681.325
К-276

М. А. КАРЦЕВ, В. А. БРИК

ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ И СИНХРОННАЯ АРИФМЕТИКА

М. А. КАРЦЕВ, В. А. БРИК

ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ И СИНХРОННАЯ АРИФМЕТИКА

МОСКВА «РАДИО И СВЯЗЬ» 1981

ББК 32.97
К21
УДК 681.32

Карцев М. А., Брик В. А.

К21 Вычислительные системы и синхронная арифметика. — М.: Радио и связь, 1981. — 360 с., ил.

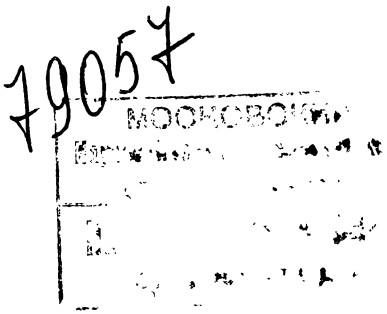
В пер.: 1 р. 20 к.

Исследуются, с одной стороны, организация структур многомашинных, многопроцессорных и конвейерных вычислительных систем и организация вычислений в них и, с другой стороны, — большинство известных методов построения быстродействующих синхронных сумматоров, умножителей и устройств для деления, используемых в вычислительных системах и машинах.

$К\frac{30502\text{-}127}{046(01)\text{-}81}$ 62-81 (С. р.) 2704080330 **ББК 32.97**
6Ф7.3

Рецензенты: д-р техн. наук проф. Ю. М. Шамаев, канд. техн. наук. Ю. Н. Глухов.

Редакция литературы по кибернетике и вычислительной технике



© Издательство «Радио и связь», 1981 г.

ПРЕДИСЛОВИЕ

Приступая к этой книге, мы отдавали себе отчет в том, что в настоящее время отнюдь не ощущается недостатка в публикациях по вычислительным системам. Наоборот, поток книг и статей по этим вопросам существенно обгоняет развитие техники и способов применения вычислительных систем. Вероятно, количество названий только монографий по этим вопросам превышает количество реализованных и действующих вычислительных систем, имеющихся в мире. Между тем в этой области есть еще слишком много неясных вопросов.

Не устоялась даже терминология. В этой книге под словами *вычислительная система*, вынесенными в заголовок, мы понимаем вычислительные средства, предназначенные для выполнения параллельных вычислений; точное определение этого понятия содержится в п. 1.1.1.

Но наименее проработанным является, на наш взгляд, вопрос о производительности вычислительных систем. Слишком часто данные, сообщаемые о производительности проектируемой или выпускаемой вычислительной системы, получаются путем простого суммирования производительности отдельных средств, входящих в состав системы. Между тем ситуации, с которыми приходится сталкиваться пользователю реальной вычислительной системы, могут быть существенно разными; соответственно различны и те данные о производительности системы, которые ему нужны.

Если система используется в составе крупного вычислительного центра, где большое количество пользователей решает свои сравнительно мелкие задачи, то суммарная производительность системы почти не интересует каждого отдельного пользователя. Она важна для него лишь постольку, поскольку от нее зависит, сколько времени он сможет получить для работы в интерактивном (диалоговом) режиме или сколько задач от него примут для решения в режиме пакетной обработки.

С очень похожим положением мы имеем дело, как правило, при использовании вычислительной системы в составе автоматизированной системы управления производством (в масштабе предприятия, отрасли, народного хозяйства в целом) и в различных информационно-поисковых системах. Здесь тоже обычно имеется ряд мало связанных между собой небольших задач, работающих, однако, над общим банком данных. Но проектировщика АСУ или информационно-поисковой системы не может, конечно, не интересовать суммарная производительность вычислительных средств, поскольку определенные наборы задач его системы должны выполняться в определенные интервалы времени (иногда — в течение нескольких часов, или нескольких суток, или нескольких недель).

С принципиально другой ситуацией мы встречаемся в автоматизированных системах управления сложными технологическими процессами, при решении крупных научных задач и в других случаях. Высокая производительность вычислительных средств здесь нужна для того, чтобы получить в течение короткого времени решение одной, но достаточно громоздкой задачи. Как нам представляется, во многих случаях ни у проектировщиков вычислительных систем, ни у тех, кто предполагает использовать эти системы для указанных целей, нет четкого представления о том, что одна и та же вычислительная система не может обеспечить одинаковую производительность при решении крупных задач разных классов и что необходимо определенное соответствие между свойствами конкретных задач и структурой вычислительной системы, чтобы возможности вычислительной системы использовались достаточно эффективно.

Когда в этой книге речь идет о *высокопроизводительных* вычислительных системах, то имеется в виду в основном именно эта последняя ситуация (в п. 1.1.2 приведены точные определения реальной пользовательской производительности и пользовательской эффективности вычислительной системы).

Гл. 1 и 2 посвящены подробному рассмотрению именно этих вопросов. В результате выделен тип *синхронных вычислительных систем*, которые потенциально могут обеспечить максимум реальной пользовательской производительности. Разумеется, эта производительность может быть реализована при решении задач определенного

класса (т. е. обладающих определенными свойствами), но класс этот достаточно широкий и включает достаточно важные задачи.

Гл. 3—6 посвящены рассмотрению наиболее сложных технических вопросов, возникающих при построении таких систем, — разработке *синхронных методов выполнения арифметических и логических операций*, т. е. методов, обеспечивающих минимальное время выполнения операции независимо от операндов, над которыми она выполняется.

Значение этих глав значительно шире, чем можно было бы заключить из предыдущего, поскольку применение синхронных методов выполнения операций необходимо и при построении обычных (однопроцессорных) управляющих машин, предназначенных для работы в реальном масштабе времени. Во многих случаях предлагаемые синхронные методы выполнения операций обеспечивают более высокое быстродействие, чем известные асинхронные методы, и их применение целесообразно при создании вообще любых быстродействующих ЦВМ.

Гл. 7 предназначена для недостаточно подготовленного читателя, который захотел бы понять до конца содержание всей книги. Ему следовало бы начать чтение книги именно с этой главы, в которой содержатся начальные сведения по основам вычислительной техники.

Гл. 1 и 2 этой книги написаны М. А. Карцевым, гл. 3—7 — В. А. Бриком.

Авторы будут признательны читателям за все замечания, которые они сочтут возможным высказать по содержанию книги.

В. А. Брик, М. А. Карцев

1. ПРОБЛЕМА ОРГАНИЗАЦИИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

1.1. ИСХОДНЫЕ ПОЛОЖЕНИЯ

1.1.1. ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ, ЦЕЛИ ИХ СОЗДАНИЯ

1°. *Вычислительной системой* мы будем называть *комплекс вычислительных средств, связанных физически и программно*, в котором *одновременно* (в один и тот же момент времени) могут выполняться *несколько арифметических или логических операций по преобразованию данных.*

Мы попытались сформулировать это определение так, чтобы исключить из понятия вычислительных систем обычные современные машины, содержащие, как правило, один центральный процессор и один или несколько периферийных процессоров — каналов. Хотя эти процессоры могут работать все одновременно, организация машины большей частью такова, что арифметические и логические операции над данными выполняются только центральным процессором, а каналы заняты простой передачей данных.

В составе центрального процессора обычной вычислительной машины может быть несколько специализированных арифметических и логических устройств разного назначения: например, для выполнения операций с числами, для формирования адресов главной памяти и др. При этом, с целью повышения быстродействия центрального процессора, временную диаграмму его работы во многих случаях строят так, чтобы операции этих устройств перекрывались по времени. Скажем, когда основное арифметико-логическое устройство выполняет операцию над числами, указанную в i-й инструкции, то в то же время формируется исполнительный адрес для обращения к главной памяти, необходимый при исполнении $(i+1)$-й инструкции, и т. д. Но и при этом вычислительная машина не подходит под данное нами определение вычислительной системы, потому что в нем речь идет об одновременном выполнении нескольких арифметических или логических операций над данными.

С другой стороны, определение вычислительных систем распространяется на различные типы многопроцессорных систем и многомашинных комплексов, системы с магистральной (конвейерной) структурой, вычислительные среды и другие средства. Различные структуры вычислительных систем рассматриваются в § 2.1.

2°. *Цели*, которые ставятся при создании вычислительной системы, могут быть различны.

В ряде случаев целью разработчика вычислительной системы является достижение более высокой *надежности*, чем может быть обеспечена одиночной вычислительной машиной.

Например, если машина должна использоваться для управления технологическим процессом, то для обеспечения непрерывности технологического процесса может быть применен комплекс из двух одинаковых машин. Когда обе машины находятся в рабочем состоянии, одна из них назначается ведущей, другая — резервной. Собственно решение всей задачи управления осуществляется ведущей машиной; резервная машина принимает от объекта управления всю информацию, которая идет и на ведущую машину, выполняет над ней те же операции, что и ведущая, но никакой информации наружу не выдает. При возникновении отказа в ведущей машине или при необходимости вывести ведущую машину на профилактику производится переназначение машин в комплексе, причем практически одновременно прекращается выдача информации с бывшей ведущей машины и открывается выдача информации с бывшей резервной машины. После восстановления работоспособности машины, выведенной в ремонт или на профилактику, она подключается к комплексу в качестве резервной.

В других случаях при создании вычислительной системы ставится цель повышения *живучести* по сравнению с одиночной вычислительной машиной, т. е. способности выполнять свои основные функции (но не все функции в полном объеме) при наличии тех или иных отказов в аппаратуре.

Например, в рассмотренном выше 2-машинном комплексе на резервную машину может быть возложена задача повышения достоверности выдаваемой информации путем оперативного сравнения своих результатов с результатами, получаемыми ведущей машиной, и блокировки выдачи информации управляемому объекту при обнаружении несовпадения. В случае отказа одной из машин другая берет на себя роль ведущей и продолжает вести управление технологическим процессом, но функции повышения достоверности управляющей информации в тот период, пока не восстановлена работоспособность обеих машин, не реализуются.

Могут быть ситуации, когда вычислительная система создается для *упрощения каналов передачи информации.*

Например, можно представить себе, что управление технологическим процессом в принципе может быть реализовано с помощью одной вычислительной машины, расположенной в центре системы управления. Однако при этом потребовалось бы организовать передачу большого количества информации от множества периферийных датчиков к центральной машине. Возможно, что у некоторых групп датчиков выгодно поставить свои периферийные машины, которые вели бы первичную обработку и объединение информации от присоединенных к ним датчиков и далее сообщались бы значительно более простыми каналами передачи информации с центральной машиной.

Чаще всего, однако, целью создания вычислительной системы является получение более высокой *производительности*, чем производительность обычной вычислительной машины.

Предпосылкой этого направления является микроминиатюризация компонентов схемы, повышение их надежности, снижение стоимости и энергопотребления. Именно эти обстоятельства позволяют объединить в одном аппаратурном комплексе больше элементов, чем в обычной вычислительной машине, и получить увеличение быстродействия не только за счет повышения скоростей отдельных элементов и сокращения длин соединений, но и за счет усложнения структуры системы.

В дальнейшем, говоря о вычислительных системах, мы будем иметь в виду главным образом системы, создаваемые с целью достижения высокой производительности при обработке информации.

1.1.2. ПРОИЗВОДИТЕЛЬНОСТЬ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Разработка структуры вычислительной системы не вызывает особых затруднений в тех случаях, когда целью создания вычислительной системы является получение высокой надежности или живучести или упрощение каналов связи в системе. Сравнительно просто этот вопрос решается и в тех случаях, когда при создании системы ставится задача достижения высокой *системной производительности*. Если же вычислительная система должна обеспечить более высокую *пользовательскую производительность*, чем производительность обычной вычислительной машины, вопрос о структуре такой системы далеко не тривиален.

1°. *Системная производительность* — это производительность, достигаемая при решении совокупности пользовательских задач.

Представим себе для примера вычислительную систему, состоящую из нескольких (n) вычислительных машин, причем на любом отрезке времени каждая из машин назначается для решения одной из пользовательских задач. Ясно, что системная производительность такой вычислительной системы примерно в n раз выше, чем производительность каждой из машин, т. е. примерно равна сумме производительностей всех входящих в систему машин, но скорость решения одной отдельно взятой пользовательской задачи примерно такая же, как и при использовании для этого одиночной машины.

Может возникнуть вопрос, имеет ли вообще смысл при этих условиях объединять машины в систему? В действительности такое объединение дает определенный выигрыш по нескольким направлениям. Он состоит прежде всего в возможности более рационального использования обобществленных ресурсов машин (внешней памяти, периферийных устройств, процессорного времени) с учетом реальных потребностей активных пользователей в этих ресурсах на каждом отдельном отрезке времени. Уменьшается общий объем памяти на внешних носителях, занимаемой программами математического обеспечения (поскольку не нужно иметь столько дубликатов каждой программы, сколько машин имеется в системе); при определенных условиях может быть достигнута и экономия в объеме внутренней памяти. Объединение машин в систему увеличивает живучесть комплекса, поскольку выход из строя одной из машин — особенно запланированный заранее — не нарушает очередности пользовательских программ, не требует от человека действий по переносу на другие машины программ, находившихся в очереди к вышедшей из строя машине, и приводит лишь к замедлению общего темпа прохождения пользовательских программ. Упрощается также решение вопросов организации контроля вычислений и др.

2°. Возможна ситуация, когда пользователь выходит на счет с множеством сравнительно небольших независимых задач. С его точки зрения производительность вычислительной системы это и есть системная производительность, о которой мы говорили в п. 1°. В более общем случае, однако, пользователя интересует производительность вычислительной системы, обеспечиваемая при решении одной крупной задачи.

Соответственно *пользовательской производительностью* мы будем называть производительность вычислительной системы, которая достигается при решении одной отдельно взятой пользовательской задачи — в условиях, когда данной задаче предоставлен приоритет в получении всех необходимых ей ресурсов системы перед задачами других пользователей.

3°. Производительность вычислительной системы — это вообще сложное (и не достаточно точно определенное) понятие, включающее скорости выполнения операций, объемы памяти, характеристики операционной системы, пропускную способность каналов внешнего обмена и другие параметры. Тем не менее в дальнейшем, рассматривая вопрос о производительности вычислительной системы, мы будем иметь в виду главным образом один из этих параметров — ее *быстродействие.* Если в качестве примера представить себе вычислительную систему как объединение нескольких вычислительных машин, то более или менее очевидно, как в общей производительности системы суммируются, скажем, объемы памяти отдельных машин (для этого достаточно обобществить запоминающие устройства и обеспечить обращения от всех машин к общей памяти либо обеспечить обмен информацией между машинами). Между тем вопрос о том, как суммируются величины быстродействия отдельных машин — особенно если речь идет о пользовательской, а не системной производительности — весьма сложен.

Говоря о быстродействии вычислительной системы, мы должны будем ввести две различные характеристики — *номинальное быстродействие* $S_н$ и *реальное быстродействие* $S_р$ системы.

Номинальным быстродействием $S_н$ будем называть суммарное быстродействие всех составных частей системы.

Реальное быстродействие, входящее в оценку пользовательской производительности, будет получаться разным для разных задач или разных классов задач и отличным от этих значений — при оценке системной производительности.

Отношение реального быстродействия к номинальному будем называть *эффективностью* вычислительной системы E,

$$E=S_р/S_н.$$

Соответственно сказанному выше необходимо различать *системную эффективность* и *пользовательские эффективности* для различных задач (различных классов задач).

Представим себе в качестве простейшего примера вычислительную систему, полученную объединением n вычислительных машин (см. п. 1.1.1).

Если машины все одинаковы и быстродействие каждой из них равно S [операций/с], то номинальное быстродействие системы равно, очевидно, $S_н = n \cdot S$.

Если система организована так, что на любом отрезке времени одна машина системы отводится всегда одной пользовательской программе, то реальное системное быстродействие примерно равно номинальному быстродействию, причем системная эффективность примерно равна единице,

$$S_{р(с)} \approx S_н = nS; \quad E_с = S_{р(с)}/S_н \approx 1,$$

а реальное пользовательское быстродействие равно примерно быстродействию одной машины — независимо от свойств пользовательских программ. Соответственно пользовательские эффективности всегда одинаковы и тем меньше, чем больше n (чем больше машин объединено в систему):

$$S_{р(п)} \approx S; \quad E_п = S_{р(п)}/S_н \approx 1/n.$$

Чтобы получить пользовательское быстродействие больше S, систему нужно строить так, чтобы приоритетная программа могла выполняться параллельно несколькими машинами системы (в пределе — n машинами). Возможность реализовать такой режим зависит от частных свойств задачи. Конкретно: требуется, чтобы в программе данной задачи на каждом этапе ее решение можно было выделить n ветвей, каждая из которых могла бы выполняться независимо от других ветвей.

В общем случае (для произвольных задач) реальное пользовательское быстродействие такой системы лежит в пределах $S_н \geqslant S_{р(п)} \geqslant S$, и соответственно $1 \geqslant E_п \geqslant 1/n$, причем $E_п$, вообще говоря, тем меньше, чем больше n.

Величины $S_{р(п)}$ и $E_п$ для некоторой величины n при указанном построении системы существенным образом зависят от того, на какое количество независимых ветвей удается разделить решение некоторой конкретной задачи пользователя на каждом этапе исполнения программы, что в свою очередь существенно зависит от частных свойств данной задачи.

Сложнее определить характеристики быстродействия системы, если она состоит из n различных машин.

Пусть быстродействие i-й машины ($i=1, 2, \ldots, n$) равно

$$S_i = 1 \Big/ \sum_{j=1}^{m} k_j t_{ij},$$

где k_j — доля операций j-го типа в общем количестве операций, необходимых для решения набора типовых задач (по которому ведется определение быстродействия машины), а t_{ij} — время выполнения одной операции j-го типа i-й машиной ($j=1, 2, \ldots, m$).

Для определения номинального быстродействия системы нужно попытаться так распределить операции между машинами, чтобы выполнялись равенства

$$\sum_{j=1}^{m} k_{1j}t_{1j} = \sum_{j=1}^{m} k_{2j}t_{2j} = \ldots = \sum_{j=1}^{m} k_{nj}t_{nj}$$

и при этом величина $T = \sum_{j=1}^{m} k_{ij}t_{ij}$ была минимальна. Здесь k_{ij} — доля операций j-го типа, исполнение которых передано i-й машине ($j=1, 2, \ldots, m$; $i=1, 2, \ldots, n$), причем, разумеется, должны выполняться равенства

$$k_j = \sum_{i=1}^{n} k_{ij} \quad (\text{для } j = 1, 2, \ldots, m).$$

В этой системе уравнений неизвестными являются величины k_{ij}; их количество равно $n \cdot m$. Количество линейных алгебраических уравнений, связывающих величины k_{ij}, равно $n-1+m$, где n и m — целые числа, $n \geqslant 2$, $m \geqslant 1$. Нетрудно видеть, что система уравнений имеет единственное решение при $m=1$, т. е. при условии, что в любой i-й машине все операции выполняются за одинаковое время $t_{i1}=t_i=1/S_i$. При этом

$$k_{11}:k_{21}:\ldots:k_{n1} = \frac{1}{t_1}:\frac{1}{t_2}:\ldots:\frac{1}{t_n} = S_1:S_2:\ldots:S_n,$$

а номинальное быстродействие вычислительной системы

$$S_н = \frac{1}{T} = S_1 + S_2 + \ldots + S_n.$$

Аналогичный результат получается при условии, что времена выполнения операций разных типов в различных машинах пропорциональны, т. е. что

$$t_{11} : t_{12} : \ldots : t_{1m} = t_{21} : t_{22} : \ldots : t_{2m} = \ldots = t_{n1} : t_{n2} : \ldots : t_{nm}$$

или (что то же самое)

$$t_{11}:t_{21}:\ldots:t_{n1} = t_{12}:t_{22}:\ldots:t_{n2} = \ldots = t_{1m}:t_{2m}:\ldots:t_{nm} = \frac{1}{S_1}:\frac{1}{S_2}:\ldots:\frac{1}{S_n}.$$

При этом тоже

$$k_{11} : k_{21} : \ldots : k_{n1} = k_{12} : k_{22} : \ldots : k_{2n} = \ldots = k_{1m} : k_{2m} : \ldots : k_{nm} =$$

$$= S_1 : S_2 : \ldots : S_n$$

и $S_н = S_1 + S_2 + \ldots + S_n$.

В общем случае, если $m > 1$ и времена выполнения однотипных операций в различных машинах не пропорциональны друг другу, то система линейных уравнений недоопределена, и нахождение величин k_{ij}, соответствующих минимальной величине $T = \sum_{j=1}^{m} k_{ij} t_{ij} = \frac{1}{S_н}$ при ограничениях $0 \leqslant k_{ij} \leqslant k_j$ для $j = 1, 2, \ldots, m$; $i = 1, 2, \ldots, n$ является задачей линейного программирования.

На самом деле решение этой задачи не имеет особого смысла, поскольку найденное таким образом номинальное быстродействие вычислительной системы $S_н$ все равно практически недостижимо, и реальное системное быстродействие $S_{р(с)}$ будет наверняка ниже $S_н$, не говоря уже о реальном пользовательском быстродействии $S_{р(п)}$, которое будет еще ниже.

В гл. 2 мы рассмотрим различные построения вычислительных систем и увидим, что оценки реального быстродействия для них существенно различны. При одном и том же номинальном быстродействии реальное системное быстродействие и в особенности реальное пользовательское быстродействие для тех или иных классов задач сильно зависят от структуры вычислительной системы и частных свойств задач. Вычислительные системы, оставаясь универсальными в том смысле, что любая задача в принципе может быть решена любой вычислительной системой, в то же время весьма специализированы, поскольку в зависимости от структуры системы для одних задач ее эффективность оказывается удовлетворительной, а для других — недопустимо низкой.

В литературе по вычислительным системам, описаниях и особенно рекламных проспектах вычислительных систем этой стороне проблемы обычно не уделяется должного внимания и сообщается лишь номинальное быстродействие вычислительной системы. Между тем номинальное быстродействие характеризует не столько возможности системы при решении определенных задач, сколько затраты на создание системы. Выбор вычислительных систем для тех или иных конкретных применений, основанный лишь на данных о номинальном быстродействии и не учитывающий указанных обстоятельств, приводит в ряде случаев к тяжелым и трудно поправимым ошибкам.

1.2. ОСОБЕННОСТИ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ, ПОЗВОЛЯЮЩИЕ ОРГАНИЗОВАТЬ ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Общие способы организации параллельных вычислений, пригодные при решении любых задач, неизвестны и скорее всего не существуют. Когда с помощью вычислительной системы необходимо получать высокое пользовательское быстродействие, то для организации параллельных вычислений используются частные свойства тех или иных задач. При этом использование тех или иных свойств возможно при определенных структурах вычислительных систем. По этой причине, собственно, вычислительные системы разных типов эффективны при решении разных классов задач (см. гл. 2).

Ниже рассматриваются те частные особенности различных задач, на основе которых могут быть организованы параллельные вычисления.

1.2.1. ЕСТЕСТВЕННЫЙ ПАРАЛЛЕЛИЗМ И ПАРАЛЛЕЛИЗМ МНОЖЕСТВА ОБЪЕКТОВ

1°. Мы будем говорить, что задача обладает *естественным параллелизмом*, если в ее исходной постановке она сводится к операциям над многомерными векторами либо над матрицами, либо над решетчатыми функциями, либо над другими аналогичными объектами. Каждый из этих объектов может быть представлен совокупностью чисел (или, может быть, булевых переменных, если речь идет о булевых векторах, матрицах и т. п.): многомерный вектор — своими компонентами, матрица — элементами матрицы либо компонентами векторов, соответствующих ее столбцам или строкам, решетчатая функция—своими значениями на множестве дискретных значений аргумента и т. п.

Большинство операций, которые должны выполняться при этом, представляет собой совокупности одинаковых операций над соответствующими парами чисел (элементов) двух аналогичных объектов либо совокупности одинаковых операций над всеми числами, характеризующими объект, и константой. Например, сложение двух векторов состоит в сложении соответствующих компонент векторов, вычитание векторов — в вычитании их компонент, умножение вектора на число — в умножении каждой из компонент вектора на это число и т. д. Ясно, что

все эти операции могут выполняться параллельно и независимо друг от друга.

Само возникновение проблемы организации параллельных вычислений для решения таких задач связано не с сутью дела, а с тем обстоятельством, что в течение многих лет существования вычислительной математики ее главной целью было создание последовательностных алгоритмов, т. е. алгоритмов, каждый шаг которых состоял бы в выполнении операции над одной парой чисел или над одним числом. Именно так может действовать человек-вычислитель, если он выполняет вычисления вручную, так же действует обычная однопроцессорная вычислительная машина. Последовательностный характер алгоритмов нашел отражение в структуре наиболее распространенных алгоритмических языков—АЛГОЛ-60, ФОРТРАН и др. Появление вычислительных систем выдвинуло проблему организации параллельных вычислений на основе известных алгоритмов, в ряде случаев — по готовым программам (написанным, возможно, на АЛГОЛе-60 или ФОРТРАНе). Однако для задач, обладающих естественным параллелизмом, такой путь организации параллельных вычислений является далеко не лучшим [1, 2].

2°. *Параллелизм множества объектов* представляет собой частный случай естественного параллелизма. Его смысл в том, что задача состоит в обработке информации о различных, но однотипных объектах по одной и той же или почти одной и той же программе.

По сравнению с общим случаем естественного параллелизма, в задачах, обладающих параллелизмом множества объектов, сравнительно меньший вес имеют так называемые *интегральные операции.*

Например, вычисление скалярного произведения для n-мерных векторов

$$\mathbf{A}\cdot\mathbf{B}=\sum_{i=1}^{n} A_iB_i$$

состоит фактически из операций двух типов: сначала вычисление n попарных произведений соответствующих компонент векторов, причем получается вновь n-мерный вектор (компонентами которого являются числа вида A_iB_i, $i=1, 2, \ldots, n$), затем «интегральная» операция — суммирование между собой всех компонент этого вектора.

Большей частью интегральные операции отличаются тем, что исходными операндами для них являются векторы или функции, или множества объектов, а результатом должно быть число. Примером такой операции может быть, скажем, вычисление определенного интеграла от функции; собственно, само название интегральных операций выбрано по ассоциации с этим примером. Однако, например, и вычисление неопределенного интеграла, когда и исходный операнд и результат являются функциями, тоже представляет собой интегральную операцию, поскольку значение результата в каждой точке зависит, вообще говоря, от нескольких или в пределе от всех значений исходной функции.

В ходе решения задачи, обладающей параллелизмом множества объектов, такие интегральные операции тоже могут встретиться, но относительно реже, чем в общем случае.

С другой стороны, при использовании параллелизма множества объектов чаще, чем в общем случае естественного параллелизма, встречаются ситуации, когда отдельные участки вычислений должны выполняться различно для разных объектов.

В других случаях такие ситуации тоже могут быть.

Представим себе, например, решение задачи Дирихле на плоскости, т. е. отыскание значений функции $\varphi(x, y)$, удовлетворяющей уравнению

$$\frac{\partial \varphi}{\partial x} + \frac{\partial \varphi}{\partial y} = 0$$

во всех точках внутри некоторой области на плоскости x, y при заданных граничных условиях (заданных значениях $\varphi(x, y)$ на границах области). Область, в которой выполняются вычисления, накрывается прямоугольной сеткой, во всех узлах сетки, кроме граничных, задаются более или менее произвольным образом в качестве начального приближения некоторые значения функции φ. Далее вычисление очередного приближения искомой функции φ состоит в том, что для каждого узла сетки вычисляется среднее арифметическое из тех значений функции в 4 соседних узлах, которые она имела в предыдущем приближении. Все эти вычисления выполняются совершенно одинаково для всех узлов сетки, кроме, однако, узлов, расположенных на границе области, где должны всегда сохраняться заданные первоначально значения (граничные условия).

При использовании параллелизма множества объектов аналогичное положение встречается сравнительно часто.

3°. В дальнейшем для нас важна будет не только качественная сторона вопроса, состоящая в том, какими

свойствами, позволяющими организовать параллельные вычисления, обладают те или иные задачи, но и *количественная характеристика* этих свойств.

Если речь идет о естественном параллелизме или, в частности, о параллелизме множества объектов, то основной количественной характеристикой является *ранг задачи* r — количество параметров, по которым должна вестись параллельная обработка (количество компонент многомерного вектора, количество точек, в которых задана функция, количество однородных объектов в множестве обрабатываемых объектов и т. д.) [3].

В некоторых случаях ранг задач может сам выступать как векторная величина. Например, функции, над которыми должны выполняться операции, могут зависеть от нескольких — скажем, двух — переменных и задаваться на плоской решетке размером $r_1 \times r_2$ узлов; при этом ранг задачи есть вектор (r_1, r_2). Аналогичным образом, если задача состоит в сопоставлении по тем или иным правилам информации о каждом из r_1 объектов, принадлежащих первому множеству объектов, с информацией о каждом из r_2 объектов, принадлежащих второму множеству объектов, ранг задачи также является вектором (r_1, r_2).

Если, однако, структура вычислительной системы не рассчитана на матричную структуру данных и ориентируется на работу с данными типа векторов, функций одной переменной и т. п., то такие задачи могут решаться параллельно-последовательно. Например, операции могут выполняться сначала с r_1 значениями функции, лежащими на одной прямой, затем — с r_1 значениями, лежащими на другой прямой, и т. д. r_2 раз. Либо — в другом примере — сначала информация об r_1 объектах первого множества сопоставляется с информацией об одном из объектов второго множества, затем снова информация об r_1 объектах первого множества — с информацией о другом объекте второго множества, и т. д. r_2 раз. При этом мы должны считать, что ранг задачи есть скалярная величина r_1. Перестроив порядок вычислений в приведенных примерах, можно считать ранг задачи равным r_2.

В некоторых случаях ранг задачи не является константой. Например, на рис. 1.2.1 показана некоторая криволинейная область задания функции двух переменных. Как бы ни велся счет — параллельно по горизонта-

лям или параллельно по вертикалям — количество значений функции, с которыми можно оперировать одновременно, меняется при переходе от одной линии к другой. Возможны случаи, когда в ходе решения задачи необходимо оперировать с функциями, заданными на разных решетках, с векторами разной размерности и т. п.

Другой важной характеристикой для задач, обладающих естественным параллелизмом, является *коэффициент расхождения* задачи, D.

Смысл его состоит в следующем.

Для простоты изложения будем полагать, что речь

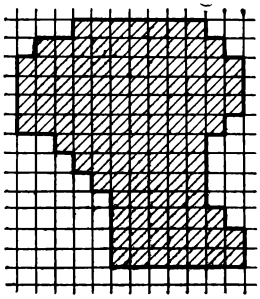


Рис. 1.2.1.

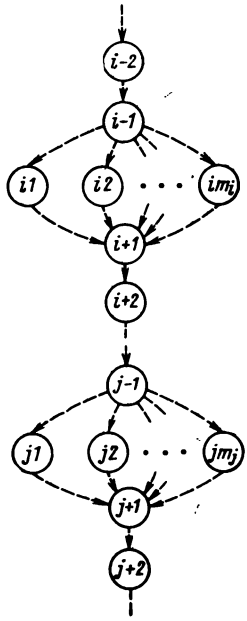


Рис. 1.2.2.

идет о параллелизме множества объектов. Информация обо всех r объектах, с которыми имеет дело данная задача (r — ранг задачи), обрабатывается по почти одинаковой программе, однако некоторые ветви программы для различных объектов различны. Например, на рис. 1.2.2 показан граф управления программы, в соответствии с которым программа обработки информации об объектах до (i—1)-го этапа включительно идет для всех объектов одинаково, далее объекты классифицируются на m_i классов (m_i=2, 3, ..., r) и — в зависимости от того, к какому из классов принадлежит данный объект — i-й этап выполняется в одном из m_i вариантов; далее

программа вновь идет для всех объектов одинаково, вплоть до $(j-1)$-го этапа включительно, затем объекты классифицируются на m_j классов и j-й этап программы выполняется в одном из m_j вариантов, и т. д. В программе однопроцессорной машины оператор $i-1$ заканчивается условной передачей управления к началу одного из операторов $i1$, $i2$, ..., im_i (в зависимости от условий, определяющих принадлежность объекта к одному из m_i классов), а операторы $i1$, $i2$, ..., im_i заканчиваются безусловной передачей управления к оператору $i+1$; аналогично — для оператора $j-1$ и операторов $j1$, $j2$, ..., jm_j.

Предположим для единства, что m_k различных вариантов имеется для любого k-го этапа программы, но для большинства этапов $m_k=1$. Предположим, кроме того, что *длина оператора* ks_k, где $k=1, 2, \ldots; s_k=1, 2, \ldots, m_k$, есть l_{ks_k}. Длина оператора — это фактически время его исполнения. Для разных вычислительных машин или разных физических реализаций вычислительных систем величины l_{ks_k} и соответственно характеристики задачи, получаемые ниже на их основе, могут быть различны. Возможно даже, что и в пределах одной вычислительной системы длина некоторого оператора зависит от того, какими средствами системы он выполняется; однако этот случай мы здесь не рассматриваем.

Суммарной длиной программы L_Σ назовем величину

$$L_\Sigma = \sum_{(k)} \sum_{s_k=1}^{m_k} l_{ks_k},$$

а *средней длиной программы* — величину $L_{\text{ср}}$, равную

$$L_{\text{ср}} = \frac{1}{r} \sum_{(k)} \sum_{s_k=1}^{m_k} r_{ks_k} \cdot l_{ks_k},$$

где r_{ks_k} — количество объектов, которые на k-м этапе относятся к s_k-му классу (т. е. для которых исполняется оператор ks_k); очевидно, что $\sum_{s_k=1}^{m_k} r_{ks_k} = r$ для любых k.

Величину *расхождения программы* D определим как отношение суммарной длины программы к ее средней длине:

$$D=\frac{L_{\Sigma}}{L_{\text{ср}}}=r\frac{\sum\limits_{(k)}\sum\limits_{(s_k)} l_{ks_k}}{\sum\limits_{(k)}\sum\limits_{(s_k)} r_{ks_k}\cdot l_{ks_k}}.$$

Нетрудно видеть, что если программа обработки информации по всем r объектам в точности одинакова, т. е. если на любом k-м этапе программы имеется всего одна ветвь программы, $m_k=1$ (причем, разумеется, $r_{k1}=r$), то расхождение равно единице, $D=1$. Чем сильнее отличаются между собой программы обработки информации о различных объектах, т. е. чем больше имеется этапов, на которых программы для разных объектов различны, чем больше разных классов объектов на каждом таком этапе (чем больше величины m_k и чем больше длины соответствующих операторов программы l_{ks_k} для этапов, где $m_k>1$), тем больше расхождение D.

Обратим внимание на один важный частный случай, который может встретиться при определении величины D. Предположим, что на некотором i-м этапе обработки информации должен циклически выполняться программный оператор длиной $l_{\text{ц}}$, причем количество повторений цикла для каждого из объектов заранее неизвестно и может быть $N_{\text{ц min}}$, $N_{\text{ц min}}+1$, $N_{\text{ц min}}+2$, ..., $N_{\text{ц max}}$. В этом случае i-й этап должен быть представлен в виде $(N_{\text{ц max}}-N_{\text{ц min}}+1)$ подэтапов $i^{(0)}, i^{(1)}, i^{(2)}, \ldots, i^{(N_{\text{ц max}}-N_{\text{ц min}})}$. На этапе $i^{(0)}$ количество вариантов $m_{i^{(0)}}$ равно 1, длина оператора равна $l_{i^{(0)}}=N_{\text{ц min}}l_{\text{ц}}$. Далее объекты разбиваются на 2 класса $(m_{i^{(1)}}=2)$: те, для которых должно быть выполнено по меньшей мере еще одно повторение цикла, и те, для которых цикл больше повторять не нужно; длины операторов, выполняемых на $i^{(1)}$-м этапе, равны соответственно $l_{i^{(1)}1}=l_{\text{ц}}$ и $l_{i^{(1)}2}=0$. Далее для $i^{(2)}$-го этапа объекты вновь классифицируются на 2 класса $(m_{i^{(2)}}=2)$ и т. д. При этом, разумеется, $r\geqslant r_{i^{(1)}1}\geqslant r_{i^{(2)}1}\geqslant\ldots\geqslant r_{i^{(N_{\text{ц max}}-N_{\text{ц min}})}1}$.

Заметим еще, что величина расхождения D — в том виде, как мы ее выше оценили количественно, — может быть вычислена для некоторой задачи только по готовой программе для решения этой задачи и с учетом конкретных технических характеристик вычислительной системы, которую предполагается использовать для этой цели. Однако на самом деле величина D, как и ранг задачи r, является характеристикой собственно задачи, а не программы для ее решения, и грубое представление о величине расхождения может быть получено на основе рассмотрения самóй задачи в ее первоначальной постановке. В последующих разделах будет показано (см. п. 2.2.4), что соответствующее построение вычислительной системы позволяет удержать величину потерь в эффективности вычислительной системы, возникающих за счет наличия расхождения, в пределах 5—10%. Поэтому ошибки в оценке величины D, которые получаются при грубом рассмотрении задачи в ее первоначальной постановке, в действительности не играют особой роли.

1.2.2. ПАРАЛЛЕЛИЗМ НЕЗАВИСИМЫХ ВЕТВЕЙ

1°. Параллелизм независимых ветвей — наиболее известный тип параллелизма вычислительных задач. Ему посвящено подавляющее большинство теоретических исследований, причем многие авторы склонны рассматривать его чуть ли не как единственный вид параллелизма, который можно использовать для организации параллельных вычислений. Поскольку реальные вычислительные системы во многих случаях ориентированы на другие виды параллелизма, их делят на «истинные» и «не истинные» системы [4].

О параллелизме независимых ветвей мы отчасти уже говорили в п. 1.1.2 в примере, поясняющем понятие пользовательской производительности вычислительной системы.

Суть этого вида параллелизма состоит в том, что в программе решения крупной задачи на тех или иных этапах могут быть выделены независимые части — *ветви*, которые при наличии в вычислительной системе соответствующих средств могут выполняться параллельно (одновременно одна с другой).

Ветвь программы Y не зависит от ветви X, если удовлетворяются четыре условия:

1) между ними нет функциональных связей; иначе говоря, ни одна из входных переменных для ветви Y не является выходной переменной ветви X либо какой-нибудь ветви, зависящей от X;

2) между ними нет связи по рабочим полям памяти; иначе говоря, если ветви программы независимы, они не должны производить запись в одни и те же ячейки памяти;

3) они независимы в программном отношении, т. е. должны выполняться по разным программам;

4) они независимы по управлению; иначе говоря, условие выполнения ветви Y не должно зависеть от признаков, вырабатываемых при выполнении ветви X или какой-либо ветви, зависящей от X.

В частности, условие (3) отличает параллелизм независимых ветвей от параллелизма множества объектов: обработку информации о различных объектах из множества однородных объектов нельзя рассматривать как независимые ветви программы, потому что эта обработка должна выполняться по одной и той же программе; фактически параллелизм множества объектов можно свести к параллелизму независимых ветвей, записав в памяти вычислительной системы столько копий программы, сколько независимых ветвей нужно получить (но, разумеется, не больше, чем количество объектов в множестве). Смысл этих различий между двумя видами параллелизма будет более понятен из п. 1.2.4, 1° и § 2.1.

2°. В теоретических исследованиях и при практическом использовании параллелизма независимых ветвей применяются: аппарат билогических графов, разработанный Г. Эстриным (G. Estrin) и др., язык p-схем Э. В. Евреинова и Ю. Г. Косарева, аппарат ярусно-параллельных форм, предложенный и разработанный Д. А. Поспеловым, близкие к нему идеи Ю. С. Голубева-Новожилова и др. [5—13].

Все эти методы в общем близки между собой. Например, при пользовании аппарата ярусно-параллельных форм программа представляется в виде «ярусов», причем в 0-й ярус входят все те ветви программы, каждая из которых не зависит ни от одной другой ветви, в 1-й ярус — ветви, зависящие только от ветвей 0-го яруса, во 2-й ярус — ветви, зависящие от ветвей 1-го яруса и, может быть, ветвей 0-го яруса и т. д. На рис. 1.2.3 в каче-

стве примера изображена ярусно-параллельная форма некоторой программы. Кружками обозначены ветви программы, внутри кружков проставлены номера ветвей, а рядом с кружками — длины этих ветвей (либо, может быть, математические ожидания длин, если длины ветвей не являются константами). Если одна и та же ветвь программы может быть исполнена n различными средствами вычислительной системы за разные времена, то

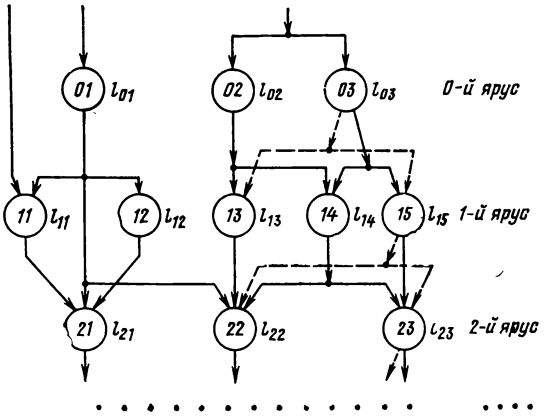


Рис. 1.2.3

вместо одного числа, обозначающего длину ветви, у каждого кружка нужно было бы проставлять n-мерный вектор. Сплошными стрелками обозначены функциональные связи между ветвями, а штриховыми, как и на рис. 1.2.2, — связи по управлению. Сплошная стрелка, входящая в некоторый кружок, обозначает входную переменную или группу входных переменных для соответствующей ветви. Если эта стрелка не выходит ни из какого другого кружка, то она соответствует входным данным программы. Стрелка, выходящая из некоторого кружка, обозначает выходную переменную или группу выходных переменных соответствующей ветви программы. Если эта стрелка не входит ни в какой другой кружок, то она обозначает выходные данные программы. Разветвления стрелок показывают, что одни и те же данные (входные данные программы или данные, получающиеся на выходе какой-либо ветви) являются входными для двух или более ветвей.

Насколько это известно авторам, строгого доказательства возможности представить любую программу в виде ярусно-параллельной формы не существует.

3°. Для того, чтобы оценить по ярусно-параллельной форме программы возможности организации параллельных вычислений, вводится ряд *количественных характеристик* ярусно-параллельной формы: b_i — ширина i-го яруса (т. е. количество независимых ветвей в i-м ярусе), B — ширина графа ярусно-параллельной формы (максимальная ширина яруса, т. е. максимум из b_i, $i=1, 2, \ldots$), l_i и L — длина яруса и длина графа, ε и θ — коэффициент заполнения ярусов и коэффициент разброса и т. д. Если при разных реализациях программы (при разных исходных данных) эти показатели оказываются различными, то вводится ряд дополнительных показателей: например, b^j_i — ширина i-го яруса в j-й реализации программы (при этом $b_i=\max\limits_{(j)}\{b^j_i\}$), B^j — ширина графа ярусно-параллельной формы в j-й реализации программы (при этом $B=\max\limits_{(j)}\{B^j\}$) и т. д. Всего для характеристики ярусно-параллельной формы вводится несколько десятков различных численных показателей, одни из которых относятся к программе в целом, другие — к ее ярусам, третьи — к определенной реализации программы, четвертые — к ярусам этой реализации.

