\}$. Функциональная и структурная организация операционных устройств, определяющая порядок их функционирования и структуру, базируется на принципе микропрограммного управления, который состоит в следующем.

1. Любая операция f_g, реализуемая устройством, рассматривается как сложное действие, которое разделяется на последовательность элементарных действий (микроопераций) над словами информации.

2. Для управления порядком следования микроопераций используются логические условия, которые в зависимости от значений слов, преобразуемых микрооперациями, принимают значения «истина» и «ложь», (1 или 0).

3. Процесс выполнения операций в устройстве описывается в форме алгоритма, представляемого в терминах микроопераций и логических условий и называемого микропрограммой. Микропрограмма определяет порядок проверки значений логических условий и следования микроопераций, необходимый для получения требуемых результатов.

4. Микропрограмма используется как форма представления функций устройства, на основе которой определяется структура и порядок функционирования во времени.

Сказанное можно рассматривать как содержательное описание принципа микропрограммного управления, из которого следует, что структура и порядок функционирования операционных устройств предопределяются алгоритмом выполнения операций F.

Концепция операционного и управляющего автоматов

В функциональном и структурном отношении операционное устройство разделяется на две части: операционный и управляющий автоматы (рис. 5.1). Операционный автомат (ОА) служит для хранения слов информации, выполнения набора микроопераций и вычисления значений логических условий, т.е. операционный автомат является структурой, организованной для выполнения действий над информацией. Микрооперации, реализуемые операционным автоматом, инициируется множеством управляющих сигналов $Y = \{y_1, ..., y_m\}$ с каждым из которых отождествляется определенная микрооперация. Значение логических условий, вычисляемые в операционном автомате, отображаются множеством осведомительных сигналов $X = \{x_1, ..., x_L\}$ каждый из которых отождествляется с определенным логическим условием. Управляющий автомат (УА) генерирует последовательность управляющих сигналов, предписанную микропрограммой и соответствующую значениям логических условий. Иначе говоря, управляющий автомат задает порядок выполнения действий в операционном автомате, вытекающий из алгоритма выполнения операций. Наименование операции, которую необходимо выполнить в устройстве, определяется кодом g операции. По отношению к управляющему автомату $g_1, ..., g_n$ (коды операций) и осведомительные сигналы $x_1, ..., x_L$, формируемые в операционном автомате, играют одинаковую роль: они влияют на порядок выработки управляющих сигналов Y. Поэтому сигналы $g_1, ..., g_n$ и $x_1, ..., x_L$ относятся к одному классу – к классу осведомительных сигналов, поступающих на вход управляющего автомата.

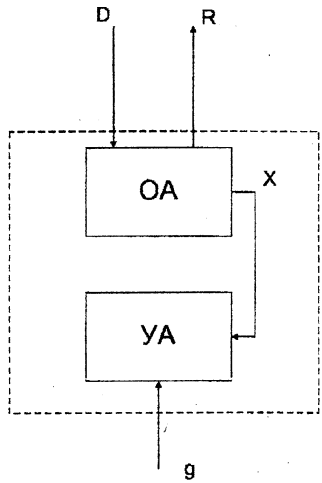


Рис. 5.1. Структура операционного устройства

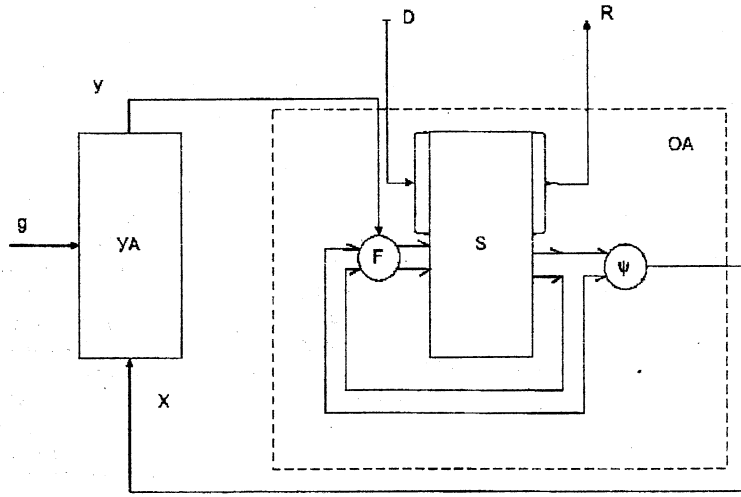


Рис. 5.2. Структурная организация операционного автомата

Операционный автомат, реализуя действия над словами информации, является исполнительной частью устройства, работой которого управляет управляющий автомат, генерирующий необходимые последовательности управляющих символов.

На данном этапе рассмотрения вопроса операционный и управляющий автоматы могут быть определены своими функциями – перечнем выполняемых ими действий, исходя из которых в дальнейшем будет определена структура автоматов.