Общим недостатком этой совокупности показателей является то, что она относится не собственно к задаче, а к определенной программе для решения этой задачи. Подбирая различные алгоритмы и составляя разными способами программу для решения одной и той же задачи (даже при использовании одного и того же математического метода решения), можно получить программы с разными характеристиками ярусно-параллельной формы и, следовательно, с разными возможностями для организации параллельных вычислений. Общие методы оптимизации программ (т. е., например, составление программы для любой задачи с максимально возможной шириной графа ярусно-параллельной формы), а также методы определения параметров оптимальной программы без ее разработки (по общему виду формулировки задачи) отсутствуют.

Другим недостатком указанной совокупности характеристик является тот факт, что даже знание численных значений всех характеристик далеко не всегда дает воз-

можность оценить пользовательскую эффективность вычислительной системы, использующей параллелизм независимых ветвей, при решении данной задачи (см. п. 2.2.2).

1.2.3. ПАРАЛЛЕЛИЗМ СМЕЖНЫХ ОПЕРАЦИЙ

1°. Суть *параллелизма смежных операций* состоит в следующем. При исполнении программы довольно часто встречаются ситуации, когда исходные данные для некоторой операции (скажем, i-й по порядку) и условия ее выполнения вырабатываются не при исполнении предыдущей операции ((i—1)-й), а раньше — при исполнении, например, (i—2)- или (i—3)-й и т. д. операции. Если это так, то соответствующее построение вычислительной системы может позволить совместить время исполнения i-й операции с исполнением (i—1)-й или соответственно (i—2)-й и т. д. операции. Вообще, если подготовка исходных данных и условий исполнения i-й операции заканчивается при исполнении (i—k)-й операции (где k=2, 3, ...) то i-ю операцию можно совместить с (i—k+1)-й, (i—k+2)-й, ..., (i—1)-й.

Следует обратить внимание, что в понятие исходных данных и условий исполнения некоторой (i-й по порядку) операции входит нечто большее, чем просто наличие операндов для нее и отсутствие между (i—k)-й и i-й операциями условных передач управления, от исхода которых зависит, должна ли вообще исполняться i-я операция. Если операнды для операции должны читаться из внутренней памяти, то важно, чтобы нужные страницы уже находились в главной памяти и были открыты для обращений к ним, чтобы для формирования исполнительных адресов обращения была готова информация в регистрах, используемых в качестве индексных и базовых, и т. д.; если при исполнении i-й операции происходит засылка информации в некоторую ячейку памяти (сверхоперативной, или главной, или в один из индексных регистров и т. д.), то важно, чтобы ни в одной операции от (i—k+1)-й до (i—1)-й не использовалась эта ячейка памяти ни для чтения из нее, ни для записи в нее. В зависимости от структуры вычислительной системы, по-видимому, необходимо требовать выполнения и других условий аналогичного характера.

2°. Как представляется на первый взгляд, между параллелизмом смежных операций и параллелизмом независимых ветвей (см. п. 1.2.2) нет существенной разницы. Если после исполнения $(i—k)$-й операции можно исполнять i-ю операцию, то i-ю операцию можно представить в виде независимой ветви, принадлежащей тому же ярусу, что и ветвь, содержащая $(i—k+1)$-ю операцию, и т. д.; длина этой ветви, возможно, составляет всего одну операцию, но с теоретической точки зрения это не имеет значения.

Однако в действительности дело обстоит таким образом только с теоретической точки зрения. В § 2.1 мы увидим, что параллелизм независимых ветвей и параллелизм смежных операций реализуются вычислительными системами существенно различной структуры.

При реализации параллелизма независимых ветвей каждый раз, как начинает или заканчивает работу одна из ветвей программы, в системе затрачивается некоторое служебное время, необходимое для проверки наличия признаков, которые разрешают работу данной ветви, и для выработки признаков окончания выполнения данной ветви (позволяющих принять решение о разрешении работы следующих ветвей). Кроме того, определенные служебные потери времени связаны с обменом информацией. В системах с разделенной памятью эти потери выступают в явном виде; в системах, где имеется, скажем, несколько процессоров, работающих над общей памятью, — в виде потерь времени, возникающих при конфликтных ситуациях, когда несколько процессоров одновременно обращается к одному физическому блоку памяти или через общие шины памяти. Определенное время затрачивается также на обращение к диспетчерской программе или операционной системе в те моменты времени, когда пользовательская программа выделяет новую ветвь или когда заканчивается работа некоторой ветви.

Ясно, что эффективное использование параллелизма независимых ветвей возможно при условии, что время работы каждой отдельной ветви будет намного больше, чем указанные служебные потери времени.

Собственно, соотношение времени работы отдельных ветвей и служебных потерь времени, связанных с организацией параллельной работы независимых ветвей, является показателем того, с каким видом параллелизма

мы имеем дело. Если это отношение велико, то мы имеем дело с параллелизмом независимых ветвей. Если оно приближается к единице (например, меньше 4—5), то количество переходит в качество и речь должна идти о параллелизме смежных операций.

3°. *Количественной характеристикой* параллелизма смежных операций, которым обладает та или иная задача, является *показатель связанности* смежных операций α — вероятность того, что результат некоторой операции будет использован в следующей за ней операции. Чем меньше для данной задачи показатель связанности α, тем больше для нее глубина параллелизма смежных операций (величина, которую мы определим ниже).

Если приближенно считать, что вероятность использования результата данной операции в любой последующей операции тоже равна α, то вероятность того, что результат данной операции не будет использован в l последующих операциях и будет использован ровно в $(l+1)$-й операции, окажется равной $\alpha(1-\alpha)^l$. Если $\alpha \ll 1$, то $\alpha(1-\alpha)^l \approx \alpha e^{-\alpha l}$ (закон Пуассона).

Для того, чтобы, начиная от данной операции, имелась цепочка, содержащая не меньше l операций, которые можно исполнить одновременно с данной, должны совпасть следующие события:

— результат данной операции не используется по меньшей мере в $(l-1)$ последующих операциях;

— результат 1-й после данной операции не используется по меньшей мере в $(l-2)$ следующих за ней операциях;

— результат 2-й после данной операции не используется по меньшей мере в $(l-3)$ следующих за ней операциях;

. . . . . .

— результат $(l-2)$-й после данной операции не используется по меньшей мере в одной следующей за ней операции.

Итак, вероятность того, что, начиная от данной операции, имеется цепочка длиной не меньше l операций, которые можно выполнить все одновременно, равна

$$\beta_l = \prod_{i=1}^{l-1} (1-\alpha)^i = (1-\alpha)^{\frac{(l-1)\,l}{2}}.$$

Далее нетрудно определить вероятность того, что, начиная от любой данной операции, в программе имеется цепочка ровно из l операций, которые можно выполнить все одновременно. Если эту вероятность обозначить через γ_l, то, очевидно,

$$\gamma_l = \beta_l (1 - (1 - \alpha)^l) = (1 - \alpha)^{\frac{(l-1)\,l}{2}} (1 - (1 - \alpha)^l) =$$

$$= (1 - \alpha)^{\frac{(l-1)\,l}{2}} + (1 - \alpha)^{\frac{l\,(l+1)}{2}}.$$

Математическое ожидание длины цепочки операций, которые можно исполнять все одновременно, назовем *глубиной параллелизма смежных операций* $L_{\text{псо}}$ для данной программы. Очевидно,

$$L_{\text{псо}} = \sum_{l=1}^{\infty} l\gamma_l = \sum_{l=1}^{\infty} l\left((1 - \alpha)^{\frac{(l-1)\,l}{2}} - (1 - \alpha)^{\frac{l\,(l+1)}{2}}\right) =$$

$$= \sum_{l=1}^{\infty} (1 - \alpha)^{\frac{(l-1)\,l}{2}}.$$

Эти соотношения однозначно связывают глубину параллелизма смежных операций с показателем связанности смежных операций и, казалось бы, незачем вводить два разных числовых показателя для характеристики рассматриваемого вида параллелизма. Однако показатель связанности α имеет смысл только при той гипотезе о распределении вероятностей использования результатов некоторой операции в последующих операциях, которая была принята в начале настоящего пункта. Соответственно и приведенные соотношения, связывающие $L_{\text{псо}}$ с α, справедливы только для принятого вероятностного распределения. С этой точки зрения величина $L_{\text{псо}}$ (глубина параллелизма смежных операций) является более общей характеристикой, так как может быть найдена и для других распределений, если они лучше описывают реальное положение дел для тех или иных конкретных задач.

Собственно, показатель связанности смежных операций α и глубина параллелизма смежных операций $L_{\text{псо}}$, как мы их определили выше, являются характеристиками не собственно задачи, а конкретной программы для

решения этой задачи. В этом отношении введенные количественные характеристики параллелизма смежных операций ничуть не лучше, чем те количественные характеристики параллелизма независимых ветвей, о которых говорилось в п. 1. 2.2, 3°.

Здесь, однако, положение облегчается тем, что получение примерно одних и тех же величин показателя связанности и соответственно глубины параллелизма смежных операций может быть достигнуто путем локальной оптимизации практически любой программы для решения некоторой определенной задачи и даже для большинства различных задач. Локальная оптимизация состоит в том, что просматриваются несколько инструкций программы, которые должны выполняться подряд, и изменяется порядок следования некоторых из них, возможно, изменяются номера регистров или адреса ячеек памяти и т. п., используемых в некоторых инструкциях, с тем, чтобы обеспечить максимально возможный параллелизм смежных операций. Эта оптимизация может выполняться либо вручную, при составлении программы на машинно-ориентированном языке, либо программным путем, при трансляции программы с языка ассемблера на машинный язык (или трансляцией с языка ассемблера на язык ассемблера или с машинного на машинный), либо специальной аппаратурой во время исполнения программы (при ее интерпретации).

Обычные значения величин показателя α, достигаемые при этом, $\alpha \approx 0{,}1—0{,}5$, что зависит не столько от конкретных свойств задачи, сколько от качества выполнения локальной оптимизации. Соответствующие значения $L_{\text{псо}} \approx 4—1{,}6$.

1.2.4. ИСКУССТВЕННЫЙ ПАРАЛЛЕЛИЗМ

Во многих случаях крупные вычислительные задачи, встречающиеся на практике, обладают в той или иной мере всеми перечисленными видами параллелизма. Соответственно с той или иной эффективностью такую задачу можно решать с помощью любой вычислительной системы: ориентированной на использование естественного параллелизма (или параллелизма множества объектов), либо на использование параллелизма независимых ветвей, либо параллелизма смежных операций.

Тем не менее сравнительно часто приходится сталкиваться с практически важными задачами, которые в их

исходной постановке не обладают ни одним из рассмотренных типов параллелизма либо обладают одним каким-нибудь видом параллелизма в то время, как система, с помощью которой должна решаться эта задача, ориентирована на другой вид параллелизма *).

Каких-либо общих методов, позволяющих создать параллелизм того или иного вида в задаче, которая не обладает ни одним из видов параллелизма, либо перейти от одного типа параллелизма к другому, не существует. В этом разделе на двух частных примерах мы рассмотрим некоторые искусственные приемы решения проблем такого рода, причем в первом примере мы будем пользоваться *эквивалентным преобразованием* исходного алгоритма решения задачи, во втором примере — *неэквивалентным преобразованием*. В принципе правила эквивалентных преобразований алгоритмов могут быть формализованы, и алгоритмист либо программно-аппаратный оптимизатор должны лишь найти, какие именно преобразования и в какой последовательности нужно выполнить, чтобы получить требуемый эффект. Неэквивалентные преобразования тесно связаны с существом решаемой задачи и сводятся фактически к подмене одного алгоритма решения данной задачи другим, обладающим требуемым видом параллелизма; такие преобразования целиком выполняются эвристическими методами.

1°. *Преобразование параллелизма множества объектов в параллелизм независимых ветвей.* Пусть задача состоит в том, что по одной и той же программе должна быть обработана информация о большом количестве (r) однотипных объектов; иначе говоря, речь идет о классическом примере задачи, обладающей параллелизмом множества объектов с рангом r (см. п. 1.2.1). Предположим также, что эта задача должна быть решена с по-

*) Неискушенному читателю эта последняя ситуация может показаться нелепой: для чего может понадобиться решать задачу с помощью вычислительной системы, которая для этой задачи заведомо не подходит? Но ведь вычислительные системы не стоят на витрине универмага, где можно приобрести систему на любой вкус; далеко не каждый пользователь имеет доступ к разным вычислительным системам, на которых он мог бы решать различные из имеющихся у него задач. Кроме того, речь может идти о разных частях одной крупной задачи, которые должны все решаться с помощью одной вычислительной системы, но из которых одна часть обладает одним видом параллелизма, другая — другим, а третья, возможно, — ни одним из описанных выше видов параллелизма.

мощью вычислительной системы, ориентированной на параллелизм независимых ветвей, причем структура системы такова, что наибольшая эффективность будет достигнута при ширине всех ярусов, равной b, где $b \ll r$.

Причины, по которым обработку по одинаковой программе информации об r разных объектах нельзя рассматривать как r независимых ветвей, объяснены в п. 1.2.2, п. 1° (об этом см. также ниже). В связи с этим граф ярусно-параллельной формы задачи в ее исходной постановке имеет вид, приведенный на рис. 1.2.4.

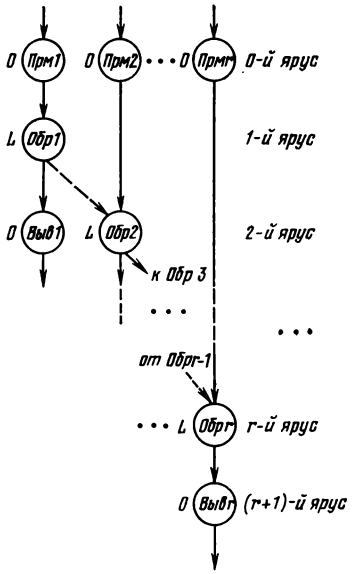


Рис. 1.2.4

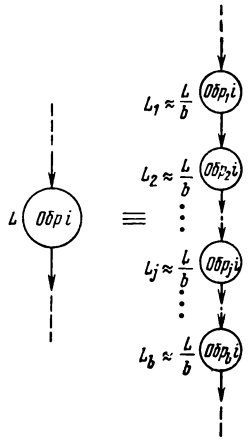


Рис. 1.2.5

В нулевом ярусе расположены фиктивные ветви программы, введенные нами для пояснения логической структуры обработки информации и обозначенные Прм1, Прм2, ..., Прм r. Это — ветви приема информации соответственно для 1-, 2-, ..., r-го объектов. Так же, как фиктивные ветви, расположенные в более низких ярусах и соответствующие выводу результатов по 1-, 2-, ..., r-му объектам (Выв1, Выв2, ..., Выв r), эти ветви исполняются специализированной аппаратурой системы (внешние устройства, каналы), а для основной процессорной части системы длины этих ветвей (время их исполнения) равны нулю.

Основная обработка информации для каждого из r объектов представлена на рисунке в виде одной ветви — соответственно Обр1, Обр2, ..., Обр r — и, как предполагается, параллелизмом независимых ветвей не обладает, вследствие чего не может быть представлена в виде нескольких подветвей, которые можно было бы расположить в одном ярусе. Ветви Обр1, Обр2, ..., Обр r информационно между собой не связаны, но они связаны по управлению, поскольку исполняются все по одинаковой программе (см. условие (3) на с. 22). Эти управляющие связи на рис. 1.2.4 проведены, как обычно, штриховыми линиями. Ввиду наличия указанных связей ветви Обр1, Обр2, ..., Обр r расположены в разных ярусах ярусно-параллельной формы, а общая длина графа рис. 1.2.4 равна Lr.

Произведем теперь эквивалентные преобразования алгоритма (и программы) решения задачи, которые позволяют эффективно использовать вычислительную систему, ориентированную на параллелизм независимых ветвей.

Прежде всего ветвь обработки информации об i-ом объекте Обр i (для $i=1, 2, \ldots, r$) представим в виде последовательных ветвей (фактически подветвей) $\text{Обр}_1 i$, $\text{Обр}_2 i$, ..., $\text{Обр}_j i$, ..., $\text{Обр}_b i$ — так, как это показано на рис. 1.2.5. Ветви Обр i разобьем на подветви так, чтобы длины подветвей были примерно одинаковы: $L_1 \approx L_2 \approx \ldots \approx L_b \approx L/b$.

Некоторая неравномерность в длинах подветвей возникает за счет того, что минимальным дискретом разделения могут быть машинные операции, длины которых, возможно, неодинаковы, а также за счет того, что длины отдельных подветвей, возможно, зависят от исходных данных. Кроме того, при использовании определенных структур вычислительных систем, ориентированных на параллелизм независимых ветвей, может потребоваться минимизация или хотя бы выравнивание объемов информации, передаваемых от j-й подветви к $(j+1)$-й (для $j=1, 2, \ldots, b-1$).

Заметим далее, что две ветви, $\text{Обр}_{j_1} i_1$ и $\text{Обр}_{j_2} i_2$ (i_1, $i_2=1, 2, \ldots, r$; $j_1, j_2=1, 2, \ldots, b$), при $i_1=i_2$ и $j_1 \neq j_2$ связаны между собой информационно; при $j_1=j_2$ и $i_1 \neq i_2$ они связаны общей программой, а при $i_1 \neq i_2$ и $j_1 \neq j_2$ полностью независимы.

С учетом этого граф ярусно-параллельной формы исходной программы, представленный на рис. 1.2.4, можно преобразовать к виду, показанному на рис. 1.2.6. Для простоты при построении рис. 1.2.6 предполагалось, что $b=3$; читатель без труда представит себе, как выглядел бы граф ярусно-параллельной формы при других значениях b.

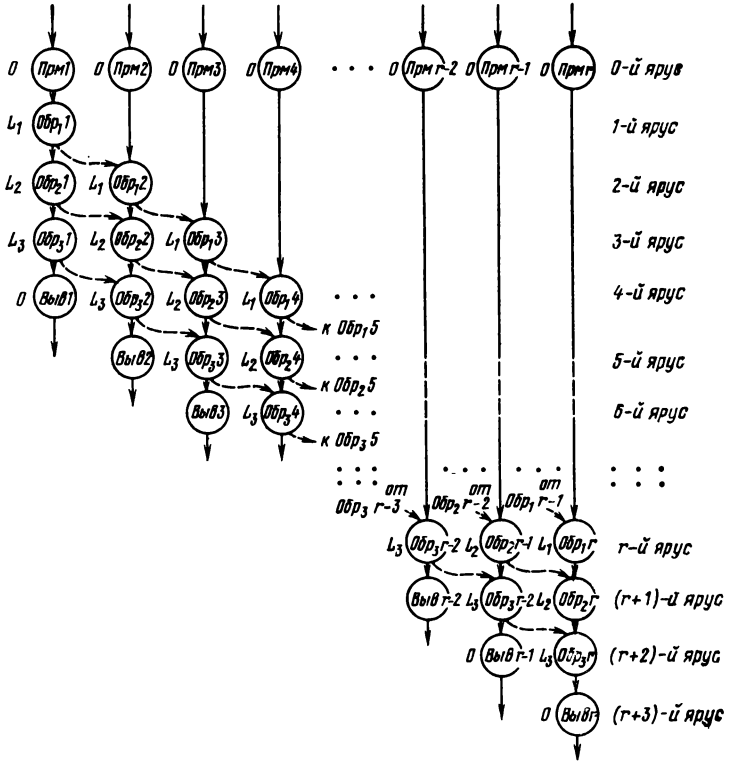


Рис. 1.2.6

После преобразования граф ярусно-параллельной формы содержит $r+b-1$ непустых ярусов (т. е. ярусов конечной длины). Ширина каждого из этих ярусов, кроме первых ($b-1$) и последних ($b-1$) ярусов, равна в точности b (фиктивная ветвь Выв i нулевой длины не

учитывается). Общая длина графа при этом равна

$$L_G = L_1 + \max\{L_1, L_2\} + \ldots + \max\{L_1, L_2, \ldots, L_{b-1}\} +$$
$$+ (r - b + 1)\max\{L_1, L_2, \ldots, L_b\} +$$
$$+ \max\{L_2, L_3, L_4, \ldots, L_b\} + \max\{L_3, L_4, \ldots, L_b\} + \ldots + L_b.$$

Если $L_1 \approx L_2 \approx \ldots \approx L_b \approx \frac{L}{b}$, то общая длина графа несколько превышает величину $\frac{r+b-1}{b} L$. Приняв при этом условие $r \gg b$, получим $L_G \approx L/b$.

Представим себе, как мы это делали в § 1.1, вычислительную систему в виде соединения n машин.

Непосредственно использовать в такой системе параллелизм множества объектов неудобно по многим причинам.

Прежде всего ясно, что если бы мы просто попытались на каждую машину возложить обработку информации по r/n объектам, то потребовались бы какие-то дополнительные устройства системы, которые распределяли бы между машинами входную информацию, поступающую об r объектах, и далее собирали бы выходную информацию. Скорее всего для этого потребовалась бы дополнительная $(n+1)$-я машина, которая вела бы весь внешний обмен, передавала принятую информацию другим n машинам и принимала от них информацию для передачи на выход; эти излишние передачи информации требовали бы дополнительного времени и вели бы к снижению эффективности системы.

Главное же возражение против такого использования системы состоит в том, что оно требует размещения в памяти каждой из n машин своего экземпляра полной программы. Таким образом, общий объем памяти для хранения программной информации в системе был бы в n раз больше, чем это необходимо, причем во всей этой памяти хранилась бы просто n раз одна и та же информация. Если учесть, что и все n устройств управления всех n машин выполняли бы одни и те же операции, то станет ясно, насколько неэффективно использовалось бы оборудование системы в этом случае.

Смысл выполненного выше преобразования программы состоит в преобразовании параллелизма множества

объектов в параллелизм независимых ветвей, т. е. в тот вид параллелизма, который более всего подходит для рассматриваемой системы. Приняв $b=n$ (фактически b можно было выбирать произвольно, лишь бы соблюдалось условие $b \ll r$), работу системы далее организуем следующим образом.

Информацию обо всех r объектах будем принимать всегда в 1-ю машину; если возможно, то принимать информацию по каждому следующему объекту будем после того, как эта машина закончит использование входной информации по предыдущему объекту, чтобы не увеличивать объем внутренней памяти, необходимый для хранения входной информации, и избежать необходимости модификации адресов (в других машинах модификация адресов не потребуется в любом случае). Эта же первая машина будет выполнять последовательно ветви $\text{Обр}_1 1$, $\text{Обр}_1 2$, ..., $\text{Обр}_1 r$. Закончив выполнение ветви $\text{Обр}_1 i$ (где $i=1, 2, \ldots, r$), 1-я машина передает выходную информацию этой ветви 2-й машине, которая последовательно выполняет ветви $\text{Обр}_2 1$, $\text{Обр}_2 2$, ..., $\text{Обр}_2 r$, и т. д. Последняя, n-я машина ($n=b$) последовательно выполняет ветви $\text{Обр}_b 1$, $\text{Обр}_b 2$, ..., $\text{Обр}_b r$ и ведет выдачу выходной информации.

Таким образом, n машин вычислительной системы образуют как бы конвейер, цикл работы которого равен

$$\max\{L_1, L_2, \ldots, L_b\}+\Delta,$$

где Δ — интервал времени, в течение которого одновременно от каждой j-й машины ($j=1, 2, \ldots, n-1$) происходит передача информации к следующей $(j+1)$-й машине; при этом, если в конце некоторого цикла от 1-й машины во 2-ю передаются результаты работы ветви $\text{Обр}_1 i$, то одновременно от 2-й машины в 3-ю передаются результаты работы ветви $\text{Обр}_2 (i-1)$, от 3-й машины к 4-й — ветви $\text{Обр}_3 (i-2)$ и т. д. Если $L_j \approx L/b$ для $j=1, 2, \ldots, b$ и если $\Delta \ll L/b$, то цикл работы конвейера лишь немного превышает величину L/b.

Каждая из n машин системы хранит примерно $1/n$ часть программы: 1-я машина — для ветви $\text{Обр}_1 i$, 2-я — для ветви $\text{Обр}_2 i$ и т. д. Суммарное количество программной информации, хранимой в памяти вычислительной системы, примерно такое же, как было бы в обычной вычислительной машине; небольшие дополнительные объемы памяти для хранения программ связаны с необ-

ходимостью организовать обмен информацией между машинами и синхронизацию конвейера.

Заметим, что предлагаемое здесь средство преобразования параллелизма множества объектов в параллелизм независимых ветвей не является универсальным. Условием, когда оно применимо, является достаточно большая длина L программы обработки информации по каждому из объектов. Если L мала, а необходимость применения вычислительной системы связана с очень большим рангом задачи r, то разделить программу на необходимое количество примерно равных подветвей b, может быть, просто нельзя. Либо может оказаться, что подветви получаются слишком короткими и потери, связанные с неравномерностью ветвей, а также на обмен информацией и синхронизацию, несоразмерно велики.

2°. *Искусственное создание параллелизма множества объектов.* На практике встречаются большие задачи, не обладающие ни одним из видов параллелизма (кроме, может быть, параллелизма смежных операций, глубина которого, как мы говорили, невелика).

Не претендуя на общность изложения, рассмотрим в качестве примера чисто последовательностной задачи нахождение корня X алгебраического или трансцендетного уравнения

$$f(x)=0,$$

которое выполняется с помощью итерационной схемы, построенной по методу Ньютона (метод касательных):

$$x_{i+1}=x_i-\frac{f(x_i)}{f'(x_i)},$$

где $x_0, x_1, \ldots, x_i, x_{i+1}, \ldots$ — последовательность, сходящаяся к X.

Если известно, что значение X расположено в интервале (a, b), а начальное приближение x_0 выбрано в середине этого интервала,

$$x_0=\frac{b-a}{2},$$

то достаточными условиями сходимости итераций являются наличие отличных от нуля первой и второй производных $f(x)$ во всем интервале (a, b) и выполнение неравенства

$$\frac{M}{4m}(b-a)<1,$$

где M и m — такие числа, что

$$|f'(x)|>m,\ |f''(x)|<M \text{ при } a\leqslant x\leqslant b.$$

При этом погрешность начального приближения

$$\delta_0<\Delta_0=\frac{b-a}{2},$$

погрешность первого приближения

$$\delta_1 < \Delta_1 = \frac{M}{2m}\Delta_0^2 = \frac{2m}{M}\left(\frac{M}{4m}(b-a)\right)^2,$$

. . . . . .

погрешность i-го приближения

$$\delta_i < \Delta_i = \frac{2m}{M}\left(\frac{M}{4m}(b-a)\right)^{2^i},$$

где через Δ_i обозначена верхняя оценка δ_i $(i=0, 1, \ldots)$.

При реализации этого алгоритма на обычной однопроцессорной вычислительной машине необходимое количество циклов итераций, $k^{(1)}$, определяется из условия

$$\Delta_{k^{(1)}} \leqslant \varepsilon,$$

где ε — допустимая погрешность окончательного значения X.

Предположим далее, что длина каждого цикла итераций L велика (из-за того, что вычисление $f(x_i)$ и $f'(x_i)$ весьма громоздко), вследствие чего суммарное время счета на обычной машине $(Lk^{(1)})$ оказывается недопустимо большим.

Пусть в нашем распоряжении имеется вычислительная система, ориентированная на естественный параллелизм или параллелизм множества объектов, которая могла бы вычислить выражения

$$x - \frac{f(x)}{f'(x)}$$

для n различных значений x одновременно — примерно за то же время, за какое это выражение вычисляется обычной машиной для одного значения x.

Попробуем использовать эту систему для ускорения получения конечного результата.

С этой целью произведем неэквивалентное преобразование алгоритма [14]. Исходный интервал (a, b) разделим на n равных интервалов длиной $(b-a)/n$; в середине каждого интервала выберем свое начальное приближение $x_{01}, x_{02}, \ldots, x_{0n}$. В первом цикле итераций вычислительная система вычислит n величин $\tilde{x}_{1j}$,

$$\tilde{x}_{1j} = x_{0j} - \frac{f(x_{0j})}{f'(x_{0j})} \quad (\text{для } j = 1, 2, \ldots, n).$$

Из условий сходимости итераций вытекает, что $f(x)$ монотонна и во всем интервале (a, b) либо вогнута вверх, либо вогнута вниз. Поэтому все вычисленные значения $\tilde{x}_{1j}$ расположены либо все слева, либо все справа от искомого значения X; ближе всего к X находится соответственно $\max\limits_{(j)}\{\tilde{x}_{1j}\}$ или $\min\limits_{(j)}\{\tilde{x}_{1j}\}$ причем отклонение этой величины от X не превышает, очевидно,

$$\tilde{\Delta}_1 = \frac{1}{n^2}\frac{2m}{M}\left(\frac{M}{4m}(b-a)\right)^2 = \frac{1}{n^2}\Delta_1.$$

Обозначим соответственно

$$\max_{(j)}\{\tilde{x}_{1j}\} = b_1, \quad \tilde{b}_1 - \tilde{\Delta}_1 = a_1$$

либо

$$\min_{(j)}\{\tilde{x}_{1j}\} = a_1, \quad \tilde{a}_1 + \tilde{\Delta}_1 = b_1.$$

Искомый корень X лежит теперь в интервале (a_1, b_1).

Разделим далее интервал (a_1, b_1) на n равных частей. Можно было бы принять середины этих подынтервалов за набор из n первых приближений (подобно тому, как ранее мы получили набор из n начальных приближений). Однако есть опасность, что оценка $\widetilde{\Delta}_1$ сильно завышена и что уход от $b_1=\max\limits_{(j)}\{\tilde{x}_{1j}\}$ (или соответственно от $a_1=\min\limits_{(j)}\{\tilde{x}_{1j}\}$) к середине ближайшего подынтервала может ухудшить точность первого приближения. Поэтому в качестве набора первых приближений $x_{11}, x_{12}, \ldots, x_{1n}$ выберем само значение b_1 (или соответственно a_1) и граничные точки остальных подынтервалов. Где бы в интервале (a_1, b_1) ни было расположено искомое значение X, всегда найдется по меньшей мере одно такое x_{1j}, которое отстоит от X не более чем на

$$\frac{\widetilde{\Delta}_1}{n}=\frac{\Delta_1}{n^2}.$$

Далее с помощью вычислительной системы произведем одновременно вычисление n величин $\tilde{x}_{2j}$:

$$\tilde{x}_{2j}=x_{1j}-\frac{f(x_{1j})}{f'(x_{1j})},$$

из которых по крайней мере одна (максимальная либо минимальная) отклоняется от X не более чем на

$$\widetilde{\Delta}_2=\frac{1}{n^6}\frac{2m}{M}\left(\frac{M}{4m}(b-a)\right)^{2^2}=\frac{1}{n^6}\Delta_2.$$

Действуя дальше аналогичным образом, получим после i-го цикла итерации

$$\widetilde{\Delta}_i=\frac{1}{(n^2)^{2^i-1}}\Delta_i.$$

Необходимое количество циклов итерации при использовании вычислительной системы, вычисляющей одновременно n очередных приближений, $k^{(n)}$ определяется из условия

$$\widetilde{\Delta}_{k^{(n)}}\leqslant\varepsilon.$$

В любом случае $k^{(n)}\leqslant k^{(1)}$, т. е. рассматриваемая система дает выигрыш в скорости по сравнению с обычной (однопроцессорной) машиной, причем отношение $k^{(1)}/k^{(n)}$ может быть и меньше, и больше n. Оно зависит от соотношения величин m и M (нижняя грань $|f'(x)|$ и верхняя грань $|f''(x)|$) длины интервала $b-a$, требуемой точности ε и величины n.

Если величины $\frac{M}{4m}(b-a)$ и n заданы, то максимум этого отношения, $k^{(1)}/k^{(n)}=(k^{(1)}/k^{(n)})_{\max}$ достигается при том значении ε, которому соответствует $k^{(n)}=1$. Так получается потому, что в последовательности $\widetilde{\Delta}_1, \widetilde{\Delta}_2, \ldots$ дискреты (отношения соседних членов) существенно крупнее, чем в последовательности $\Delta_1, \Delta_2, \ldots$.

При прочих равных условиях чем ближе к единице величина $\frac{M}{4m}(b-a)$, т. е. чем медленнее сходятся итерации, тем больше может быть отношение $k^{(1)}/k^{(n)}$. Это видно из следующей таблицы:

$\frac{M}{4m}(b-a)$	$(k^{(1)}/k^{(n)})_{\max}$		
	$n=4$	$n=8$	$n=16$
0,99	8	9	9
0,999	11	11	12

В некоторых случаях, как мы говорили, может оказаться, что $k^{(1)}/k^{(n)}>n$. В частности, в таблице выделены жирным шрифтом те значения $(k^{(1)}/k^{(n)})_{\max}$, которые больше n. Эти случаи соответствуют тем относительно редким ситуациям, о которых говорилось в п. 1.1.2, когда пользовательская эффективность вычислительной системы получается больше единицы (в таблице видно, что она может быть равна 2 и даже почти 3). При этом область, где достигается такой выигрыш (при прочих равных условиях) тем шире, чем меньше n.

2. СТРУКТУРЫ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

В настоящей главе рассматриваются и сравниваются между собой различные структуры вычислительных систем. Полагая по-прежнему, что главной целью создания системы является достижение высокого реального пользовательского быстродействия $S_{p(\pi)}$ при приемлемых значениях эффективности системы E_π *), мы попробуем показать, каким образом структура системы должна соответствовать конкретным свойствам задач, которые будут решаться с ее помощью. Рассматриваются и некоторые другие особенности разных типов вычислительных систем.

2.1. ПРИНЦИПЫ ОРГАНИЗАЦИИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

2.1.1. МНОГОМАШИННЫЕ КОМПЛЕКСЫ И МНОГОПРОЦЕССОРНЫЕ СИСТЕМЫ С РАЗДЕЛЬНЫМ УПРАВЛЕНИЕМ (СИСТЕМЫ ТИПОВ I И II)

1°. *Многомашинные комплексы* — наиболее давно известный и широко используемый вид вычислительных систем. Именно вычислительные системы этого типа рас-

*) Определение этих понятий см. п. 1.1.2, 2°.

сматривались нами в § 1.1 в качестве примера для пояснения различных понятий и определений, вводившихся для вычислительных систем вообще, а также в п. 1.2.4, 1°.

Многомашинный комплекс представляет собой соединение нескольких независимых машин, каждая со своим центральным процессором, каналами, внешними устройствами, операционной системой. Большей частью (если целью создания комплекса является достижение высо-

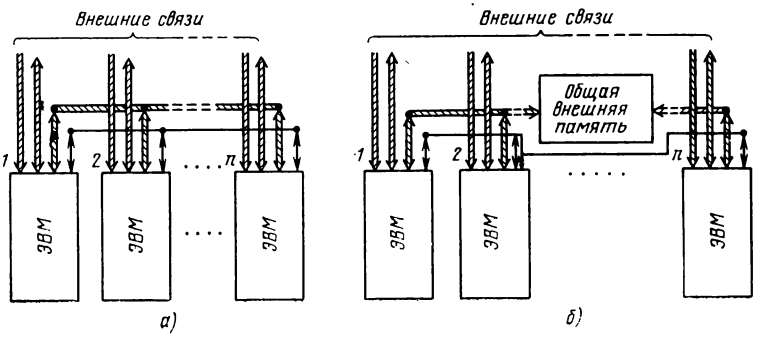


Рис. 2.1.1

кой пользовательской производительности для широкого класса задач или повышение надежности) все машины в комплексе одинаковы. Информационные связи между ними осуществляются по принципу «канал — канал» (рис. 2.1.1,*а*) либо через общее поле памяти. При этом, как правило, речь идет об общем поле внешней памяти, т. е. об общих магнитных дисках или лентах и т. п. (рис. 2.1.1,*б*).

Управляющие воздействия от одной машины к другой могут передаваться либо по тем же цепям, что и информация (в особенности, если речь идет об информационных связях «канал — канал»), либо по отдельным цепям, показанным на рис. 2.1.1 тонкими линиями. Например, управляющие воздействия для других машин могут формироваться специальными внешними устройствами, присоединенными к каналам данной машины, а управляющие воздействия от других машин восприниматься непосредственно системой прерывания программ, имеющейся в составе устройства центрального управления данной машины.

Конфигурация информационных и управляющих связей на рис. 2.1.1 намеренно не показана. Здесь возможно множество различных вариантов. Применяются, например, отдельные соединения каждой машины с любой другой; связи через общие магистрали, когда каждая из машин может выдать информацию или управляющие воздействия на любую свободную магистраль и осуществить прием с любой магистрали; ограниченные связи, например, каждой машины только с двумя соседними; «веерные» («радиальные») связи, когда одна из машин комплекса связана со всеми остальными, а все остальные машины совсем не связаны или почти не связаны друг с другом.

Веерные (радиальные) связи характерны для многомашинных комплексов с так называемым *централизованным управлением*, — в которых все функции управления системой возложены на одну *ведущую* машину. Ведущая машина может быть того же типа, что и все остальные машины комплекса («ведомые»), а может быть и другого типа.

Наряду с многомашинными комплексами с централизованным управлением известны многомашинные комплексы с *распределенным управлением*, в которых функции управления системой распределены между всеми машинами комплекса.

В дальнейшем многомашинные комплексы будем называть вычислительными системами *типа I*.

2°. *Многопроцессорные системы с раздельным управлением* представляют собой следующий шаг по пути интеграции вычислительных средств в единую вычислительную систему по сравнению с многомашинными комплексами.

Одно из возможных построений многопроцессорной системы с раздельным управлением показано на рис. 2.1.2. Система состоит из n процессоров, каждый из которых содержит свое устройство управления (У), арифметико-логическое устройство (А) и свое устройство памяти (П). Все процессоры связаны между собой через общую память, и с ней же связаны общие для всех процессоров каналы внешнего обмена.

В отличие от многомашинных комплексов, где общая память, если она имеется (см. рис. 2.1.1), является скорее всего внешней памятью, здесь общая память — это часть внутренней памяти системы. Возможно, что

устройства раздельной памяти в составе каждого процессора вообще отсутствуют, либо представляют собой только сверхоперативную память, либо включают в себя как сверхоперативную, так и главную память; соответственно общее поле памяти либо включает в себя все ступени внутренней памяти, либо главную память и большую (внутреннюю память 1-й ступени), либо только большую память *).

Возможны и такие построения многопроцессорных систем с раздельным управлением, в которых общей внутренней памяти вообще нет, а обмен информацией между процессорами идет через специальные цепи. При этом обобществленные каналы адресуются ко всей внутренней памяти системы, указывая в качестве адреса обращения номер процессора и адрес его памяти. Такая многопроцессорная система по своей структуре очень близка к многомашинному комплексу.

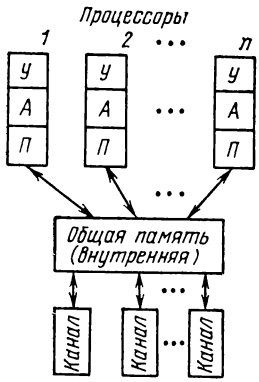


Рис. 2.1.2

В дальнейшем многопроцессорные системы с раздельным управлением будем называть вычислительными системами *типа II*.

Вычислительные системы типа I и II иногда называют системами *с множественным потоком команд и множественным потоком данных* («МПК — МПД») — см., например, [2]. По этой терминологии обычная однопроцессорная машина должна называться системой *с одиночным потоком команд и одиночным потоком данных* («ОПК—ОПД»), а многопроцессорные системы с общим управлением, ориентированные на естественный параллелизм, которые мы рассмотрим в п. 2.1.3 (где они называются системами типа IV) — системами *с одиночным потоком команд и множественным потоком данных* («ОПК—МПД»). При этом остается неясным, как следует называть многопроцессорные системы с общим управлением, ориентированные на параллелизм смежных

*) Здесь используется та же терминология, что и в [1].

операций (см. п. 2.1.2, системы типа III), а также какие системы следует относить к классу «множественный поток команд—одиночный поток данных» («МПК—ОПД»). Вообще эта терминология нам не представляется достаточно удачной.

3°. *Области применения.* Будем считать, как мы условились в § 1.1, основной целью создания вычислительной системы достижение высокой пользовательской производительности.

Ясно, что системы типа I и II лучше всего приспособлены к использованию параллелизма независимых ветвей. При этом возможности системы будут использованы наилучшим образом, если ширина каждого яруса ярусно-параллельной формы b будет в точности равна или кратна количеству машин в многомашинном комплексе либо количеству процессоров n в многопроцессорной системе с раздельным управлением: $b=kn$, где $k=1, 2, 3, \ldots$ (натуральное число).

Кроме того, важно, чтобы все ветви программы, входящие в один i-й ярус, были по возможности одинаковой длины l_i и чтобы эта длина l_i была достаточно велика.

Конечно, равенство или кратность ширины всех ярусов количеству машин (процессоров) в системе и равные длины всех ветвей в ярусе сами собой могут получиться только в силу стечения почти невероятных обстоятельств. На такую ситуацию можно рассчитывать либо в специализированной системе, построенной для решения определенной задачи (определенного класса задач), либо как на результат специального преобразования программы решения задачи — например, такого, как было описано в п. 1.2.4, 1°.

При использовании систем указанного вида лучше было бы рассчитывать на ситуацию другого рода — когда при прохождении программы среднее по времени количество исполняемых и готовых к исполнению ветвей значительно больше, чем количество n машин (процессоров) в системе (подробнее эти вопросы рассматриваются в п. 2.2.2).

Поэтому в большинстве случаев вычислительные системы рассматриваемых типов с относительно большим количеством машин (процессоров) используются для достижения высокой системной производительности. Если при этом в потоке задач, проходящих через систему, встречается задача, которая могла бы эф-

фективно использовать n_1-машинную или n_1-процессорную систему ($n_1 \leqslant n$), то ей выделяется подсистема из n_1 машин или соответственно n_1 процессоров. Таким образом попутно — там, где это возможно — решается проблема повышения пользовательской производительности.

4°. Причины, по которым в многомашинных комплексах и в многопроцессорных системах с раздельным управлением нельзя использовать параллелизм смежных операций, ясны непосредственно из определения параллелизма смежных операций (см. п. 1.2.3, 2°).

В п. 1.2.4, 1° на одном частном примере были показаны обстоятельства, препятствующие использованию в системах рассматриваемого типа параллелизма множества объектов. Речь шла о трудностях распределения работ (и соответственно входной информации об объектах) между машинами многомашинного комплекса, сбора выходной информации о разных объектах от разных машин для передачи на общий выход и, главное, о необходимости размножать в n экземплярах — по количеству машин в комплексе — программу обработки информации, одинаковую для всех объектов.

В случае, когда мы имеем дело с системой типа II над общей памятью (рис. 2.1.2), программу в принципе можно не размножать, а держать ее в общей памяти, к которой обращались бы устройства управления всех n процессоров системы. Однако при этом постоянно возникали бы конфликты между процессорами при обращениях к общей памяти, в результате чего процессоры работали бы фактически не параллельно, а последовательно во времени и никакого выигрыша в пользовательской производительности мы не получили бы.