Функция операционного автомата определяется следующей совокупностью сведений:

1. Множество входных слов $D = \{d_1, \ldots, d_k\}$, вводимых в автомат в качестве операндов.
2. Множеством выходных слов $R = \{r_1, \ldots, r_Q\}$, представляющих результаты операций.
3. Множеством внутренних слов $S = \{S_1, \ldots, S_N\}$, используемых для представления информации в процессе выполнения операций. В дальнейшем будем предполагать, что входные и выходные слова совпадают определенными внутренними словами, т.е. $D \subseteq S$ и $R \subseteq S$.
4. Множеством логических условий $X = \{x_i\}, i = 1 \ldots L$, где $X_i = \varphi_i(S)$, $\varphi_i(S)$– булева функция.

Таким образом, функция операционного автомата задана, если определены множества D, R, S, X, Y. Заметим, что время не является аргументом функции операционного автомата. Функция устанавливает список действий – микроопераций и логических условий, которые может выполнять автомат, но никак не определяет порядок следования этих действий во времени. Иначе говоря, функция операционного автомата характеризует средства, которые могут быть использованы для вычислений, но не сам вычислительный процесс. Порядок выполнения действий во времени определяется в форме функций управляющего автомата.

Функция управляющего автомата представляется как операторная схема алгоритма (микропрограммы), функциональными операторами которой являются символы $y_1, \ldots, y_m$, отождествляемые с микрооперациями, и в качестве логических условий используются булевы переменные $x_1, \ldots, x_L$. Операторная схема алгоритма наиболее часто представляется в виде граф-схемы или логической схемы алгоритма. Каждая из этих форм определяет вычислительный процесс в последовательном аспекте – устанавливает порядок проверки логических условий $x_1, \ldots, x_L$ и порядок следования микроопераций.

Постановка задачи проектирования УА

Структурная реализация микроопераций и логических условий может более детально представляться так (рис 5.2). Операционный автомат ОА разделяется на три части: память S; комбинационную схему F, реализующую микрооперации; комбинационную схему φ, вычисляющую значения логических условий. Память S обеспечивает хранение слов $S_1, \ldots, S_N$, которые представляют значения операндов D, промежуточные значения и конечные результаты R. Для выполнения микрооперации $Y = \{y_m\}$ служит комбинационная схема А. Управляющие сигналы Y, формируемые управляющим автоматом УА, инициируют выполнение необходимых микроопераций.

Задача проектирования операционного устройства ставится следующим образом. Заданы функции операционного устройства, определяемые множествами D входных и R выходных слов и множеством операций F, а также требования к его быстродействию. Требуется синтезировать схему устройства, обеспечивающую реализацию заданных функций с заданным быстродействием и являющуюся минимальной в смысле количества используемого оборудования.

Чтобы синтезировать схемы операционного устройства, необходимо принять некоторый способ выполнения операций в устройстве и описать его в форме микропрограммы. Микропрограмма, представляющая функцию операционного устройства безотносительно к средствам, называется функциональной микропрограммой. Она фиксирует в себе алгоритм выполнения операций, рекомендуемый проектировщиком и используется как исходная форма представления функций устройства, на основе которой синтезируется структура.

Для записи функциональных микропрограмм необходима какая-либо алгоритмическая система – язык функционального микропрограммирования. Средства языка должны обеспечивать описание алгоритмов выполнения операций – слов, микроопераций, логических условий и порядка их выполнения – с такой степенью детализации, которая обеспечит синтез структурного устройства.

Традиционно используют следующие микрооперации:

1. Установка осуществляет присваивание слову значения константы, например, $A := 0$, $B := 1111$, $C(0) := 1$, $C(1:17) = 127_{10}$ (указание номера разряда или их нескольких номеров говорит о тех разрядах, с которыми выполняются операции, например, $C(0)$ – с нулевым разрядом, $A := 0$ – со всеми разрядами, $A(1:15)$ – с разрядами с 1 по 15).

2. Инвертирование обеспечивает изменение значения на инверсное, например, $A := \bar{A}$, $C(0) := \bar{C}(0)$.

3. Передача является присваиванием слову значения другого слова, в том числе инверсии или составного слова, например, $A := B$, $A(0) := B(0)$, $C := 11.A(1:15)$ – две первые единицы предшествуют разрядам с 1 по 15 числа A.

4. Сдвиг изменяет положения разрядов слова по отношению к начальному путем перемещения каждого разряда слова на k позиций влево $L_k(A(1:15))$ или вправо $R_k(A(1:12))$. Любая микрооперация сдвига может быть представлена в форме оператора присваивания.

5. Счет обеспечивает изменение значения слова на единицу, например, $A := A + 1$, $B := B - 1$, $C := A + 1$, $C(1:4) = C(1:4) + 1$.

6. Сложение служит для присваивания слову значения суммы слагаемых, например, $C := C + A$.

Некоторые из используемых в программе микроопераций могут выполняться параллельно во времени, в то время как другие – только последовательно. Свойство совокупности микроопераций, гарантирующее возможность их параллельного выполнения, называется совместимостью. Микрооперации, не обладающие указанными свойствами, называются несовместимыми. Совместимость микроопераций обусловлена, во-первых, содержанием операторов, представляющих микрооперации, - так называемая функциональная совместимость – и, во-вторых, структурой операционного устройства, допускающей или исключающей возможность параллельного выполнения нескольких микроопераций, - так называемая структурная совместимость.

Содержательный граф микропрограммы строится с использованием вершин четырех типов (рис. 5.3) и дуг, связывающих вершины. Начальная вершина отмечает начало алгоритма и имеет единственный выход, из которого исходит дуга к первой выполняемой вершине графа. Для обозначения операторов алгоритма используются вершины двух типов: функциональные и условные. Функциональная (основная) вершина определяет действие – совокупность функционально совместимых микроопераций, выполняемых параллельно. Микрооперации в вершине представляются в виде операторов присваивания. В функциональную вершину может входить любое (не меньше одной) число дуг и из вершины выходит только одна дуга. Условная вершина используется для разветвления вычислительного процесса в одном из двух возможных направлений, выбор которого определяется текущим значением логического условия, указанного в вершине. Если

условие имеет значение 0, вычислительный процесс развивается по дуге, отмеченной символом 0, в противном случае – по дуге, отмеченной символом 1. В условную вершину может входить любое число дуг, но выходят всегда две дуги. Конечная вершина отмечает конец программы. В конечную вершину может входить любое число дуг.

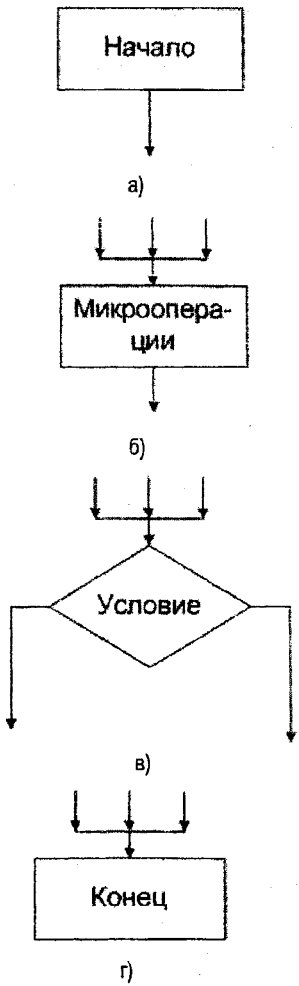


Рис. 5.3. Вершины графа микропрограммы: а) начальная; б) функциональная (операторная); в) условная; г) конечная

Граф, представляющий микропрограмму, считается корректным, если выполняются следующие условия:

1. Граф содержит только одну начальную и только одну конечную вершины.

2. В любую вершину, кроме начальной, должна входить хотя бы одна дуга, исходящая из другой вершины графа.

3. Из каждого выхода любой вершины, кроме конечной, должна выходить одна дуга, ведущая к некоторой вершине графа.

4. При всевозможных значениях слов должен существовать путь из начальной вершины в конечную.

Условия 2 и 3 требуют чтобы каждая вершина имела хотя бы одну предшествующую и одну последующую вершины, причем функциональная вершина может иметь только одну последующую вершину, а условная – точно две последующие вершины, соответствующие выходам 0 и 1 условной вершины.

Примеры функциональных программ. Пусть операционное устройство предназначено для выполнения только операции умножения $F = \{X\}$ над двоичными числами, с фиксированной запятой, представляемыми словами следующего формата:

0	15
±	Цифровые разряды

Результат операции (произведение) будем представлять в таком же формате.

Сосредоточим внимание лишь на процедуре построения функциональных микропрограмм и поэтому будем считать, что можно использовать любые алгоритмы выполнения операций независимо от затрат времени на их реализацию.

Для умножения будем использовать алгоритм вычисления произведения, начиная от младших разрядов множителя, который состоит из следующих действий:

1. Произведение полагается равным нулю.

2. Если младший разряд множителя равен 1, произведение увеличивается на значение модуля множимого.

3. Произведение и множитель сдвигаются на один разряд вправо, в результате чего в младший разряд множителя вводится очередная цифра множителя.

4. Действия 2, 3 повторяются для обработки всех 15 цифр множителя.

5. Если знаки сомножителей одинаковы, произведению присваивается знак плюс; если знаки разные, произведению присваивается знак минус.

Определим слова, которые, необходимы для выполнения умножения по описанному алгоритму. Операнды – множимое и множитель будем представлять словами $A(0:15)$ и $B(0:15)$. Значения присваиваются словам A и B вне алгоритма до начала операции, поэтому эти слова должны иметь тип i (входные). Затем слова A и B используются в процессе выполнения алгоритма, что отмечается присваиванием им типа L (внутренние). Следовательно, слова A, B должны иметь тип i_L. Значение произведения будем представлять словом $C(0:15)$. Слово $C(0:15)$ должно иметь тип L_0, поскольку оно обрабатывается в микропрограмме и по окончании операции представляет значение результата, используемое вне программы. Поскольку все цифры множителя обрабатыва-

ются одинаково, целесообразно организовать цикл. Для определения момента завершения цикла необходим счетчик числа повторения цикла. Перед началом операции счетчик устанавливается в состояние 15 и после обработки каждой цифры его состояние должно уменьшаться на единицу. Переход счетчика в состояние 0 свидетельствует об окончании цикла. Чтобы закодировать значение 15, необходимо 4 двоичных разряда, т.е. счетчик должен иметь формат $СЧ$ (1:4). Значение счетчика используется только в микропрограмме, поэтому слово $СЧ$ должно иметь тип L (внутреннее).