Можно себе представить, конечно, что главная память, входящая в состав общей памяти системы, содержит большое количество ($\geqslant n$) блоков с независимым обращением и что адресация этой памяти организована так, что номер блока указывается младшими разрядами адреса (иначе говоря, последовательные инструкции находятся всегда в разных блоках). Тогда ситуация могла бы сложиться следующим образом.

Допустим, что при исполнении некоторой инструкции — например 1-й, возник конфликт между всеми n процессорами, которые все одновременно обращаются за этой инструкцией, скажем, к 1-му блоку памяти. Инст-

рукцию получает, конечно, только один из процессоров — например 1-й, все остальные в данном такте простаивают. Но уже в следующем такте 1-й процессор обращается за 2-й инструкцией ко 2-му блоку памяти, а из 1-го блока памяти 1-ю инструкцию получает 2-й процессор; затем 1-й процессор получает 3-ю инструкцию из 3-го блока памяти, 2-й — вторую инструкцию из 2-го блока памяти, 3-й процессор — 1-ю инструкцию из 1-го блока памяти и т. д. Через n тактов все процессоры выстраиваются в цепочку, все они выполняют одну и ту же программу со сдвигом по времени друг относительно друга на один такт (на одну инструкцию) и конфликтов между ними не возникает.

Однако столь благополучная ситуация может существовать только на длинных линейных участках программы. Любая передача управления в программе с высокой вероятностью вызывает новые конфликты при обращениях к общей памяти и новые потери времени. Действительно, если в результате выполнения передачи управления 1-й процессор обращается за очередной инструкцией к тому блоку памяти, к которому должен был обратиться i-й процессор ($n>i>1$), и, как и прежде, имеет приоритет, то в данном такте простаивает i-й процессор; но в следующем такте к тому же блоку памяти обратится 2-й процессор, а простаивать будут уже i- и $(i+1)$-й процессоры, и т. д. Аналогичным образом цепочка процессоров, выбирающих без потерь времени, такт за тактом последовательные инструкции из разных блоков памяти, разрушается обращениями за данными — например, за константой, расположенной в одном из блоков памяти (где расположены и некоторые из инструкций программы) и нужной последовательно всем процессорам.

Кроме того, идея нумерации физических блоков памяти младшими разрядами адреса приводит к тому, что любая программа и ее данные оказываются расписанными по всем физическим блокам памяти, что противоречит идее сохранения живучести комплекса за счет наличия в его составе большого числа однотипных модулей (в частности, модулей памяти). Повреждение хотя бы одного блока памяти приводит к повреждению всех программ, хранимых в общей памяти, в том числе, возможно, и программ операционной системы, которые как раз и должны обеспечивать живучесть средств.

Если бы речь шла не о параллелизме множества объектов, а о более общем типе естественного параллелизма, то дополнительные трудности в системах типа I и II вызвала бы реализацию «интегральных» операций (см. п. 1.2.1, 2°).

В силу указанных обстоятельств структуры вычислительных систем, рассмотренные в настоящем разделе, не могут эффективно использовать ни естественный параллелизм (в частности, параллелизм множества объектов), ни параллелизм смежных операций, но ориентированы главным образом на параллелизм независимых ветвей.

2.1.2. МНОГОПРОЦЕССОРНЫЕ СИСТЕМЫ С ОБЩИМ УПРАВЛЕНИЕМ, ОРИЕНТИРОВАННЫЕ НА ПАРАЛЛЕЛИЗМ СМЕЖНЫХ ОПЕРАЦИЙ (СИСТЕМЫ ТИПА III)

1°. *Принцип построения* многопроцессорной системы с общим управлением, ориентированной на использование параллелизма смежных операций, иллюстрируется рис. 2.1.3.

Под общим управлением У собрано n арифметико-логических процессоров А. Каждый из них может иметь свою небольшую память, но может и не иметь ее (на рисунке блоки раздельной памяти П показаны штриховыми линиями). В основном процессоры работают над общей памятью. Из нее же устройством управления У прочитываются инструкции, по которым оно работает.

Имеется два принципиальных варианта организации работы такой системы.

Первый вариант состоит в том, что устройство управления прочитывает инструкции «вперед» со скоростью, намного превышающей скорость исполнения одной инструкции одним арифметико-логическим процессором. Прочитав очередную инструкцию, устройство управления анализирует, имеются ли условия для того, чтобы начать ее исполнение, если возможно, — поручает ее исполнение любому свободному процессору (либо первому освобождающемуся процессору, если все они были заняты), анализирует следующую инструкцию, и т. д. Просмотр программы «вперед» и исполнение отдельных ее инструкций может продолжаться и в том случае, когда выясняется, что исполнение очередной инструкции нужно отло-

жить; инструкции, исполнение которых отложено, накапливаются в буфере устройства управления.

Во втором варианте устройство управления прочитывает из памяти непосредственно векторы-инструкции. Один вектор-инструкция содержит в пределе n компонент — по числу процессоров, каждая из которых указывает, какую операцию должен выполнить соответствующий процессор.

В первом варианте анализ программы и организация параллельных вычислений производится устройством управления непосредственно на этапе исполнения программы, методом интерпретации, во втором варианте — на этапе программирования, при трансляции программы на машинный язык. По-видимому, оптимизация программы методом трансляции может быть выполнена более эффективно, чем методом интерпретации, поскольку при трансляции можно обеспечить просмотр программы вперед на большее количество шагов.

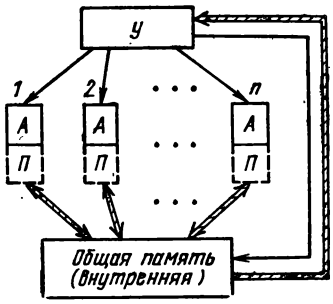


Рис. 2.1.3

В любом случае характерной особенностью систем рассматриваемого типа является тот факт, что несмотря на наличие общего для всех n процессоров устройства управления, все они управляются разными инструкциями или разными компонентами вектора-инструкции.

В дальнейшем такие вычислительные системы будем называть системами *типа III*.

2°. Требования к памяти, предъявляемые одним и другим вариантом на первый взгляд кажутся разными: в первом варианте требуется высокое быстродействие памяти (чтобы инструкции можно было читать намного быстрее, чем идет их исполнение), во втором — требуется широкий формат обращения к памяти, — чтобы можно было за одно обращение прочесть вектор-инструкцию. В действительности требования эти в значительной мере взаимозаменяемы. Если возможен широкий формат обращения к памяти, то для первого варианта несколько последовательных инструкций могут быть получены за

одно обращение к памяти, время выполнения которого при этом может быть не на много меньше, чем время исполнения инструкции процессором. С другой стороны, если быстродействие памяти намного больше быстродействия процессорной части, то вектор-инструкция для второго варианта может быть прочитан из памяти за несколько обращений при обычном формате обращения.

Необходимо учесть, что когда мы говорим «высокое быстродействие» или «широкий формат» памяти, то эти понятия связаны с количеством процессоров n. «Высокое быстродействие» — это время обращения к памяти примерно в n раз меньшее, чем время выполнения операции в процессоре, «широкий формат» — это возможность выборки ориентировочно n слов за одно обращение. Поскольку глубина параллелизма смежных операций обычно невелика, количество процессоров в системе типа III никогда не делается больше, чем 2—6 (см. п. 2.2.3). Если, скажем, n=4, то реализация быстродействия памяти в 4 раза более высокого, чем у процессора, или формата обращения до четырех слов не связана с особыми затруднениями.

2.1.3. МНОГОПРОЦЕССОРНЫЕ СИСТЕМЫ С ОБЩИМ УПРАВЛЕНИЕМ, ОРИЕНТИРОВАННЫЕ НА ЕСТЕСТВЕННЫЙ ПАРАЛЛЕЛИЗМ (СИСТЕМЫ ТИПА IV)

1°. *Принцип построения* многопроцессорной системы с общим управлением, ориентированной на использование естественного параллелизма или, в частности, параллелизма множества объектов, в весьма упрощенном виде показан на рис. 2.1.4. Основное отличие от построения, показанного на рис. 2.1.3, состоит в том, что общее устройство управления У раздает всем n арифметико-логическим процессорам А одинаковые команды для каждого шага, а не различные, как было в построении рис. 2.1.3. Все процессоры одновременно выполняют одну и ту же операцию — каждый над своими данными.

Эти данные выбираются, может быть, каждым процессором из своей отдельной памяти П, если такие устройства памяти существуют (на рисунке они показаны штриховыми линиями), причем все процессоры по одинаковым адресам обращаются каждый к своей памяти. Возможно также, что массив данных выбирается по

одному адресу, передаваемому устройством управления У, из общей памяти (которая должна иметь достаточно широкий формат обращения), а далее каждый элемент массива передается соответствующему процессору.

Многопроцессорные системы рассматриваемого типа будем в дальнейшем называть системами *типа IV*.

2°. Чтобы система типа IV работала достаточно эффективно, в ней желательно предусмотреть ряд важных усовершенствований, которые на рис. 2.1.4 не отражены.

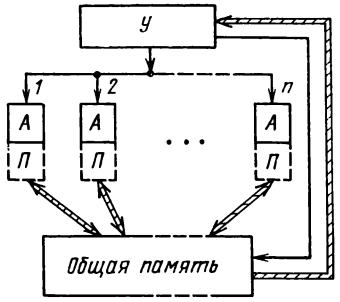


Рис. 2.1.4

Первое из них — относительно простое — состоит в том, чтобы предусмотреть *размножение операндов*, т. е. возможность передачи из общей памяти одного операнда всем процессорам. Термин «размножение» связан с тем, что цепи передачи от общей памяти вообще устроены так, чтобы каждому процессору передавать свой элемент массива; здесь же как бы формируется массив, в котором n раз повторен один и тот же элемент.

В общем случае использования естественного параллелизма размножение операндов необходимо для выполнения таких операций, как умножение вектора на число, умножение матрицы на число и т. п. Представляет интерес также операция сравнения всех значений функции, заданной в узлах решетки, с константой (выделение линии уровня); однако использование этой возможности связано с необходимостью иметь в составе системы также механизм масок, о чем мы скажем ниже.

В частном случае использования параллелизма множества объектов тоже возможны случаи, когда в некоторых операциях по обработке информации о разных объектах множества используется одна и та же константа и поэтому необходим механизм размножения. Существенно более важной представляется однако ситуация, когда в обрабатываемом множестве нужно выделить подмножество объектов, удовлетворяющих определенным условиям. В некоторых случаях для выполнения

операций такого рода в состав системы вводятся специальные *ассоциативные устройства* *). Более универсальным средством решения этой проблемы является передача во все процессоры одновременно, через механизм размножения, некоторых параметров (порогов, эталонов) и выполнение в каждом из процессоров сравнения с этими параметрами тех или иных переменных, характеризующих обрабатываемый в данном процессоре объект. При этом должна быть предусмотрена возможность использовать в дальнейших вычислениях выработанные в каждом из процессоров условия (результаты сравнения), и здесь мы снова подходим к необходимости введения механизма масок.

3°. *Механизм масок* представляет собой второе важное усовершенствование структуры рис. 2.1.4.

В наиболее развитом виде механизм масок содержит специальный процессор, оперирующий с массивами булевых переменных. Входная информация для этого процессора может поступать из основных процессоров — когда в них выполняются операции сравнения (признаки «равно», «больше», «меньше») или при выполнении обычных арифметических операций (признаки «результат равен нулю», «результат меньше нуля», «переполнение разрядной сетки» и т. п.), а также прочитываться из памяти. Под управлением общего устройства управления (по специальным инструкциям или по специальным указаниям в инструкциях) процессор обработки массивов булевых переменных производит различные логические операции над этими массивами. Выходная информация (тоже массивы булевых переменных) либо записывается в память для последующих операций, либо поступает прямо в основные процессоры — каждый разряд массива в свой процессор — на отдельные управляющие входы в качестве «масок».

В тех кодах операции, которые устройство управления передает основным процессорам, должно содержаться дополнительное указание: выполнять ли операцию безусловно, или под прямой маской, или под инверсной маской. Последние указания означают соответственно, что данная операция должна исполняться только в тех процессорах, для которых маска содержит единицу

*) Такое решение применено, в частности, в системе PEPE, см., например, [3, 4].

(**true**), либо только в тех процессорах, для которых маска содержит нуль (**false**); операции в других процессорах при этом блокируются.

Если на некотором, k-м этапе исполнения программы объекты классифицируются не на два, а на большее количество классов m_k, $m_k>2$ (см. п. 1.2.1, 3°), то сначала под маской, которая содержит единицы для объектов 1-го класса и нули для всех других объектов, исполняется ветвь программы, относящаяся к объектам 1-го класса; затем под маской, которая содержит единицы для объектов 2-го класса и нули для всех других объектов, исполняется ветвь, относящаяся к объектам 2-го класса и т. д. Возможен и более сложный вариант, когда устройство управления передает всем процессорам m_k-компонентный вектор-инструкцию, а маска представляет собой вместо массива из r булевых переменных массив из r m_k-ичных величин *).

Таким образом с помощью механизма масок могут быть реализованы расхождения в почти одинаковых программах обработки информации о разных объектах или для разных точек решетчатой функции и т. п., о которых говорилось в п. 1.2.1, 2°, 3°.

4°. Введение *аппаратуры параллельно-последовательной обработки* [8] — это третье усложнение, которое может существенно улучшить эффективность систем рассматриваемого типа.

Рассмотрим его суть для того частного случая, когда используется параллелизм множества объектов. Как будет видно из 2.2.4, наибольшая эффективность систем рассматриваемого типа достигается, когда ранг задачи r в точности равен количеству процессоров, $r=n$, либо, что более вероятно, при $r\gg n$. В последнем случае задача, как обычно, должна решаться циклически: сначала для первой группы из n объектов, затем для следующей группы и т. д., $\left]\frac{r}{n}\right[$ раз, где квадратные скобки] [

*) В зачаточном виде механизм масок присутствовал еще в проекте SOLOMON [5]: каждый из его процессоров содержал «триггер состояния»; специальные операции настройки устанавливали эти триггеры в состояние «активен» или «пассивен»; дальнейшие операции исполнялись только активными процессорами. В одном из ранних вариантов Illiac IV каждый процессор мог с помощью настройки устанавливаться в одно из четырех состояний и соответственно исполнять одну из четырех программ, транслируемых одновременно устройством управления. (Об Illiac IV см., например, [6, 7].)

означают ближайшее не меньшее целое; если r не кратно n, то при прохождении последнего цикла часть процессоров маскируется. В принципе можно циклически повторять либо всю программу вычислений, либо каждую операцию, входящую в программу; однако пооперационные циклы потребовали бы в обычных условиях слишком больших затрат времени и количества инструкций в программе для служебных целей: организации циклов, смены маски в последнем цикле, модификации адресов.

Идея состоит в том, чтобы в состав устройства управления ввести аппаратуру, которая без излишних затрат времени и избыточной информации в программе могла бы организовать пооперационные циклы.

Одно из возможных построений такой аппаратуры в общих чертах показано на рис. 2.1.5.

В регистр r в начале выполнения всей программы обработки информации о множестве объектов специальной операцией заносится количество объектов в обрабатываемом множестве (r). В регистре n хранится количество процессоров в системе; эта величина может быть константой для данной конфигурации системы, а может быть и переменной величиной — зависящей от технического состояния процессоров. Регистр kn в начале каждой операции обнуляется. Любая арифметическая или логическая операция выполняется сначала для первой группы из n объектов; одновременно к содержимому регистра kn добавляется содержимое регистра n, причем на первом шаге в нем образуется величина n. Далее операция выполняется для следующей группы из n объектов, а в регистре kn получается величина $2n$, и т. д. Так продолжается до тех пор, пока компаратор не обнаруживает, что величина $r-kn$ (k — целое) не стала меньше n.

Если $0<r-kn<n$, то последний цикл выполняется под дополнительной маской, формируемой из величины $r-kn$ (действующей, может быть, наряду с обычной маской), после чего устройство управления переходит к выполнению следующей операции. Если $r-kn=0$ (в случае, когда величина r оказалась кратной n), устройство управления сразу переходит к выполнению следующей операции.

Содержимое регистра kn используется также для модификации адресов данных от цикла к циклу (естественно, если данные выбираются без размножения), а содержимое регистра n задает формат обращения к памяти за данными.

Преимущества, достигаемые введением аппаратуры параллельно-последовательной обработки, очевидны:

— инвариантность программы при изменении числа процессоров n (в частности, программа не меняется и для $n=1$, т. е. фактически не отличается от программы для обычной однопроцессорной машины);

— исключение операций проверки окончания цикла, передачи управления к началу цикла, модификации ад-

ресов; если r велико по сравнению с n, а длина программы обработки информации по каждому из объектов l мала, это может дать эффективность системы больше единицы (см. п. 2.2.4);

— уменьшение количества обращений к памяти за инструкциями во время исполнения программы (каждая инструкция читается из памяти один раз вместо $]\frac{r}{n}[$ раз, как было бы при отсутствии указанной аппаратуры).

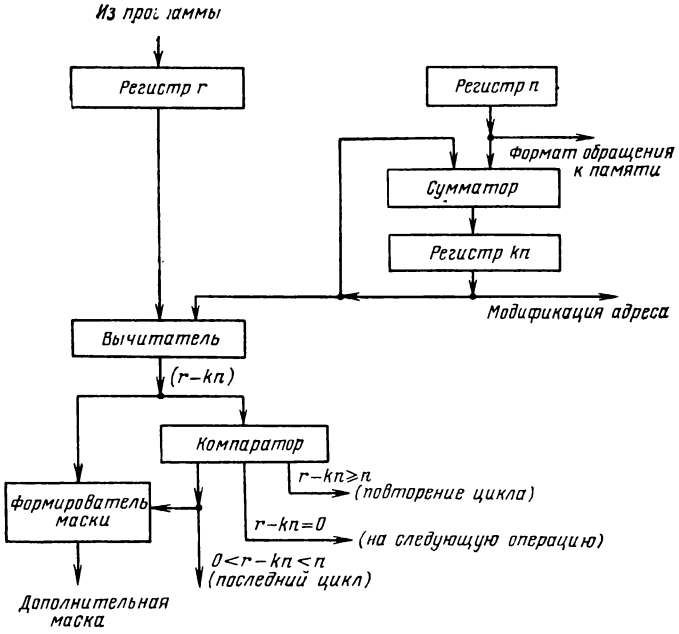


Рис. 2.1.5

5°. Введение в состав системы специального *процессора переформирования массивов* — четвертое усовершенствование, которое желательно предусмотреть в системе, рассмотренной в п. 1°, — в особенности если мы ориентируемся главным образом на параллелизм множества объектов. Цель введения такого процессора — уменьшить потери в эффективности системы, обусловленные наличием более или менее значительных расхож-

дений в программах обработки информации о разных объектах, в условиях, когда $r \gg n$. (Если бы количество объектов r было одного порядка с n, то делить множество объектов на два подмножества и обрабатывать эти подмножества раздельно не имело бы смысла, потому что при обработке таких подмножеств многие процессоры все равно бы простаивали. При использовании естественного параллелизма в общей форме расхождения в программах обычно бывают невелики и, кроме того, существенную информацию несет само расположение данных в массивах — см. примеры в п. 1.2.1.)

В частном случае параллелизма множества объектов расположение данных о некотором объекте в массиве данных обо всех объектах, как правило, не играет роли. При этом небольшие расхождения в программах целесообразно реализовать с помощью механизма масок: после того, как выработана маска, содержащая для объектов одного класса единицы, а для объектов второго класса нули, система выполняет сначала операции для объектов одного класса под прямой маской, а потом операции для объектов другого класса — под инверсной маской.

Если расхождение в программе велико, то выгоднее поступить иначе. Сразу после классификации объектов (получения маски) включается в работу процессор переформирования массивов. Входной информацией для него являются массивы данных об r объектах и массив маски, содержащий r булевых переменных. И то и другое читается из общей памяти системы. Процессор переформирования массивов разделяет каждый массив данных на два подмассива: один — относящийся к объектам, для которых маска содержит единицы, другой — к объектам, для которых маска содержит нули, и в таком виде записывает их в память. Одновременно определяются размеры этих подмассивов, r_1 и $r_0 = r - r_1$. Конечно, на работу этого процессора затрачивается определенное время, зато дальше основные процессоры могут обрабатывать раздельно подмножество r_1 и подмножество r_0, без потерь в эффективности. Когда это необходимо, два подмножества могут снова рассматриваться как одно множество r, причем для этого никаких дополнительных операций производить не нужно.

При необходимости классифицировать объекты на большее количество классов m_k, $m_k > 2$, процессор пере-

формирования массивов мог бы сделать это последовательно: сначала разделить исходный массив на массив r_1 объектов 1-го класса и массив $r—r_1$ объектов остальных классов (2-, 3-, ..., m_k-го), затем оставшийся массив объектов остальных классов—на массив r_2 объектов 2-го класса и массив $r—r_1—r_2$ объектов 3-, 4-, ..., m_k-го классов и т. д. Возможно и более сложное построение процессора переформирования массивов, позволяющее сразу переформировать исходный массив в m_k массивов.

Если количество процессоров n в системе невелико, то имеет смысл ввести еще один механизм для уменьшения потерь в эффективности системы при наличии расхождений в программах—*схему проверки однородности выборки.* Она может быть полезна как при отсутствии в системе процессора переформирования массивов, так и при его наличии. В первом случае эта схема работает всегда при наличии расхождений в программе, во втором — при условии, что расхождение относительно невелико и процессор переформирования массивов включать нецелесообразно.

Идея состоит в том, чтобы при выполнении операции под прямой либо под инверсной маской над очередной группой из n объектов проверить, имеется ли в соответствующих n разрядах маски соответственно хотя бы одна единица или хотя бы один нуль. Например, если операция выполняется под прямой маской, т. е. только для тех объектов, для которых соответствующие разряды маски содержат единицы, а в очередной выборке n объектов ни один из соответствующих разрядов маски не содержит единицы (т. е. выборка n объектов из r оказалась однородной), то операцию можно просто пропустить и не затрачивать без надобности время на ее использование.

Аппаратура схемы проверки однородности выборки чрезвычайно проста.

Эффект от введения схемы проверки однородности выборки, очевидно, тем больше, чем меньше n: при случайном распределении r объектов в множестве вероятность того, что очередная выборка из n объектов окажется однородной, тем больше, чем меньше n. Подробнее эти вопросы мы рассмотрим в п. 2.2.4.

6°. Последнее из усовершенствований системы рис. 2.1.4, о котором следует сказать и которое требуется

в особенности, если речь идет об использовании естественного параллелизма в общем виде (а не его частного случая—параллелизма множества объектов), — это введение *общих арифметико-логических цепей и локальных связей* между процессорами.

Эти цепи нужны, например, для вычисления скалярного произведения векторов (для чего нужно просуммировать произведения соответствующих компонент векторов, полученные в каждом из процессоров), при вычислении производной от функции, заданной своими значениями в дискретных точках (для чего каждый процессор должен получить из соседних процессоров значения функции в соседних точках), и т. п.

Пока используется параллелизм множества объектов, все основные процессоры системы представляют собой просто группу из n процессоров.

Для операций рассматриваемого здесь типа важно, как эти процессоры расположены один относительно другого (разумеется, не в пространстве, а по логическим связям): представляют ли они собой просто линию из n процессоров, или замкнуты в кольцо длиной n, или расположены в виде прямоугольника $n_1 \times n_2$ (где $n_1 \cdot n_2 \leqslant n$), или этот прямоугольник свернут в цилиндр с образующей n_1 и длиной окружности n_2 (либо наоборот, с образующей n_2 и длиной окружности n_1) или, может быть, прямоугольник свернут в тор.

Можно предусмотреть оперативное перестроение связей между процессорами и общих арифметических цепей. Специальные операции, применяемые для этого, называют операциями *геометрического управления.*

2.1.4. КОНВЕЙЕРНЫЕ (МАГИСТРАЛЬНЫЕ) ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

1°. *Принцип построения.* В вычислительных системах, рассматривавшихся в п. 2.1.1—2.1.3, возможность выполнения одновременно нескольких операций по преобразованию данных обеспечивалась наличием в системе нескольких процессоров. Все эти процессоры в принципе могли начать исполнение каждый своей операции одновременно.

Ниже рассматриваются системы принципиально другого типа.

Процессор в системе всего один. Но он разделен на n частей (ступеней), выполняющих последовательные этапы операции (1-, 2-, ..., n-й). При этом процессор устроен так, что в то время, когда j-я ступень процессора выполняет свой, j-й этап некоторой i-й операции, $(j-1)$-я ступень может выполнять $(j-1)$-й этап для $(i+1)$-й операции, $(j-2)$-я ступень — $(j-2)$-й этап для $(i+2)$-й операции, ..., 1-я ступень процессора — 1-й этап $(i+j-1)$-й операции (где $j=2, 3, \ldots, n$). В системе этого типа тоже может выполняться одновременно несколько (n) операций по преобразованию данных (что соответствует определению вычислительной системы, данному в п. 1.1.1, 1°). Однако эти операции в некоторый момент времени обязательно находятся на разных этапах исполнения и в принципе не могут начинаться все одновременно.

Заметим, что организованный подобным образом конвейер операций широко применяется и в обычных однопроцессорных вычислительных машинах (см., например, [1], п. 1. 4. 2). Скажем, выполнение операции может быть разделено на следующие этапы: формирование адреса инструкции — чтение инструкции из памяти — формирование адреса операнда — чтение операнда из памяти — собственно выполнение арифметической или логической операции. В то время, когда в арифметико-логическом устройстве машины идет выполнение арифметической операции, заданной i-й инструкцией, из памяти читается операнд для выполнения операции, заданной $(i+1)$-й инструкцией; одновременно узел формирования адресов операндов формирует адрес для чтения операнда $(i+2)$-й инструкции, производится чтение из памяти $(i+3)$-й инструкции (если это возможно, т. е. при условии, что $(i+3)$-я инструкция читается из другого блока памяти и по другим шинам, чем операнд $(i+1)$-й инструкции), а в узле формирования адреса очередной инструкции формируется адрес $(i+5)$-й инструкции.

Такая машина отличается от вычислительной системы лишь в тонких деталях, либо вообще чисто терминологически. Если считать, что этапы, на которые разделено выполнение операции в машине (формирование адреса инструкции, чтение инструкции и т. д.), являются этапами выполнения арифметической или логической операции, то любая вычислительная машина с конвейерной организацией должна называться вычислительной систе-

мой. Если же настаивать на том, что в вычислительной системе по определению (см. п. 1.1.1, 1°) одновременно должны исполняться несколько собственно арифметических или логических операций над данными, то машина указанного типа превращается в вычислительную систему только при условии, что само арифметико-логическое устройство разделено на несколько частей, включенных в общий конвейер и выполняющих одновременно различные этапы разных арифметических или логических операций (может быть, одинаковых операций над разными данными).

Для дальнейшего изложения, однако, эти различия, имеющие на самом деле несколько схоластический характер, не играют никакой роли.

2°. *Принципы организации управления* в системе и соответственно использования того или иного вида параллелизма которые были рассмотрены в предыдущих разделах применительно к n-процессорным вычислительным системам, можно применить и к конвейерным вычислительным системам, в которых единственный процессор разделен на n ступеней, работающих с совмещением по времени.

В частности, на рис. 2.1.6 показана конвейерная система с раздельным управлением. В ней имеется n устройств управления У — по числу частей, на которые разделен процессор. Принципиально важным является наличие n раздельных узлов формирования адреса следующей инструкции и регистров инструкции; возможно, что у каждого устройства управления имеются свои индексные и базовые регистры, регистры ключей защиты и другие индивидуальные цепи. Часть цепей может быть общей для различных устройств управления (например, формирователь адреса операнда, дешифратор кода операции и др.) и использоваться всеми устройствами управления по очереди. Эти общие цепи на рисунке включены в состав ступеней процессора ($У^{(j)}_0$, $j=1, 2, \ldots, n$).

Устройства управления подсоединены к ступеням процессора через кольцевой коммутатор. Коммутатор устроен так, что в 1-м такте работы системы к 1-й ступени процессора подключено 1-е устройство управления, которое начинает выполнение своей операции; во 2-м такте 1-е устройство управления переключается на 2-ю ступень процессора, а к 1-й ступени подключается 2-е

устройство управления, и т. д. В n-м такте 1-е устройство управления подключено к n-й ступени процессора, где завершается исполнение его 1-й операции, к $(n-1)$-й ступени подключено 2-е устройство управления, ..., к 1-й ступени подключено n-е устройство управления. После завершения 1-й операции 1-го устройства управления, в $(n+1)$-м такте 1-е устройство управления снова подключается к 1-й ступени процессора, где начинает исполнение 2-й операции своей программы, и т. д.

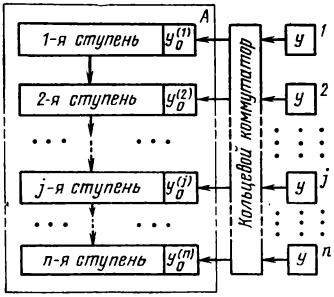


Рис. 2.1.6

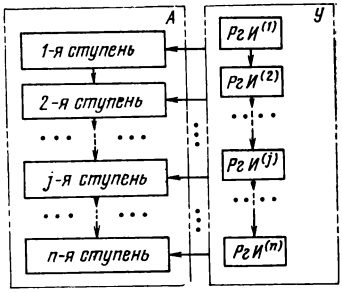


Рис. 2.1.7

Ясно, что система, построенная указанным образом, аналогична по своим возможностям n-процессорной системе типа II, рассмотренной в п. 2.1.1, 2° (рис. 2.1.2) и ориентирована на использование параллелизма независимых ветвей (если целью является достижение высокой пользовательской производительности). Такие конвейерные системы мы будем также относить к вычислительным системам *типа II*.

На рис. 2.1.7 показана другая организация управления в конвейерной системе — ориентированная на использование параллелизма смежных операций.

Устройство управления здесь одно, но регистров для хранения инструкции (РгИ) столько же, сколько ступеней имеется в процессоре. 1-я инструкция программы прочитывается в РгИ$^{(1)}$, и в 1-й ступени процессора начинается ее исполнение. В следующем такте 1-я инструкция передается в РгИ$^{(2)}$ и ее исполнение продолжает 2-я ступень процессора, а в РгИ$^{(1)}$ прочитывается 2-я инструкция и в 1-й ступени процессора начинается ее исполнение, и т. д.

В случае, когда оптимизация программы выполняется в процессе трансляции на машинный язык, в инструкциях программы могут быть расставлены при необходимости указания, на сколько тактов нужно задержать исполнение данной инструкции от момента ее получения. Если таких указаний нет, то в схеме устройства управления должно быть предусмотрено сравнение вновь полученной инструкции с инструкциями, находящимися в последующих регистрах РгИ (т. е. с предшествующими ей, но еще не исполненными до конца инструкциями), с целью выявления возможности выполнения очередного этапа данной инструкции.

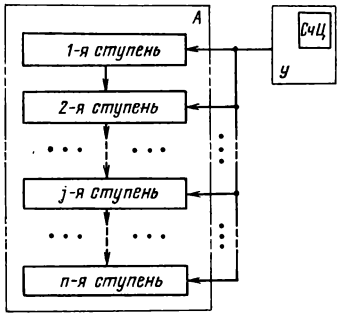


Рис. 2.1.8

Ясно, что конвейерная система, построенная указанным образом, аналогична многопроцессорным системам с общим управлением, ориентированным на параллелизм смежных операций (системы типа III — см. п. 2.1.2), и тоже должна относиться к вычислительным системам *типа III*.

Наконец, рис. 2.1.8 иллюстрирует построение управления в конвейерной системе, ориентированной на параллелизм множества объектов.

Устройство управления на рис. 2.1.8 одно. В его составе имеется аппаратура, предназначенная для организации циклического повторения одной и той же операции для всех объектов из обрабатываемого множества (счетчик циклов — СчЦ). По своему назначению и структуре эта аппаратура аналогична рассмотренной в п. 2.1.3, 4° (рис. 2.1.5), но существенно проще.

Получив очередную инструкцию, устройство управления начинает выполнение указанной операции для 1-го объекта. Когда эта операция пройдет через 1-ю ступень процессора и поступит во 2-ю ступень, в 1-й ступени начинается выполнение той же операции для второго объекта, и т. д. Когда n-я ступень заканчивает выполнение операции для 1-го объекта, в 1-й ступени начинается выполнение операции для n-го объекта; начиная от этого такта, загружены все ступени процессора. В следующем,

$(n+1)$-м такте в n-й ступени заканчивается выполнение операции для 2-го объекта, а в первой начинается для $(n+1)$-го объекта и т. д. Когда в 1-й ступени начинается выполнение операции для r-го объекта, то в n-й ступени заканчивается выполнение операции для $(r-n+1)$-го объекта. Далее устройство управления выжидает еще $n-1$ тактов, пока выполнение операции закончится для $(r-n+2)$-, $(r-n+3)$-, ..., r-го объектов. В течение последних $n-1$ тактов (так же как в течение первых $n-1$ тактов) используются не все ступени процессора.

После окончания выполнения операции для r-го объекта устройство управления прочитывает следующую инструкцию и снова исполняет ее для всех r объектов (в течение $r+n-1$ тактов) и т. д.

Ясно, что конвейерные системы, построенные указанным образом, аналогичны многопроцессорным системам с общим управлением, ориентированным на параллелизм множества объектов (типа IV), которые были рассмотрены в п. 2.1.3, и тоже должны относиться к вычислительным системам *типа IV*.

По этому принципу построена, в частности, вычислительная система STAR-100 [10].

2.2. ЭФФЕКТИВНОСТЬ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

2.2.1. ОБЩИЕ СООБРАЖЕНИЯ

1°. В § 2.2 нас будет интересовать в основном реальное пользовательское быстродействие и пользовательская эффективность вычислительных систем различных типов. В соответствии с определениями, данными в п. 1.1.2, реальное пользовательское быстродействие $S_{р(п)}$ — это быстродействие вычислительной системы, достигаемое при решении одной отдельно взятой пользовательской задачи, а пользовательская эффективность ($E_п$) — это отношение реального пользовательского быстродействия к номинальному быстродействию системы.

Нетрудно видеть, что определенная таким образом пользовательская эффективность представляет собой среднее по всем средствам системы значение отношений $T_{пi}/T_с$, где $T_с$ — общее время работы вычислительной системы при решении данной задачи, а $T_{пi}$ — та часть времени $T_с$, в течение которой i-е средство системы занято полезными операциями. Веса, с которыми усредняются отношения $T_{пi}/T_с$ для разных i (разных средств систе-

мы), пропорциональны быстродействию соответствующих средств:

$$E_{\text{п}} = \frac{\sum\limits_{(i)} S_i \frac{T_{\text{п}i}}{T_{\text{с}}}}{\sum\limits_{(i)} S_i} = \frac{\sum\limits_{(i)} S_i T_{\text{п}i}}{T_{\text{с}} \sum\limits_{(i)} S_i}.$$

В этом выражении $\sum\limits_{(i)} S_i T_{\text{п}i}$ — количество полезных операций, выполняемых вычислительной системой за время решения данной задачи, а $T_{\text{с}} \cdot \sum\limits_{(i)} S_i$ — полное количество операций, которые могли быть выполнены за это время, если бы все составные части системы были полностью загружены. В первом приближении можно считать, что $\sum\limits_{(i)} S_i T_{\text{п}i}$ — это то количество операций, которое необходимо выполнить в обычной (однопроцессорной) вычислительной машине для решения данной задачи.

Если быстродействие всех n средств системы (всех машин, входящих в многомашинный комплекс, всех процессоров многопроцессорной системы, всех ступеней конвейерной системы) одинаково, то

$$E_{\text{п}} = \frac{1}{n} \sum_{(i)} T_{\text{п}i} / T_{\text{с}}.$$

Именно этим случаем мы будем главным образом заниматься ниже.

В п. 1.1.2 мы говорили о том, что номинальная производительность вычислительной системы характеризует не столько ее возможности, сколько затраты на ее создание и эксплуатацию. Фактически величина пользовательской эффективности показывает, во сколько раз выигрыш в реальном пользовательском быстродействии вычислительной системы по сравнению с обычной (однопроцессорной) машиной отличается от соотношения соответствующих затрат. Если величина $E_{\text{п}}$ близка к единице (еще лучше, — если она больше единицы), то создание и использование вычислительной системы рационально, если существенно меньше единицы, — нерационально (разумеется, когда целью применения вычисли-

тельной системы является достижение высокой пользовательской производительности).

Говоря здесь о затратах на создание и эксплуатацию вычислительной системы, мы не обязательно имеем в виду стоимость системы в денежном выражении. Это может быть также площадь, занимаемая вычислительной системой, или ее объем, или масса, или энергопотребление и т. п. Важно только, чтобы критерии, по которым оцениваются «затраты» на вычислительную систему и на обычную вычислительную машину, были одинаковы.

2°. Утверждение о том, что номинальная производительность вычислительной системы пропорциональна затратам на ее создание и эксплуатацию, справедливо в разной мере для разных типов вычислительных систем. Ближе всего к истине это утверждение для многомашинных комплексов (вычислительных систем типа I), рассмотренных в п. 2.1.1, 1°. Достижение в многомашинном комплексе номинального быстродействия nS, в n раз более высокого, чем быстродействие одиночной вычислительной машины, требует объединения в комплексе n одинаковых машин с быстродействием S *).

Некоторые отклонения от прямой пропорциональности затрат и номинального быстродействия являются результатом двух причин. С одной стороны, — это необходимость иметь в составе комплекса каналы связи машин между собой и некоторое другое дополнительное оборудование, а также дополнительную операционную систему комплекса. С другой стороны, — в направлении уменьшения общих затрат — действует возможность обобществления и более рационального использования определенной части аппаратуры.

Те же причины, вызывающие отклонение от прямой пропорциональности затрат на создание вычислительной системы ее номинальному быстродействию, действуют и для многопроцессорных систем с раздельным управлением (типа II), рассмотренных в п. 2.1.1, 2°. Но степень интеграции вычислительных средств здесь более высо-

*) Конечно, затраты на вычислительную машину определяются не только ее быстродействием, но также объемом памяти, пропускной способностью внешнего обмена, затратами на программное обеспечение и пр. Известно однако, что, например объем памяти, суммарная пропускная способность внешнего обмена и количество периферийных устройств в общем случае находятся в прямой зависимости от быстродействия центрального процессора [1].

кая, поэтому отклонения от пропорциональности более существенны. Грубо можно считать, что если затраты на n-машинный комплекс при n порядка 3—6 в n раз выше, чем на одиночную машину, то для n-процессорной системы с раздельным управлением они выше всего в $(0{,}8—0{,}9)n$ раз, а при большем n—в $(0{,}7—0{,}8)n$ раз *).

Ясно, что отношение затрат на n-процессорную систему с общим управлением (типа III или типа IV — см. п. 2.1.2 и 2.1.3) к затратам на аналогичную однопроцессорную машину должно еще сильнее отклоняться от n.

В чистом виде в этих системах по сравнению с однопроцессорной машиной повторено n раз только арифметико-логическое устройство.

Устройство управления всего одно, — правда, более сложное, чем в обычной машине.

Внутренняя память для систем типа III должна быть по объему, вероятно, примерно в n раз больше, чем для обычной машины с одним процессором того же быстродействия, что и процессор системы, — просто потому, что от n-процессорной системы мы рассчитываем получить примерно в n раз большее быстродействие (см. [1]).

Что касается внутренней памяти многопроцессорной системы типа IV, то здесь следовало бы учитывать два фактора, действующие в противоположных направлениях. С одной стороны, программа для решения некоторой задачи с помощью такой системы несколько короче, чем для решения той же задачи с помощью обычной машины, потому что каждая операция в системе определена сразу для многомерных векторов или матриц, или функций, или для множеств объектов, а не для отдельных чисел. С другой стороны, количество рабочих ячеек, используемых при работе системы, больше, чем для обычной машины: если бы, например, обычная (однопроцессорная) система циклически обрабатывала информацию о множестве объектов, то рабочие ячейки, используемые в каждом цикле, могли бы быть одними и теми же. В многопроцессорной системе типа IV с пооперационной организацией циклов (см. п. 2.1.3, 4°) количество рабо-

*) Эти данные, как и приводимые ниже аналогичные данные для других типов вычислительных систем, основаны на весьма примерных оценках одного из авторов. Их следует воспринимать, разумеется, как сугубо ориентировочные.

чих ячеек в r раз больше (где r — ранг задачи, количество объектов в обрабатываемом множестве), а при обычной организации циклов, когда объекты обрабатываются группами по n (где n — количество процессоров), — в n раз больше.

В целом, однако, можно считать, что и для многопроцессорных систем типа IV общий объем внутренней памяти должен быть примерно в n раз выше, чем для обычной вычислительной машины с одним процессором того же типа, которые применены в системе.

Наконец, оценивая затраты на многопроцессорную систему типа IV, нужно было бы учесть те усовершенствования структуры системы, о которых говорилось в п. 2.1.3, 2°—6°. Однако сопоставить в общем виде сложность соответствующей аппаратуры со сложностью устройств, которые входят в состав обычной машины, трудно. Поскольку объем этой аппаратуры в общем не очень значителен по сравнению со всей аппаратурой системы, мы позволим себе просто не учитывать затраты на нее. В виде компенсации допускаемой тем самым погрешности не будем учитывать также и тот факт, что указанная аппаратура наряду с чисто системными функциями, направленными на повышение эффективности основных процессоров системы, выполняет иногда и операции, связанные непосредственно с обработкой информации, причем более эффективно, чем аппаратура обычной вычислительной машины. В качестве примера таких операций можно назвать логические операции над массивами булевых переменных (аппаратура, рассмотренная в п. 2.1.3, 3°), подсчет количества объектов, удовлетворяющих определенным условиям, либо выяснение, имеются ли вообще в данном множестве объекты, удовлетворяющие определенным условиям (аппаратура, рассмотренная в п. 2.1.3, 5°), и т. д.

Грубо можно считать, что при небольшом количестве процессоров n (порядка 3—6) отношение затрат на n-процессорную систему типа III или IV к затратам на обычную однопроцессорную машину составляет $(0{,}6—0{,}8)n$. Многопроцессорные системы типа IV имеет смысл, как мы увидим из дальнейшего (см. п. 2.2.4), строить и с бо́льшим количеством процессоров. Чем больше количество процессоров n, тем сильнее отклоняются затраты на систему от пропорциональности числу процессоров; например, при n=10—15 затраты на n-процессорную

систему типа IV составляют всего $(0{,}4—0{,}5)n$ затрат на одиночную машину.

3°. Для вычислительных систем конвейерного типа (п. 2.1.4) предположение о пропорциональности затрат на систему ее номинальному быстродействию еще менее справедливо.

Если говорить, например, о конвейерных системах типа IV (рис. 2.1.8 на с. 60), то избыточная аппаратура в них по сравнению с обычной вычислительной системой практически ограничивается n регистрами (или n парами регистров) для хранения промежуточных результатов различных этапов операции. Их назначение состоит в том, чтобы развязать разные ступени процессора (одного!) и обеспечить возможность одновременного выполнения в каждой из ступеней своего этапа операции — одной и той же для всех ступеней, но над разными данными в разных ступенях.

В конвейерных системах типа III, кроме n регистров (или n пар регистров), разделяющих ступени процессора, имеется n регистров инструкции и, возможно, повторены n раз отдельные цепи устройства управления. Например, если на несколько ступеней разделено арифметико-логическое устройство и каждой ступени соответствует свой регистр инструкции (см. рис. 2.1.7 на с. 59), то в составе устройства управления должно быть соответственно несколько дешифраторов кода операции. Кроме того, в составе устройства управления системы типа III должны быть цепи, сравнивающие содержимое последовательных регистров инструкции и проверяющие возможности исполнения определенного этапа некоторой инструкции до окончания исполнения предшествующих.

Наконец, в конвейерной системе типа II с количеством ступеней n (см. рис. 2.1.6 на с. 59) должно быть n независимых устройств управления, и пропорциональность затрат на процессорную часть системы ее номинальному быстродействию соблюдается точнее всего.

Ориентировочно можно считать, что для конвейерной системы типа IV при разумном количестве ступеней n *)

*) Части, на которые делится арифметико-логическое устройство, не могут быть слишком мелкими — иначе в длительности такта работы конвейера (времени срабатывания каждой из ступеней) слишком значительным будет собственное время срабатывания разделительных регистров. Кроме того, минимальный такт работы конвейера ограничен длительностью обращения к памяти; расслоение памяти отчасти ослабляет это ограничение, но не снимает его полностью.

затраты на процессорную часть системы мало зависят от числа ступеней n и примерно в 1,5—2 раза превышают затраты на процессорную часть обычной вычислительной машины.

Для n-ступенных конвейерных систем типа II и III затраты на процессорную часть превышают затраты на процессорную часть обычной машины соответственно примерно в $(0{,}3—0{,}4)n$ раз и в $(0{,}4—0{,}5)n$ раз.

4°. В п. 2.2.2—2.2.4 мы будем заниматься определением пользовательской эффективности $E_{\text{п}}$ вычислительных систем разных типов при решении разных классов задач, т. е. величиной отношения реального пользовательского быстродействия $S_{\text{р(п)}}$ к номинальному быстродействию системы $S_{\text{н}}$. При этом те различия между оценкой номинального быстродействия системы и оценкой затрат на ее создание и использование, о которых говорилось в п. 2° и 3°, можно было бы учитывать просто в том, какой считать минимально допустимую величину пользовательской эффективности. Например, в соответствии со сказанным в п. 2° можно было бы считать, что при проектировании или планировании использования многомашинного комплекса (системы типа I) приемлемой является величина $E_{\text{п}}$, близкая к единице, для многопроцессорной системы типа II — порядка 0,8—0,9 при малых n и порядка 0,7—0,8 при больших n, и т. д.

Возможности такого подхода, однако, весьма ограничены. Особенно ясно это видно на примере конвейерных систем. В п. 3°, в частности, сравнивая затраты на конвейерную вычислительную систему с ее номинальным быстродействием, мы имели в виду главным образом затраты на процессорную часть системы; при этом, как мы говорили, затраты на процессорную часть конвейерной системы сравнительно слабо связаны с количеством ступеней и, следовательно, с номинальным быстродействием системы. Между тем, имеется прямая связь между быстродействием процессорной части и необходимым объемом главной памяти, общим объемом внутренней памяти, необходимой пропускной способностью внешнего обмена и другими характеристиками системы [1]. Об этом мы упоминали уже на с. 63. При проектировании вычислительной системы характеристики всех ее устройств согласуются, естественно, с номинальным быстродействием процессорной части. Если при конкрет-

ном использовании вычислительной системы ее эффективность оказывается низкой, то в общем случае это значит, что недостаточно используется не только процессорная часть (затраты на которую, может быть, и непропорциональны номинальному быстродействию), но недоиспользуются также память системы, каналы внешнего обмена и другие возможности системы.

В общем мы бы считали, что приемлемыми являются значения пользовательской эффективности $E_{\text{п}}$ не ниже 0,6—0,9, причем самые низкие из указанных значений могут быть допущены для конвейерных систем типа IV, а наиболее высокие следует требовать от многопроцессорных систем типа II и многомашинных комплексов (систем типа I).

2.2.2. ЭФФЕКТИВНОСТЬ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ТИПА I И II

1°. Вопрос об определении в общем виде пользовательской эффективности вычислительных систем типа I и II, ориентированных на параллелизм независимых ветвей, значительно сложнее, чем определение пользовательской эффективности для вычислительных систем других типов (о чем речь будет идти далее, в п. 2.2.3 и 2.2.4).

Проще всего пользовательскую эффективность систем типа I и II определить для случая, когда все N ярусов ярусно-параллельной формы имеют ширину b, равную n — количеству машин в многомашинном комплексе или процессоров в многопроцессорной системе, или количеству ступеней конвейерной системы типа II. Если это так, то

$$E_{\text{п}} = \frac{\sum_{i=0}^{N-1} \sum_{j=1}^{b} l_{ij}}{n \sum_{i=0}^{N-1} \left(\max_{j=1}^{b} \{ l_{ij} + \Delta^{(1)} l_{ij} + \Delta^{(2)} l_{ij} \} + (n-1)\, \Delta^{(3)} l_i \right)},$$

где l_{ij} — длина j-й ветви ($j=1, 2, \ldots, b$) в i-м ярусе ($i=0, 1, \ldots, N-1$); $\Delta^{(1)} l_{ij}$ — приращение длины ветви с номером ij (j-й ветви i-го яруса), связанное с необходимостью выполнения операций по проверке условий, позволяющих начать исполнение данной ветви (до начала ее исполнения), и операций по формированию

условий запуска последующих ветвей (после окончания исполнения данной ветви); $\Delta^{(2)}l_{ij}$ — приращение длины ветви с номером ij, связанное с необходимостью обмена информацией между машинами многомашинного комплекса или процессорами многопроцессорной системы типа II, если они работают над раздельной памятью. Для многопроцессорных и конвейерных систем типа II, работающих над общей внутренней памятью, приращение $\Delta^{(2)}l_{ij}$ обусловливается конфликтами при обращениях разных процессоров или разных ступеней конвейера к общей памяти; $(n-1)\Delta^{(3)}l_i$ — приращение длины i-го яруса, характерное для конвейерных систем типа II и обусловленное тем, что разные ветви, принадлежащие i-му ярусу, начинают и заканчивают работу не одновременно, а со сдвигом в $(n-1)$ тактов (причем мы предполагаем здесь, что ни одна из ветвей $(i+1)$-го яруса не может начинать работу, пока не закончено исполнение всех ветвей i-го яруса). Если длины ветвей и соответственно ярусов отождествить с временами их исполнения, то величина $\Delta^{(3)}l_i$ для всех ярусов (для всех i) равна просто длительности такта работы конвейера.

В приведенном выше выражении для $E_{\text{п}}$ величина $\sum_{i=0}^{N-1}\sum_{j=1}^{b} l_{ij}$, находящаяся в числителе,— это продолжительность исполнения программы однопроцессорной машиной. Разделив эту величину на n, получим $\sum_{i=0}^{N-1}\sum_{j=1}^{b} l_{ij}/n$ — время, в течение которого программа должна была бы исполняться вычислительной системой, если бы ее реальное быстродействие было равно номинальному (в n раз выше, чем быстродействие обычной однопроцессорной машины), а эффективность системы была бы равна единице. Величина $\sum_{i=0}^{N-1}\left(\max_{j=1}^{b}\{l_{ij}+\Delta^{(1)}l_{ij}+\Delta^{(2)}l_{ij}\}+(n-1)\Delta^{(3)}l_{ij}\right)$, находящаяся в знаменателе выражения для $E_{\text{п}}$, представляет собой то реальное время, за которое данная программа исполняется вычислительной системой.

Ясно, что для любого i-го яруса, $i=0, 1, \ldots, N-1$,

$$n\left(\max_{j=1}^{b}\{l_{ij}+\Delta^{(1)}l_{ij}+\Delta^{(2)}l_{ij}\}+(n-1)\Delta^{(3)}l_i\right)>\sum_{j=1}^{b} l_{ij}$$

(напомним, что здесь $n=b$), и поэтому $E_{\text{п}}<1$. Величина эффективности $E_{\text{п}}$ тем ближе к единице, чем меньше различаются между собой длины ветвей, принадлежащих одному ярусу, и чем больше эти длины по сравнению с величинами $\Delta^{(1)}l_{ij}$, $\Delta^{(2)}l_{ij}$ и $(n-1)\Delta^{(3)}l_i$. Иначе говоря, желательно, чтобы для любого i-го яруса соблюдалось соотношение

$$l_{i1}\approx l_{i2}\approx\ldots\approx l_{ib}\gg\Delta^{(1)}l_{ij},\ \Delta^{(2)}l_{ij},\ (n-1)\Delta^{(3)}l_i.$$

Заметим, однако, что рассматриваемый выше случай, когда ширина любого яруса программы b_i в точности равна n, является наиболее благоприятным для получения высокой эффективности вычислительных систем типа I и II и в общем случае мало реален.

Другой случай, в котором выражение для эффективности системы $E_{\text{п}}$ может быть записано в аналитической форме, состоит в том, что для разных ярусов ярусно-параллельной формы программы величины b_0, b_1, b_2, ..., b_{N-1} вообще различны и отличаются от n, но зато длины всех ветвей, входящих в один и тот же i-й ярус, одинаковы между собой: $l_{i1}=l_{i2}=\ldots=l_{ib_i}=l_i$ для $i=0, 1, 2, \ldots, N-1$. Кроме того, сохраним в силе предположение, что исполнение ни одной ветви $(i+1)$-го яруса не может быть начато до того, как закончится исполнение всех ветвей i-го яруса.

Очевидно, что для этого случая время исполнения программы обычной (однопроцессорной) машиной должно было бы быть $\sum\limits_{i=0}^{N-1} b_i l_i$. При использовании вычислительной системы все b_i ветвей, входящих в i-й ярус ($i=0, 1, 2, \ldots, N-1$), нужно разделить на $\left]\frac{b_i}{n}\right[$ групп, из которых $\left(\left]\frac{b_i}{n}\right[-1\right)$ групп содержат ровно по n ветвей каждая, а последняя группа — от 1 до n ветвей. Время исполнения всех ветвей одного яруса пропорционально, очевидно, количеству групп $\left]\frac{b_i}{n}\right[$. (Напомним, что $\left]\frac{b_i}{n}\right[$ — это ближайшее не меньшее целое для числа b_i/n.) Таким обра-

зом, выражение для эффективности системы имеет в данном случае вид:

$$E_{\text{п}} = \frac{\sum_{i=0}^{N-1} b_i l_i}{n \sum_{i=0}^{N-1} \left(\left] \frac{b_i}{n} \right[\cdot (l_i + \Delta^{(1)} l_i + \Delta^{(2)} l_i) + \left(b_i - 1 - n \left(\left] \frac{b_i}{n} \right[- 1 \right) \Delta^{(3)} l_i \right) \right)}.$$

Здесь $\Delta^{(1)} l_i$, $\Delta^{(2)} l_i$, $\Delta^{(3)} l_i$ имеют тот же смысл, что и выше. Коэффициент при $\Delta^{(3)} l_i$ показывает, сколько тактов потребуется в n-ступенной конвейерной системе для завершения последней операции последней ветви i-го яруса сверх тех n тактов, которые отводятся для выполнения последней операции в последней $\left(\left] \frac{b_i}{n} \right[\right)$-й группе ветвей. Если бы b_i было кратно n, т. е. если бы выполнялось равенство $\left] \frac{b_i}{n} \right[= \frac{b_i}{n}$ и последняя группа была полной (содержала бы n ветвей), то этот коэффициент был бы равен

$$b_i - 1 - n \left(\left] \frac{b_i}{n} \right[- 1 \right) = b_i - 1 - n \left(\frac{b_i}{n} - 1 \right) = n - 1.$$

Видно, что в рассматриваемом случае пользовательская эффективность тем выше, чем больше величины b_0, b_1, ..., b_{N-1} по сравнению с n.

В более общем случае, когда различны не только величины b_i для разных ярусов, но и длины ветвей l_{ij}, входящих в один ярус, для достижения наибольшей эффективности системы следовало бы поступать иначе. Все b_i ветвей, входящих в один ярус, нужно разбить на n групп — по числу машин в многомашинном комплексе или процессоров в многопроцессорной системе, или устройств управления в конвейерной системе типа II; соответственно каждая группа ветвей будет исполняться своей машиной или своим процессором, или своим устройством управления конвейерной системы. В среднем в каждую группу будет входить b_i/n ветвей, но вообще-то группировка ведется с таким расчетом, чтобы по возможности были равны между собой суммы длин ветвей, входящих в каждую группу. Если бы группи-

ровку удалось провести так, чтобы суммы длин ветвей для всех групп были в точности равны между собой, эффективность системы была бы больше всего.

Задача о группировке b_i чисел l_{i1}, l_{i2}, ..., l_{ib_i} в n групп, таких, чтобы суммы чисел, входящих в каждую группу, минимально отличались друг от друга (т. е. чтобы разность между наибольшей и наименьшей суммами была минимальной), требует проведения весьма значительного (при больших b_i и n) перебора. Если все характеристики ярусно-параллельной формы программы и характеристики вычислительной системы известны заранее и постоянны, то эта задача может решаться до запуска программы — как говорят, *в статике*. Если характеристики программы b_i и l_{ij} ($i=0, 1, \ldots, N-1$; $j=1, 2, \ldots, b_i$) зависят от исходных данных, а величина n зависит от состояния технических средств системы и, возможно, от наличия в системе других задач и их приоритетов по отношению к данной программе, то планирование загрузки средств системы должно вестись в ходе исполнения программы — как говорят, *в динамике*.

В действительности задача планирования загрузки вычислительной системы типа I или II значительно сложнее, чем можно было бы заключить из предыдущего. Мы исходили до сих пор из предположения, что условием, разрешающим начать выполнение любой ветви $(i+1)$-го яруса, является окончание исполнения в с е х ветвей i-го яруса. На практике это во многих случаях не так. Хотя программу и можно формально представить в виде ярусно-параллельной формы, оказывается, что такое представление не отражает действительных возможностей организации параллельных вычислений, а численные характеристики графа ярусно-параллельной формы, о которых говорилось в п. 1.2.2, 3° и которыми мы пользовались выше, не позволяют оценить, какая эффективность системы может быть достигнута при реализации данной программы.

В качестве простейшего примера рассмотрим фрагмент графа ярусно-параллельной формы программы, представленный на рис. 2.2.1,*а*, и предположим, что эта программа должна реализоваться 4-машинным комплексом или 4-процессорной вычислительной системой типа II. Цифры, проставленные на рис. 2.2.1 рядом с кружками, символизирующими ветви, обозначают длины ветвей в условных единицах времени. Предполагается, что величины $\Lambda^{(1)}l_{ij}$ и $\Lambda^{(2)}l_{ij}$ настолько малы по сравнению с длинами ветвей, что ими можно пренебречь.

Формальный подход к делу подсказывает, что поскольку ширина i-го и $(i+1)$-го ярусов в точности равна количеству машин в комплексе или процессоров в многопроцессорной системе,

$$b_i = b_{i+1} = n = 4,$$

то при планировании загрузки системы у нас, казалось бы, нет выбора, и расписание работы системы должно быть таким, как показано на рис. 2.2.1,*б*. Продолжительность исполнения i-го и $(i+1)$-го ярусов равна сумме их длин, $l_i + l_{i+1} = 14$ единицам времени, а эффективность системы равнялась бы при этом:

$$E_{\text{п}} = \frac{\sum_{i}^{i+1} \sum_{j=1}^{b_i} l_{ij}}{n \sum_{i}^{i+1} \max_{j=1}^{b_i} \{l_{ij}\}} = \frac{(1+3+3+4)+(10+10+10+1)}{4\,(4+10)} =$$

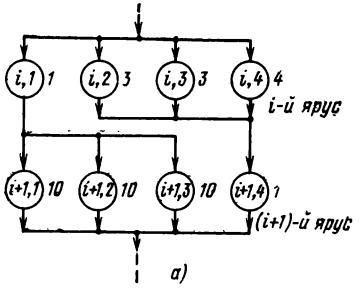
$$= \frac{42}{56} = 0{,}75.$$

Некоторое усовершенствование расписания, представленного на рис. 2.2.1,*б*, может состоять в том, что прохождение ветвей $i+1,1$, $i+1,2$ и $i+1,3$ может начинаться в 1-, 2-, 3-й машинах раньше, чем в 4-й машине закончится прохождение ветви $i,4$, сразу после освобождения машин от исполнения ветвей $i,1$; $i,2$; $i,3$, поскольку все эти ветви не зависят от $i,4$. Общее время исполнения i-го и $(i+1)$-го ярусов при этом сократилось бы до 13 единиц, а эффективность повысилась бы до величины $E_{\text{п}} = \frac{42}{4 \cdot 13} \approx 0{,}81$.

Между тем непосредственное рассмотрение графа, изображенного на рис. 2.2.1,*а*, позволяет построить еще более выгодное расписание работы системы — такое, как показано на рис. 2.2.1,*в*. Длительность выполнения i- и $(i+1)$-го ярусов в соответствии с этим расписанием составляет 11 ед. времени, а эффективность $E_{\text{п}} = \frac{42}{4 \cdot 11} \approx 0{,}95$. Несложным перебором можно показать, что это расписание является оптимальным, т. е. что достигаемая при его использовании величина $E_{\text{п}}$ — максимально возможная.

Известны различные алгоритмы планирования работы вычислительных систем. Необходимо только учитывать, что сама планирующая программа исполняется теми же средствами системы, которые она распределяет для исполнения основной программы. Чем ближе к оптимальному должно быть получено расписание, тем должна быть сложнее и тем дольше будет работать планирующая программа. Поэтому имеется некоторый оптимум в том, к каким отклонениям от оптимального расписания реализации пользовательских программ следует стремиться.

2°. Те аспекты организации работы вычислительных систем типа I или II, которые рассматривались нами в п. 1° и которые представляются весьма привлекательными для различных теоретических построений, тем не менее мало приблизили нас к решению основных практических вопросов, связанных с построением и использованием вычислительных систем указанных типов. Фактически на их основе мы можем сделать лишь два весьма приблизительных вывода.



Первый из них состоит в том, что вычислительные системы типа I и II могут быть достаточно эффективными в двух случаях. Либо такая система должна

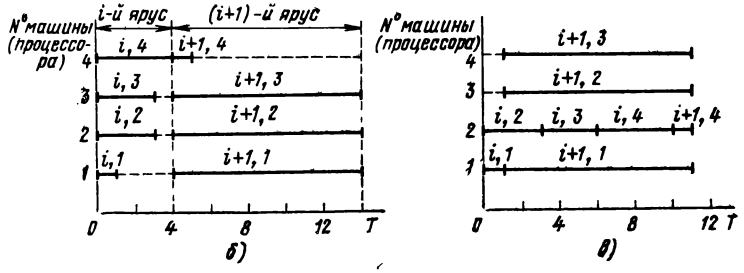


Рис. 2.2.1

строиться как строго специализированная, предназначенная для решения определенной задачи. При этом количество машин многомашинного комплекса, или количество процессоров многопроцессорной системы, или количество ступеней конвейерной системы n должно быть в точности равно или являться целым делителем ширины каждого из ярусов ярусно-параллельной формы этой задачи, а длины всех ветвей, принадлежащих одному ярусу, должны быть одинаковы между собой. Либо — если систему типа I или II предполагается использовать для достижения высокой пользовательской производительности при решении широкого круга задач — в число этих задач должны войти такие, в ходе исполнения которых в каждый момент времени t количество исполняемых и готовых к исполнению ветвей, $B(t)$, велико по

сравнению с n. Наоборот: чем шире должен быть класс задач, при решении которых данная система будет достаточно эффективна, тем меньше должно быть n. В пределе, при $n=1$, обычная однопроцессорная система одинаково эффективна при решении любых задач.

Второй вывод из сказанного в п. 1° состоит в том, что для достаточно эффективного использования систем типов I и II необходимо, чтобы длины независимых ветвей в программах были намного больше, чем чисто системные потери времени: $\Delta^{(1)}l$ — на проверку и формирование условий запуска ветвей, $\Delta^{(2)}l$ — на обмен информацией между ветвями, $\Delta^{(3)}l$ — на завершение последних операций ветвей в конвейерных системах. Наоборот: чем лучше реализованы в системе функции организации ее работы, т. е. чем меньше времени отнимают диспетчерские функции, обмен информацией между средствами системы и т. п., тем более широкий круг задач может быть эффективно решен с помощью системы. В частности, по этой причине пользовательская эффективность вычислительных систем типа II при прочих равных условиях несколько выше, чем для систем типа I.

Оба эти вывода, однако, весьма тривиальны. К сожалению, оба они исходят из предположения, что заранее задана ярусно-параллельная форма программы и известны ее характеристики.

Значительно содержательнее был бы ответ на вопрос о том, каковы могут быть оптимальные в определенном смысле характеристики ярусно-параллельной формы программы для некоторой определенной задачи, либо ответ на вопрос, каким требованиям должны отвечать условия задач для того, чтобы ярусно-параллельные формы программ для их решения имели требуемые характеристики. Однако авторам неизвестны общие способы решения подобных вопросов.

Те или иные свойства программ, позволяющие при их реализации с определенной эффективностью использовать вычислительные системы типа I или II, могут быть результатом сознательных усилий программиста в этом направлении. При этом от программиста большей частью требуется, чтобы он сам расставил в тексте своей программы операторы разветвления типа **fork** <список ветвей>, означающие, что, начиная от данной точки программы, можно одновременно запускать ветви, перечисленные в списке, и операторы ожидания типа **joint**

<список ветвей> и **quitt** <список ветвей>, означающие соответственно, что дальнейшее продолжение программы возможно после окончания всех или хотя бы одной из ветвей, перечисленных в списке. Возможен также вариант, когда соответствующие операции вставляются в текст программы транслятором при переводе программы с языка высокого уровня на машинный язык (см. п. 2.4.1, 4°). Если программа написана безотносительно к структуре системы, то транслятор обеспечит возможности параллельного исполнения отдельных ветвей программы лишь в той мере, в какой это окажется возможным.

Таким образом, получается, что возможности эффективного использования вычислительной системы типа I или II для решения некоторой задачи зависят не столько от свойств этой задачи, сколько от искусства программиста либо,— если программа составлялась безотносительно к структуре системы,— от обстоятельств почти случайного характера.

Итог нашего весьма пространного рассмотрения вопроса довольно скуден. Фактически, если целью создания вычислительной системы является получение высокой пользовательской производительности, то мы не знаем, для каких классов задач целесообразно применять вычислительные системы типа I и II и каковы рациональные характеристики этих систем.

В этой ситуации большой интерес представляет ряд работ по моделированию процессов организации параллельных вычислений при решении разнообразных реальных задач. В частности, в одной из этих работ [9] рассматривался широкий круг задач, связанных с прогнозом погоды, рентгеноструктурным анализом и др. В другой американской работе, проводившейся в плане программы высадки человека на Марс, исследовались задачи, возлагаемые на бортовые вычислительные средства космического корабля: навигационные, автоматического управления и т. п. (на борту корабля предполагалось использовать 3-машинный вычислительный комплекс Cerber).

Результаты этих и ряда других работ аналогичного характера более или менее совпадают между собой и в усредненной форме представлены на рис. 2.2.2. На рисунке показаны зависимости реального пользовательского быстродействия $S_{р(п)}$ и соответствующей пользова-

тельской эффективности $E_н$ от n (числа машин в многомашинном комплексе или процессоров в многопроцессорной системе, или ступеней в конвейерной системе типа II).

Из рассмотрения рисунка видно, что если рассчитывать на получение от системы высокого пользовательского быстродействия для широкого круга задач, то не имеет смысла объединять больше 3—4, в крайнем случае 5 машин (процессоров, устройств управления в конвейер-

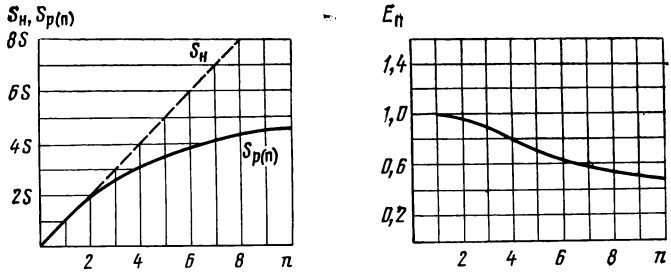


Рис. 2.2.2

ной системе): слишком медленно растет при дальнейшем увеличении n реальное пользовательское быстродействие. Как говорилось уже в 2.1.1, 3°, вычислительные системы рассматриваемых типов с большим n строятся главным образом в расчете на получение высокой системной производительности, а повышение пользовательской производительности решается как попутная задача — для тех программ из общего потока, проходящего через систему, для которых это выгодно.

2.2.3. ЭФФЕКТИВНОСТЬ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ТИПА III

Оценка эффективности вычислительных систем типа III наталкивается фактически на те же методические трудности, которые помешали нам получить в общем виде сколько-нибудь удовлетворительные результаты при попытке оценить эффективность систем типа I и II: количественные характеристики параллелизма смежных операций, на который ориентированы системы типа III, относятся на самом деле не к собственно решаемым задачам, а к программам для их решения. Однако, как мы говорили в п. 1.2.3, 3°, положение здесь облегчается тем, что сами эти характеристики (показатель связанности

смежных операций α и глубина параллелизма смежных операций $L_{\text{псо}}$) более зависят от качества локальной оптимизации программы, чем от свойств задачи. Во всяком случае для различных задач вычислительного характера *) при использовании одних и тех же методов локальной оптимизации программ достигаются примерно одни и те же значения α и $L_{\text{псо}}$ (см. п. 1.2.3, 3°).

Если величины α (или $L_{\text{псо}}$) и n известны, то дальше определение пользовательской эффективности $E_{\text{п}}$ для n-процессорной или n-ступенной конвейерной системы типа III не представляет особых затруднений. Естественно, что $E_{\text{п}}$ является функцией α (или $L_{\text{псо}}$) и n: $E_{\text{п}}=E_{\text{п}}(\alpha, n)$ или $E_{\text{п}}=E_{\text{п}}(L_{\text{псо}}, n)$.

Для упрощения изложения будем в дальнейшем предполагать, что оптимизация программы (с целью улучшения возможностей использования параллелизма смежных операций) выполнена в процессе трансляции программы либо не выполняется вообще. На самом деле допущение возможности выполнения оптимизации одновременно с интерпретацией программы ничего не изменило бы по существу. Однако каждый раз, говоря об очередной операции программы, о цепочке операций, которые можно выполнить одновременно с данной операцией, и т. д., мы должны были бы упоминать, что речь идет не о порядке операций в тексте исполняемой программы, а о том порядке, который получается в результате ее оптимизации. Такое же упрощение нами фактически уже было принято в п. 1.2.3, 3°, где вводились численные характеристики параллелизма смежных операций.

Сохраним обозначения, принятые в п. 1.2.3, 3°.

Очевидно, что в любом такте работы вычислительной системы типа III все n ее процессоров (или все n ступеней, если речь идет о конвейерной системе) могут быть заняты полезными операциями только при условии, что, начиная от очередной операции, в программе имеется цепочка длиной не менее n операций, которые можно выполнить все одновременно. Вероятность существования такой цепочки была обозначена нами β_n, причем

$$\beta_n=(1-\alpha)^{\frac{(n-1)\,n}{2}}.$$

*) Так мы называем задачи, в которых главную роль играет выполнение вычислений — в отличие от информационно-поисковых и логических задач.

В других случаях занято меньшее количество процессоров (ступеней) l, где $1 \leqslant l \leqslant n-1$. Вероятность того, что занято ровно l процессоров (ступеней), есть γ_l — вероятность ситуации, когда, начиная от очередной операции в программе, имеется ровно l операций, которые можно

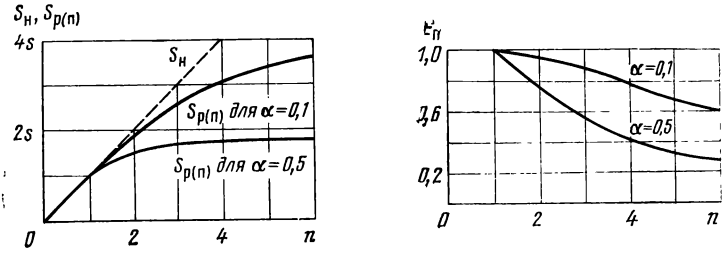


Рис. 2.2.3

выполнять все одновременно. Как мы говорили в п. 1.2.3, 3°,

$$\gamma_l = (1-\alpha)^{\frac{(l-1)\,l}{2}} - (1-\alpha)^{\frac{l\,(l+1)}{2}}.$$

Итого, эффективность вычислительной системы типа III равна

$$E_п(\alpha,\ n) = \frac{1}{n}\left(\sum_{l=1}^{n-1} l\gamma_l + n\beta_n\right) = \frac{1}{n}\left(\sum_{l=1}^{n-1} l\left((1-\alpha)^{\frac{(l-1)\,l}{2}} - \right.\right.$$

$$\left.\left. - (1-\alpha)^{\frac{l\,(l+1)}{2}}\right) + n\,(1-\alpha)^{\frac{(n-1)\,n}{2}}\right) = \frac{1}{n}\sum_{l=1}^{n}(1-\alpha)^{\frac{(l-1)\,l}{2}}.$$

На рис. 2.2.3 показаны зависимости от n реального пользовательского быстродействия $S_{р(п)}$ и пользовательской эффективности $E_п$ для вычислительных систем типа III при различных значениях α.

Можно также записать:

$$E_п(L_{псо},\ n) = \frac{1}{n}(L_{псо} - R(n)),$$

где $R(n)$ — некоторый остаточный член, тем меньший, чем больше n. При том вероятностном распределении использования результатов предыдущих операций в по-

следующих операциях программы, которое было принято ранее,

$$R(n)=\sum_{l=1}^{\infty}(1-\alpha)^{\frac{(l-1)\,l}{2}}-\sum_{l=1}^{n}(1-\alpha)^{\frac{(l-1)\,l}{2}}=$$

$$=\sum_{l=n+1}^{\infty}(1-\alpha)^{\frac{(l-1)\,l}{2}}.$$

Ясно, что при прочих равных условиях величина $E_{\text{п}}$ тем меньше, чем больше n. Чем больше коэффициент связанности смежных операций α или чем меньше глубина параллелизма смежных операций $L_{\text{псо}}$, тем меньшее количество процессоров n целесообразно объединять в многопроцессорную систему типа III (или предусматривать ступеней в конвейерной системе типа III). Если считать допустимыми значения пользовательской эффективности порядка 0,6—0,8, то при обычных значениях α и $L_{\text{псо}}$ (α=0,1—0,5; $L_{\text{псо}}$=1,6÷4 — см. п. 1.2.3, 3°) целесообразная величина n для вычислительных систем типа III лежит в пределах 2—6.

2.2.4. ЭФФЕКТИВНОСТЬ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ТИПА IV

1°. Рассмотрение вопроса об эффективности вычислительных систем типа IV, ориентированных на использование естественного параллелизма (или, в частности, параллелизма множества объектов), начнем для определенности с оценки эффективности *многопроцессорных* систем этого типа. Как мы увидим из дальнейшего, аналогичная оценка для конвейерных систем типа IV получается несколько иной.

Основным фактором, определяющим эффективность многопроцессорных систем типа IV, является соотношение ранга задачи r и количества процессоров n. В общем случае r не равно и не кратно n. Без учета других, менее значительных факторов, влияющих на величину эффективности, пользовательская эффективность n-процессорной системы типа IV при решении задачи ранга r может быть приближенно оценена отношением

$$E_{\text{п}}\approx\frac{r}{n}:\left\rceil\frac{r}{n}\right\lceil=\frac{r}{n\cdot\left\rceil\frac{r}{n}\right\lceil},$$

где через $\left]\frac{r}{n}\right[$ обозначено ближайшее не меньшее целое для числа r/n. Здесь r — это то количество повторений каждой из операций в программе решения задачи ранга r, которое потребовалось бы, если бы эта программа исполнялась обычной однопроцессорной машиной; r/n — это количество повторений каждой операции в n-процессорной системе, которое должно было бы получиться, если бы пользовательское быстродействие n-процессорной системы было действительно в n раз выше быстродействия однопроцессорной машины (т. е. если бы пользовательская эффективность была равна единице); $\left]\frac{r}{n}\right[$ — это реальное количество повторений каждой операции в n-процессорной системе, причем $\left]\frac{r}{n}\right[\cdot n - r$ есть количество процессоров, простаивающих в последнем цикле исполнения операции (из-за чего, собственно, пользовательская эффективность и оказывается вообще меньше единицы).

Если r равно или кратно n, то $\left]\frac{r}{n}\right[=\frac{r}{n}$; во всех других случаях $\left]\frac{r}{n}\right[>\frac{r}{n}$. Поэтому

$$E_{\text{п}}=\frac{r}{n}:\left]\frac{r}{n}\right[\leqslant 1.$$

На рис. 2.2.4 в качестве примера показаны зависимости величины $\frac{r}{n\cdot\left]\frac{r}{n}\right[}$ от n для $r=8$ и $r=12$. Видно, что $r\Big/\left(n\cdot\left]\frac{r}{n}\right[\right)$ представляет собой не монотонную функцию n. Общая тенденция, однако, состоит в том, что $\frac{r}{n\cdot\left]\frac{r}{n}\right[}$ тем ближе к единице, чем больше r и чем меньше n.

В действительности величина $\frac{r}{n\cdot\left]\frac{r}{n}\right[}$ однозначно свя-

зана с отношением r/n. Эта зависимость представлена на рис. 2.2.5. В отличие от рис. 2.2.4, где мы условно соединили линиями точки, соответствующие целочисленным значениям аргумента (n), функция, представленная на рис. 2.2.5, имеет смысл при любых положительных рациональных значениях аргумента (r/n). Она представляет собой кусочно-линейную функцию с бесконечным количеством разрывов — во всех точках, где r/n —

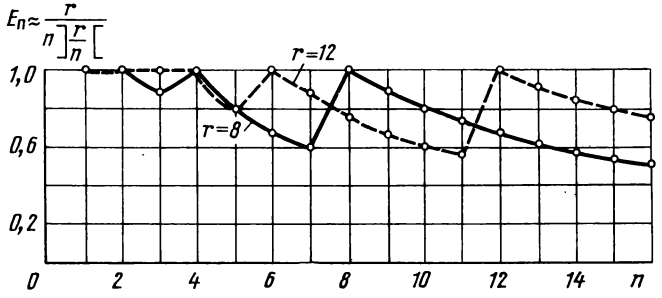


Рис. 2.2.4

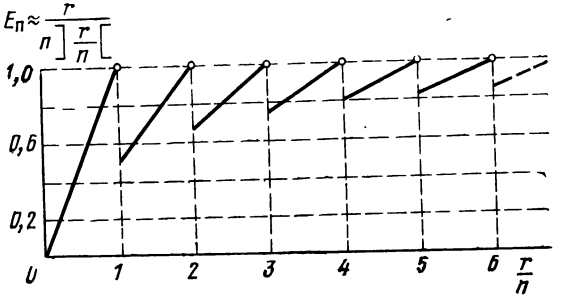


Рис. 2.2.5

целое число, причем в этих точках значения функции равны единице (предел слева).

Если на разных этапах решения задачи ее ранг не остается постоянным и принимает значения из множества $\{r_i\}$, то

$$E_n \approx \frac{1}{n} \sum_{(i)} p_i \frac{r_i}{\left]\frac{r_i}{n}\right[},$$

где p_i — вероятность того, что ранг задачи есть r_i.

Другие факторы, влияющие на пользовательскую эффективность многопроцессорных систем типа IV, учтем введением коэффициентов:

$K_р$ — коэффициент, учитывающий наличие расхождений в программе;

$K_ц$ — коэффициент, учитывающий возможности аппаратуры последовательно-параллельной обработки, если таковая имеется в системе (см. п. 2.1.3, 4°);

$K_о$ — коэффициент, учитывающий влияние на эффективность системы наличия в программе «интегральных» операций (см. п. 1.2.1, 2°) и возможности общих арифметико-логических цепей и локальных связей между процессорами, если таковые имеются в системе (см. п. 2.1.3, 6°).

Полное выражение для пользовательской эффективности многопроцессорной системы типа IV при этом должно быть записано в виде:

$$E_п = \frac{1}{n} \sum_{(i)} p_i \frac{r_i}{\left\lceil \frac{r_i}{n} \right\rceil} K_р K_ц K_о.$$

Ниже мы рассмотрим более подробно влияние перечисленных факторов на величину $E_п$. Мы увидим, что $K_р$, $K_ц$ и $K_о$ определенным образом зависят как от характеристик решаемых задач, так и от количества основных процессоров, состава и характеристик другого оборудования системы.

Точное определение величин $K_р$, $K_ц$ и $K_о$ связано во многих случаях с весьма кропотливыми расчетами и необходимостью знать очень детальные характеристики решаемой задачи. Забегая, однако, вперед, можно сказать, что все эти факторы оказывают не очень значительное влияние на величину пользовательской эффективности, т. е. что для широкого круга задач, подходящих для решения с помощью вычислительных систем рассматриваемого типа, произведение $K_р K_ц K_о$ близко к единице, а зависимость $E_п$ от n и r близка к приведенной выше (рис. 2.2.4, 2.2.5).

2°. Рассмотрим сначала *влияние на пользовательскую эффективность расхождений в программе* обработки различных компонент векторов или матриц, или различных точек функции, или информации о различных объектах из множества однородных объектов. Для упрощения изложения будем иметь в виду в дальнейшем главным

образом параллелизм множества объектов, а не общий случай естественного параллелизма. Как мы уже говорили, именно в этом частном случае расхождения в программах играют наиболее значительную роль и именно для этого случая целесообразно ввести в систему, например, процессор переформирования массивов.

Если в системе отсутствуют и процессор переформирования массивов, и схема проверки однородности выборки, то все расхождения в программе реализуются только за счет механизма масок, а коэффициент $K_р$, входящий в выражение для $E_п$, есть просто величина, обратная расхождению программы D, $K_р \approx 1/D$. Знак приближенного равенства мы поставили здесь потому, что в самом определении величины D, приведенном в п. 1.2.1, 3°, имеется некоторая неточность (вернее, недоговоренность), состоящая в следующем. В формуле для величины D (с. 20)

$$D = r \frac{\sum\limits_{(k)} \sum\limits_{s_k=1}^{m_k} l_{ks_k}}{\sum\limits_{(k)} \sum\limits_{s_k=1}^{m_k} r_{ks_k} l_{ks_k}}$$

молчаливо предполагается, что длины ветвей l_{ks_k} (т. е. длины подпрограмм обработки информации об объектах s_k-го класса, $s_k = 1, 2, \ldots, m_k$, на k-м этапе исполнения программы) одинаковы в многопроцессорной системе и в обычной однопроцессорной машине. Иначе в числителе и в знаменателе выражения для D нужно было бы записать разные величины l_{ks_k}: в числителе — длины ветвей в программе многопроцессорной системы, в знаменателе — соответствующие длины для однопроцессорной машины.

При идентичной системе команд величины l_{ks_k} для многопроцессорной системы и для однопроцессорной машины действительно можно с достаточной точностью считать одинаковыми, но только в том случае, когда величины l_{ks_k} не слишком малы — например, если величины l_{ks_k}, измеренные в количестве машинных операций, больше 5 — 6.

Для пояснения рассмотрим простейший пример. Пусть расхождение в программе обработки информации о разных объектах на k-м этапе описывается (в терминах АЛГОЛ-60) условным оператором ... **if** <отношение> **then** <арифметическое выражение 1> **else** <арифметическое выражение 2>; ... Иначе говоря, на k-м этапе объекты делятся на 2 класса ($m_k=2$): те, для которых отношение истинно, и те, для которых отношение ложно. Для первых вычисляется арифметическое выражение 1, для вторых — арифметическое выражение 2.

На машинном языке для однопроцессорной машины выполнение этого оператора описывается программой:

— . . . . . .

— вычисление логического выражения, соответствующего заданному отношению;
— условная передача управления (на продолжение либо к M1);
— вычисление арифметического выражения 1;
— безусловная передача управления к M2;
M1: вычисление арифметического выражения 2;

M2: . . . . . .

То же для многопроцессорной системы выглядит следующим образом:

— . . . . . .

— вычисление логического выражения, соответствующего заданному отношению, для группы объектов (формирование массива **маски**);
— вычисление арифметического выражения 1 для группы объектов под прямой маской;
— вычисление арифметического выражения 2 под инверсной маской;

— . . . . . .

Сопоставление программ для однопроцессорной машины и для многопроцессорной системы показывает, что длина ветви вычислений для объектов 1-го класса, l_{k1}, в программе однопроцессорной машины содержит две «лишние» операции по сравнению с программой многопроцессорной системы: условную передачу управления (на продолжение либо к M1) и безусловную передачу управления к M2. Для объектов 2-го класса (для которых заданное отношение ложно) отличие в величине l_{k2} составляет одну операцию — условной передачи управления.

Если бы расхождения имелись на всех этапах исполнения программы и были при этом только указанного вида, то коэффициент K_p должен был бы определяться по формуле

$$K_p = \frac{\sum\limits_{(k)} (r_{k1} l_{k1} + r_{k2} l_{k2})}{r \sum\limits_{(k)} ((l_{k1} - 2) + (l_{k2} - 1))}.$$

При условии l_{k1}, $l_{k2} \gg 3$ в этом случае действительно

$$K_p \approx \frac{1}{D} = \frac{\sum\limits_{(k)} (r_{k1} l_{k1} + r_{k2} l_{k2})}{r \sum\limits_{(k)} (l_{k1} + l_{k2})}.$$

Однако вообще в этом случае $K_p > \frac{1}{D}$, а если l_{k1} и l_{k2} малы, то может оказаться $K_p > 1$. В качестве предельного случая можно представить себе ситуацию, в которой вычисления арифметических выражений 1 и 2 содержат всего по одной арифметической операции; тогда $l_{k1}=3$ (условная передача управления — арифметическая операция — безусловная передача управления), а $l_{k2}=2$ (условная передача управления — арифметическая операция). Если это так, то

$$K_p = \frac{\sum\limits_{k=1}^{N} (3r_{k1} + 2r_{k2})}{rN} \geqslant 1$$

(где N — количество этапов, $k=1, 2, \ldots, N$). В пределе, при $r_{k1} \longrightarrow r$ и соответственно $r_{k2} = r - r_{k1} \longrightarrow 0$, $K_p \longrightarrow 3$.