Функциональная микропрограмма, описывающая алгоритм операции умножения содержательно представлена графом микропрограммы (рис 5.4). Выполнение операции начинается с присваивания произведению C нулевого значения и установки счетчика $СЧ$ в начальное состояние 15. Эти микрооперации являются совместимыми, что позволяет объединить их в один функциональный оператор. Вычисление произведения начинается с анализа значения младшей цифры множителя. Если $B(15) = 1$, произведение C увеличивается на значение модуля множимого, представляемого в разрядах 1-15 слова A.

В следующем операторе производится сдвиг цифровых разрядов 1-15 множителя B с целью выделения очередной цифры множителя, сдвиг произведения C и уменьшение значения счетчика на 1. Чтобы определить необходимость повторения цикла выработки произведения и момент окончания формирования значения произведения, проверяется условие $СЧ = 0$. До окончания обработки 15 цифр множителя счетчик находится в состоянии, отличном от 0. Поэтому условие имеет значение 0 («ложь») и в микропрограмме выполняется переход на обработку следующей цифры. Когда $СЧ = 0$, выработка произведения заканчивается. Значение логического условия $A(0) \oplus B(0)$ определяет знак произведения. Если $A(0) \oplus B(0) = 0$, знаки сомножителей одинаковы и в знаковом разряде C(0) произведения сохраняется значение 0, соответствующее знаку плюс. Если $A(0) \oplus B(0) = 1$, в знаковый разряд $C(0)$ произведения заносится значение 1, соответствующее знаку минус.

Содержательный граф микропрограммы обычно используется для синтеза управляющего и операционного автоматов.

Покажем, как строится из нее граф УА Мура.

Если каждый из прямоугольников на граф-схеме (рис. 5.4) пометить своей буквой (начало и конец одинаковой), например, $a_0, a_1, \ldots, a_k$, то тогда, обозначив этими буквами вершины графа автомата, и, проведя стрелки, показывающие переход рассматриваемого состояния в следующее под воздействием соответствующего значения X_i или $\overline{X}_i$ одиночных или группы входов (в случае безусловного перехода $X_i = 1$), получим искомый граф. Групповая комбинация входных сигналов возникает при проверке цепочки из нескольких логических условий.

Начальную вершину (состояние a_0) будем интерпретировать как состояние, в котором автомат находится в покое и оно переходит в само себя ($\overline{X}_0 = 0$) до пуска автомата ($X_0 = 1$). В итоге получим граф-схему (рис. 5.5) УА Мура, которая может служить основой для синтеза его функциональной схемы.

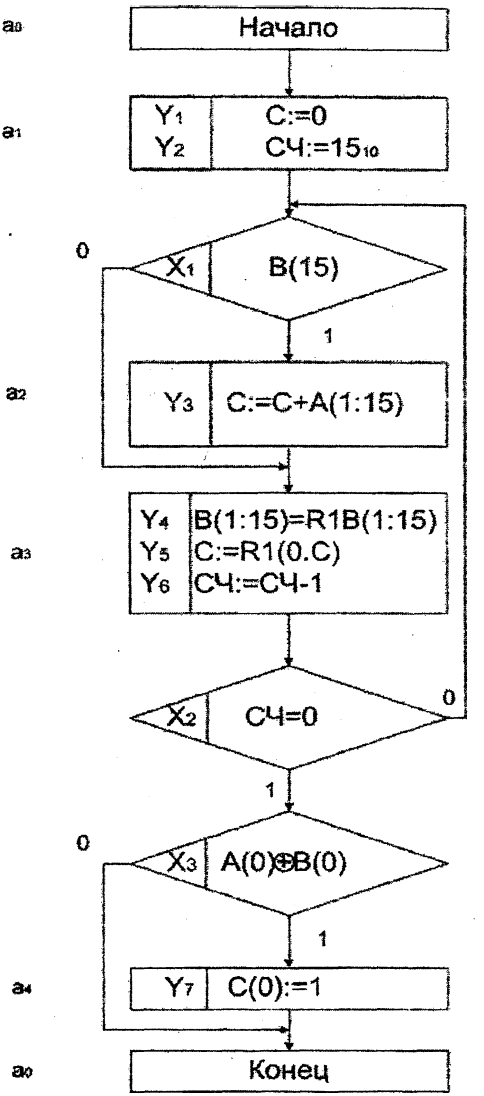


Рис. 5.4. Содержательный граф микропрограммы умножения

Задание №7

Заключается в построении микропрограммы и на ее основе граф-схемы автомата Мура для вычисления величины $D = (L \text{ или } R)_k[A^i(0{:}p) \times B(0{:}p) \pm C(0{:}p)]$.

Чтобы вычислять D, необходимо конкретизировать набор $L, R, k, i, p, +, -$, который определяется по последовательности букв фамилии и имени студента. Форматы чисел A, B, C выбираются после вычисления p, которые принимаются равными сумме букв в фамилии и имени (при получении этой суммы более 15 принимается $p = 10$); k (количество разрядов сдвига) принимается равным количеству гласных букв в имени; L – левый сдвиг выбирается при начале имени с согласной, в противном случае выбирается R (правый сдвиг); $i = 2$ полагается при нечетном количестве букв имени, а в противном случае $i = 1$; знак «+» выбирается при нечетном количестве букв фамилии, в противном случае выбирается «−».

Получим вариант для вычисления D для студента с фамилией и именем Иванов Николай: $D = L_3[A^2(0{:}13) \times B(0{:}13) - C(0{:}13)]$.

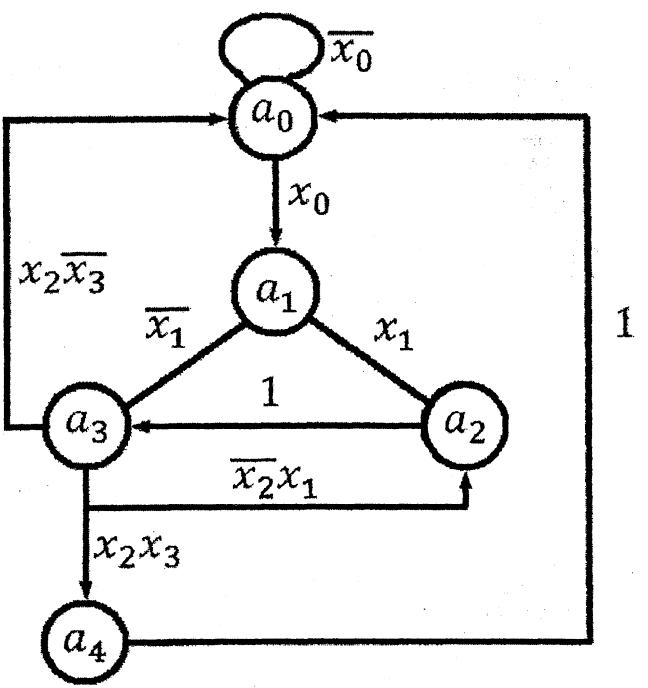


Рис. 5.5. Граф-схема УА Мура

РАЗДЕЛ 6. КОНТРОЛЬ И ДИАГНОСТИКА ЦИФРОВЫХ АВТОМАТОВ (ЗАДАНИЕ №8)

Общие подходы к обеспечению нормального функционирования ЦА

При возникновении какого-либо нарушения нормального функционирования ЦА результат будет неверным, однако пользователь об этом может не узнать, если не будут предусмотрены сигналы о появлении ошибки. Поэтому при конструировании ЦА надо предусмотреть возможность обнаружения ошибок или использовать методы и схемы, позволяющие их исправлять. Эти функции возлагаются на систему контроля работы ЦА.

Контроль работы ЦА разделяется на профилактический (предупреждение возможных ошибок) и оперативный (проверка правильности выполнения всех операций).

Решение задач контроля и надежного функционирования ЦА требует определенной избыточности его структуры, которая создается за счет схемных или логических средств, а также использования дополнительной информации.

Так как позиционные системы не несут в себе избыточности информации при представлении чисел, то они в чистом виде не годятся для контроля.

Большой интерес для особо ответственных блоков ЦА представляют различные мажоритарные структуры, когда процесс вычислений продолжается лишь при наличии совпадения большинства результатов, выполняемых на нескольких (3 и более) идентичных устройствах.

Ошибка в работе ЦА может быть систематической, возникающей в результате отказа, и случайной, возникающей в результате сбоя.

Сбоями называют кратковременные нарушения правильной работы ЦА, происходящие, как правило, в результате помех (электрические поля, вибрации и т.п.). Сбоями в телевидении обычно можно пренебречь, а в ВТ нужно бороться с ошибками. Это делается двумя путями: увеличением надежности отдельных элементов и узлов, а также посредством выявления и исправления ошибок. Мы будем рассматривать в основном второй путь, который связан с введением избыточности в перерабатываемую информацию, что является основой аппаратурного контроля ЦА.

Возможность обнаружения ошибок базируется на простой идее, которая заключается в том, что в n – разрядном двоичном слове используются не все $N = 2^n$ возможных комбинаций (от 0...0 до 1...1 = $2^n - 1$). Остальные $N - N_p = N_з$ комбинации являются запрещенными. Ошибки в двоичных словах заключаются в том, что цифры в неверных разрядах заменяются на взаимообратные. Если в результате ошибок кодовая комбинация перешла в разряд запрещенных, то этим самым вскрывается наличие ошибок.