Конечно, рассмотренный пример весьма условен. Однако он показывает, что расхождения, имеющиеся в программах, не обязательно снижают пользовательскую эффективность вычислительной системы типа IV. В некоторых частных случаях может оказаться, что простейший механизм масок без каких-либо дальнейших усовершенствований (введения процессора переформирования массивов и схемы проверки однородности выборки) может обеспечить значительное повышение пользовательской эффективности системы. При наиболее благоприятных ситуациях пользовательская эффективность $E_п$ может получиться даже существенно больше единицы (в нашем примере до 3).

Теперь остановимся на вопросе о целесообразности введения в состав системы *схемы проверки однородности выборки* (см. п. 2.1.3, 5°).

Пусть, как и прежде, r — ранг задачи (количество объектов, по которым ведется обработка информации), n — количество процессоров в системе, m_k — количество подмножеств (классов), на которые разделяется на k-м этапе обработки множество из r объектов, r_{ks_k} — количество объектов, принадлежащих s_k-му подмножеству.

l_{ks_k} — длина ветви обработки информации на k-м этапе выполнения программы по объекту, принадлежащему s_k-му подмножеству ($s_k=1, 2, \ldots, m_k$).

Для упрощения дальнейшего изложения отвлечемся от рассмотренных выше обстоятельств, в силу которых длины ветвей в программе многопроцессорной системы могут отличаться от длин соответствующих ветвей в программе однопроцессорной машины. Иначе говоря, все соотношения будем записывать так, будто величины l_{ks_k} достаточно велики. Однако схема проверки однородности выборки одинаково работает при любых длинах ветвей, и тот выигрыш в эффективности системы, который достигается с ее помощью, может просто накладываться на выигрыш, получаемый за счет других факторов.

Отношение r_{ks_k}/r можно приравнять вероятности того, что некоторый произвольно выбранный объект из множества, содержащего r объектов, принадлежит s_k-му классу. Будем полагать, что объекты каждого s_k-го класса распределены в множестве из r объектов равномерно. Тогда вероятность того, что в выборке из n очередных объектов имеется хотя бы один объект, принадлежащий s_k-му классу, есть, очевидно, $1-(1-r_{ks_k}/r)^n$. Если r достаточно велико, то при наличии в системе схемы проверки однородности выборки величину $1-(1-r_{ks_k}/r)^n$ можно рассматривать как частоту, с которой при обработке групп по n объектов на k-м этапе должна исполняться ветвь l_{ks_k}. При этом, очевидно,

$$K_{\text{р}}=\frac{\sum\limits_{(k)}\sum\limits_{s_k=1}^{m_k} r_{ks_k}l_{ks_k}}{r\sum\limits_{(k)}\sum\limits_{s_k=1}^{m_k}\left(1-(1-r_{ks_k}/r)^n\right)l_{ks_k}}>\frac{1}{D}=\frac{\sum\limits_{(k)}\sum\limits_{s_k=1}^{m_k} r_{ks_k}l_{ks_k}}{r\sum\limits_{(k)}\sum\limits_{s_k=1}^{m_k} l_{ks_k}}.$$

Ясно, что эффект от введения схемы проверки однородности выборки тем больше, чем меньше количество объектов r_{ks_k} (по сравнению с общим количеством объектов r), принадлежащих s_k-му классу, для которого со-

ответствующая длина ветви l_{ks_k} велика по сравнению с длинами других ветвей, и чем меньше количество процессоров n.

Например, если бы на k-м этапе объекты делились на два примерно равных класса ($m_k=2$; $r_{k1}\approx r_{k2}\approx r/2$) и длины ветвей для этих классов были бы примерно равны ($l_{k1}\approx l_{k2}\approx l_k$), то в знаменателе выражения для $\frac{1}{D}$ k-й этап был бы представлен слагаемым

$$l_{k1}+l_{k2}\approx 2l_k,$$

а в знаменателе приведенного выше выражения для $K_р$, учитывающего наличие схемы проверки однородности выборки, — слагаемым

$$(1-(1-r_{k1}/r)^n)l_{k1}+(1-(1-r_{k2}/r)^n)l_{k2}\approx 2\cdot(1-2^{-n})l_k.$$

При $n=2$

$$2(1-2^{-n})l_k\approx 0{,}75(2l_k),$$

при $n=4$

$$2(1-2^{-n})l_k\approx 0{,}94(2l_k),$$

т. е. даже при малых значениях n выигрыш от введения схемы проверки однородности выборки невелик.

Другое дело, если бы классы были резко различны по своим характеристикам. Представим себе, например, что количество объектов 1-го класса составляет всего $^1/_{10}$ от общего числа объектов, а остальные $^9/_{10}$ объектов относятся ко второму классу и при этом длина ветви обработки для объектов 1-го класса в 10 раз больше, чем для объектов 2-го класса; иначе говоря, $r_{k1}=0{,}1r$, $r_{k2}=0{,}9r$, $l_{k1}=10l$, $l_{k2}=l$, где l — некоторая условная единица длины ветви. Если это так, то в знаменателе для $1/D$ k-й этап представлен слагаемым

$$l_{k1}+l_{k2}=11l,$$

а в знаменателе выражения для $K_р$, учитывающего наличие схемы проверки однородности выборки, — слагаемым

$$(1-(1-r_{k1}/r)^n)l_{k1}+(1-(1-r_{k2}/r)^n)l_{k2}=$$
$$=(1-0{,}9^n)10l+(1-0{,}1^n)l.$$

При $n=2$

$$(1-0{,}9^2)10l+(1-0{,}1^2)l\approx 0{,}26(11l),$$

при $n=4$

$$(1-0{,}9^4)10l+(1-0{,}1^4)l\approx 0{,}40(11l),$$

при $n=6$

$$(1-0{,}9^6)10l+(1-0{,}1^6)l\approx 0{,}52(11l).$$

Как видим, выигрыш здесь достаточно значителен.

Рассмотрим, наконец, *эффект* от введения в состав системы *процессора переформирования массивов*

(п. 2.1.3, 5°) и его влияние на величину коэффициента $K_р$.

Процессор переформирования массивов, если он есть в составе системы, имеет смысл включать в работу на тех этапах вычислений, когда расхождение в программах образуется сравнительно длинными ветвями.

Сам процессор переформирования массивов требует для своей работы некоторого времени $T_{пм}$, которое в общем случае является неубывающей функцией трех переменных: r — количества объектов в обрабатываемом множестве, m_k — количества классов, на которые должно быть разделено обрабатываемое множество из r объектов, M_k — количества параметров, которыми описывается каждый из r объектов к началу k-го этапа вычислений и которые потребуются для выполнения k-го или любого последующего этапа. Таким образом,

$$T_{пм}^{(k)} = T_{пм}(r, m_k, M_k),$$

где верхний индекс (k) указывает, что речь идет о времени работы процессора на k-м этапе исполнения программы.

Зависимость $T_{пм}$ от M_k требует, возможно, дополнительного пояснения. Допустим, что решаемая задача состоит в обработке траекторий движения заряженных частиц, причем r — количество обрабатываемых траекторий. Допустим также, что к началу k-го этапа обработки для каждой i-й частицы ($i=1, 2, \ldots, r$) известны ее координаты в пространстве (для определенности, скажем, декартовы) x_i, y_i, z_i, момент времени t_i, к которому относятся эти координаты, составляющие скоростей $\dot{x}_i$, $\dot{y}_i$, $\dot{z}_i$, а также некоторый признак U_i, характеризующий тип частицы; таким образом, общее количество параметров, которыми описывается траектория каждой частицы к началу k-го этапа вычислений, равно 8. Может оказаться, что какой-либо (или какие-либо) из этих параметров не нужен ни для выполнения k-го этапа, ни вообще в последующих этапах; если это не так, то $M_k=8$. Ясно, что на предыдущих и на последующих этапах вычислений количество параметров, описывающих каждый из объектов, может быть другим.

Общая память n-процессорной системы должна иметь либо широкий формат обращения — так, чтобы за одно обращение из нее можно было выбрать одновременно n слов (по числу процессоров), расположенных в последовательных ячейках, либо высокую скорость по сравнению с процессорной частью, чтобы эти n слов могли быть достаточно быстро выбраны по последовательным адресам. Так или иначе M_k характеристик для объектов должны быть расположены в ней в виде M_k массивов, каждый из которых занимает r последовательных ячеек. Например, в первом массиве, начиная от определенного адреса, расположены в последовательных ячейках все x_i: сначала x_1, затем x_2 и т. д. до x_r; во втором массиве — y_1, y_2, ..., y_r, и т. д.

Когда речь идет о том, чтобы процессор переформирования массивов разделил по определенным маскам множество из r объектов на m_k подмножеств, то это значит, что он должен переформировать фактически M_k массивов под управлением одного и того же набора масок, разделив каждый из них на m_k подмассивов.

Поэтому $T_{\text{пм}}^{(k)}$ есть вообще не убывающая, может быть, даже линейная функция от M_k.

При определенной структуре вычислительной системы и пользовательской программы время работы процессора переформирования массивов может быть полностью или частично перекрыто с временем выполнения других операций основными процессорами системы. Поэтому будем считать, что в полное время исполнения k-го этапа программы оно входит в виде слагаемого $\varkappa T_{\text{пм}}^{(k)} = \varkappa T_{\text{пм}}^{(k)}(r, m_k, M_k)$, где $0 \leqslant \varkappa \leqslant 1$.

После того, как процессор переформирования массивов разделил исходное множество объектов на m_k подмножеств — в соответствии с количеством разных ветвей для k-го этапа — потери времени, связанные с расхождением программы на k-м этапе, должны, казалось бы, исчезнуть. Однако в общем случае вместо них появляются некоторые дополнительные потери. Они связаны с тем, что прежде мы имели одно множество из r объектов и в последнем цикле его обработки у нас один раз простаивали $\left]\frac{r}{n}\right[\cdot n - r$ процессоров, теперь же вместо этого имеется m_k множеств, содержащих по $r_{k_1}, r_{k_2}, \ldots, r_{km_k}$ объектов и в последнем цикле обработки каждого из них простаивает соответственно $\left]\frac{r_{ks_k}}{n}\right[\cdot n - r_{ks_k}$ процессоров, где $s_k = 1, 2, \ldots, m_k$.

С учетом этих обстоятельств оказывается, что включение на k-м этапе процессора переформирования массивов целесообразно при условии, что

$$\varkappa T_{\text{пм}}(r, m_k, M_k) + \sum_{s_k=1}^{m_k} \left]\frac{r_{ks_k}}{n}\right[\cdot l_{ks_k} <$$

$$< \left]\frac{r}{n}\right[\cdot \sum_{s_k=1}^{m_k} (1 - (1 - r_{ks_k}/r)^n)\, l_{ks_k},$$

где в правой части приняты во внимание также возможности схемы проверки однородности выборки. Время исполнения k-го этапа программы при этом равно левой либо правой части неравенства — в зависимости от того, какая из них меньше.

Поскольку всегда $\left]\frac{r_{ki}}{n}\right[+ \left]\frac{r_{kj}}{n}\right[\leqslant \left]\frac{r_{ki}+r_{kj}}{n}\right[$, а $T_{\text{пм}}^{(k)}$ является неубывающей функцией m_k, то здесь имеется некоторая возможность оптимизации. Она состоит в том, что при условии, когда на k-м этапе программы обработки информации по объектам s_{ki}-го и s_{kj}-го классов мало отличаются друг от друга, может быть, имеет смысл при переформировании массивов не разделять между собой эти два класса, чтобы уменьшить таким образом общее количество классов m_k (и соответственно величину $T_{\text{пм}}^{(k)}$) и получить вместо двух классов объектов, содержащих по r_{ki} и r_{kj} объектов, один класс, содержащий $r_{ki}+r_{kj}$ объектов. Зато дальше, при обработке информации по объектам этого объединенного класса, расхождение в программах придется проходить с помощью обычного механизма масок, используя также возможности схемы проверки однородности выборки.

Итак, если расхождение в программах образуется за счет коротких разветвлений — и может быть, весьма многочисленных, так что суммарная величина D велика,— эти расхождения эффективно реализуются механизмом масок; при этом $K_{\text{р}}$ может получиться даже больше единицы. Если суммарная величина расхождения D образуется за счет достаточно длинных разветвлений на отдельных этапах программы, то может быть эффективно использован процессор переформирования массивов, в результате чего коэффициент $K_{\text{р}}$ получится хотя и меньше единицы, но весьма к ней близок. Наихудшим положение оказывается в случае, когда суммарная величина расхождения D образуется разветвлениями «средней» длины: выигрыш в количестве операций управления, которые дает вообще механизм масок, здесь уже незначителен, а включать процессор переформирования массивов на каждом из этапов, где встречается такое расхождение, еще нецелесообразно; величина $K_{\text{р}}$ при этом близка к $1/D$ — лишь немного больше за счет уменьшения количества операций управления и, возмож-

но, за счет наличия схемы проверки однородности выборки. Впрочем, схема проверки однородности выборки дает выигрыш и при «малых» разветвлениях.

Опыт показывает, что совокупный эффект рассмотренных факторов позволяет получить для большинства задач, обладающих естественным параллелизмом (или, в частности, параллелизмом множества объектов) величину $K_р > 0{,}9$.

3°. *Коэффициент* $K_ц$ учитывает, как было сказано выше, влияние на эффективность системы аппаратуры последовательно-параллельной обработки. Если этой аппаратуры в системе нет, то $K_ц = 1$. При ее наличии всегда $K_ц > 1$, но величина $K_ц$ существенно зависит от конкретных свойств задачи.

Представим себе в качестве крайнего примера, что задача, решаемая вычислительной системой, состоит в основном в суммировании многомерных (r-мерных) векторов, умножении таких векторов на число и т. п. операций. Программа для обычной (однопроцессорной) машины при этом содержит короткие, но повторяющиеся много раз циклы. Например, для получения суммы r-мерных векторов $Z = X + Y$ программа (в терминах АЛГОЛ-60) выглядит следующим образом:

... **for** $I := 1$ **step** 1 **until** R **do** $Z[I] := X[I] + Y[I]$; ... .

Реализация этого цикла в программе на машинном языке имеет примерно следующий вид:

— . . . . . .
— засылка единицы в индексный регистр ($I := 1$);
M: выполнение сложения очередных компонент векторов и запись результата в память ($Z[I] := X[I] + Y[I]$);
— модификация индексного регистра ($I := I + 1$);
— вычитание $R - I$;
— условная передача управления по алгебраическому знаку результата предыдущей операции (если $R - I \geqslant 0$, то к M, если $R - I < 0$, то на продолжение программы);
— . . . . . .

Мы не говорим здесь в точности об особенностях машинного языка, но ясно, что в любом случае количество полезных операций в цикле (сложение $X[I] + Y[I]$) в 2—3 раза меньше, чем количество служебных операций для организации цикла (модификация индексного регистра, вычитание $R - I$, условная передача управления).

В n-процессорной системе с аналогичным машинным языком и без аппаратуры последовательно-параллельной обработки единственное отличие в программе на машинном языке состояло бы в том, что вместо операции $I := I + 1$ нужно было бы предусмотреть операцию $I := I + n$, потому что в каждом цикле одновременно бы

суммировалось n пар компонент. Такой цикл повторялся бы $]r/n[$ раз — вместе со всеми своими полезными и служебными операциями; иначе говоря, коэффициент $K_ц$ был бы равен 1 — как и должно быть при отсутствии аппаратуры последовательно-параллельной обработки.

Если в n-процессорной системе предусмотрена аппаратура, описанная в п. 2.1.3, 4° (см. рис. 2.1.5), то программа на машинном языке выглядит следующим образом:

— . . . . . .
— засылка размерности векторов в регистр r;
— выполнение сложения соответствующих компонент и запись результатов $(Z[I] := X[I] + Y[I])$;
— . . . . . .

Служебные операции в цикле здесь отсутствуют совсем, поэтому $K_ц \approx 3$—4.

Конечно, приведенный пример не очень реален. Для обычных задач $K_ц$ получается меньше чем 3—4, но достигает иногда величины 1,3—1,5.

Можно, разумеется, возразить, что и в обычной (однопроцессорной) машине может быть предусмотрена аппаратура для автоматической реализации циклов — даже более простая, чем на рис. 2.1.5,— которая без специальных инструкций производила бы в каждом цикле преобразование индексного регистра и проверку условия выхода из цикла. Понятны, однако, причины, по которым эту аппаратуру целесообразно применять именно в многопроцессорных системах типа IV, где расчет идет на задачи, обладающие естественным параллелизмом с высоким рангом r и где одно устройство управления работает сразу с n арифметико-логическими процессорами. В обычной универсальной машине, рассчитанной на более широкий круг задач и содержащей одно устройство управления на одно арифметико-логическое устройство, такая аппаратура, как правило, отсутствует.

4°. *Коэффициент* K_o, учитывающий влияние на эффективность системы общих арифметико-логических цепей и локальных связей между процессорами, оценить в сколько-нибудь общем виде трудно.

Если интегральные операции в задаче вообще отсутствуют или их мало, $K_o = 1$ независимо от того, какие

общие арифметико-логические цепи предусмотрены в системе. С этим случаем мы в основном имеем дело, когда речь идет о параллелизме множества объектов. Сравнительно редкие интегральные операции, которые встречаются в таких задачах, состоят, например, в том, чтобы одинаковым образом проверить некоторое отношение для каждого из объектов (например, проверить, больше ли определенный параметр, характеризующий данный объект, чем заданное — одинаковое для всех объектов — число), а затем определить, сколько имеется всего объектов (из r), удовлетворяющих указанному отношению. Механизм размножения операндов (п. 2.1.3, 2°) и процессор переформирования массивов (2.1.3, 5°) в сочетании с обычными возможностями многопроцессорной системы типа IV обеспечивают достаточно эффективную реализацию подобных операций без каких-либо дополнительных усложнений системы.

В общем случае при решении задач с естественным параллелизмом величина K_0 существенно зависит от конкретных свойств задачи и возможностей общих арифметико-логических цепей и локальных связей, имеющихся в системе. В обычных ситуациях $K_0 \approx 0{,}5—1{,}5$.

5°. В итоге проведенного выше рассмотрения мы видим, что для многопроцессорных систем типа IV коэффициенты $K_р$, $K_ц$, K_0 зависят от конкретных свойств задач и особенностей систем, но в первом приближении — особенно, если расхождение D не слишком велико — можно считать, что $K_р K_ц K_0 \approx 1$.

При этом эффективность многопроцессорной системы типа IV зависит главным образом от отношения ранга задачи r к количеству процессоров n — так, как об этом говорилось в п. 1° (см. также рис. 2.2.5). Если требовать, например, чтобы эффективность n-процессорной системы была порядка 0,8 или выше, то для решения с помощью этой системы должны отбираться задачи ранга $r > (4—5)n$. Чем меньше n, тем более универсальна система.

Для задач с большим значением расхождения D (скажем, $D \approx 2—3$), прежде чем ставить их решение с помощью многопроцессорной системы типа IV, желательно проводить специальное рассмотрение вопроса о том, как справятся аппаратные средства системы с этим расхождением и не слишком ли низким окажется коэффициент $K_р$ (см. п. 2°).

6°. Пользовательская *эффективность конвейерной системы* типа IV — так же, как и эффективность многопроцессорной системы типа IV — в основном зависит от величины отношения r/n, где r — ранг задачи, n — количество ступеней в конвейерной системе.

Ясно, что выполнение любой операции над r операндами или r парами операндов должно занять в n-ступенной конвейерной системе $r+n-1$ тактов. При этом

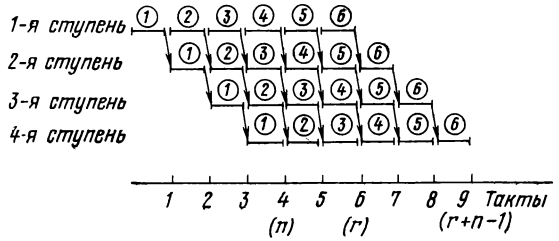


Рис. 2.2.6

1-я ступень занята полезными операциями в течение первых r тактов, а последние $n-1$ тактов простаивает; 2-я ступень работает от 2- до $(r+1)$-го тактов включительно, 3-я — от 3- до $(r+2)$-го тактов, ..., n-я ступень — от n- до $(r+n-1)$-го тактов. (Для пояснения на рис. 2.2.6 показана временная диаграмма работы системы для $r=6$, $n=4$, цифры в кружках обозначают номер обрабатываемого операнда или обрабатываемой пары операндов.)

Таким образом, каждая ступень системы из общего количества $r+n-1$ тактов занята полезными операциями r тактов и простаивает $n-1$ тактов. В первом приближении поэтому можно считать, что пользовательская эффективность n-ступенной конвейерной системы типа IV при решении задачи ранга r есть

$$E_{\text{п}} \approx \frac{r}{r+n-1} = \frac{r/n}{\frac{r}{n}+1-\frac{1}{n}}.$$

В отличие от приведенного в 1° аналогичного выражения для пользовательской производительности n-процессорной системы типа IV, здесь величина $E_{\text{п}}$ зависит не только от отношения r/n, но до некоторой степени

также от величины n. Однако эта зависимость сказывается при малых количествах ступеней n; если $n \gg 1$, то $E_{\text{п}} \approx \frac{r/n}{(r/n)+1}$.

Зависимости $E_{\text{п}}$ от r/n для конвейерных систем типа IV для разных n представлены на рис. 2.2.7. На этом же рисунке показаны зависимости $E_{\text{п}}$ от n при некоторых фиксированных r.

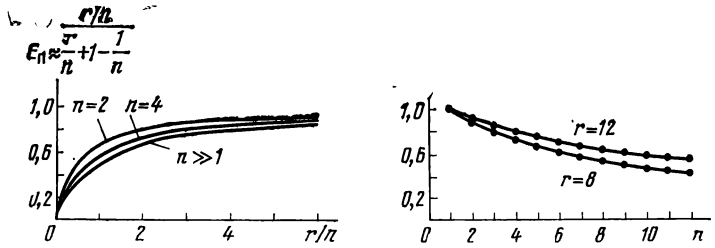


Рис. 2.2.7

Полное выражение для $E_{\text{п}}$ должно включать также коэффициенты $K_{\text{р}}$, $K_{\text{о}}$, $K_{\text{ц}}$, аналогичные введенным в п. 1° для многопроцессорных систем типа IV:

$$E_{\text{п}} = \frac{r/n}{\frac{r}{n}+1-\frac{1}{n}} K_{\text{р}} K_{\text{ц}} K_{\text{о}}.$$

Свойства задач, от которых зависят эти коэффициенты для конвейерных систем, те же, что и для многопроцессорных систем, однако соответствующая аппаратура конвейерной системы выглядит иначе, чем в многопроцессорной системе, и конкретные значения коэффициентов $K_{\text{р}}$, $K_{\text{ц}}$, $K_{\text{о}}$ могут получаться другими.

Например, когда в n-процессорной системе встречаются расхождения в программах, т. е. требуется выполнение определенных операций под маской, то основные затруднения происходят фактически потому, что имеющиеся в системе n процессоров обеспечиваются одновременно n операндами за счет широкого формата обращения к памяти. При каждом обращении к памяти прочитывается одновременно n слов, расположенных в последовательных ячейках. Если переформирование массивов заранее не сделано, то части из этих n операндов могут соответствовать единицы в маске, другой

части — нули. Соответственно при выполнении операции под маской часть процессоров будет простаивать.

В конвейерной системе типа IV операнды читаются из памяти по одному — в принципе тоже по последовательным адресам. Нетрудно, однако, предусмотреть схему просмотра маски «вперед». Если должна выполняться операция под маской (скажем, прямой), то эта схема пропускает чтение тех операндов, для которых в маске содержатся нули, и сразу выбирает очередной операнд, для которого в маске содержится единица, обеспечивая одновременно соответствующую модификацию адреса. Таким образом, схема просмотра маски вперед заменяет процессор переформирования массивов (а схема проверки однородности выборки вообще становится ненужной), причем ее работа практически не требует специального времени.

На самом деле, конечно, некоторое время для работы такой схемы требуется; но она может быть включена на правах дополнительной ступени конвейера, предшествующей ступени чтения очередного операнда.

Может оказаться, что просмотреть маску вперед на произвольное количество разрядов (до r) трудно, и схема просматривает только группы из ограниченного количества разрядов маски (скажем, до 8 или до 16): тогда определенная вероятность пропуска тактов при выполнении операций под маской остается — тем меньшая, чем больше разрядов маски просматривается одновременно.

Аналогичным образом на правах дополнительных ступеней конвейера могут быть включены арифметико-логические цепи для реализации интегральных операций.

Что касается аппаратной реализации циклов (аналогичной аппаратуре параллельно-последовательной обработки в многопроцессорных системах), то она заложена в самой идее построения конвейерной системы типа IV.

В итоге оказывается, что произведение $K_{\text{р}}K_{\text{п}}K_{\text{о}}$ для конвейерных систем типа IV при прочих равных условиях несколько выше, чем для многопроцессорных систем этого типа. В то же время нетрудно заметить, что кривая $E_{\text{п}}=E_{\text{п}}(r/n)$ для конвейерной системы при $n\gg 1$ на рис. 2.2.7 является нижней огибающей кусочно-линейной функции, представляющей зависимость $E_{\text{п}}$ от r/n для многопроцессорных систем (рис. 2.2.5). Поэтому

с достаточной точностью можно считать, что зависимости пользовательской эффективности $E_п$ от отношения r/n для многопроцессорных и для конвейерных систем типа IV примерно одинаковы.

2.3. КОМБИНИРОВАННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

2.3.1. ЗАДАЧА ПОСТРОЕНИЯ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ С МАКСИМАЛЬНЫМ ПОЛЬЗОВАТЕЛЬСКИМ БЫСТРОДЕЙСТВИЕМ

1°. Пусть требуется создать вычислительную систему с максимально возможным пользовательским быстродействием, предназначенную для решения особо крупных задач. Допустим, что соображения, связанные со стоимостью, габаритами и т. п. характеристиками, не слишком существенны. Тем не менее определенные критерии целесообразности должны быть соблюдены. Например, вряд ли целесообразно удваивать количество процессоров в многопроцессорной системе, если известно, что это приведет к увеличению пользовательского быстродействия лишь на несколько процентов. Обобщенным критерием такой целесообразности, собственно, и является пользовательская эффективность вычислительной системы.

Особо крупные задачи, на которые мы здесь рассчитываем, обладают обычно в той или иной степени всеми видами параллелизма, о которых говорилось в § 1.2. Следовательно, в принципе можно было бы строить систему любого из рассмотренных выше типов.

Однако рассмотрение, например графиков на рис. 2.2.2 показывает, что в общем случае (кроме специальных задач) пользовательское быстродействие вычислительной системы типа I или II при более или менее разумных значениях пользовательской эффективности может быть получено лишь в 4—4,5 раза выше, чем быстродействие обычной машины. Для этого n (количество машин в многомашинном комплексе или процессоров в многопроцессорной системе, или ступеней и соответственно устройств управления в конвейерной системе типа II) должно быть порядка 6—10.

Аналогичным образом рассмотрение рис. 2.2.3 показывает, что при наиболее тщательной оптимизации программ с целью выявления в них параллелизма смежных операций вычислительная система типа III может обес-

печить лишь в 3—3,5 раза более высокое быстродействие, чем обычная вычислительная машина. При этом количество процессоров в ней n (или количество ступеней, если речь идет о конвейерной системе) должно быть порядка 4—6.

Несколько более обнадеживающие результаты дает рассмотрение графиков пользовательской эффективности для вычислительных систем типа IV, приведенных на рис. 2.2.5 и 2.2.7. Например, нетрудно видеть, что если бы ранг решаемых задач был не меньше 10, то, имея 10 процессоров в многопроцессорной системе типа IV (или соответственно 10 ступеней в конвейерной системе), можно было бы получить выигрыш в быстродействии по сравнению с обычной машиной не менее чем в 5 раз — больше, чем можно получить от тех же 10 машин (или процессоров, или ступеней конвейера) в вычислительной системе типа I или II, и больше, чем можно получить от вычислительной системы типа III.

Чем больше ранг естественного параллелизма, на который можно рассчитывать, тем больше можно принять величину n (число процессоров в многопроцессорной системе или ступеней в конвейерной системе *)) для вычислительной системы типа IV и тем больше получится ее пользовательское быстродействие. Скажем, если бы можно было считать, что ранг естественного параллелизма будет порядка 100, то, приняв $n=100$, можно было бы получить увеличение пользовательского быстродействия не менее, чем в 50 раз по сравнению с однопроцессорной машиной. Такого результата нельзя, очевидно, достигнуть в вычислительных системах типа I, II или III.

Вообще говоря, «особо крупные задачи», о которых здесь идет речь, во многих случаях как раз и являются особо крупными потому, что имеют высокий ранг естественного параллелизма: они состоят, например, в интегрировании дифференциальных уравнений в частных производных конечно-разностными методами, причем количество узлов решетки велико, в операциях над матрицами большого ранга, в обработке информации о большом количестве более или менее однородных объектов и т. п. Тем не менее ясно, что чем меньше количество процессоров (ступеней конвейера) n в вычислительной

*) См., однако, сноску на с. 66.

системе типа IV и чем меньше, следовательно, может быть допустимый ранг естественного параллелизма r, тем шире круг задач, для решения которых можно будет эффективно использовать такую систему.

2°. *Построение комбинированных систем* представляет собой более универсальный путь создания вычислительных систем с максимальной пользовательской производительностью.

Как видно из предыдущего, для вычислительных систем любого типа пользовательская эффективность $E_{\text{п}}$ зависит от n (числа машин в многомашинном комплексе или процессоров в многопроцессорной системе, или ступеней конвейерной системы)

$$E_{\text{п}}=E_{\text{п}}(n),$$

причем в общем случае это функция убывающая. Некоторым исключением являются многопроцессорные системы типа IV, для которых при фиксированном r и при $n\leqslant r$ функция $E_{\text{п}}(n)$ немонотонна (см. рис. 2.2.4 на с. 82); однако и здесь обнаруживается общая тенденция к уменьшению $E_{\text{п}}$ с ростом n.

Идея построения комбинированных систем состоит в том, что снижение пользовательской эффективности с увеличением n в системах типа I или II, III и IV происходит по разным причинам, хотя результаты, получающиеся вследствие этих разных по существу факторов, могут быть похожими (ср., например, рис. 2.2.2, где показана зависимость $E_{\text{п}}(n)$ для систем типа I или II, с аналогичными зависимостями для систем типа III на рис. 2.2.3 и для систем типа IV на рис. 2.2.4 и 2.2.7). Если задача, как мы предполагали в п. 1°, в той или иной мере обладает всеми видами параллелизма, то в построении вычислительной системы можно попытаться использовать их все. При этом использование одного вида параллелизма никак не сказывается (или, может быть, почти никак не сказывается) на возможностях использования других видов параллелизма.

Скажем, в задаче на каждом этапе ее решения можно выделить несколько независимых ветвей, которые могут исполняться одновременно. Каждая такая ветвь представляет собой часть общей задачи обработки информации о некотором множестве объектов (либо, возможно, разные ветви задачи имеют дело с обработкой информации о разных множествах объектов, либо обла-

дают естественным параллелизмом в более общей форме). Организовав одновременное исполнение нескольких независимых ветвей задачи, мы ничего не потеряем в возможности использовать при исполнении каждой ветви параллелизм множества объектов (либо естественный параллелизм вообще) и можем получить таким образом дальнейшее увеличение пользовательского быстродействия системы. Мы считаем также, что задача обладает параллелизмом смежных операций. Иначе говоря, в программе обработки информации о каждом из объектов на всех ее этапах вслед за любой инструкцией с достаточной вероятностью имеется еще несколько инструкций, которые можно выполнить одновременно с данной. Ясно, что использование этой возможности для повышения пользовательской производительности системы не зависит от того, выполняется ли одновременно несколько независимых ветвей программы и идет ли при исполнении каждой ветви обработка информации только об одном или одновременно о нескольких однородных объектах.

2.3.2. МНОГОПРОЦЕССОРНАЯ КОМБИНИРОВАННАЯ СИСТЕМА

1°. На рис. 2.3.1 показано одно из возможных построений комбинированной вычислительной системы, реализующей идею п. 2.3.1, 2° [11].

Основу системы составляют линии арифметико-логических процессоров (А), объединенных под общим управлением (У) и, более того, присоединенных к общему выходу устройства управления. Фактически каждая такая линия представляет собой многопроцессорную систему типа IV (ср. с рис. 2.1.4), ориентированную на использование естественного параллелизма или, в частности, параллелизма множества объектов; поэтому количество процессоров в каждой линии мы обозначили через n_4. Все процессоры одной линии в каждый момент времени выполняют одну и ту же операцию, но над разными данными (скажем, относящимися к разным объектам).

Каждое устройство управления на рис. 2.3.1 имеет, однако, несколько выходов, и к каждому из них присоединена своя линия процессоров. В этой части рассматриваемое построение идентично построению много-

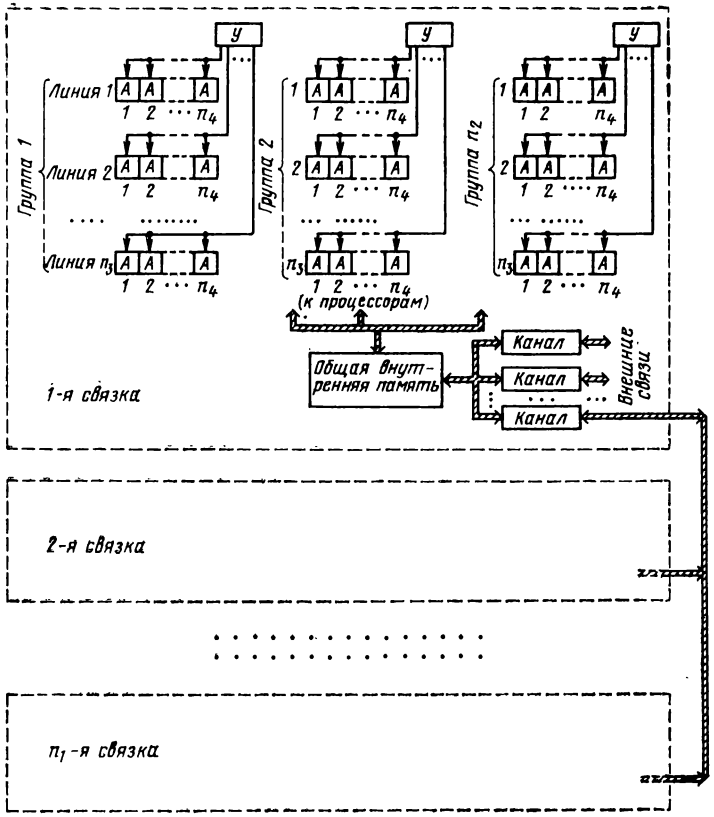


Рис. 2.3.1

процессорной системы типа III (ср. с рис. 2.1.3), но к раздельным выходам устройства управления присоединены не одиночные процессоры, а линии процессоров; соответственно количество линий, присоединенных к одному устройству управления, обозначено n_3. Присоединение к разным выходам устройства управления n_3 линий процессоров ориентировано на использование параллелизма смежных операций. Разным линиям процессоров устройство управления выдает либо последовательные инструкции из записанной обычным образом однолинейной программы (если в результате анализа, выполняемого устройством управления, выясняется, что

эти инструкции могут быть выполнены одновременно), либо различные компоненты вектора-инструкции (если выявление параллелизма смежных операций проведено на этапе программирования или трансляции программы на машинный язык системы) — ср. с п. 2.1.2.

Далее n_2 групп линий собраны над общей внутренней памятью по типу многопроцессорной системы типа II. Сделано это в точности по аналогии с многопроцессорной системой типа II, показанной на рис. 2.1.2 (с. 42) *).

Несколько (n_1) таких объединений, которые на рис. 2.3.1 названы связками, соединены между собой так же, как соединяются отдельные машины в многомашинный комплекс (вычислительную систему типа I). На рисунке показано, что эти соединения выполнены по схеме канал — канал, т. е. так же, как на рис. 2.1.1,*а*, но их можно было бы выполнить и через общую внешнюю память — как на рис. 2.1.1,*б*.

Фактически и объединение n_2 групп в одну связку, и объединение n_1 связок в один комплекс ориентированы на один и тот же вид параллелизма — параллелизм независимых ветвей. Можно было бы не устраивать объединения связок, а вместо этого предусмотреть внутри одной (единственной) связки $n_2 n_1$ групп, что дало бы, возможно, несколько более высокую пользовательскую эффективность (п. 2.2.2, 2°). Однако наличие в составе комплекса полностью функционально-законченных частей, каковыми являются связки, создает определенные удобства в организации технического обслуживания вычислительной системы (в частности, проведения профилактических или ремонтных работ на отдельных связках без вывода из строя всей системы), а также в организации режима разделения оборудования [11] — когда необходимо получить не только максимальную пользовательскую производительность для особо крупных задач, но и удовлетворительную системную производительность для потока задач со случайными характеристиками. Функционально-законченная связка может являться удобным дискретом наращивания вычислительной системы.

---

*) На рис. 2.3.1 для простоты не показаны индивидуальные ЗУ, которые могут быть у отдельных процессоров, у линий процессоров и у групп линий.

Построение рис. 2.3.1 послужило в свое время прообразом структуры вычислительной машины М-10 [16].

2°. Ясно, что для системы, изображенной на рис. 2.3.1, *пользовательская эффективность* равна

$$E_{\text{п}}^{(\text{к})}(n_1 \cdot n_2 \cdot n_3 \cdot n_4) \approx E_{\text{п}}^{(1,2)}(n_1 \cdot n_2)\, E_{\text{п}}^{(3)}(n_3)\, E_{\text{п}}^{(4)}(n_4),$$

где верхний индекс (к) означает, что речь идет об эффективности комбинированной системы, а верхние индексы (1, 2), (3), (4) относятся соответственно к вычислительным системам типа I или II, типа III и типа IV. Произведение $n_1n_2n_3n_4$ является общим количеством процессоров в системе. Нетрудно убедиться, что $E_{\text{п}}^{(\text{к})}(n_1n_2n_3n_4) > E_{\text{п}}^{(1,2)}(n_1n_2n_3n_4)$, $E_{\text{п}}^{(\text{к})}(n_1n_2n_3n_4) > E_{\text{п}}^{(3)}(n_1n_2n_3n_4)$ и в общем случае, при некотором фиксированном и не очень большом r (по сравнению с общим количеством процессоров), $E_{\text{п}}^{(\text{к})}(n_1n_2n_3n_4) > E_{\text{п}}^{(4)}(n_1n_2n_3n_4)$. Реальное пользовательское быстродействие такой системы равно

$$S_{\text{р(п)}} = E_{\text{п}}^{(\text{к})}(n_1n_2n_3n_4)\, n_1n_2n_3n_4 S$$

(где S — быстродействие одиночного процессора) — больше, чем могло бы быть максимальное пользовательское быстродействие системы, построенной по одному какому-нибудь типу и ориентированной на одну какую-либо разновидность параллелизма.

В приведенном выше выражении для $E_{\text{п}}^{(\text{к})}$ мы поставили знак приближенного равенства, потому что эффективность соединения n_2 групп линий процессоров в одну связку и эффективность соединения n_1 связок в один комплекс несколько меньше, чем эффективность объединения в многомашинный комплекс n_1 обычных машин или объединение n_2 отдельных процессоров в многопроцессорную систему типа II. Дело в том, что при определении пользовательской эффективности вычислительных систем типа I и II определенную роль играют соотношения между длинами независимых ветвей и служебными потерями на обмен информацией, выработку и проверку условий запуска отдельных ветвей и т. п. (см., например, выражение для $E_{\text{п}}$ на с. 68 в п. 2.2.2, 1°). Когда по образцу многопроцессорной системы типа II объединяются вместо отдельных процессоров группы линий про-

цессоров или по образцу вычислительной системы типа I объединяются вместо обычных машин мощные многопроцессорные подсистемы-связки, то длины независимых ветвей при прочих равных условиях становятся меньше, а относительный вес служебных потерь времени — больше.

2.3.3. КОНВЕЙЕРНЫЕ И СМЕШАННЫЕ КОМБИНИРОВАННЫЕ СИСТЕМЫ

1°. Возможные построения комбинированных вычислительных систем весьма разнообразны. В отдельных случаях эффективность использования тех или иных видов параллелизма может оказаться более высокой, чем в обычных системах — в отличие от случая, рассмотренного в п. 2.3.2, 2°, где эффективность использования параллелизма независимых ветвей в комбинированной системе была несколько ниже, чем в обычной вычислительной системе.

Остановимся сначала на фрагменте структуры комбинированной системы, показанном на рис. 2.3.2. Конвейерная n-ступенная система, изображенная на этом рисунке, ориентирована на использование одновременно и естественного параллелизма, и параллелизма смежных операций (ср. с рис. 2.1.7 на с. 59 и 2.1.8).

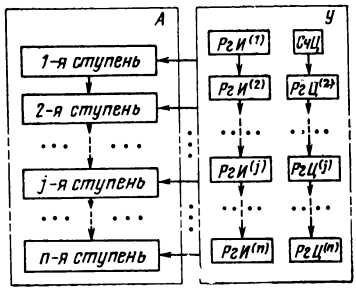


Рис. 2.3.2

Так же, как в конвейерной системе типа IV, в составе устройства управления (У) имеется счетчик циклов (СчЦ), который сохраняет в регистре инструкции (в данном случае—в РгИ$^{(1)}$) одну и ту же инструкцию в течение r тактов, где r — ранг естественного параллелизма. Однако регистр инструкции здесь не один, а n — по количеству ступеней конвейера: РгИ$^{(1)}$, РгИ$^{(2)}$, ..., РгИ$^{(n)}$ — как в конвейерной системе типа III. Если счетчик циклов в течение r тактов удерживает в РгИ$^{(1)}$ одну и ту же инструкцию, то во 2-м из них эта инструкция передается в РгИ$^{(2)}$, в 3-м — в РгИ$^{(3)}$ и т. д.; при $r \geqslant n$ в конце концов оказывается, что во всех регистрах

инструкции находится одна и та же инструкция и все ступени конвейера заняты выполнением одной и той же операции, но на разных этапах этой операции и над разными данными. Рассматриваемая система при этом работает так же, как конвейерная система типа IV. Поскольку на отдельных этапах исполнения операции содержимое счетчика цикла может использоваться для

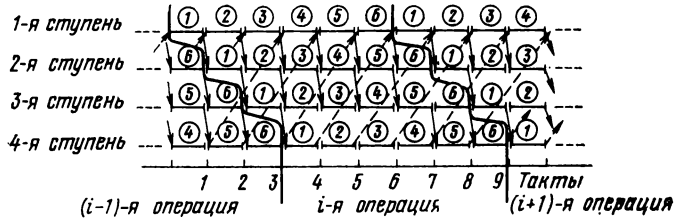


Рис. 2.3.3

модификации адреса обращения к памяти, то во всех ступенях устройства управления предусмотрены свои регистры РгЦ, в которых хранятся предыдущие состояния счетчика циклов *).

Если бы мы имели дело с обычной конвейерной системой типа IV, то выполнение каждой операции занимало бы $r+n-1$ тактов, из которых каждая ступень занята полезными операциями в течение r тактов и простаивает $n-1$ тактов. В рассматриваемой системе дело обстоит иначе.