Если же кодовая комбинация является ошибочной и переходит в разрешенную, то такая ошибка не обнаруживается. Чем больше избыточность, тем выше корректирующая способность применяемого кода.

Любой код, обладающий ненулевой избыточностью, называется корректирующим, независимо от того, исправляет он или только обнаруживает ошибки. В ВТ коды разделяются на посылочные и арифметические, так как задачи обнаружения (исправления) ошибок при передаче (хранении) информации и выполнении арифметических и логических операций в общем случае значительно различаются. Задача проектирования ЦА сводится к выбору такого кода, который при возможно меньшей избыточности обеспечивает нужную корректирующую способность.

Автоматическая диагностика (программным или аппаратурным путем) отдельных узлов ЦА в основном базируется на сравнении получаемых результатов функционирования ЦА на заранее известном наборе входных тестов (последовательностей), для которых имеются эталоны при исправной работе машины, а иногда и эталоны результатов при некаторах видах отказов элементов или узлов ЦА.

Все методы диагностики ЦА делятся на два основных класса. К первому классу относятся методы, при реализации которых диагностирование осуществляется в процессе функционирования ЦА. Их преимущество в том, что неисправность немедленно обнаружи-

вается или же происходит автоматическая корректировка результатов. Однако это влечет и дополнительные затраты на постоянно функционирующее избыточное оборудование.

Ко второму классу относятся методы, основанные на внесении временной избыточности, – тестовое диагностирование. При их реализации ЦА находится в режиме тестирования и не функционирует по своему прямому назначению. Для этого случая готовятся заранее специальные последовательности входных воздействий – тесты, которые позволяют обнаружить заданные виды неисправностей.

Для реализации тестового контроля часто используется схема, основанная на применении эталонного ЦА. В этом случае нет необходимости в запоминании эталонных сигналов на выходе, что упрощает стендовое оборудование и дает больше возможности для поиска неисправностей.

Эффективность тестового диагностирования зависит от выбора входных воздействий, позволяющих обнаруживать определенный класс неисправностей.

Контроль правильности передачи и хранения информации

Введем основные понятия из области корректирующих кодов. Количество единиц в двоичной кодовой комбинации называется ее кодовым весом W. (Если $[X_{\text{ПР}}] = 1,1010_2$, то $W = 3$).

Количество разрядов d, в которых не совпадают двоичные цифры в двух кодовых комбинациях, называется расстоянием между этими комбинациями. Оно обычно определяется методом поразрядного сложения по модулю два с последующим вычислением веса суммы. ($[X_1]_{\text{ПР}} = 1,010, [X_2]_{\text{ПР}} = 0,1011, [C]_{\text{ПР}} = [X_1]_{\text{ПР}} \oplus [X_2]_{\text{ПР}} = 1,001, W = 2, d = 2$).

Минимальным кодовым расстоянием d_{min} называется самое малое кодовое расстояние, возможное между двумя любыми кодовыми комбинациями из рассматриваемых множеств. В обычном двоичном коде n-разрядных чисел возможны кодовые расстояния от 1 до n, т.е. здесь $d_{min} = 1$.

Обнаружение ошибки

В общем случае, когда в кодовой комбинации могут появляться ошибки любой кратности $i \leq n$ (одиночная, двойная, тройная и т.д.), для обнаружения некоторой ошибки требуется корректирующий код с минимальным расстоянием $d_{min} = i + 1$.

Следовательно, обнаружение одиночной ошибки можно обеспечить ценой минимальной избыточности – добавлением к слову всего одного контрольного разряда.

Практически в дополнительный разряд может проставляться единица, если исходное число содержит четное число единиц, а иначе – проставляется 0. Тогда возникновение одиночной ошибки (или же ошибки с нечетным изменением количества разрядов) приводит к нарушению нечетности веса всей комбинации. Можно дополнять единицу также с целью контроля по четности, но в этом случае трудно отделить передачу нулевой информации от ее отсутствия.

Таким образом, избыточность кодовой комбинации на 1 разряд позволяет обнаруживать все нечетные групповые ошибки и как частный случай одиночную ошибку.

В местах возможного возникновения кратковременных сбоев или выхода из строя отдельных элементов применяются так называемые коды с обнаружением ошибок, а для особо ответственных схем – коды с автоматическим исправлением ошибок. В таких кодах (обычно двоичных) имеются дополнительные контрольные разряды.

Идея (двоичного) кода с исправлением одиночной ошибки состоит в следующем. Пусть исходный (незащищенный) код имеет m двоичных разрядов. Выберем $n > m$ так, чтобы $\frac{2^n}{n+1} \geq 2^m$. Полученный защищенный код будет иметь $k = n - m$ контрольных разрядов, причем $2^k \geq m + k + 1 = n + 1$. Каждому из n разрядов присваивается номер

от 1 до n. Далее для любого значения защищенного (n-разрядного) кода составляется k контрольных сумм $S_1, \ldots, S_k$ по модулю 2 значений специально выбранных разрядов этого кода. Для S_i выбираются разряды, для которых двоичные коды номеров имеют в i-ом разряде единицу. Для суммы S_1 это будут, очевидно, разряды с номерами 1, 3, 5, 7,..., для суммы S_2 – разряды с номерами 2, 3, 6, 7, 10, 11, ... и т.д.

При любом исходном коде контрольные разряды могут быть выбраны так, чтобы все контрольные суммы были равны нулю. Проверочный (двоичный) код $S = S_k \ldots S_1$ при этом будет нулевым. При таком условии защищенный код называется правильным. При возникновении одиночной ошибки в этом коде в любом (безразлично – основном или контрольном) разряде проверочный код будет отличен от нуля. При этом в силу способа выбора контрольных сумм значение проверочного кода S будет представлять собой двоичный код номера разряда защищенного кода, в котором возникла ошибка. Для исправления этой ошибки достаточно изменить значение указанного разряда на противоположное (если оно равно 0, то 1, а если 1 – то на 0).

Код, когда в качестве контрольных берут 1, 2, 4, ..., 2^n разряды называется кодом Хемминга. Код Хемминга может либо исправлять одиночные ошибки (при гипотезе, что ошибки более высокой кратности невозможны), либо обнаруживать любые ошибки, не кратные трем. Тройные ошибки считаются маловероятными и поэтому иногда говорят, что код Хэмминга позволяет обнаруживать одиночные и двойные ошибки.

Покажем на примере, как ликвидируется одиночная ошибка с помощью кода Хемминга. Чтобы искать и исправлять одиночную ошибку нужно поэтапно проделать следующую работу:

1. Задать незащищенное двоичное слово, Для иллюстрации используем слово 10100111 из восьми разрядов ($m = 8$).

2. Вычислить необходимый размер (n) разрядной сетки защищенного слова и количество контрольных разрядов k. Решив неравенство $\frac{2^n}{n+1} \geq 2^m = 2^8$, получим $n = 12$, $k = 12 - 8 = 4$.

3. Определим номера контрольных разрядов в защищенном слове: $2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8$.

4. Построим защищенное слово в 12 разрядной сетке. Занесем исходное слово в порядке следования его разрядов, пропуская места контрольных разрядов.

1	2	3	4	5	6	7	8	9	10	11	12
		1		0	1	0		0	1	1	1
0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

Определим значения защищенных разрядов исходя из равенства сумм S_i нулю. Чтобы сумма S_1 была равна нулю, входящий в нее контрольный разряд должен быть равен 0. В сумму S_1 должны войти значения разрядов, расположенных на нечетных местах $(1,3,5,7,9,11) = (0001,0011,\ldots,1011)$.

$$S_1 = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 0.$$

Равенство достигается при нулевом первом контрольном разряде. Аналогично отыскиваются и другие контрольные разряды.

В S_2 войдут разряды: $(0010,0011,0110,0111,1010,1011)$.

$$S_2 = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0.$$

В S_3 войдут разряды: $(0100, 0101, 0110, 0111)$.

$$S_3 = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 0.$$

В S_4 войдут разряды: $(1000, 1001, 1010, 1011, 1100)$.

$$S_4 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1.$$

По результатам вычисления контрольных разрядов и значениям разрядов передаваемого слова строим защищенное слово.

1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	0	0	1	0	1	0	1	1	1
0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

Полученное защищенное слово передается в нужное место и там определяется правильность его передачи. В месте приема переданного слова снова вычисляются контрольные суммы S_1, S_2, S_3, S_4, но уже при известных значениях заштрихованных контрольных разрядов. При равенстве всех сумм нулю считается, что передача произошла правильно. Предположим, что в процессе передачи произошло искажение только одного разряда (неважно какого: контрольного или основного) Для примера предположим, что неправильно передан разряд №10. Это означает, что вместо 1 поступил в десятый разряд 0. Тогда после вычисления контрольных сумм получим: $S_1 = 0, S_2 = 1, S_3 = 0, S_4 = 1$. Располагаем эти суммы в порядке убывания их номеров и получаем $S = S_4 S_3 S_4 S_1 = 1010$. Это двоичное число соответствует номеру 10. Тогда можно автоматически исправить ошибку в 10 разряде, заменив пришедший 0 на 1.

Задание №8 (а)

В соответствии с фамилией студента задать исходное слово, записав по порядку следования в ней букв на месте гласных 1 и на месте согласных 0. Провести защиту этого слова, допустить ошибку в разряде равном количеству букв в фамилии и обнаружить ее.

Контроль выполнения арифметических операций

Коды с проверкой на четность и код Хемминга нельзя непосредственно применять для контроля правильности арифметических операций, так как у них не существует однозначной связи между контрольными разрядами исходных чисел и контрольными разрядами результата операций.

Поэтому для целей контроля этих операций используется специальные арифметические корректирующие коды. Они также делятся на классы кодов обнаруживающих и исправляющих ошибки.