Через r тактов от начала некоторой (i-й) операции — сразу же после окончания исполнения ее в 1-й ступени конвейера — устройство управления запускает, если возможно, следующую (($i+1$)-ю) операцию, используя параллелизм смежных операций.

При этом в 1-м такте исполнения ($i+1$)-й инструкции эта инструкция попадает в РгИ(1) и одновременно обнуляется и начинает счет от начала счетчик циклов.

*) В обычной конвейерной системе типа IV (см. рис. 2.1.8) такие регистры не были нужны, поскольку счетчик циклов никогда не сбрасывался до окончания исполняемой операции над последними данными в последней ступени конвейера. Здесь же, как мы увидим из дальнейшего, такие ситуации возможны.

В этом такте r последующих регистров инструкции РгИ (если $r<n$) хранят еще i-ю инструкцию, следующие за ними — $(i-1)$-ю, и т. д.; если $r\geqslant n$, то все последующие РгИ, начиная от $\text{РгИ}^{(2)}$ до $\text{РгИ}^{(n)}$, в 1-м такте исполнения $(i+1)$-й инструкции хранят еще i-ю инструкцию. Соответственно регистры РгЦ хранят предыдущие состояния счетчика циклов. В следующем такте $(i+1)$-я инструкция передвигается также в $\text{РгИ}^{(2)}$, а предыдущие инструкции сдвигаются ниже по цепочке РгИ, и т. д.

2°. Легко видеть, что эффективность использования параллелизма смежных операций в рассматриваемой системе намного выше, чем в обычной вычислительной системе типа III.

Действительно. Рассмотрим временную диаграмму работы системы, приведенную на рис. 2.3.3. Как и временная диаграмма для конвейерной системы типа IV на рис. 2.2.6 (с. 95), она построена в предположении $r\geqslant n$ (на рисунке $r=6$, $n=4$). Видно, что в том такте, когда 1-я ступень конвейера освобождается для начала исполнения $(i+1)$-й операции (в первую очередь — над данными, относящимися к 1-му объекту из множества, или к 1-й компоненте r-мерных векторов, и т. п.), то i-я операция над данными, относящимися к 1-му объекту или к 1-й компоненте r-мерного вектора и т. п., уже завершена. К началу 2-го такта исполнения $(i+1)$-й операции при $r\geqslant n$ наверняка завершена уже i-я операция над данными, относящимися ко 2-му объекту из обрабатываемого множества объектов, и т. д.

Ясно, что простои конвейера здесь возможны только в том случае, если i-я операция является интегральной (т. е. ее результат зависит от данных по всем объектам обрабатываемого множества или от всех компонент r-мерного вектора и т. п.) и при этом результат i-й операции сразу же используется в $(i+1)$-й операции. Что касается всех предыдущих операций ($(i-1)$-й, $(i-2)$-й и т. д.), то они (при $r\geqslant n$) к началу $(i+1)$-й операции завершены полностью. Вероятность использования результата i-й операции в $(i+1)$-й мы назвали в свое время показателем связанности операций α. Обозначив дополнительно вероятность наличия в программе интегральных операций через $p_{\text{и}}$ (разумеется, $p_{\text{и}}<1$, а для многих задач $p_{\text{и}}\ll 1$), найдем, что вероятность простоя конвейера в построении рис. 2.3.2 равна $p_{\text{и}}\alpha$. Длительность простоя в этих случаях равна, очевидно, $n-1$ так-

там (до завершения i-й операции). Поэтому при $r \geqslant n$ получим:

$$E_{\text{п}}^{(3-4)}(n) = \frac{r}{r + p_{\text{и}}\alpha(n-1)},$$

где символом $E_{\text{п}}^{(3-4)}$ обозначена пользовательская эффективность системы, сочетающей принципы построения конвейерных систем типа III и IV. Легко видеть, что $E_{\text{п}}^{(3-4)}(n) > E_{\text{п}}^{(4)}(n)$ и $E_{\text{п}}^{(3-4)}(n) > E_{\text{п}}^{(3)}(n)$. Поскольку, как говорилось выше, $\alpha \approx 0{,}1-0{,}5$, то произведение $p_{\text{и}}\alpha \ll 1$, а эффективность рассматриваемого построения получается весьма высокой.

3°. Мы говорили о построении рис. 2.3.2 как о фрагменте структуры вычислительной системы потому, что на самом деле это построение может входить в качестве составной части в вычислительные системы самых разнообразных конфигураций.

Представим себе, например, что количество ступеней конвейера n велико, и для тех задач, которые предполагается решать с помощью системы, нельзя рассчитывать всегда на выполнение условия $r \geqslant n$. При $r < n$ к началу $(i+1)$-й операции еще не будет выполнена i-я операция над данными, относящимися к 1-му объекту из обрабатываемого множества (к 1-м компонентам обрабатываемых векторов и т. п.) и не будет полностью закончено выполнение $(i-1)$-й операции. При $r < n/2$ к началу $(i+1)$-й операции не будут завершены i-, $(i-1)$- и $(i-2)$-я операции, и т. д. Разумеется, это приведет к снижению пользовательской эффективности.

В этой ситуации выгодно предусмотреть, как это показано на рис. 2.3.4, несколько устройств управления, включенных через кольцевой коммутатор так, как это делается в конвейерных системах типа II (ср. с рис. 2.1.6 на с. 59). Количество устройств управления n_2 здесь меньше количества ступеней n и является целым делителем n *). Система рассчитана на выполнение одновременно n_2 независимых ветвей.

---

*) Последнее условие мы вводим только для простоты дальнейшего изложения. Фактически можно любое число ступеней n мысленно дополнить до ближайшего кратного n_2 несколькими фиктивными ступенями, после чего все излагаемое ниже остается в силе.

Каждое из устройств управления имеет свой узел формирования адреса очередной инструкции и первый регистр инструкции РгИ, свой счетчик циклов СчЦ (для реализации в выполняемой им ветви естественного параллелизма) и свои цепи, анализирующие возможность начать очередную операцию программы до полного окончания предыдущей операции (для реализации параллелизма смежных операций в выполняемой им ветви программы). Кроме того, в состав каждого устройства управления входят, может быть, автономные индексные и базовые регистры, цепи защиты памяти и т. п.

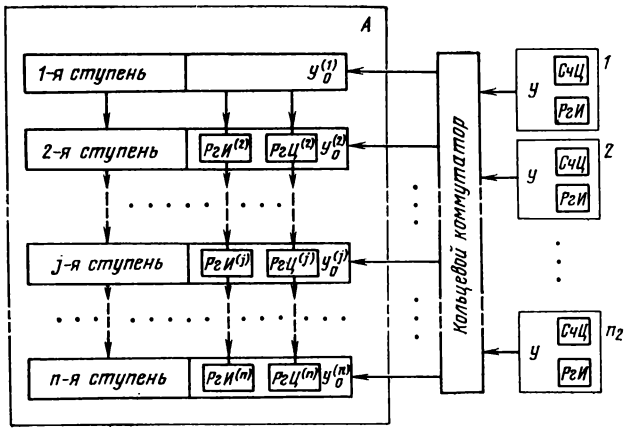


Рис. 2.3.4

Когда некоторое устройство управления через кольцевой коммутатор подсоединяется к 1-й ступени процессора, в общие управляющие цепи 1-й ступени $У_0^{(1)}$ передаются содержимое РгИ и СчЦ данного устройства управления, причем СчЦ увеличивается на единицу (это происходит раз за n/n_2 тактов). Однако во всех последующих ступенях процессора к общим управляющим цепям отнесены также регистры $РгИ^{(j)}$ и $РгЦ^{(j)}$, где $j=2, 3, \ldots, n$; в конце каждого такта работы системы все $РгИ^{(j)}$ принимают содержимое $РгИ^{(j-1)}$, а все $РгЦ^{(j)}$ — состояния $РгЦ^{(j-1)}$. В результате оказывается, что в том такте, когда 1-е устройство управления под-

ключается к 1-й ступени управления, его инструкции находятся также в РгИ$^{(n_2+1)}$, РгИ$^{(2n_2+1)}$, ...

Это могут быть такие же инструкции, как передаваемая в 1-ю ступень (поскольку используется естественный параллелизм), а могут быть и предыдущие инструкции, проходившие через данное устройство управления (поскольку оно использует параллелизм смежных операций). В соответствующих регистрах РгЦ находятся предыдущие состояния счетчика циклов 1-го устройства управления.

Кольцевой коммутатор устроен так, что в каждом такте любое из n_2 устройств управления подключается одновременно к n/n_2 ступеням конвейера. Скажем, если 1-е устройство управления подключено к 1-й ступени, то в этом же такте оно подключено к (n_2+1)-, $(2n_2+1)$-, $(3n_2+1)$-й, ... ступеням (как раз к тем, где в РгИ и РгЦ хранятся соответственно предыдущие инструкции и состояния счетчика циклов 1-го устройства управления). В следующем такте кольцевой коммутатор подключает 1-е устройство управления ко 2-, (n_2+2)-, $(2n_2+2)$-, $(3n_2+2)$-й, ... ступеням, а 2-е устройство управления подключается к 1-, (n_2+1)-, $(2n_2+1)$-, $(3n_2+1)$-й, ... ступеням. Затем 1-е устройство управления подключается к 3-, (n_2+3)-, $(2n_2+3)$-, $(3n_2+3)$-й, ... ступеням, 2-е устройство управления — ко 2-, (n_2+2)-, $(2n_2+2)$-, $(3n_2+2)$-й, ... ступеням, а 3-е устройство управления — к 1-, (n_2+1)-, $(2n_2+1)$-, $(3n_2+1)$-й, ... ступеням, и т. д.

В итоге получается, что система рис. 2.3.4 представляет собой как бы n_2 вложенных друг в друга и объединенных по принципу конвейерной системы типа II построений рис. 2.3.2, каждое из n/n_2 ступеней. Фактически комбинированная вычислительная система рис. 2.3.4 является конвейерным эквивалентом многопроцессорной связки рис. 2.3.1, которая также использует все три вида параллелизма.

Пользовательская эффективность такой системы равна

$$E_{\text{п}}^{(\text{к})}(n) = E_{\text{п}}^{(1,\,2)}(n_2)\, E_{\text{п}}^{(3-4)}(n/n_2).$$

4°. В п. 3° мы рассматривали случай, когда ранг естественного параллелизма r недостаточно велик по сравнению с количеством ступеней конвейера n. Может быть и обратное положение, когда для построения

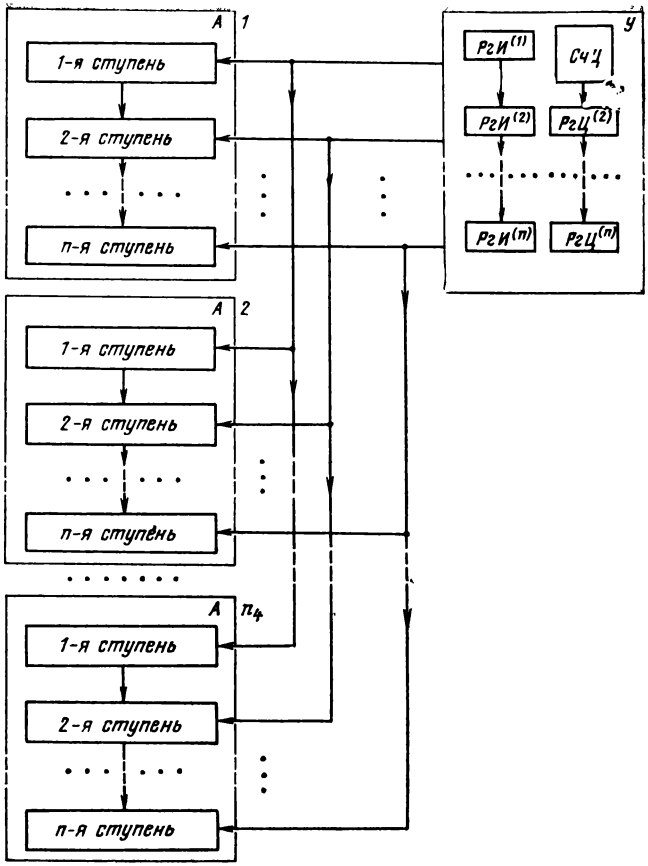


Рис. 2.3.5

рис. 2.3.2 $r \gg n$ и возможности естественного параллелизма для повышения суммарного пользовательского быстродействия системы используются недостаточно.

Если это так, то к одному устройству управления У, имеющемуся на рис. 2.3.2, можно присоединить несколько (n_4) одинаковых арифметико-логических процессоров конвейерного типа (А) — тоже таких же, как на рис. 2.3.2. Соответствующее построение показано на рис. 2.3.5. Главное отличие в работе устройства управления в составе этого построения от работы аналогичного устройства, изображенного на рис. 2.3.2, состоит

в том, что при использовании естественного параллелизма счетчик циклов СчЦ в каждом такте добавляет вместо единицы величину n_4 — потому, что компоненты многомерных векторов или данные по разным объектам из множества и т. п. берутся в обработку не по одному, а группами по n_4 последовательных компонентов, объектов и т. п. Если r не кратно n_4, то в последнем такте операции может потребоваться, возможно, маскирование части арифметико-логических процессоров.

Рис. 2.3.5 представляет фактически один из множества возможных вариантов комбинированных вычислительных систем, построенных по *смешанному* принципу, т. е. сочетающих — с целью достижения максимальной производительности — как конвейерную организацию, так и возможности объединения в одну систему нескольких процессоров. Одновременно с этим в смешанных системах возможно и использование принципа объединения в одну систему нескольких независимых машин либо более сложных образований — подобно тому, как говорилось в 2.3.1 применительно к многопроцессорным комбинированным системам («связки» на рис. 2.3.1, с. 102).

Например, несколько построений рис. 2.3.2, 2.3.5 или даже рис. 2.3.4 (если количество устройств управления n_2 в этом построении не слишком велико) может быть объединено по типу многопроцессорной системы типа II, а несколько таких объединений — по типу многомашинного комплекса (вычислительной системы типа I). Ясно, что такие вычислительные системы ориентированы на использование всех видов параллелизма, которыми может обладать пользовательская задача.

Рассмотреть все возможные построения смешанных систем не представляется возможным — слишком велико количество различных вариантов.

2.4. ДОПОЛНИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

2.4.1. О ПРОГРАММИРОВАНИИ ДЛЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

1°. Наиболее распространенные из современных языков программирования создавались в период, когда основные вычислительные средства, выпускаемые промышленностью, представляли собой обычные однопроцессорные машины; собственно, такое положение в зна-

чительной мере сохраняется и сейчас. Хотя создатели, например, языка АЛГОЛ-60 специально декларировали независимость языка от конкретных свойств той или иной вычислительной машины, все же при разработке языка они имели в виду, разумеется, обычную однопроцессорную машину и соответственно последовательностные алгоритмы решения различных задач.

Ясно, что эти языки не могут оказаться подходящими для описания параллельных вычислительных процессов. Наиболее радикальным решением при переходе к использованию вычислительных систем было бы создание принципиально новых языков программирования. Имеются, однако, некоторые препятствия к тому, чтобы немедленно идти по этому пути.

Первое из них состоит в наличии в обращении большого количества программ, написанных на современных языках. Если бы с порога было объявлено, что все эти программы не подходят для вычислительных систем, то это привело бы, прежде всего, к существенному сокращению возможных областей применения вычислительных систем и как следствие — к замедлению развития собственно техники вычислительных систем. Положение усугубляется факторами субъективного и психологического характера, поскольку вместе со значительным фондом программ, написанных на современных языках, имеется и значительное количество пользователей и профессиональных программистов, изучивших и имеющих опыт работы с современными языками программирования.

Другое препятствие состоит в том, что разные вычислительные системы используют разные свойства задач для организации параллельных вычислений. Однако язык высокого уровня должен быть в достаточной мере независимым от конкретных свойств вычислительной системы. Как частный случай весьма желательно, чтобы новый язык программирования, приспособленный специально для описания параллельных вычислений, мог транслироваться и на обычную вычислительную машину и далее интерпретироваться в виде обычного последовательностного вычислительного процесса. Поэтому переход от современных языков программирования к языкам, специально приспособленным для описания параллельных вычислительных процессов, можно представить себе в виде ряда последовательных этапов:

1) введение в трансляторы с современных языков специальных оптимизирующих блоков, выявляющих те свойства программы, которые могут быть использованы для организации параллельных вычислений, и соответствующим образом отражающих эти свойства в программе на машинном языке для конкретной вычислительной системы;

2) знание программистом устройства оптимизирующих блоков и использование этих знаний при составлении программы — с тем, чтобы после трансляции получить наилучшую программу для системы;

3) введение в состав современных языков дополнительных средств для описания параллельных вычислительных процессов; при этом транслятор для вычислительной системы должен получать специальное указание, использованы ли в данной конкретной программе дополнительные средства или транслятор должен организовать по возможности параллельные вычисления с помощью собственных оптимизирующих блоков, а транслятор для обычной машины или для системы, не использующей соответствующий вид параллелизма, должен уметь игнорировать дополнительные описания либо формировать записанную с их использованием программу в виде последовательностной программы.

В соответствии с этими этапами мы будем в дальнейшем рассматривать применительно к различным видам параллелизма следующие аспекты проблемы:

— на какие особенности конкретной программы могли бы опереться оптимизирующие блоки транслятора с целью организации параллельных вычислений по программе, составленной на одном из современных (последовательностных) языков;

— каким образом программист, работающий на современных языках программирования, может улучшить результаты работы транслятора, если ему заранее известно, на какой вид параллелизма ориентирована вычислительная система и как устроены оптимизирующие блоки транслятора;

— какие дополнения целесообразно ввести в языки для описания параллельных вычислений, опирающихся на определенные виды параллелизма.

Последнее имеет отношение, возможно, и к вопросу разработки принципиально новых языков.

Еще один аспект проблемы состоит в том, как построить трансляторы для вычислительных систем, чтобы в самом процессе трансляции наилучшим образом использовались возможности системы по организации параллельных операций. Но этого вопроса мы ниже не касаемся.

2°. *Использование параллелизма смежных операций* представляет собой, пожалуй, единственный случай применения вычислительной системы, когда практически не требуется никаких изменений или дополнений в современных языках программирования. Как говорилось выше, организация параллельных вычислений с использованием этого вида параллелизма возлагается на блоки локальной оптимизации в составе транслятора либо даже выполняется аппаратно, в ходе интерпретации программы на машинном языке.

Однако при составлении программы — даже на языке высокого уровня — программисту иногда полезно знать свойства системы, которую он предполагает использовать для решения задачи (количество параллельных процессоров или ступеней конвейера, ориентированных на параллелизм смежных операций), и свойства оптимизирующих блоков ее транслятора или соответствующей аппаратуры.

Предположим, например, что система позволяет выполнять параллельно до 4 независимых операций, а блоки локальной оптимизации или аппаратура просматривают программы на глубину до 8 последовательных операций.

Рассмотрим далее фрагмент программы, записанной, для примера, на АЛГОЛ-60.

```
.        .
I:=I+1;
X[I]:=((X[I]+CONS1)/8)↑I;
SUMX:=SUMX+X[I];
J:=J+1;
Y[J]:=((Y[J]+CONS2)/2)↑J;
SUMY:=SUMY+Y[J];
K:=K+1;
Z[K]:=((Z[K]—CONS3)×2)↑K;
SUMZ:=SUMZ+Z[K];
L:=L+1;
U[L]:=((U[L]—CONS4)×8)↑L;
SUMU:=SUMU+U[L];
. . . . . .
```

При указанных выше параметрах системы и блоков локальной оптимизации, как нетрудно видеть, такая программа загрузит систему далеко не оптимальным образом. В частности, на первом шаге просматривается возможность выполнения восьми операций:

1) I+1;
2) X[I]+CONS1;
3) (X[I]+CONS1)/8;
4) ((X[I]+CONS1)/8)↑I;
5) SUMX+X[I];
6) J+1;
7) Y[J]+CONS2;
8) (Y[J]+CONS2)/2.

Из них могут быть выполнены одновременно (на первом шаге) только две — 1- и 6-я; для выполнения любой из остальных операций нужен результат предыдущей.

На втором шаге просмотр дойдет, очевидно, до операции SUMY+Y[J] (ранее выполненные операции не учитываются), а выполнять можно будет опять только две операции: X[I]+CONS1 и Y[J]+CONS2. На третьем шаге просмотр дойдет до операции Z[K]—CONS3, а выполнять можно будет три операции: (X[I]+ +CONS1)/8, (Y[J]+CONS2)/2 и K+1, и т. д.

Между тем очевидное изменение записи программы на исходном языке позволило бы получить полную загрузку системы (возможность одновременного выполнения 4 операций) на каждом шаге:

. . . . . ;
begin I:=I+1; J:=J+1; K:=K+1; L:=L+1 **end**
begin X[I]:=X[I]+CONS1; Y[J]:=Y[J]+CONS2;
Z[K]:=Z[K]—CONS3; U[L]:=U[L]—CONS4 **end**
begin X[I]:=X[I]/8; Y[J]:=Y[J]/2;
Z[K]:=Z[K]×2; U[L]:=U[L]×8 **end**
begin X[I]:=X[I]↑I; Y[J]:=Y[J]↑J;
Z[K]:=Z[K]↑K; U[L]:=U[L]↑L **end**
begin SUMX:=SUMX+X[I]; SUMY:=SUMY+Y[J];
SUMZ:=SUMZ+Z[K]; SUMU:=SUMU+U[L] **end**.

Дополнительные операторные скобки (**begin—end**) использованы здесь для того, чтобы выделить группы операций, выполняемых одновременно. В них заключено ровно по 4 операции; однако и при $n<4$ полезная загрузка системы при работе по указанной программе была бы полной.

Конечно, к рассмотренному примеру следует относиться как к чисто иллюстративному. Введение дополнительных операторных скобок для операторов, которые могут выполняться одновременно (неважно, сколько их — чем больше, тем лучше), разумеется, никак не скажется на работе транслятора для обычной вычислитель-

ной машины или для системы другой конфигурации; но и для системы, рассмотренной в примере конфигурации, не очень большое усложнение транслятора делает эти скобки излишними. Что касается сделанного в примере преобразования программы по существу (изменения порядка операций), то и его требовать от программиста в обычных условиях трудно, да и нет необходимости. Дело в том, что обычно глубина просмотра программы в процессе ее оптимизации в 3—4 раза превышает n — количество инструкций, которые могут выполняться одновременно (а не в 2 раза, как в рассмотренном примере). При этом пользовательская эффективность системы и так получается достаточно хорошей. Для программиста же просмотр программы на $(3—4)n$ операций вперед представляет собой слишком трудоемкую работу.

3°. *Использование естественного параллелизма* или в частном случае параллелизма множества объектов в ситуации, когда программирование ведется на одном из современных языков, требует исследования в процессе трансляции программных циклов и, если возможно, организации параллельного выполнения тех вычислений, которые предполагалось вести циклически.

Удобнее всего параллельное выполнение циклов организовать в том случае, когда список цикла задан в виде (обозначения АЛГОЛ-60):

for I := <начало> **step** 1 **until** <конец> **do**,

где через I обозначена некоторая целочисленная переменная, используемая как индекс. Приращение, равное 1, нам потребовалось для того, чтобы переменные с индексами, выбираемые для выполнения последовательных циклов, располагались в памяти по последовательным адресам. Однако, если бы в цикле переменные имели в качестве индексов выражения типа 2I+1, 5I—4, I↑2 и т. п., то эта закономерность была бы нарушена.

Для того, чтобы последовательные циклы можно было выполнять параллельно, должен быть выдержан ряд ограничений.

Например, в цикле вообще могут использоваться разные переменные с индексом I либо зависящим от I — например, X[I], Y[I—1] и т. д. Однако, если одна и та же переменная встречается внутри цикла с разными индексными выражениями, зависящими от I, например, Y[I] и Y[I—1], то это допускается только при усло-

вии, что в цикле вообще нет оператора присваивания этой переменной либо что оператор присваивания расположен после операторов, в которых эта переменная используется с разными индексными выражениями. Иначе последовательные циклы оказались бы зависящими друг от друга. При этом в случае, когда переменная с индексом, зависящим от I, впервые встречается в левой части оператора присваивания (и далее может встречаться в программе цикла обязательно с тем же индексным выражением), важно, чтобы значение этого индексного выражения было связано взаимно-однозначной зависимостью со значением I.

В программе цикла могут встречаться также переменные без индексов или переменные с индексами, не зависящими от I (т. е. не меняющимися от цикла к циклу). Для того чтобы циклы были независимы друг от друга, должно соблюдаться следующее правило:

переменная без индекса (с индексом, не зависящим от I) в программе цикла либо должна встречаться сначала в левой части оператора присваивания (причем присваиваемое ей значение не должно зависеть от ее предыдущего значения), а дальше может встречаться уже в любых контекстах, либо эта переменная вообще не должна встречаться в левой части операторов присваивания.

В первом случае мы имеем дело как бы с рабочими ячейками, которые при параллельном выполнении циклов могут быть назначены разными для разных циклов. Во втором случае речь идет как бы о константах, которые при исполнении программы цикла используются во всех циклах одинаково и могут прочитываться из одной и той же ячейки для любого цикла (например, если речь идет о многопроцессорной системе типа IV, то через аппарат размножения операндов — см. 2.1.3, 2°).

Исключение может быть сделано для операторов, описывающих интегральные операции, которые интерпретируются общими арифметико-логическими цепями, если таковые имеются в составе вычислительной системы (см. п. 2.1.3, 6°). Например, если общие арифметико-логические цепи позволяют вычислить в качестве одной операции системы скалярное произведение векторов, то в программе цикла допустим оператор, скажем, следующего вида:

$$\mathrm{PROD} := \mathrm{PROD} + \mathrm{X[I]} \times \mathrm{Y[I]};,$$

где одна и та же переменная без индекса (PROD) встречается и в левой, и в правой части оператора присваивания. Точно так же, если общие арифметико-логические цепи позволяют интерпретировать в качестве одной операции отыскание минимального элемента в массиве, то в программе цикла допустим оператор, скажем, вида

if Z[I] < MINZ **then** MINZ := Z[I];,

где переменная без индекса (MINZ) и используется при вычислении отношения и встречается в левой части оператора присваивания.

Имеется еще несколько столь же очевидных, правил, которые должны учитываться оптимизирующими блоками транслятора, организующими параллельное выполнение системой циклических вычислений.

В общем правила, по которым транслятор с одного из современных языков программирования разворачивает циклические вычисления в параллельное выполнение циклов для вычислительной системы типа IV, сравнительно просты. Программист тоже может без труда понять и запомнить эти правила — с тем, чтобы при составлении программы «помочь» транслятору организовать параллельные вычисления.

При этом иногда желательно, чтобы программист учитывал еще некоторые дополнительные обстоятельства.

Например, программу сложения матриц

$$\|Z\| = \|X\| + \|Y\|$$

можно записать двояко (в АЛГОЛ-60):

```
for K:=1 step 1 until LASTK do
    for I:=1 step 1 until LASTI do
        Z[I, K]:=X[I, K]+Y[I, K];
```

либо

```
for I:=1 step 1 until LASTI do
for K:=1 step 1 until LASTK do
Z[I, K]:=X[I, K]+Y[I, K];.
```

В первом случае информацию в памяти системы типа IV желательно разместить в порядке возрастания I, а ранг задачи будет равен размерности матриц по X (т. е. величине LASTI), в другом же случае информацию желательно разместить в порядке возрастания K, а ранг задачи будет равен размерности матриц по Y (т. е. LASTK).

Поскольку пользовательская эффективность вычислительной системы типа IV в общем случае тем больше, чем больше ранг задачи, а транслятор разворачивает в виде параллельных вычислений, как правило, внутренний цикл программы, то при наличии нескольких вложенных циклов желательно, чтобы внутренним был цикл,

повторяющийся наибольшее количество раз. Более сложный транслятор мог бы учитывать это обстоятельство автоматически.

С более общей точки зрения, однако, ясно, что когда программист расписывает в виде выполняемых последовательно циклов вычисления, которые заведомо можно выполнить параллельно (и, возможно, соблюдает определенные правила, чтобы транслятор сумел развернуть их в параллельные вычисления), а затем оптимизирующие блоки транслятора превращают эту запись в программу параллельных вычислений, то по существу дважды делается ненужная работа. Значительно выгоднее было бы дополнить исходный язык средствами для описания параллельных вычислений.

Например, в АЛГОЛЕ-60 уже имеются средства для описания массивов. Массивы данных появляются в программах во многих случаях тогда, когда программа обладает параллелизмом множества объектов. Остается только определить операции над массивами. Скажем, если считать, что сложение массивов одинаковых размерностей и с одинаковыми границами состоит в сложении элементов этих массивов с одинаковыми индексами, то программу сложения двух матриц

$$\|Z\| = \|X\| + \|Y\|$$

можно было бы записать в виде

begin array X, Y, Z [1 : LASTI, 1 : LASTK];
Z:=X+Y
end,

где LASTI и LASTK — переменные, являющиеся для данного блока глобальными, либо фиксированные величины (числа).

Наряду с обычными операциями над массивами желательно определить также операции выработки масок и операции под масками (см. 2.1.3, 3°). Например, если MASK и COND — булевы массивы, X и Y — действительные массивы, имеют одинаковые размерности и одинаково определенные границы, то оператор

COND:=X⩾Y;

означал бы, что каждому элементу массива COND присваивается значение **true**, если соответствующий элемент массива X больше или равен соответствующему элементу массива Y, и значение **false** в противном случае. Оператор

MASK:=MASK∧COND;

означал бы присваивание каждому элементу массива MASK логического произведения его предыдущего значения на соответствующий элемент массива COND. Оператор

X:=**if** MASK **then** X—Y;

означал бы, что каждому элементу массива X при условии, что значение соответствующего элемента массива MASK есть **true**, присваивается разность между предыдущим значением данного элемента массива X и значением соответствующего элемента массива Y. Те

элементы массива X, для которых соответствующие элементы массива MASK имеют значение **false**, сохраняют предыдущее значение.

Аналогично действительным, целым и булевым массивам могут быть определены такие объекты языка, как действительные, целые булевы векторы, матрицы и т. п., и операции над ними:

Например, объекты языка:

vector<идентификатор>[<границы>] (или **real vector**),
Integer vector<идентификатор>[<границы>],
Boolean vector<идентификатор>[<границы>],
matrix<идентификатор>[<границы строки, границы столбца>] (или **real matrix**),
integer matrix<идентификатор>[<границы строки, границы столбца>],
Boolean matrix<идентификатор>[<границы строки, границы столбца>]
и т. д.

Если E, H — векторы, A, B — матрицы, K — число, то дополнительными операциями могут быть, например,

E·H — скалярное произведение векторов,

E×H — векторное произведение,

E·K — умножение вектора на число (совпадает с умножением одномерного массива на число),

E+H, E—H — сложение и вычитание векторов (совпадают со сложением и вычитанием одномерных массивов),

A·E — произведение матрицы на вектор,

A·B — произведение матриц

и т. д.

Введение дополнительных объектов языка и дополнительных операций не лишает язык его универсальности, делая одновременно запись программ на этом языке более компактной, а работу программиста более простой. Если транслятор переводит программу на язык обычной (однопроцессорной) машины или на язык вычислительной системы, не использующей естественный параллелизм (системы типа I, II или III), то операции над массивами, векторами, матрицами и т. п. должны разворачиваться им в циклические вычисления. Реализовать такую возможность в трансляторе однопроцессорной машины или вычислительной системы типа I, II или III несложно.

Аналогичные во многом идеи высказывались также в [12—14].

4°. Использование параллелизма независимых ветвей в программах, написанных на одном из современных языков программирования, связано с исследованием свойств отдельных разделов программы.

В частности, если программа написана на АЛГОЛе-60, то оптимизирующие блоки транслятора, имеющие целью выявить в программе параллелизм независимых ветвей, должны были бы исследовать блоки оптимизируемой программы. При этом исследованию подлежат такие пары блоков, из которых ни один не является подблоком другого.

Два блока могут исполняться параллельно при условии, что в одном (любом) из них вообще не упоминаются те глобальные для обоих блоков переменные, которые упоминаются в левой части хотя бы одного оператора присваивания в другом блоке. Определенные тонкости связаны также с использованием процедур.

В общем правила, по которым оптимизирующие блоки транслятора могут выявить в программе параллелизм независимых ветвей и организовать на этой основе параллельные вычисления с помощью системы типа I или II, сформулировать не очень сложно. Возможно, что формулировка этих правил окажется даже проще, чем формулировка правил, по которым оптимизирующие блоки транслятора должны были бы исследовать программные циклы с целью выявления естественного параллелизма и организации параллельных вычислений с помощью системы типа IV (см. п. 3°). Однако характер работы, возлагаемой на транслятор в одном и в другом случае, существенно различен.

Исследование программных циклов, выполняемое в интересах организации работы системы типа IV, представляет собой некоторым образом локальную задачу. Хотя оператор, выполняемый в цикле, может быть сам по себе очень длинным, а исследовать его придется весь целиком, исследование одного цикла ведется независимо от исследования других циклов. Только если требуется высокая степень оптимизации программы и не предполагается требовать каких-либо специальных усилий от программиста, то транслятор должен, может быть, исследовать совместно вложенные циклы (см. с. 119).

Исследование блоков программы (в АЛГОЛе-60 или подпрограмм в ФОРТРАНе) требует попарного сопоставления всех блоков одного уровня. По сути дела необходимо производить перебор всех пар, потому что из возможности параллельного выполнения блоков A и B и блоков A и C не следует возможность параллельного выполнения блоков B и C.

Скорее всего, когда речь идет о достаточно сложной программе — такой, для решения которой как раз и необходима вычислительная система с высокой пользовательской производительностью, — такая работа вряд ли может быть выполнена транслятором с приемлемым качеством и приемлемой скоростью.

Реально приходится рассчитывать главным образом на разметку программы программистом путем введения в нее операторов типа **fork**, **joint** и **quitt**, как об этом говорилось в п. 2.2.2,2°. Аналогичные средства для выделения независимых ветвей имеются в составе языка программирования PL/1.

Заметим еще, что дополнения к современным языкам программирования, о которых говорилось выше (п. 3°) в связи с использованием естественного параллелизма в вычислительных системах типа IV, вели в общем к упрощению труда программиста. Требование же о расстановке специальных указателей для организации параллельных вычислений в вычислительной системе типа I или II, наоборот, создает излишнюю нагрузку для программиста. В целом проблема программирования для вычислительных систем типа I и II выглядит существенно более сложной, чем для систем типа IV.

2.4.2. ОБ ОСОБЕННОСТЯХ АППАРАТУРЫ

До сих пор, говоря о структурах вычислительных систем, мы оставляли в стороне те особенности отдельных устройств, которые необходимы для использования их в составе системы. Дело выглядело таким образом, будто обычные устройства — такие же, как применяются в составе однопроцессорных машин — просто объединяются в разных комбинациях в те или иные структуры вычислительных систем. Между тем некоторые требования к свойствам применяемых устройств все же нужно предъявить, хотя, может быть, и не очень значительные.

Об особенностях запоминающих устройств, применяемых в составе вычислительных систем, мы отчасти уже говорили в п. 2.1.2 и 2.1.3. Вопрос этот, так же как проблема организации связей в системе и некоторые другие, заслуживает отдельного тщательного рассмотрения (см., например, [11]).

Однако в дальнейшем главное внимание уделяется наиболее сложной части проблемы — особенностям арифметико-логических устройств, на основе которых строятся вычислительные системы.

По сути дела только вычислительные системы типа I, если они не используются в контуре управления в реальном масштабе времени (об этом см. ниже), не предъявляют никаких специальных требований к арифметико-логическим устройствам (да и вообще ко всем устройствам машин). Системы этого типа являются *асинхронными:* каждая машина работает в своем темпе, независимом от темпа других машин; это могут быть даже машины разных типов. Только в определенных точках вычислительного процесса, после окончания и

перед началом исполнения отдельных ветвей программы, машины обмениваются между собой управляющими сигналами и организуют обмен информацией между собой. Даже в эти периоды времени, как правило, не требуется синхронизации работы машин между собой. Канал одной машины в соответствии с принятым интерфейсом входит в связь с каналом другой машины или устройством, являющимся частью общего поля внешней памяти, формирует запрос на передачу или на прием очередной порции информации, получает в ответ сигнал готовности, выдает или получает очередную порцию информации со своим сигналом сопровождения, формирует новый запрос, и т. д.

Многопроцессорные вычислительные системы типа II являются обязательно системами *синхронизированными*, т. е. системами, работающими от единого тактового генератора. Одна операция, например, в арифметико-логических устройствах (и вообще в процессорах) может в принципе выполняться за разное время, в зависимости от типа операции, от операндов, даже от свойств процессора, если процессоры не все одинаковы, но это время обязательно кратно длительности такта, единого для всех процессоров. Иначе обмен информацией процессоров между собой либо между процессорами и общей памятью требовал бы каждый раз затрат нескольких дополнительных тактов на синхронизацию и существенно снижал бы общее быстродействие системы.

Также синхронизированной может быть многопроцессорная система типа III, если ее работа построена по принципу опережающего анализа инструкций и передачи очередной инструкции, для которой выполняются условия запуска, свободному процессору.

Многопроцессорные системы типа III, работающие с векторами-инструкциями, а также многопроцессорные системы типа IV и конвейерные системы всех типов являются системами *синхронными*. Однако под этим общим термином объединяются несколько разные по жесткости требования, предъявляемые к устройствам системы в разных из указанных структур.

Для многопроцессорных систем типа IV важно, чтобы любая операция выполнялась за некоторое фиксированное для данной операции время, не зависящее от операндов, участвующих в ней. Ясно, что применение, например, в арифметико-логических процессорах

такой системы методов ускорения операций, направленных на минимизацию среднего для совокупности всех возможных операндов времени выполнения операции, бессмысленно. Длительность цикла работы системы, в котором n процессоров выполняют одну и ту же операцию над разными операндами, все равно определялась бы длительностью работы того процессора, где операнды оказались наименее удачными в данной выборке из n. При этом аппаратура, затраченная на минимизацию среднего времени выполнения операции, фактически оказалась бы бесполезной.

В многопроцессорной системе типа III, работающей с вектором-инструкцией, желательно, чтобы длительность выполнения операции не зависела не только от операндов, но и от типа операции, потому что цикл работы системы и здесь подстраивается под наибольшее время работы любого из n процессоров, но теперь эти процессоры выполняют, может быть, разные операции. Возможен, конечно, вариант, в котором, например, длительность умножения равна $k_{\text{у}}T$, а длительность деления — $k_{\text{д}}T$, где T — длительность всех других операций (логических, сложения, вычитания и т. п.), а $k_{\text{у}}$ и $k_{\text{д}}$ — целые числа. При этом вслед за вектором-инструкцией, в котором для некоторого процессора задана операция умножения или деления, должно следовать $k_{\text{у}}$—1 или соответственно $k_{\text{д}}$—1 векторов-инструкций, в которых для данного процессора не предусматривалось бы никаких новых операций. Однако такое решение существенно усложнило бы процесс оптимизации программы и привело бы к усложнению устройства управления. В частности, устройство управления должно было бы уметь выбирать из памяти векторы-инструкции переменной длины, помнить, какие процессоры заняты операциями, заданными в предыдущих циклах, раздавать в зависимости от этого компоненты вектора-инструкции процессорам в разном порядке (а не всегда первую компоненту 1-му процессору, вторую — 2-му и т. д.).

Наконец, для конвейерных систем всех типов желательно, чтобы не только длительность выполнения операции не зависела от операндов и типа операции, но и чтобы длительности отдельных этапов, на которые разделены операции, были фиксированы. Возможны, конечно, построения конвейеров с переменным тактом, а также асинхронного типа [1]. Однако использование этих

возможностей связано в любом случае с непроизводительными простоями аппаратуры системы.

Таким образом, создание вычислительных систем во многих случаях требует разработки *методов выполнения операций и* соответственно *арифметико-логических устройств*, которые мы в дальнейшем будем называть *синхронными*, понимая под этим алгоритмы выполнения операций и структуры арифметико-логических устройств, *рассчитанные на фиксированные* (не зависящие от операндов) *длительности выполнения операций и на достижение минимальной длительности операции при любых операндах*. Именно этой проблеме посвящены в основном последующие разделы книги.

В заключение обратим внимание на два важных обстоятельства. Большинство простейших (неускоренных) методов выполнения арифметических и логических операций являются в своей основе синхронными методами. Появление большого числа различных асинхронных методов и структур является результатом сознательных усилий разработчиков, предпринимавшихся в течение ряда лет для снижения среднего времени выполнения операций; иногда при этом достигалось и уменьшение максимального времени выполнения арифметической операции — соответствующего наименее благоприятному сочетанию операндов — но не это считалось главным (см., например, [15]). Между тем при создании вычислительных систем мы, как правило, не можем удовлетвориться простейшими методами выполнения операций, но должны искать такие синхронные методы и структуры, которые обеспечивали бы именно м и н и м а л ь н ы е длительности операций при любых операндах. Из предыдущего видно, что в составе, например, многопроцессорной системы выгоднее иметь меньшее количество более сложных и более быстродействующих процессоров, чем большее количество простых, но более медленных процессоров. Равное номинальное быстродействие, которое может быть получено в обоих случаях, при меньшем количестве процессоров соответствует большему реальному пользовательскому быстродействию.

Заметим еще, что область применения синхронных методов выполнения операций и соответствующих структур арифметико-логических устройств не ограничивается лишь многопроцессорными системами типа III (с векто-

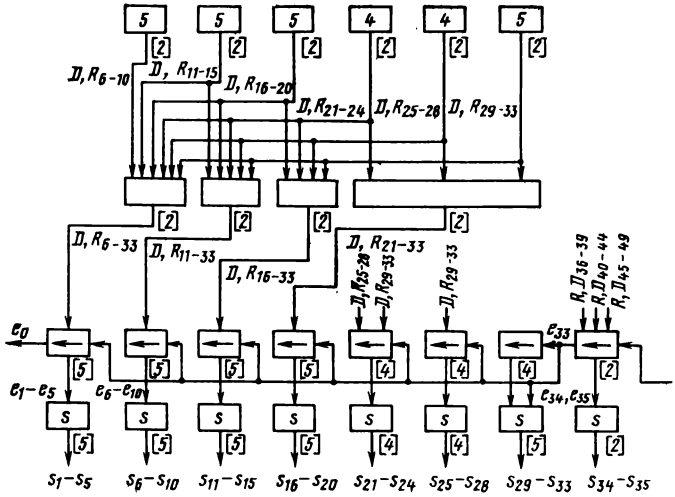
ром-инструкцией) и IV и конвейерными системами, но на самом деле намного шире. По сути дела в любом случае, когда вычислительная система или даже обычная вычислительная машина используется для управления некоторым процессом, протекающим в реальном времени, и темп, в котором должна вестись обработка информации и выдача управляющих воздействий, задается извне, необходимо применять синхронные методы выполнения операций. В противном случае существенно затруднен даже процесс отладки и испытаний системы управления в целом, поскольку без специального исследования нельзя выявить наихудшие комбинации исходных данных, при которых время работы вычислительных средств окажется максимальным. Перебор ситуаций, выбранных на основе других соображений, не гарантирует от появления в реальных условиях сбоев, вызванных нехваткой времени на обработку информации и выработку управляющих сигналов.

Альтернативой к использованию синхронных методов выполнения операций в этих случаях могло бы быть применение вычислительных средств с большим запасом быстродействия. Однако это связано, естественно, с бо́льшими затратами. Во всяком случае аппаратура, применяемая в таких вычислительных средствах для уменьшения *среднего* времени выполнения операций, наверняка оказалась бы избыточной.

3. БЫСТРОДЕЙСТВУЮЩИЕ СИНХРОННЫЕ СУММИРУЮЩИЕ УСТРОЙСТВА

3.1. СВЕРХПАРАЛЛЕЛЬНЫЕ СУММАТОРЫ

Рассмотрение ускоренных синхронных сумматоров мы начнем с описания одного из самых быстрых (но и одного из самых громоздких) построений — сверхпараллельного сумматора (сумматор с параллельным переносом, сумматор с одновременным переносом, simultaneous carry adder, carry look—ahead adder). Идея такого построения состоит в том, что сумматор разбивается на сравнительно короткие группы разрядов, которые объединяются в более крупные группы, те — в еще



более крупные и т. д., причем в каждом разряде и в каждой группе разрядов производится выработка специальных подготовительных функций, которые затем используются для организации ускоренного пробега переносов вдоль всего сумматора. Самым существенным здесь является то, что эти подготовительные функции не зависят от значения сигнала переноса, поступающего в соответствующий разряд или группу разрядов. Поэтому указанные функции можно вырабатывать одновременно.

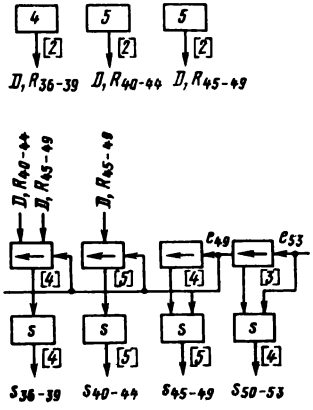
Две подготовительные функции, которые мы будем обозначать буквами D и R, являются соответственно функцией переноса, возникшего внутри разряда (группы разрядов), и функцией разрешения распространения переноса через разряд (группу разрядов).

На рис. 3.1.1 приведена структурная схема сумматора, являющегося первым из описанных в литературе сверхпараллельных сумматоров [1]. В каждом из разрядов этого 53-разрядного сумматора вырабатываются поразрядные подготовительные функции

$$D_i = a_i c_i, \quad R_i = a_i \bar{c}_i + \bar{a}_i c_i \qquad (3.1.1)$$

(знаком «+» обозначается логическая функция ИЛИ). Здесь a_i, c_i — двоичные цифры слагаемых в i-м разряде (i=1, 2, ..., 53). Узлы выработки сигналов D_i, R_i на рисунке не показаны.

Рис. 3.1.1



Групповые сигналы D и R вырабатываются многоярусными (в данном случае 2-ярусными) схемами. В первом ярусе, как это видно из рисунка, вырабатываются функции D и R для 4- и 5-разрядных групп, на которые разбит сумматор (количество разрядов указано внутри прямоугольников, изображающих схемы выработки сигналов D и R). Групповые сигналы D и R формируются из соответствующих разрядных сигналов D_i и R_i. Так, например, для 25—28 разрядов вырабатываются два сигнала:

$$\left.\begin{aligned} D_{25-28} &= D_{25} + D_{26}R_{25} + D_{27}R_{26}R_{25} + D_{28}R_{27}R_{26}R_{25}, \\ R_{25-28} &= R_{25}R_{26}R_{27}R_{28}. \end{aligned}\right\} \quad (3.1.2)$$

(На рисунке оба эти сигнала для простоты обозначены одной стрелкой и надписью D, R_{25-28}. В квадратных скобках на рисунке указаны действительные количества сигналов, изображенных одной стрелкой.)

Принцип формирования групповых сигналов D и R ясно виден из соотношений (3.1.2).

В группе возникает и выходит из нее перенос, если

а) в старшем разряде $D=1$ или

б) в следующем разряде $D=1$, а через старший разряд разрешено пройти переносу ($R=1$), или

в) в третьем слева разряде $D=1$ и при этом в каждом из двух старших разрядов $R=1$ и т. д.;

через группу разрешается распространение переноса, если оно разрешено в каждом из разрядов группы.

Сигналы D и R в группе, объединяющей несколько более «мелких» групп, формируются по такому же принципу и по аналогичным формулам: если в предыдущем предложении под словом «группа» понимать группу, объединяющую «мелкие» группы, а под словом «разряд» понимать «мелкую» группу, то все предложение остается верным.

Соотношения (3.1.1) и (3.1.2) подтверждают, что как поразрядные, так и групповые сигналы D и R не зависят от сигнала переноса, поступающего в данный разряд или группу разрядов.

Из рис. 3.1.1 видно, как группы разрядов объединяются в более крупные группы и как снова вырабатываются функции D и R. Таким образом составляется многоярусная (на рис. 3.1.1 — двухъярусная) схема, в каждом из ярусов которой функции D и R формируются из аналогичных функций предыдущих ярусов. В схеме на рис. 3.1.1, например,

$$D_{16-33}=D_{16-20}+D_{21-24}R_{16-20}+D_{25-28}R_{21-24}R_{16-20}+$$

$$+D_{29-33}R_{25-28}R_{21-24}R_{16-20}, \quad R_{16-33}=R_{16-20}R_{21-24}R_{25-28}R_{29-33}.$$

Сигналы переносов и поразрядных сумм формируются в схемах, которые можно считать отделенными от схем формирования сигналов D и R. Сигналы переноса e_i вырабатываются как функции сигналов D и R, а также некоторых сигналов переноса более младших разрядов. Сигнал e_{53} считается внешним. Сигналы сумм s_i получаются из сигналов переносов и из сигналов D_i, R_i, a_i, c_i (на рисунке сигналы D_i, R_i, a_i, c_i не показаны, из сигналов e_i показаны только некоторые). Каждый прямоугольник с горизонтальной стрелкой внутри изображает схемы формирования нескольких сигналов переноса, которые вырабатываются одновременно. Нетрудно понять, что сигналы переноса формируются следующим образом:

$$e_{52}=D_{53}+e_{53}R_{53},$$

$$e_{51}=D_{52}+D_{53}R_{52}+e_{53}R_{53}R_{52},$$

$$\vdots$$

$$e_{49}=D_{50}+D_{51}R_{50}+D_{52}R_{51}R_{50}+D_{53}R_{52}R_{51}R_{50}+$$
$$+e_{53}R_{53}R_{52}R_{51}R_{50},$$

$$e_{48}=D_{49}+e_{49}R_{49},$$

$$\vdots$$

$$e_{45}=D_{46}+D_{47}R_{46}+D_{48}R_{47}R_{46}+D_{49}R_{48}R_{47}R_{46}+$$
$$+e_{49}R_{49}R_{48}R_{47}R_{46},$$

$$e_{44}=D_{45-49}+e_{49}R_{45-49},$$

$$e_{43}=D_{44}+D_{45-49}R_{44}+e_{49}R_{45-49}R_{44},$$

$$\vdots$$

$$e_{21} = D_{22} + D_{23}R_{22} + D_{24}R_{23}R_{22} + D_{25-28}R_{24}R_{23}R_{22} +$$
$$+ D_{29-33}R_{25-28}R_{24}R_{23}R_{22} + e_{33}R_{29-33}R_{25-28}R_{24}R_{23}R_{22},$$

$$\vdots$$

$$e_0 = D_1 + D_2R_1 + D_3R_2R_1 + D_4R_3R_2R_1 + D_5R_4R_3R_2R_1 +$$
$$+ D_{6-33}R_5R_4R_3R_2R_1 + e_{33}R_{6-33}R_5R_4R_3R_2R_1.$$

Сверхпараллельный сумматор может не иметь цепи кольцевого переноса (рис. 3.1.1), но может, как это будет показано ниже, и содержать ее.

Общим признаком, объединяющим различные модификации сверхпараллельных сумматоров, можно считать наличие многоярусных схем выработки функций D и R для групп разрядов.

В сумматоре, показанном на рис. 3.1.1, в одном и том же ярусе имеются пересекающиеся группы разрядов. Во втором ярусе таких групп четыре: разряды 6—33, 11—33, 16—33, 21—33. Любые две из них содержат некоторые «общие» разряды (например, разряды 11—33 для первых двух групп). В то же время эти группы являются независимыми в том смысле, что их сигналы D и R вырабатываются раздельными схемами. Кроме того, следует заметить, что в нижнем ярусе схем D, R после всех объединений осталось несколько групп (например, группы 6—33, 36—39, 40—44, 45—49). Разряды 1—5, 34—35 вообще не вошли ни в какие группы (для этих разрядов не формируются групповые сигналы D, R).

Наличие пересекающихся групп разрядов в одном ярусе и нескольких групп — в нижнем ярусе объединения не является, вообще говоря, характерным для сверхпараллельных сумматоров. Часто встречаются структуры, в которых

а) схемы формирования функций D и R, как и схемы формирования переносов, имеют многоярусную структуру; во всех ярусах разбиение на группы схем формирования сигналов D, R совпадает с разбиением на группы схем образования переносов;

б) в любом ярусе нет пересекающихся групп;

в) в нижнем ярусе схем формирования функций D, R имеется только одна группа (охватывающая все разряды);

г) в каждом ярусе используется один коэффициент объединения: в первом ярусе разряды объединяются в группы по k_1 разрядов, во втором ярусе объединяются по k_2 групп первого яруса, в третьем ярусе — по k_3 групп второго яруса и т. д.; часто бывает, что $k_1=k_2=\ldots=k_M=k$, где M — число ярусов; при этом разрядность сумматора, очевидно, равна k^M;

д) в любом ярусе формирования сигналов D, R или сигналов переноса все выходные сигналы вырабатываются параллельно (одновременно).

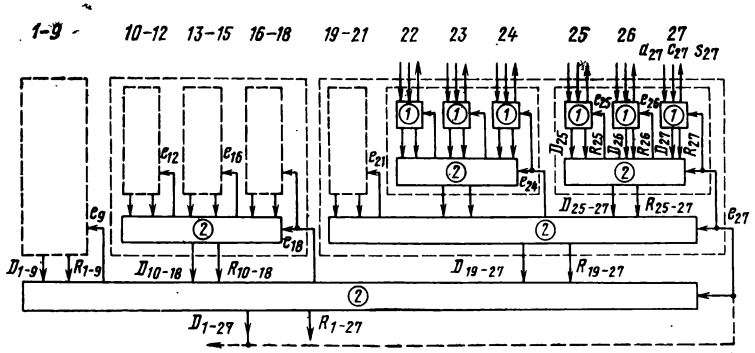


Рис. 3.1.2

В реальной аппаратуре встречаются, конечно, те или иные отклонения от этих правил. Например, в первом ярусе все группы, кроме одной, могут содержать по k_1 разрядов, а одна группа — меньше чем k_1 разрядов.

На рис. 3.1.2 показана подробная схема сумматора, полностью соответствующего перечисленным правилам. Графическое изображение этого сумматора отличается от рис. 3.1.1 тем, что на рис. 3.1.2 схемы образования сигналов D, R в ярусах объединены с соответствующими схемами формирования переносов. При таком изображении получается, что сначала сверху вниз распространяется процесс образования сигналов D, R, а затем снизу вверх — процесс формирования переносов. В этом трехъярусном сумматоре $k_1=k_2=k_3=k=3$, $M=3$. Разрядность сумматора равна $k^M=27$. Сумматор состоит из узлов двух типов (*1* и *2*). Возможная реализация узла *1* показана на рис. 3.1.3, где знаком =1 обозначен элемент, реализующий функцию ИСКЛЮЧАЮЩЕЕ ИЛИ:

$a\bar{c}+\bar{a}c$. На рис. 3.1.4 приведена схема узла 2, на которой разряды (группы), объединяемые данным узлом, пронумерованы цифрами 1, 2, 3, возрастающими в сторону младших разрядов. При вычитании чисел с использованием обратного кода сигнал D_{1-27} возникновения переноса внутри 27-разрядной группы является одновременно сигналом кольцевого переноса e_{27}. В сумматоре без кольцевого переноса цепь D_{1-27}—e_{27} следует разомкнуть. При этом D_{1-27} может служить сигналом переполнения сумматора.

Чтобы оценить время работы сверхпараллельного

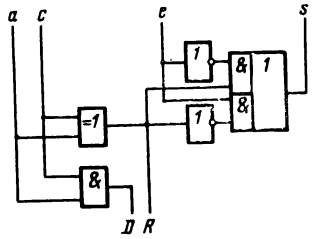


Рис. 3.1.3

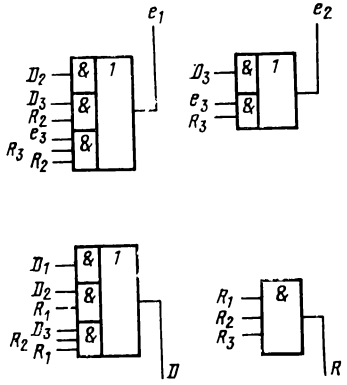


Рис. 3.1.4

сумматора, допустим, что время выработки каждого из поразрядных сигналов D_i, R_i равно 1τ, за такое же время, допустим, вырабатываются групповые сигналы D, R в каждом из M ярусов. Время выработки сигналов переноса e_i в каждом ярусе пусть тоже будет равно 1τ. Наконец, допустим, что сигналы s_i появляются через 2τ после появления сигналов e_i. Тогда полное время работы M-ярусного сверхпараллельного сумматора при наличии цепи кольцевого переноса равно

$$T=(2M+3)\tau \qquad (3.1.3)$$

(1τ тратится на выработку поразрядных сигналов D_i, R_i, еще $M\tau$ — на групповые сигналы D, R в M ярусах, $M\tau$ — на сигналы переноса в M ярусах и еще 2τ — на выработку s_i). При отсутствии цепи кольцевого переноса время работы сумматора уменьшается на 1τ (так как выработка сигналов переноса начинается не после формирования сигнала D в нижнем ярусе, а на 1τ раньше—

после формирования сигналов D, R в предпоследнем ярусе).

Если в каждом ярусе используется только один коэффициент объединения k, то, как уже отмечено,

$$n=k^M.$$

Подставляя $M=\log_k n$ в (3.1.3), получаем:

$$T=(2\log_k n+3)\tau.$$

Видно, что время работы сумматора T при данном n (или максимальная разрядность n, достижимая при заданном T) сильно зависит от коэффициентов объединения. При $k=3$ трехъярусный сумматор может содержать $3^3=27$ разрядов и время его работы, если справедливы допущения, приведшие к формуле (3.1.3), равно 9τ. При $k=5$ таким же быстродействием обладает уже сумматор с разрядностью $n=5^3=125$. Очевидно, что для достижения высокого быстродействия следует увеличивать коэффициенты объединения. Однако при увеличении k растет

а) максимальное количество входов в схеме И,

б) максимальное количество входов в схеме ИЛИ,

в) нагрузка на элементы.

Существующие ограничения этих трех параметров приводят к тому, что на практике чаще всего в сверхпараллельных сумматорах используются коэффициенты объединения $k=4$ и $k=5$.

Следует отметить одну особенность сверхпараллельных сумматоров с кольцевым переносом, которая хорошо видна, например, из рис. 3.1.2: схема, реализующая переносы, в действительности не является кольцевой, т. е. замкнутой в кольцо схемой. Поэтому в сверхпараллельных сумматорах с цепью кольцевого переноса нет опасности генерации сигналов в этой цепи.

Еще одно общее замечание по структуре сверхпараллельного сумматора: из сказанного в этом разделе должно быть ясно, что функционирование сумматора не изменится, если разрядные сигналы $R_i=a_i\bar{c}_i+\bar{a}_i c_i$ будут заменены более простыми сигналами a_i+c_i, что обычно и делается *). Используя обозначения формул (3.1.1), можно сказать, что в этих случаях вместо сигналов R_i используются сигналы R_i+D_i.

*) В схеме на рис. 3.1.3 при этом ИСКЛЮЧАЮЩИЕ ИЛИ заменяется простым ИЛИ, а сигнал s должен вырабатываться по формуле $eD+e\bar{R}+\bar{e}R\bar{D}$, а не $e\bar{R}+\bar{e}R$.

Организацию ускорения переносов в сверхпараллельном сумматоре можно осуществлять при помощи не сигналов D и R, а сигналов, реализующих другие подготовительные функции. Например, существуют серийно выпускаемые интегральные микросхемы двух типов, из которых можно, в частности, строить сверхпараллельные сумматоры с коэффициентами объединения $k=4$. Особенность этих сумматоров состоит в том, что в любой группе вместо групповых сигналов

$$D=D_1+D_2R_1+D_3R_2R_1+D_4R_3R_2R_1,$$

$$R=R_1R_2R_3R_4$$

(индексы 1, 2, 3, 4 соответствуют четырем разрядам или четырем группам, входящим в данную группу; номера возрастают в сторону младших разрядов) вырабатываются два других сигнала:

$$\Sigma D=D_1+D_2+D_3+D_4,$$

$$R+D=D_1+D_2R_1+D_3R_2R_1+D_4R_3R_2R_1+$$

$$+R_4R_3R_2R_1,$$

которые и используются для организации ускоренных переносов. Подробности структуры таких сумматоров будут рассмотрены в § 6.2.

Более узкой группой сверхпараллельных сумматоров можно считать такие сумматоры, принцип действия которых состоит в выработке сигналов D и R многоярусными схемами (что имело место и в рассмотренных схемах) и последующей *одновременной* выработке сигналов переноса для всех разрядов (вместо формирования переносов в многоярусных схемах). Будем называть такие построения *сумматорами с одновременным переносом*. По существу в первой публикации по сверхпараллельным сумматорам была приведена схема с одновременным переносом — см. рис. 3.1.1. Если сигналы e_{49} и e_{33} обозначить как D_{50-53} и D_{34-53}, то станет ясно, что любой из сигналов e_i вырабатывается прямо из сигналов R и D. Еще яснее отличие сумматоров с одновременным переносом от обычных сверхпараллельных сумматоров видно из сопоставления сумматора, показанного на рис. 3.1.2, с изображенным на рис. 3.1.5 аналогичным сумматором с одновременным переносом (показан сумматор без кольцевого переноса). Из рис. 3.1.5 видно, что

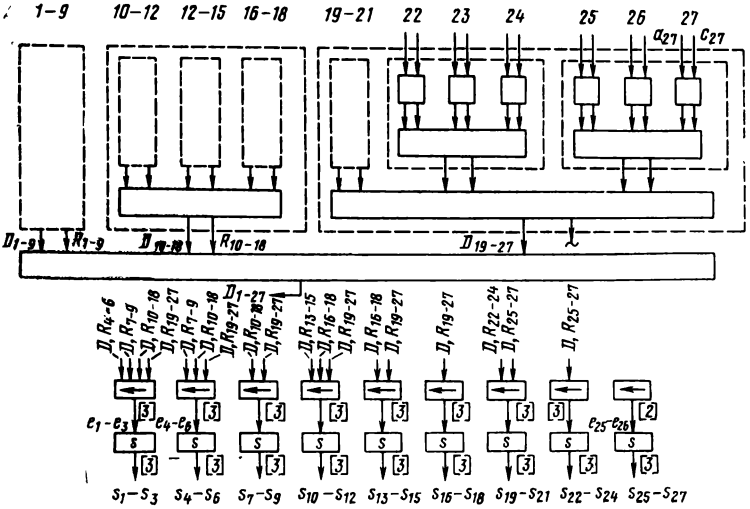


Рис. 3.1.5

сигналы D, R вырабатываются так же, как и в схеме на рис. 3.1.2. Что касается сигналов e_i, то они вырабатываются в узлах, изображенных в виде прямоугольников со стрелками, непосредственно из сигналов D, R. Например, сигналы e_1—e_3 вырабатываются следующим образом:

$$e_3 = D_{4-6} + D_{7-9}R_{4-6} + D_{10-18}R_{7-9}R_{4-6} + \\ + D_{19-27}R_{10-18}R_{7-9}R_{4-6},$$

$$e_2 = D_3 + D_{4-6}R_3 + D_{7-9}R_{4-6}R_3 + D_{10-18}R_{7-9}R_{4-6}R_3 + \\ + D_{19-27}R_{10-18}R_{7-9}R_{4-6}R_3,$$

$$e_1 = D_2 + D_3R_2 + D_{4-6}R_3R_2 + D_{7-9}R_{4-6}R_3R_2 + \\ + D_{10-18}R_{7-9}R_{4-6}R_3R_2 + D_{19-27}R_{10-18}R_{7-9}R_{4-6}R_3R_2.$$

(На рис. 3.1.5, как и на рис. 3.1.1, сигналы R_i, D_i, a_i, c_i на входах узлов, вырабатывающих сигналы e_i и s_i, не показаны.)

Видно, что дополнительное увеличение скорости (благодаря отсутствию многоярусных схем выработки переносов e_i) дается в последней схеме ценой нарастающей справа налево сложности схем выработки переносов и

ценой повышенных требований к нагрузочным способностям групповых сигналов D, R.

Структуру сверхпараллельного сумматора можно условно считать исходной для многих разновидностей ускоренных сумматоров. Рассмотрим кратко несколько возможных изменений, приводящих к другим построениям. Некоторые из этих построений хорошо известны. Рассмотрение будем иллюстрировать упрощенными блок-схемами сумматоров (рис. 3.1.6). Характер упрощений

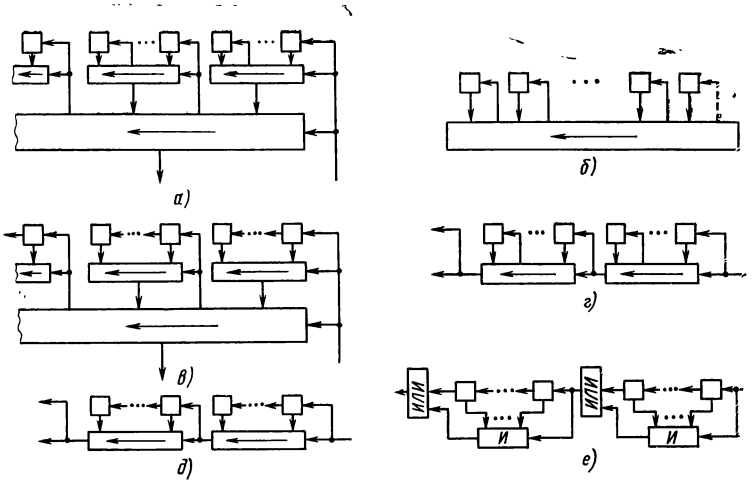


Рис. 3.1.6

виден из сравнения изображений сверхпараллельного сумматора на рис. 3.1.2 и 3.1.6,*а*.

1°. Если разрядность сумматора невелика по сравнению с возможностями, которыми обладает используемая система элементов, то можно построить весь сумматор как одну группу сверхпараллельного сумматора (рис. 3.1.6,*б*). Такое построение называется *параллельно-параллельной логикой* [2]. Принимая уже сделанные допущения о скоростях работы элементов, можно сказать, что в параллельно-параллельном сумматоре 1τ тратится на выработку разрядных сигналов D_i, R_i, еще 1τ — на одновременную выработку всех сигналов переноса (как при наличии, так и при отсутствии цепи коль-

цевого переноса) и еще 2τ — на выработку сигналов суммы. Следовательно,

$$T=4\tau.$$

Подробнее параллельно-параллельные сумматоры рассмотрены в § 3.2.

2°. Вторая разновидность сверхпараллельного сумматора мало отличается от основной схемы. Единственное отличие состоит в том (рис. 3.1.6,*в*), что внутри каждой группы первого яруса все разрядные сигналы переноса вырабатываются не одновременно за 1τ после поступления сигнала переноса в младший разряд группы, а последовательно, как в обычном неускоренном сумматоре, т. е. схемы выработки переносов e_i внутри группы, содержащей k разрядов, теперь описываются не соотношениями

$$\left.\begin{aligned}
&e_{k-1}=D_k+e_kR_k,\\
&e_{k-2}=D_{k-1}+D_kR_{k-1}+e_kR_kR_{k-1},\\
&\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\\
&e_1=D_2+D_3R_2+D_4R_3R_2+\dots+D_kR_{k-1}R_{k-2}\dots\\
&\qquad\qquad\dots R_2+e_kR_kR_{k-1}\dots R_2,
\end{aligned}\right\}\qquad(3.1.4)$$

а соотношениями

$$e_i=D_{i+1}+e_{i+1}R_{i+1}\ \ (i=k-1,\ k-2,\ \dots,\ 1)$$

(разряды в группе пронумерованы от 1 до k в сторону младших разрядов). Этим достигается существенная экономия аппаратуры. Что касается времени суммирования, то оно, естественно, увеличивается и становится равным

$$T=((2M+3)-1+(k-1))\tau=(2M+k+1)\tau,$$

где $(2M+3)\tau$ — время суммирования в исходной схеме — см. (3.1.3); $(k-1)\tau$ — максимальное время распространения переноса в группе первого яруса (считается, что k — это разрядность каждой группы).

Таким образом, потери времени по сравнению с исходной схемой составляют $(k-2)\tau$, что при $k=4-5$ равно всего лишь $(2-3)\tau$.

3°. Третья разновидность (рис. 3.1.6,*г*) сильно отличается от основной схемы. Это — одноярусная структура. Можно сказать, что эта разновидность получается путем последовательного соединения групп первого яруса

сверхпараллельного сумматора. Группы могут иметь, вообще говоря, разные длины. Пусть m — число групп, k_i — разрядность i-й группы (i=1, 2, ..., m). При этом число разрядов в сумматоре равно

$$n=\sum_{i=1}^{m} k_i.$$

В частности, при $k_1=k_2= \ldots =k_m=k$

$$n=mk.$$

Переносы внутри каждой k-разрядной группы вырабатываются так же, как в сверхпараллельном сумматоре, т. е. по формулам (3.1.4). Групповые сигналы D, R не вырабатываются. Вместо этого в каждой группе формируется сигнал группового переноса, выходящего из данной группы и поступающего в следующую группу:

$$e_0=D_1+D_2R_1+D_3R_2R_1+ \ldots +D_kR_{k-1}R_{k-2} \ldots$$
$$\ldots R_1+e_kR_kR_{k-1} \ldots R_1 \qquad (3.1.5)$$

(здесь тоже разряды группы пронумерованы от 1 до k; номера возрастают от старших разрядов к младшим). Полагая, что e_0 формируется из e_k за время, равное 1τ, можно определить T:

$$T=(m+4)\tau$$

в сумматоре с кольцевым переносом и

$$T=(m+3)\tau$$

в сумматоре без кольцевого переноса. В первом случае 1τ тратится на выработку разрядных сигналов D_i, R_i, еще $m\tau$ — на поочередную выработку в m группах сигналов переноса, выходящих из этих групп (рассматривается самый «тяжелый» случай, когда перенос, возникнув в одной из групп, обегает по кольцу весь сумматор и возвращается в ту группу, где он возник), 1τ уходит на формирование переносов внутри группы, в которую поступает последний из групповых переносов, и, наконец, 2τ тратится на выработку сигналов суммы в этой группе. При отсутствии цепи кольцевого переноса слагаемое m в сумме $(1+m+1+2)\tau$ меняется на $m-1$, поэтому $T=(m+3)\tau$.

4°. Четвертая разновидность (рис. 3.1.6,*д*) получается путем упрощения предыдущей схемы: внутри каждой

группы одновременная выработка переносов заменяется последовательной; перенос из группы остается ускоренным. Если все группы имеют одинаковую длину k, то $n=mk$, где m — число групп. Время работы такого сумматора с кольцевым переносом следует считать равным

$$T=(m+k+2)\tau.$$

Это время соответствует самому «тяжелому» случаю, когда перенос возникает в старшем разряде некоторой группы (в этом разряде $a=c=1$), проходит сквозь все остальные группы (в каждом разряде этих групп $a\neq c$), поступает в младший разряд той группы, где возник перенос, и проходит сквозь $k-1$ младших разрядов этой группы (в них тоже $a\neq c$). При такой ситуации 1τ тратится на выработку сигналов D_i, R_i, еще $m\tau$ уходит на поочередное формирование ускоренных переносов, выходящих из m последовательно соединенных групп, $(k-1)\tau$ требуется на распространение переносов через $k-1$ разряд указанной группы и, наконец, 2τ нужно на выработку последнего (по времени появления) сигнала суммы. Итого,

$$1+m+k-1+2=m+k+2.$$

Если цепь кольцевого переноса отсутствует, то в самом «тяжелом» случае перенос возникает в младшей группе и заканчивается в старшей. При этом

$$T=(1+m-1+k-1+2)\tau=(m+k+1)\tau. \qquad (3.1.6)$$

В табл. 3.1.1 приведены величины T и n, соответствующие некоторым значениям m и k. В каждой клетке таблицы над наклонной чертой указано T (единица

Таблица 3.1.1

	m					
	2	3	4	5	6	7
2	5/4	6/6	7/8	8/10	9/12	10/14
3	6/6	7/9	8/12	9/15	10/18	11/21
4	7/8	8/12	9/16	10/20	11/24	12/28
5	8/10	9/15	10/20	11/25	12/30	13/35
6	9/12	10/18	11/24	12/30	13/36	14/42
7	10/14	11/21	12/28	13/35	14/42	15/49

измерения — 1τ), под чертой — n. Таблица составлена для сумматора без кольцевого переноса.

В § 3.7 будет показано, что если в сумматоре этого типа длины групп не обязательно одинаковы, а могут быть разными и выбираются оптимальным образом, то можно достичь гораздо большего n при заданном T (или меньшего T при заданном n).

Отдельные группы сумматоров только что рассмотренного типа (рис. 3.1.6,*д*) тоже выпускаются в виде интегральных схем. Таковы, например, микросхемы *CD*4008 и *CD*4008A фирмы *RCA* [3]. В этих схемах $k=4$.

5°. Последняя из рассматриваемых в этом разделе разновидностей ускоренных сумматоров, родственных сверхпараллельному сумматору, получается путем некоторого упрощения и изменения предыдущей схемы. Изменение состоит в том (рис. 3.1.6,*е*), что схема ускоренного группового переноса из k-разрядной группы строится не по соотношению (3.1.5), а по формуле

$$e_0=D_1+e_1R_1+e_kR_kR_{k-1}\ \ldots\ R_1,$$

т. е. на выходе группы схемой ИЛИ складывается неускоренный перенос $D_1+e_1R_1$ и ускоренный «обходный» перенос $e_kR_kR_{k-1}\ \ldots\ R_1$. Построения такого типа называются *сумматорами с обходным переносом* (сумматоры с упреждающим переносом, carry skip adders). Эти сумматоры применяются достаточно часто, так как при небольших дополнительных затратах аппаратуры обеспечивают сравнительно высокое быстродействие. В § 3.3 будут подробнее описаны структуры сумматоров с обходными переносами, в том числе и оптимальные структуры, т. е. построения, позволяющие достичь максимального быстродействия при заданной разрядности (или максимальной разрядности при заданном быстродействии).

В частном случае, когда n-разрядный сумматор состоит из m одинаковых k-разрядных групп, время суммирования двух чисел равно

$$T=(m+2k-1)\tau \qquad (3.1.7)$$

как при наличии, так и при отсутствии цепи кольцевого переноса. Формула (3.1.7) соответствует самому «тяжелому» случаю, когда перенос возникает в младшем разряде младшей группы, проходит поочередно сквозь все

разряды этой группы, на что тратится $k\tau$, затем «перепрыгивает» $m-2$ группы (на это требуется еще $(m-2)\tau$), поступает на вход старшей группы, поочередно проходит сквозь $k-1$ младших разрядов этой группы (на это тратится $(k-1)\tau$) и, наконец, за 2τ вырабатывается сигнал суммы в старшем разряде сумматора. Итого,

$$k+m-2+k-1+2=m+2k-1.$$

Можно заметить, что в сумматорах, показанных на рис. 3.1.6,*г—е*, при наличии цепи кольцевого переноса возможна паразитная генерация сигналов в этой цепи. Генерация может возникнуть, если в каждом разряде $a_i \neq c_i$ и если, кроме того, по некоторой причине хотя бы в одном из разрядов хотя бы на короткое время выработается сигнал $e=1$. Подробнее с этим вопросом можно познакомиться, например, в работе [4].

Можно заметить, что в разных частях сумматоров, показанных на рис. 3.1.6, в различных комбинациях используются три способа организации переноса:

а) последовательные переносы (внутри групп первого яруса на рис. 3.16,*в*, внутри групп на рис. 3.1.6,*д*, *е*, между группами на рис. 3.1.6,*г—е*);

б) параллельные (одновременные) переносы (в каждом ярусе на рис. 3.1.6,*а*, внутри групп на рис. 3.1.6,*б*, *г*, переносы на входах групп в каждом ярусе на рис. 3.1.6,*в*);

в) обходные переносы — логическое сложение ускоренного и неускоренного переносов (переносы из групп на рис. 3.1.6,*е*).

Существуют и многие другие построения сумматоров, в которых эти три способа применяются в иных комбинациях [5].

Известны также различные типы ускоренных синхронных сумматоров, построенных на совершенно иных принципах. Некоторые из них рассмотрены в § 3.4—3.6.

3.2. ПАРАЛЛЕЛЬНО-ПАРАЛЛЕЛЬНЫЕ СУММАТОРЫ

Параллельно-параллельные сумматоры [2] следует считать частным и предельным случаем сумматоров с одновременным переносом. В этом построении все n разрядов сумматора рассматриваются как одна группа первого яруса (рис. 3.1.6,*б*).

Если сумматор не имеет цепи кольцевого переноса, то сигнал переноса в каждый из разрядов формируется отдельной схемой, анализирующей поразрядные функции D_i, R_i всех разрядов, более младших, чем данный, а также сигнал переноса e_n в самый младший разряд сумматора. Схема строится в соответствии с уравнением

$$e_i = D_{i+1} + D_{i+2}R_{i+1} + D_{i+3}R_{i+2}R_{i+1} + \ldots + \\ + D_nR_{n-1}R_{n-2} \ldots R_{i+1} + e_nR_nR_{n-1} \ldots R_{i+1},$$

где $D_i = a_ic_i$, $R_i = a_i\bar{c}_i + \bar{a}_ic_i$ или $R_i = a_i + c_i$.

Если сумматор содержит цепь кольцевого переноса, то сигнал переноса в любой i-й разряд ($i=1, 2, \ldots, n$) вырабатывается на основе анализа всех разрядных сигналов D, R:

$$e_i = D_{i+1} + D_{i+2}R_{i+1} + D_{i+3}R_{i+2}R_{i+1} + \ldots + \\ + D_nR_{n-1}R_{n-2} \ldots R_{i+1} + D_1R_nR_{n-1} \ldots R_{i+1} + \\ + D_2R_1R_nR_{n-1} \ldots R_{i+1} + \ldots + D_iR_{i-1}R_{i-2} \ldots \\ \ldots R_1R_nR_{n-1} \ldots R_{i+1}.$$

Сигналы переносов во всех разрядах в параллельно-параллельном сумматоре вырабатываются практически одновременно.

Параллельно-параллельная схема может быть использована тогда, когда применяются логические элементы с достаточно большим числом входов и достаточно большой нагрузочной способностью. Так, в [2] параллельно-параллельная логика описана применительно к модифицированным схемам NOR (ИЛИ—НЕ), имеющим до 25 входов и выдерживающим до 25 нагрузок.

Параллельно-параллельным можно также считать сумматор, в котором все сигналы переноса вырабатываются одновременно независимыми схемами не при помощи сигналов R_i, D_i или каких-либо других вспомогательных сигналов, а прямо из входных сигналов сумматора. Например, в сумматоре без кольцевого переноса можно вырабатывать сигналы переноса непосредственно по формулам:

$$e_{n-1} = a_nc_n + e_n(a_n + c_n),$$

$$e_{n-2} = a_{n-1}c_{n-1} + a_nc_n(a_{n-1} + c_{n-1}) + \\ + e_n(a_n + c_n)(a_{n-1} + c_{n-1}),$$

$$e_{n-3} = a_{n-2}c_{n-2} + a_{n-1}c_{n-1}(a_{n-2} + c_{n-2}) + a_n c_n (a_{n-1} + c_{n-1})(a_{n-2} + c_{n-2}) + e_n (a_n + c_n)(a_{n-1} + c_{n-1})(a_{n-2} + c_{n-2})$$

. . . . . .

Конечно, в этом случае должны использоваться (при прочих равных условиях, т. е. при той же разрядности) элементы с еще большими возможностями.

3.3. СУММАТОРЫ С ОБХОДНЫМИ ПЕРЕНОСАМИ

Принцип обходного переноса (упреждающий перенос, carry skip, carry bypass, carry bridging, anticipatory carry, carry look-ahead) предлагался еще Ч. Бэбиджем [6] для десятичной системы и был развит для двоичной си-

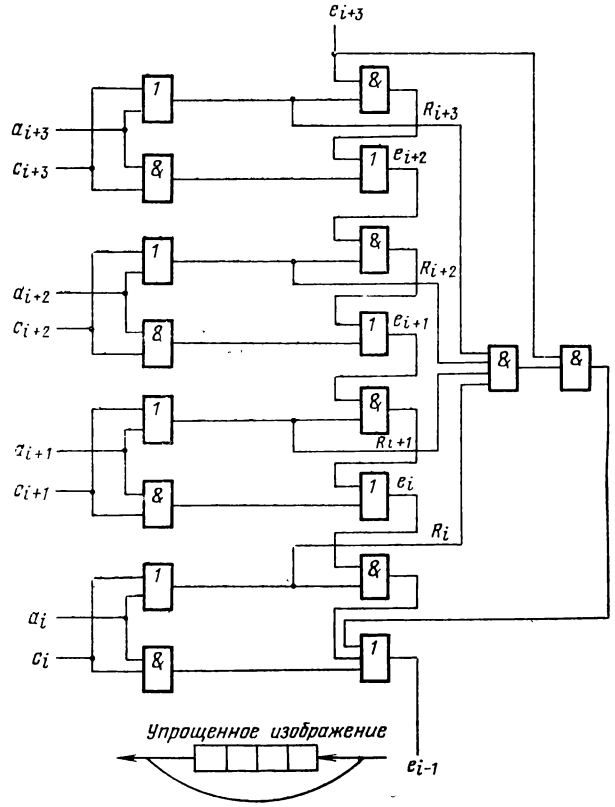


Рис. 3.3.1

стемы многими авторами [7—10]. При использовании этого метода ускорения параллельный сумматор разбивается на последовательно соединенные группы разрядов, которые охватываются цепями ускоренных обходных переносов. Выход цепи обходного переноса при помощи схемы ИЛИ объединяется с выходом цепи неускоренного переноса из соответствующего разряда сумматора. Эффект ускорения в группе, охваченной цепью обходного переноса, достигается только в том случае, если на вход младшего разряда группы поступает перенос, а в каждом из разрядов соблюдается условие разрешения распространения переноса, т. е. условие $a_i \neq c_i$ (или условие $a_i \vee c_i = 1$). Простейшая схема обходного переноса для одной четырехразрядной группы показана на рис. 3.3.1. Более сложная «двухслойная» группа показана в упрощенном виде на рис. 3.3.2. Видно, что в сумматорах с многослойными обходными переносами в одной схеме ИЛИ неускоренный перенос может объединяться с несколькими обходными переносами.

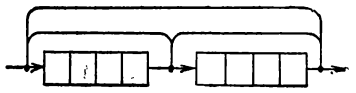


Рис. 3.3.2

Из приведенных рисунков видно, что характерной чертой схем обходного переноса является выработка и использование в них групповых сигналов типа R (сигналы разрешения распространения переноса); сигналы типа D в этих построениях не используются. Другой характерной чертой таких схем является суммирование схемой ИЛИ сигналов ускоренного и неускоренного переноса.

Последнее обстоятельство накладывает некоторые ограничения на характер управления сумматором при использовании потенциальной системы элементов. Процесс суммирования должен быть организован таким образом, чтобы не могли возникать переключения сигналов переноса с уровня 1 на уровень 0, так как в таких случаях неускоренный перенос, равный единице, пройдя сквозь схему ИЛИ, в которой он логически складывается с обходным переносом, фактически блокирует действие обходного переноса. Иными словами, в сумматоре с обходными переносами обеспечивается ускорение распространения сигналов переноса, переключающихся с уровня 0 на уровень 1 («пробег единицы»), а при обратных переключениях («пробег нуля») ускорения нет.

Защитой от этого явления может, например, служить предварительная установка в нулевое состояние всех входов сумматора перед началом суммирования.

3.3.1. ОПТИМАЛЬНЫЕ СТРУКТУРЫ СУММАТОРОВ С ОБХОДНЫМИ ПЕРЕНОСАМИ

От способа разбиения на группы сумматора с обходными переносами зависят аппаратные затраты и быстродействие сумматора. Естественно, возникает задача оптимизации структуры.

Оптимальность можно оценивать с точки зрения того или иного критерия. В этом разделе критерием оптимальности будет служить быстродействие (будем стремиться к обеспечению минимального значения максимального времени T распространения переноса при заданной разрядности n сумматора).

1°. Рассмотрим вначале задачу оптимизации структуры сумматора с группами равной длины [8]. Обозначим длину группы буквой k, а количество групп — буквой m, так что $n=mk$. За единицу времени примем время прохождения сигнала через две последовательно соединенные схемы И, ИЛИ.

Очевидно, что наихудшим случаем (время распространения переноса максимально) для сумматора без кольцевого переноса является случай, когда распространение переноса начинается в младшем, n-м, разряде сумматора, а заканчивается в старшем:

$$\left.\begin{aligned} &a_n = c_n = 1,\\ &a_i \neq c_i \ (i = 2, 3, \ldots, n-1),\\ &a_1 = c_1 = 0. \end{aligned}\right\}$$

В этом случае время распространения переноса равно

$$T = k + (m-2) + (k-1) = 2k + m - 3 \qquad (3.3.1)$$

(k — время выработки и распространения переноса в младшей группе, $m-2$ — время прохода по цепям обходного переноса, $k-1$ — время распространения переноса в старшей группе) *). Отсчет времени начинается

*) Отличие формулы (3.3.1) от (3.1.7) объясняется тем, что в (3.3.1) не учтено время, затрачиваемое на выработку сигналов суммы из сигналов переноса; в § 3.1 это время полагалось равным 2τ.

с момента поступления слагаемых на входы схем образования переносов и заканчивается формированием всех сигналов переноса (после чего могут быть определены сигналы суммы). Следует отметить, что соотношение (3.3.1) относится и к сумматору с кольцевым переносом. В этом случае оно соответствует не только описанной ситуации, но и ситуации, когда перенос возникает в младшем разряде любой группы и заканчивается в старшем разряде соседней младшей группы.