Простейшим арифметическим кодом, обнаруживающим ошибки, является код с проверкой по модулю. В его контрольные разряды записывается остаток от деления исходного кодируемого числа на некоторое заранее заданное число (модуль) d. При этом все машинные числа условно рассматриваются как целые, хотя на самом деле как числа с фиксированной запятой, так и с плавающей запятой представляют собой сложные слова.

Арифметичность кода основана на том, что остаток от деления на d суммы (произведения) должен быть равным сумме (произведению) остатков от деления на d исходных чисел.

Например, $d = 3$, выполним сложение и умножение чисел $x_1 + x_2$ и $x_1 \cdot x_2$ для чисел $x_1 = 13, x_2 = 16$, остаток $r_1 = 1, r_2 = 1$. Тогда $C = x_1 + x_2 = 13(1) + 16(1) = 29(2)$, $r_C = 2; C = x_1 \cdot x_2 = 13(1) \cdot 16(1) = 208(1), r_c = 1$.

Если остаток r_c будет превышать d, то от него самого вычисляется r_c' от r_c по модулю d. Учитывая, что все арифметические операции в ЭВМ сводятся к сложению, то достаточно ввести контроль только операции сложения.

К выбору модуля d предъявляются следующие требования:

1. Модуль должен обеспечить выявление как можно большего числа ошибок при обязательном выявлении одиночных ошибок.
2. Вычисление остатка от деления числа на модуль должно определяться простыми и быстрыми методами.
3. Модуль должен быть небольшим, чтобы получаемые остатки также имели небольшую разрядность.

Обычно модуль выбирается из ряда чисел $(2^i - 1)$ при $i = 2, 3, 4, \ldots$. На практике себя оправдал выбор модулей в интервале $3 \le d \le 15$. Приведем пример для сложения двоичных чисел.

$[X_1]_{\text{ПР}} = 0{,}01101_{(2)}$ и $[X_2]_{\text{ПР}} = 0{,}10000_{(2)}$, $d = 3_{(10)}$, $r_1 = 1$, $r_2 = 1$, $r_3 = r_1 + r_2 = 10_2$. $[C]_{\text{ПР}} = 0{,}11101$, $r_3 = 10_2$.

Задание №8 (б)

Сложить и умножить числа A и B, используя модуль $d = 3$. В качестве A взять количество букв фамилии студента, а в качестве B взять сумму букв в имени и отчестве студента. Пример, Иванов Петр Леонидович: $A = 6$, $B = 14$.

Контроль выполнения логических операций

Контроль выполнения логических операций может выполняться аналогично арифметическим. Покажем это для дизъюнкции и конъюнкции.

Известна теорема, что остаток по произвольному модулю от результата поразрядной дизъюнкции r_v двух слов равен сумме остатков операндов минус остатков r_k от результата поразрядной конъюнкции операндов, т.е. $r_v = (r_1 + r_2 - r_k) \bmod d$.

Таким образом, параллельно необходимо выполнить и поразрядную конъюнкцию над операндами, чтобы выполнить контроль дизъюнкции ($d = 3_{10}$ или $d = 11_2$).

Например, $x_1 = 11011$, $x_2 = 10111$, $x_1 \vee x_2 = 11111$, $x_1 \cdot x_2 = 10011$, $r_2 = 0$, $r_2 = 2$, $r_k = 1$, $r_v = (0 + 2 - 1) \bmod 3 = 1$.

Контроль выполнения конъюнкции аналогичен в соответствии с законом двойственности $r_k = (r_1 + r_2 - r_v) \bmod d$.

Задание №8 (в)

Выполнить и проверить логические операции $A \wedge B = C$ и $A \vee B = C$, используя $d = 3_{10} = 11_2$. A и B сформировать как пятиразрядные двоичные числа по последовательности букв в фамилии, имени и отчестве студента так, что гласные буквы замещаются нулями, а согласные 1 (выбирается подряд 10 букв). Пример, Иванов Ким Леонидович: $A = 01010$, $B = 11011$.

Учебное издание

Составитель:
Матюшков Леонид Петрович

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ В ЭВМ И СИНТЕЗ ЦА ДЛЯ ИХ МИКРОПРОГРАММНОГО ВЫПОЛНЕНИЯ

Методическое указания к выполнению практических занятий по дисциплине
«Организация и функционирование традиционных и интеллектуальных компьютеров»
для студентов специальности
1-40 03 01 «Искусственный интеллект»

Ответственный за выпуск: Матюшков Л.П.
Редактор: Строкач Т.В.
Компьютерная верстка: Боровикова Е.А.
Корректор: Никитчик Е.В.

Подписано к печати 18.10.2007 г. Формат 60x84 1/16. Бумага «Снегурочка».
Усл. п. л. 3,0. Уч.-изд. л. 3,25. Тираж 100 экз. Заказ № 1098. Отпечатано на ризографе учреждения образования «Брестский государственный технический университет».
224017, г. Брест, ул. Московская, 267.