Перепишем (3.3.1):

$$T=2k+n/k-3. \qquad (3.3.2)$$

Считая оптимальным сумматор, обладающий минимальным T при данном n, приравняем нулю производную dT/dk:

$$dT/dk=2-n/k^2=0,$$

откуда

$$k=\sqrt{n/2} \qquad (3.3.3)$$

и

$$m=\sqrt{2n}. \qquad (3.3.4)$$

Соотношения (3.3.3) и (3.3.4) указывают общий характер зависимости между n, m, k в оптимальном сумматоре. В конкретных случаях можно подбирать оптимальные целочисленные значения, близкие к тем, которые получаются из этих соотношений. Иногда параметры n, m, k могут в точности удовлетворять приведенным соотношениям (например, при $n=32$, $m=8$, $k=4$). В этих случаях T, как это следует из (3.3.2) и (3.3.3), равно

$$T=2\sqrt{2n}-3,$$

откуда

$$n=(T+3)^2/8. \qquad (3.3.5)$$

В работе [8] показаны некоторые возможности дальнейшего уменьшения величины T путем использования групп переменной длины и введения многослойных обходных переносов, но подробнее эти возможности исследованы в работе [10], к краткому изложению результатов которой мы теперь перейдем.

2°. В работе [10] найдены оптимальные структуры сумматоров с группами неодинаковой длины. Рассмотрены построенные на элементах И, ИЛИ или на элементах ИЛИ—НЕ сумматоры с однослойными и двухслойными обходными переносами, с цепью кольцевого пере-

носа и без нее. Оптимальным в работе считается сумматор, обладающий при заданной разрядности n минимальным значением максимального времени T распространения переноса (за единицу времени принимается задержка пары элементов И, ИЛИ или элемента ИЛИ—НЕ) и минимальным для этого T количеством цепей обходного переноса. Кроме того, найдены структуры «экономичных» сумматоров, обладающих при заданных n и T минимальным количеством цепей обходного переноса.

В частности, показано, что при данном T максимальное количество $n^{(T)}$ разрядов в построенном на элементах И, ИЛИ сумматоре с однослойными обходными переносами, не содержащем цепи кольцевого переноса, равно

$$n^{(T)}=[(T+4)T/4)]\ ^{*)}. \qquad (3.3.6)$$

Последовательные группы при этом содержат

$$2, 3, 4, \ldots, T-\left[\frac{T}{2}\right]-1,\quad T-\left[\frac{T}{2}\right],\quad \left[\frac{T}{2}\right]+$$
$$+1,\quad \left[\frac{T}{2}\right], \ldots, 3, 2, 1$$

разрядов (последовательность указана, начиная со старших разрядов). Например, при $T=10$ группы содержат 2, 3, 4, 5, 6, 5, 4, 3, 2, 1 разрядов.

При наличии цепи кольцевого переноса

$$n^{(T)}=[(T+3)^2/8]. \qquad (3.3.7)$$

В этом случае сумматор при T, дающем при делении на 4 остаток, не равный 2, содержит группы одинаковой длины

$$k=\left[\frac{T+3}{4}+\sqrt{\frac{1}{2}(\nu_T-n^{(T)})}\right],$$

где

$$\nu_T=\frac{(T+3)^2}{8};$$

при T, дающем при делении на 4 остаток 2, сумматор состоит из групп с чередующимися длинами по k и $k-1$ разрядов, причем

$$k=\left[\frac{T+5}{4}+\sqrt{\frac{1}{2}(\nu_T-n^{(T)})}\right].$$

---

* $[x]$ обозначает целую часть x.

Например, при $T=10$ группы содержат 4, 3, 4, 3, 4, 3 разрядов.

Из сказанного можно заключить, что обходные переносы позволяют при сравнительно небольших затратах аппаратуры получить довольно высокое быстродействие.

3.4. СУММАТОРЫ С ПИРАМИДОЙ ПЕРЕНОСОВ

Пирамида переносов (piramid carry) была предложена М. Надлером [11]. Позже аналогичная схема была запатентована Е. Вагнером [12]. Структурная схема 8-разрядного двоичного сумматора, использующего пирамиду переносов, показана на рис. 3.4.1, из которого

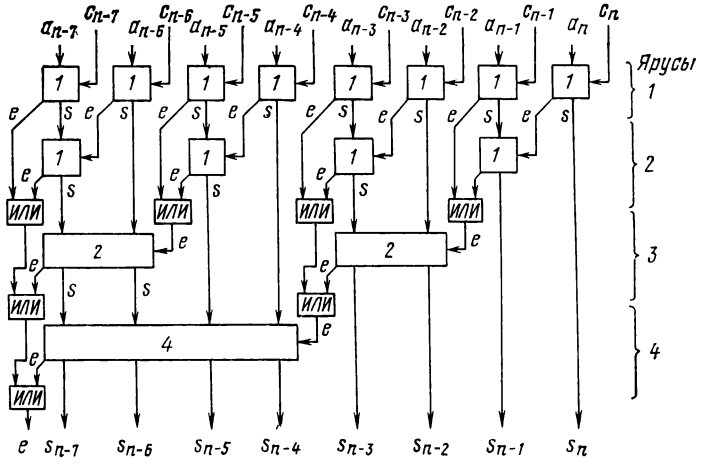


Рис. 3.4.1

видно, что этот сумматор состоит из четырех ярусов. В общем случае n-разрядный сумматор содержит M ярусов, причем

$$M=[\log_2(n-1)]+2^{*)}.$$

На входы каждого i-го яруса ($i=1, 2, \ldots, M$) подаются два числа, которые следует сложить. Для первого яруса ($i=1$) — это исходные слагаемые A и C. Числа, посту-

*) Формула справедлива при $n\geqslant 2$. Знак [ ], как и раньше, обозначает целую часть числа.

пающие на входы любого другого i-го яруса (i=2, 3, ..., M) с выходов предыдущего яруса, представляют собой сигналы суммы (n-разрядное число) и сигналы переносов (число, содержащее $n/2^{i-1}$ цифр при $n=2^k$, где k — целое). На выходах i-го яруса также вырабатываются два числа, причем цифры в числе, состоящем из сигналов переноса, расположены вдвое реже, чем на входах яруса.

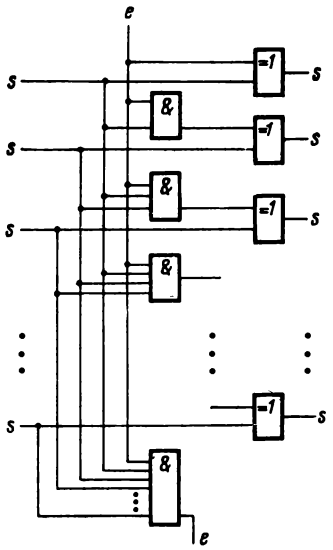


Рис. 3.4.2

Первый ярус состоит из n одноразрядных полусумматоров. Любой другой i-й ярус содержит одинаковые суммирующие схемы (счетчики), каждая из которых складывает 2^{i-2}-разрядное двоичное число с сигналом переноса, поступающего в младший разряд этой схемы. Суммирующие схемы на рис. 3.4.1 изображены в виде прямоугольников с указанной внутри них разрядностью поступающего числа. Из рисунка, в частности, видно, что первый и второй ярусы содержат одинаковые узлы (полусумматоры). В последнем ярусе вырабатывается только один сигнал переноса — сигнал переполнения сумматора (цепь кольцевого переноса в таком сумматоре отсутствует *)).

Ускорение операции сложения-вычитания в пирамиде переносов достигается благодаря тому, что указанные суммирующие схемы легко сделать ускоренными, используя, например, принцип параллельно-параллельной ло-

*) Чтобы получить сумматор с кольцевым переносом, можно было бы заменить полусумматор, складывающий цифры a_n и c_n, одноразрядным сумматором и завести на него перенос из старшего разряда. Но такая схема, во-первых, работала бы примерно вдвое медленней (это видно, например, из анализа ситуации $a_1=c_1=c_2=\ldots=c_n=1$, $a_2=a_3=\ldots=a_n=0$) и, во-вторых, нужно было бы принимать особые меры для борьбы с возможностью генерации сигналов в цепи кольцевого переноса.

гики. Одна из возможных реализаций такой суммирующей схемы приведена на рис. 3.4.2.

Время работы сумматора с пирамидой переносов определяется количеством ярусов. Если предположить, что время срабатывания каждого яруса равно 1τ, то полное время работы n-разрядного сумматора следует считать равным $([\log_2(n-1)]+2)\tau$.

3.5. СУММАТОРЫ С УСЛОВНЫМИ СУММАМИ

Сумматоры с условными суммами или условные сумматоры (conditional sum adder) по своему построению похожи на сумматоры с пирамидой переносов, однако принцип их действия иной [13].

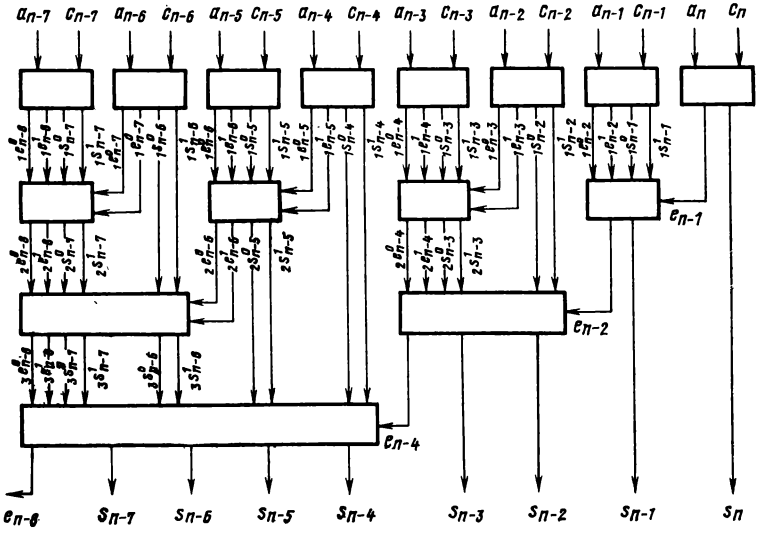


Рис. 3.5.1

Структурная схема 8-разрядного условного сумматора приведена на рис. 3.5.1. Как и пирамида сумматоров (см. рис. 3.4.1), эта схема состоит из

$$M=[\log_2(n-1)]+2$$

ярусов и количество сигналов переносов также уменьшается вдвое при переходе от предыдущего яруса к по-

следующему. Однако сами сигналы сумм и переносов во всех узлах каждого яруса (кроме крайних правых узлов) вырабатываются двойными — один комплект сигналов (с верхним индексом «0») соответствует условию равенства нулю сигнала переноса, поступающего в данную группу разрядов, другой комплект (с верхним индексом «1») соответствует переносу, равному единице. Например, узел первого яруса работает в соответствии с табл. 3.5.1, а узел второго яруса — в соответствии с табл. 3.5.2, в которой показано, чему равны выходы узла в зависимости от значений управляющих сигналов ${}_1e^0_i$, ${}_1e^1_i$ (левый нижний индекс указывает номер яруса, правый нижний — номер разряда). Из табл. 3.5.2 видно, что узел второго яруса является довольно простым коммутатором, работающим в одном из трех режимов. Аналогично работают и узлы последующих ярусов.

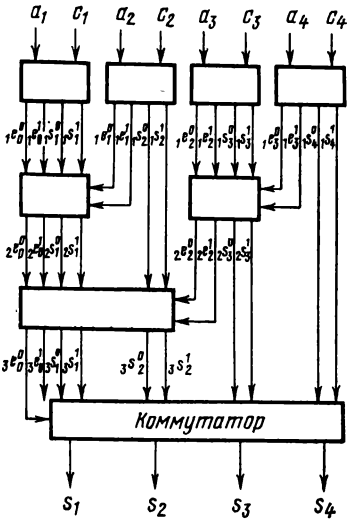


Рис. 3.5.2

Вдоль нижнего края сумматора справа налево протекает процесс установления истинных (а не условных) значений сигналов переноса и эти сигналы управляют выбором истинных значений сумм из двух поступающих сверху комплектов условных сумм. Максимальное время работы такого сумматора, как и сумматора с пирамидой переносов, пропорционально двоичному логарифму от количества разрядов.

На первый взгляд кажется, что условный сумматор не может работать в режиме с кольцевым переносом. Этот недостаток можно устранить следующим образом (см. пример для случая $n=4$ на рис. 3.5.2). Сделаем правый узел в каждом ярусе таким же, как остальные узлы яруса. Тогда на выходах сумматора образуется два комплекта сумм s^0, s^1 и два сигнала переноса e^0_0, e^1_0 из старшего разряда. Установим на выходах сумматора до-

полнительный коммутатор, управляемый сигналом e^0_0. При $e^0_0=1$ этот коммутатор будет пропускать сквозь себя в качестве окончательных результатов комплект сумм s^1, а при $e^0_0=0$ — комплект s^0. Этим, очевидно, и обеспечивается режим кольцевого переноса. Время сложения при этом увеличивается только на время срабатывания указанного коммутатора.

Таблица 3.5.1

$a_i c_i$	${}_1s^0{}_i$	${}_1e^0{}_{i-1}$	${}_1s^1{}_i$	${}_1e^1{}_{i-1}$
0 0	0	0	1	0
0 1	1	0	0	1
1 0	1	0	0	1
1 1	0	1	1	1

Таблица 3.5.2

${}_1c^0{}_i$ ${}_1c^1{}_i$	${}_2s^0{}_i$	${}_2e^0{}_{i-1}$	${}_2s^1{}_i$	${}_2e^1{}_{i-1}$
0 0	${}_1s^0{}_i$	${}_1e^0{}_{i-1}$	${}_1s^0{}_i$	${}_1e^0{}_{i-1}$
1 1	${}_1s^1{}_i$	${}_1e^1{}_{i-1}$	${}_1s^1{}_i$	${}_1e^1{}_{i-1}$
0 1	${}_1s^0{}_i$	${}_1e^0{}_{i-1}$	${}_1s^1{}_i$	${}_1e^1{}_{i-1}$

3.6. СУММАТОРЫ С ВЫБОРОМ ПЕРЕНОСА

В так называемых сумматорах с выбором переноса (carry select adder) [14] используется по существу та же идея, что и в условных сумматорах, но схемное решение здесь иное. Отличие состоит в том, что весь сумматор сразу разбивается на группы разрядов (например, по m разрядов в группе) и для каждой такой группы строятся два m-разрядных сумматора, в младший разряд одного из которых принудительно вводят перенос $e=0$, а в младший разряд другого — перенос $e=1$. Истинные значения сигналов переноса, поступающих в каждую из таких групп, вырабатываются специальными схемами ускоренного распространения переноса. Эти сигналы, как и в условном сумматоре, управляют выбором одного из двух комплектов условных сумм. Остальные

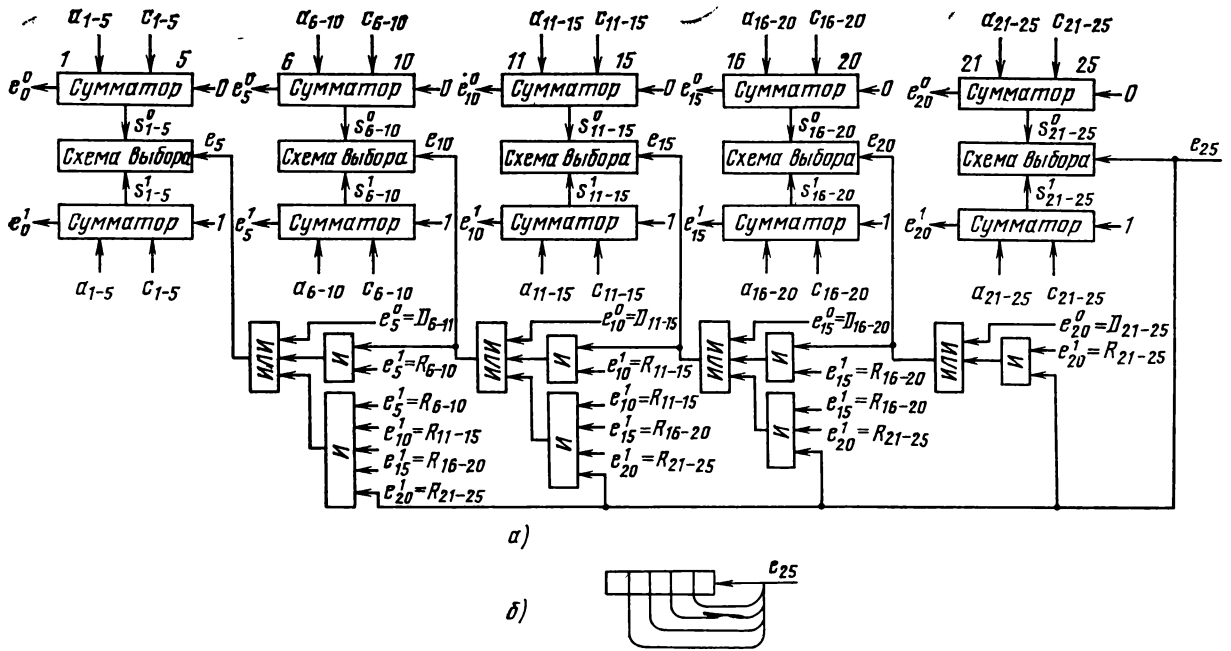


Рис. 3.6.1

ные детали видны из рис. 3.6.1,*а*, на котором показана структурная схема сумматора с выбором переноса для случая $n=25$, $m=5$.

Из рисунка видно, что внутри сумматора между 5-разрядными группами организован по существу последовательный пробег переносов, возникающих из этих групп, но, кроме того, имеются 3 цепи обходных переносов — от сигнала e_{25} до входов старших трех групп. Изображая каждую 5-разрядную группу одним квадратом, можно весь сумматор нарисовать так, как это сделано на рис. 3.6.1,*б*. Видно, что мы имеем дело с типичной схемой обходных переносов между группами. Однако не это существенно для сумматоров с выбором переноса, так как распространение переноса между 5-разрядными группами можно организовать и по другим известным принципам, например так, как это делается в сверхпараллельном сумматоре. Существенно само построение группы, позволяющее, в частности, использовать в качестве групповых сигналов D и R соответственно сигналы переноса e^0 и e^1, получающиеся на выходе старшего разряда каждой группы. В [14] показано, что можно легко образовать сигналы D и R для всего 25-разрядного сумматора, построив схемы этих двух сигналов в соответствии с соотношениями

$$\left.\begin{aligned} D&=e^0_0+e^0_5e^1_0+e^0_{10}e^1_5e^1_0+e^0_{15}e^1_{10}e^1_5e^1_0+\\ &+e^0_{20}e^1_{15}e^1_{10}e^1_5e^1_0,\\ R&=e^1_0e^1_5e^1_{10}e^1_{15}e^1_{20}. \end{aligned}\right\} \qquad (3.6.1)$$

Далее можно построить, например, 100-разрядный сумматор из четырех описанных сумматоров, организовав распространение переноса между 25-разрядными группами при помощи сигналов (3.6.1), например так, как это делается в сверхпараллельном сумматоре (см. рис. 3.1.2). Однако и это, естественно, не является отличительной или тем более обязательной чертой сумматоров с выбором переноса.

3.7. ПРИМЕР ОПТИМИЗАЦИИ СТРУКТУРЫ СУММАТОРА

В этом параграфе приводится расчет оптимальных длин групп разрядов сумматора, состоящего из последовательно соединенных групп, в каждой из которых производится ускорение как переноса, идущего «в обход»

группы, так и переноса, возникшего внутри группы и вышедшего из нее. Схема одной такой группы показана на рис. 3.7.1 (изображены только цепи образования переносов и не показаны узлы, формирующие сигналы суммы). Эта группа отличается от приведенной на рис. 3.3.1 группы с обходными переносами наличием отмеченных звездочкой схем И, заменивших схему И со входными сигналами R_i, e_i. При этом не только обходный перенос за кратчайшее время проходит сквозь группу, но и «местный» перенос, возникший внутри группы, вырабатывается за такое же время. Внутри группы схема распространения переноса осталась последовательной. Иными словами, в рассматриваемом сумматоре имеются последовательные переносы внутри групп и между группами, параллельно-параллельные переносы из групп. Сумматоры такого типа были кратко описаны в § 3.1 (см. рис. 3.1.6,*д*).

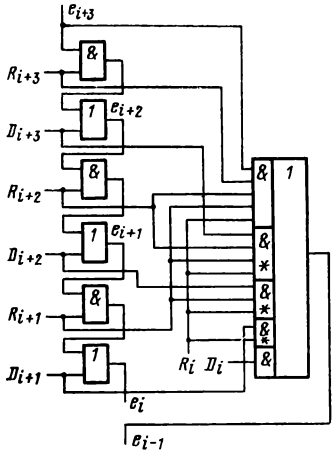


Рис. 3.7.1

Рассматриваемый сумматор должен, очевидно, обладать бо́льшим быстродействием, чем соответствующий сумматор с однослойными обходными переносами (при прочих равных условиях) и, кроме того, он свободен от указанного в § 3.3 недостатка (медленный пробег нуля). Исследуем задачу определения оптимальных длин групп, обеспечивающих максимальную длину сумматора $n=n^{(T)}$ при заданном максимальном времени T распространения переноса [15]. Сумматор рассматриваемого типа с такими параметрами будем называть *оптимальным*.

Обозначим через m количество групп в сумматоре и через $k_i (i=1, 2, \ldots, m)$ — количество разрядов в i-й группе. Очевидно, что

$$\sum_{i=1}^{m} k_i = n.$$

Будем считать, что номера групп возрастают по направлению от младших разрядов сумматора к старшим. За единицу времени, как и в п. 3.3.1, примем время прохождения сигнала через пару схем И, ИЛИ.

Заметим, что в сумматоре без кольцевого переноса цепочка переноса, обладающая максимальным временем распространения, при данном типе ускорения должна обязательно начинаться в младшей группе сумматора. Это следует из того, что время пробега переноса, возникшего в некоторой j-й группе ($j \neq 1$) и закончившегося в некотором месте i-й группы, наверняка меньше времени пробега, начавшегося в первой группе и закончившегося в том же месте той же i-й группы. Будем говорить только о таких переносах и будем называть *переносом, закончившимся в i-й группе*, перенос, возникший в первой группе и достигший выхода некоторого разряда i-й группы.

Обозначим через t_i максимальное время пробега переноса, оканчивающегося в i-й группе. Понятно, что

$$t_i = (i-1) + \max((k_i - 1), 1), \quad i = 1, 2, \ldots, m. \tag{3.7.1}$$

Здесь член $(i-1)$ соответствует времени распространения переноса от младшей группы сумматора до начала i-й группы; член $k_i - 1$ обозначает максимальное время пробега внутри i-й группы; если $k_i = 1$, то вместо величины $k_i - 1$ следует взять 1, на что и указывает член $\max((k_i - 1), 1)$.

Максимальное время распространения переноса в сумматоре определяется соотношением

$$T = \max_i t_i = \max_i \{(i-1) + \max_i ((k_i - 1), 1)\}^{*}.$$

Легко показать, что в оптимальном сумматоре без кольцевого переноса каждая группа должна содержать не менее двух разрядов. В самом деле, допустим, что $k_j = 1$. Тогда

$$t_j = (j-1) + 1 = j.$$

---

*) При $k_1 = k_2 = \ldots = k_m = k$ эта формула превращается в $T = m - 1 + k - 1 = m + k - 2$, что на 3 меньше, чем оценка, получающаяся по формуле (3.1.6). Разница объясняется тем, что в (3.1.6) учтено время (1τ), затрачиваемое на выработку сигналов R_i, D_i, и (2τ), затрачиваемое на получение сигналов суммы из сигналов переноса.

Если мы увеличим k_j на единицу, то t_j станет равно

$$t_j=(j-1)+(k_j-1)=j,$$

т. е. не изменится. Остальные t_i при этом тоже не изменяются. Иными словами, мы сумели удлинить сумматор, не увеличив T, что было бы невозможно, если бы сумматор был оптимальным.

Теперь докажем, что в оптимальном сумматоре без кольцевого переноса количество разрядов должно быть равно

$$n^{(T)}=T(T+3)/2. \qquad (3.7.2)$$

Доказательство. Очевидно, что в оптимальном сумматоре $t_i=T$ $(i=1, 2, \ldots, m)$. В самом деле, если $t_j<T$, то мы можем увеличить k_j по крайней мере на единицу и при этом будет удовлетворяться неравенство

$$t_j\leqslant T,$$

а остальные t_i не изменяются, т. е. T тоже не изменится.

Полагая $t_i=T$ и $k_i>1$, получаем из (3.7.1):

$$T=(i-1)+(k_i-1)=i+k_i-2 \quad (i=1, 2, \ldots, m),$$

т. е.

$$k_i=T-i+2.$$

Отсюда следует, что оптимальное распределение длин групп имеет вид:

$$k_1=T+1, \quad k_2=T, \quad k_3=T-1, \ldots, \quad k_m=2, \quad m=T,$$

$$n^{(T)}=\sum_{i=1}^{m} k_i=\frac{T(T+3)}{2}.$$

В табл. 3.7.1 показаны оптимальные распределения для ряда значений T.

Таблица 3.7.1

T	$n^{(T)}$	Длины групп	T	$n^{(T)}$	Длины групп
1	2	2	6	27	2, 3, 4, 5, 6, 7
2	5	2, 3	7	35	2, 3, 4, 5, 6, 7, 8
3	9	2, 3, 4	8	44	2, 3, 4, 5, 6, 7, 8, 9
4	14	2, 3, 4, 5	9	54	2, 3, 4, 5, 6, 7, 8, 9, 10
5	20	2, 3, 4, 5, 6	10	65	2, 3, 4, 5, 6, 7, 8, 9, 10, 11

На рис. 3.7.2 в упрощенном виде изображен сумматор с оптимальными длинами групп для случая $T=7$, указаны номера групп и номера разрядов.

Нетрудно показать, что схема ускоренного переноса в нескольких старших группах оптимального сумматора может быть заменена более простой схемой обходного переноса. Говоря точнее, в сумматоре рассматриваемого типа, разбитом на группы с длинами $k_m=2$, $k_{m-1}=3$, ..., $k_i=m-i+2$, ..., $k_1=m+1$, максимальное время распространения переноса не изменится, если в группах с номерами i, удовлетворяющими неравенству

$$i \geqslant (m+2)/2, \qquad (3.7.3)$$

заменить рассматриваемую схему ускоренного переноса на схему обходного переноса.

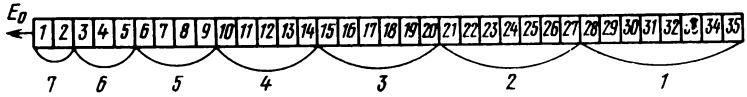


Рис. 3.7.2

В самом деле, до изменения схем переноса сигнал на выходе любой i-й группы вырабатывался не позже, чем в момент времени i. Следовательно, схему ускоренного переноса можно заменить на схему обходного переноса в тех группах, в который после этой замены сигнал переноса, возникший внутри группы, выйдет из нее не позже, чем за такое же время. Группа с номером i содержит $m-i+2$ разряда. Поэтому максимальное время выработки «местного» переноса из i-й группы, содержащей схему обходного переноса, равно $m-i+2$. Следовательно, можно установить схему обходного переноса во всех группах, для которых

$$m-i+2 \leqslant i,$$

откуда и следует (3.7.3).

Таким образом, почти в половине групп можно установить более простые схемы обходного переноса, сохранив то же быстродействие.

В сумматоре с кольцевым переносом максимальному времени распространения переноса соответствует ситуация, при которой перенос возникает в некоторой группе и, пробежав по всему сумматору, заканчивается в этой же группе. Поэтому

$$T = m + \max_i k_i - 1, \qquad (3.7.4)$$

где m — время распространения переноса, возникшего в группе, имеющей максимальную длину, до ее входа, $\max_i k_i - 1$ — время пробега переноса внутри этой группы.

Из (3.7.4) следует, что при оптимальном разбиении все группы должны иметь одинаковую длину $k=n/m$.

В самом деле, если

$$k_j < \max_i k_i,$$

то длину j-й группы можно увеличить до величины $\max_i k_i$, сохранив, как это следует из п. 3.7.4, то же T.

Теперь докажем, что в оптимальном сумматоре с кольцевым переносом при заданном T количество разрядов равно

$$n^{(T)}=[(T+1)^2/4]. \qquad (3.7.5)$$

Доказательство. Пусть $k_1=k_2= \ldots =k_m=k$. Тогда

$$km=n,\ k+m=T+1 \qquad (3.7.6)$$

(второе из этих уравнений следует из (3.7.4)).

Используя теорему Виета о корнях квадратного уравнения, приходим к тому, что k и m должны быть корнями уравнения

$$x^2+(T+1)x+n=0. \qquad (3.7.7)$$

Так как

$$x_{1,2}=\frac{T+1}{2}\pm\frac{T+1}{2}\sqrt{1-\frac{4n}{(T+1)^2}} \qquad (3.7.8)$$

и так как k и m — действительные числа, то подкоренное выражение в (3.7.8) должно быть неотрицательным числом, т. е. должно удовлетворяться неравенство

$$n\leqslant(T+1)^2/4. \qquad (3.7.9)$$

Если T — нечетное, то $(T+1)^2/4$ — целое число и его следует принять в качестве $n^{(T)}$:

$$n^{(T)}=(T+1)^2/4. \qquad (3.7.10)$$

Если T — четное, то максимальное действительное n, при котором удовлетворяется (3.7.9), равно

$$n_{\max}=(T+1)^2/4=T^2/4+2T/4+1/4.$$

Первые два слагаемых в этой сумме — целые числа. Так как $n^{(T)}$ — максимальное целое число, удовлетворяющее неравенству

$$n^{(T)}\leqslant n_{\max},$$

то

$$n^{(T)}=T^2/4+2T/4=T(T+2)/4. \qquad (3.7.11)$$

Объединяя (3.7.10) и (3.7.11), получим (3.7.5).

При $n=n^{(T)}=(T+1)^2/4$ корни уравнения (3.7.7) имеют вид

$$x_{1,2}=(T+1)/2,$$

т. е. при нечетном T оптимальным является следующее разбиение сумматора:

$$m=(T+1)/2,\ k_1=k_2=\ \ldots\ =k_m=(T+1)/2.$$

При

$$n=n^{(T)}=T(T+2)/4,$$

т. е. при четном T, корни уравнения имеют вид

$$x_{1,2}=\frac{T+1}{2}\pm\frac{T+1}{2}\sqrt{\frac{(T+1)^2-T(T+2)}{(T+1)^2}}=\frac{T+1}{2}\pm\frac{1}{2},$$

и так как система уравнений (3.7.6) симметрична относительно k и m, то имеются два оптимальных разбиения сумматора с кольцевым переносом:

$$x_1=m=T/2,\quad x_2=k_1=k_2=\ \ldots\ =k_m=T/2+1$$

и

$$x_1=k_1=k_2=\ \ldots\ =k_m=T/2,\quad x_2=m=T/2+1.$$

Таблица 3.7.2

T	$n^{(T)}$	Длины групп	T	$n^{(T)}$	Длины групп
3	4	2, 2	8	20	4, 4, 4, 4, 4
4	6	2, 2, 2	8	20	5, 5, 5, 5
4	6	3, 3	9	25	5, 5, 5, 5, 5
5	9	3, 3, 3	10	30	5, 5, 5, 5, 5, 5
6	12	3, 3, 3, 3	10	30	6, 6, 6, 6, 6
6	12	4, 4, 4	11	36	6, 6, 6, 6, 6, 6
7	16	4, 4, 4, 4			

В табл. 3.7.2 приведены оптимальные распределения для нескольких значений T.

На рис. 3.7.3 показаны графики полученных зависимостей $n^{(T)}$ от T, построенные по соотношениям (3.7.2), (3.7.5). Для сравнения штриховой линией изображены аналогичные графики для оптимальных сумматоров с однослойными обходными переносами (см. п. 3.3.1). Из рис. 3.7.3 и формул (3.7.2), (3.7.5), (3.3.6), (3.3.7) видно, что при одинаковом быстродействии и

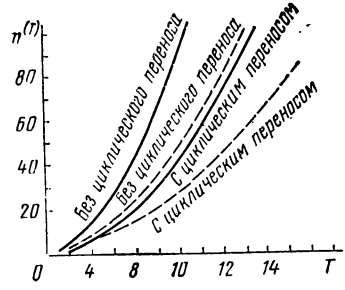


Рис. 3.7.3

прочих равных условиях сумматоры, рассмотренные в этом параграфе, обладают примерно вдвое большей длиной, чем оптимальные сумматоры с однослойными обходными переносами.

4. БЫСТРОДЕЙСТВУЮЩИЕ СИНХРОННЫЕ УМНОЖИТЕЛИ

При решении большинства вычислительных задач требуется выполнять умножение. Задачи, включающие деление, часто могут быть решены при помощи умножений, сложений и вычитаний. Деление можно выполнять, например, путем использования умножений и хранящихся в памяти таблиц обратных величин. Имеется много способов «обойти» деление, т. е. произвести его, не имея специальных устройств, предназначенных для этого.

Умножение нельзя «обойти». Единственная возможность избежать непосредственного вычисления произведения состоит в использовании таблиц умножения. Однако применение этого метода в машинах параллельного действия требует большого объема памяти. Любой программный метод умножения требует выполнения многократных сложений, которые не могут быть произведены так же быстро, как выполняется умножение при использовании ускоренных аппаратных методов. Поэтому очень часто при проектировании арифметического устройства стремятся к тому, чтобы умножение выполнялось с помощью быстрого аппаратного метода.

Все известные синхронные методы ускорения умножения сводятся по существу к раздельному или комплексному использованию следующих трех путей:

1. Уменьшение времени, требуемого для выполнения суммирования частичных произведений, т. е. произведений множимого на отдельные разряды или группы разрядов множителя.

2. Уменьшение количества частичных произведений путем использования предварительно формируемых кратных множимого.

3. Использование вычитания при умножении, позволяющее уменьшить количество дополнительных сумматоров, необходимых для формирования кратных множимого.

В основу такой классификации положены логические и математические идеи, реализация которых приводит к повышению быстродействия. Каждое из указанных направлений воплощено в группе методов ускорения умножения, аппаратные решения которых могут сильно отличаться друг от друга.

В данной главе рассматриваются основные синхронные методы ускоренного выполнения умножения.

4.1. УМЕНЬШЕНИЕ ВРЕМЕНИ СУММИРОВАНИЯ ЧАСТИЧНЫХ ПРОИЗВЕДЕНИЙ

Уменьшить время сложения всех частичных произведений можно тремя способами: ускорять процедуру сложения очередного частичного произведения с суммой предыдущих частичных произведений, начинать прибавление следующего частичного произведения до завершения прибавления предыдущего частичного произведения и, наконец, строить схемы, складывающие текущую сумму частичных произведений сразу с несколькими очередными частичными произведениями.

Основной реализацией *первого способа* является применение ускоренного сумматора для сложения очередного частичного произведения с суммой предыдущих. К этой группе методов при желании можно также отнести использование хорошо известных схем умножения, позволяющих совмещать во времени сложение со сдвигом множимого (рис. 7.7.1, *б*, *г*).

Второй способ воплощен в методе умножения, при котором используется так называемый сумматор с запоминанием переносов (carry save adder) [1, п. 4.3.2]. Идея этого метода состоит в формировании текущей суммы очередного частичного произведения с суммой предыдущих частичных произведений в виде двухрядного кода, т. е. в виде двух чисел. В предельном случае одно из этих двух чисел составлено из поразрядных сумм s, а другое — из поразрядных переносов e. Принцип работы такого устройства иллюстрируется упрощенной структурной схемой, приведенной на рис. 4.1.1. Во время каждого цикла в комбинационном сумматоре См склады-

вается очередное частичное произведение П с суммой предыдущих частичных произведений. В конце цикла значения сигналов e и s запоминаются в регистрах Рг3, Рг4, а в начале следующего цикла передаются в Рг1, Рг2. Выигрыш в скорости достигается благодаря тому, что отсутствует процесс распространения переносов в сумматоре См, время срабатывания которого в данном случае уменьшается, очевидно, до времени срабатывания одного одноразрядного сумматора.

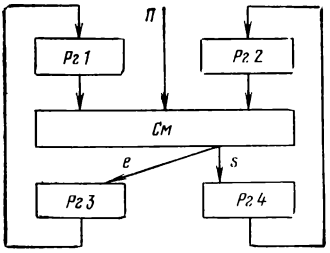


Рис. 4.1.1

При более экономичных (но и более медленных) вариантах весь сумматор разбивается не на отдельные разряды, а на сравнительно короткие, q-разрядные ($1<q<n$) сумматоры. При этом первое из двух указанных чисел составляется из выходов сумм этих сумматоров, а второе число, содержащее в q раз меньше значащих цифр, — из сигналов переноса, вырабатываемых в q-разрядных сумматорах (по одному сигналу на каждый такой сумматор).

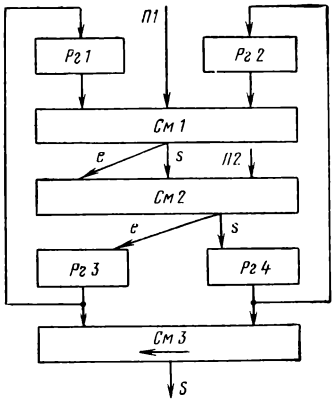


Рис. 4.1.2

В любом из этих вариантов после получения двухрядного кода окончательного произведения выполняется преобразование двухрядного кода в однорядный, т. е. обычное сложение. Его можно осуществить на обычном параллельном, а еще лучше — на ускоренном параллельном сумматоре. В устройстве, показанном на рис. 4.1.1, заключительное сложение может выполняться на сумматоре См, который должен в этом случае после проведения всех циклов сложений с частичными произведениями «перестроиться» с режима преобразования трехрядного кода (т. е. трех чисел) в двухрядный — на режим преобразования двухрядного кода в однорядный. Вместо этого заключительное сложение можно выполнять на отдель-

ном сумматоре, который следует установить после регистров Рг3, Рг4. Такой вариант устройства будет показан на рис. 4.1.2.

Третий способ ускорения процесса суммирования частичных произведений предполагает введение в арифметическое устройство дополнительных сумматоров. Количество m частичных произведений, которые одновременно прибавляются к текущей сумме частичных произведений, может быть различным. Операция умножения при этом состоит из ряда циклов, во время каждого из которых прибавляется m новых частичных произведений. Обычно при этом используется и метод запоминания переносов, обеспечивающий дополнительное ускорение. На рис. 4.1.2 в качестве примера показано, как можно на каждом цикле к текущей сумме частичных произведений прибавлять еще два ($m=2$) частичных произведений П1 и П2 и при этом формировать сумму в виде двухрядного кода. Заключительное сложение в этой схеме делается сумматором См3, который целесообразно сделать ускоренным.

В предельном варианте умножитель складывает сразу все частичные произведения. Умножение в этом случае производится за один цикл. Умножители такого вида называются *однотактными* (матричными, одновременными, пирамидами сумматоров и т. д.). Анализ и описание таких устройств приведены в § 4.4, 6.3.

4.2. ПРЕДВАРИТЕЛЬНОЕ ФОРМИРОВАНИЕ КРАТНЫХ МНОЖИМОГО

При этом методе умножения множитель разбивается на группы по q разрядов и может иметься еще одна группа, содержащая меньше чем q разрядов. Каждая группа разрядов расшифровывается независимо от других как обычное двоичное число. Если условно считать, что запятая расположена справа от группы, то при расшифровке вырабатывается одно из чисел (сигналов)

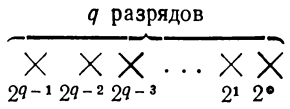
$$0,\ 1,\ 2,\ 3,\ \ldots,\ 2^q-1$$

и в качестве очередного частичного произведения соответственно используется сдвинутое необходимым образом относительно текущей суммы частичных произведений одно из 2^q кратных множимого C:

$$0,\ 1C,\ 2C,\ 3C,\ \ldots,\ (2^q-1)C.$$

Правило расшифровки одной группы для случая $q=3$ показано в табл. 4.2.1.

Таким образом, отдельным разрядам группы при расшифровке соответствуют естественные веса:



Обозначим количество групп, на которые разбит множитель (или, что то же, — количество частичных произ-

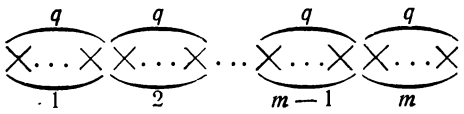
Таблица 4.2.1

Разряды группы	Кратное	Разряды группы	Кратное	Разряды группы	Кратное	Разряды группы	Кратное
000	0	010	$2C$	100	$4C$	110	$6C$
001	$1C$	011	$3C$	101	$5C$	111	$7C$

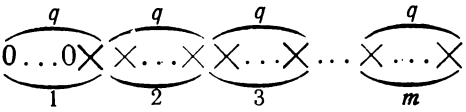
ведений), через m. Соотношение между n, q и m имеет следующий вид:

$$m=\left]\frac{n}{q}\right[^{*)}. \qquad (4.2.1)$$

Крайними случаями разбиения множителя на группы является случай, когда все группы «полные»:



(сверху указаны длины групп, снизу — их номера), и случай, когда неполная группа содержит только один разряд:



*) $]d[$ — наименьшее целое число, не меньшее чем d.

(неполная группа показана слева, но она может стоять и в любом другом месте множителя). Остальные случаи являются промежуточными между этими двумя. Сказанное иллюстрируется двойным неравенством

$$(m-1)q+1 \leqslant n \leqslant mq,$$

правая и левая границы которого соответствуют двум указанным крайним случаям разбиения. Соотношение (4.2.1) справедливо во всех случаях.

Те кратные KC, для которых K нечетно и больше единицы, предварительно формируются при помощи вспомогательных сумматоров, количество которых, очевидно, равно $2^{q-1}-1$. Остальные кратные множимого получаются путем дополнительных сдвигов из C и из кратных, выработанных вспомогательными сумматорами. Таким образом, при $q=2$ требуется сумматор для формирования $3C$, при $q=3$ нужны сумматоры $3C$, $5C$, $7C$, при $q=4$ — сумматоры $3C$, $5C$, $7C$, $9C$, $11C$, $13C$, $15C$ и т. д.

Одна группа вспомогательных сумматоров может вычислять кратные KC, используя в качестве исходных чисел величины $\pm C2^t$ (t целое), получаемые простым сдвигом множимого. Например: $3C=C+2C$, $5C=4C+C$, $7C=8C-C$, $9C=8C+C$, $15C=16C-C$ и т. д. Другие вспомогательные сумматоры вынуждены пользоваться выходами сумматоров первой группы. Например: $13C=8C+5C$ или $13C=16C-3C$ и т. д.

Очевидно, что при использовании данного метода расшифровка групп может начинаться (т. е. умножение может начинаться) с любого конца множителя. Возможна также одновременная расшифровка нескольких групп разрядов множителя; такая необходимость возникает, если в умножителе складываются одновременно несколько частичных произведений, как это было, например, показано на рис. 4.1.2.

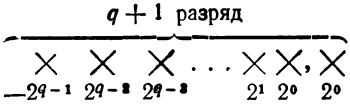
Можно считать, что при использовании данного метода время умножения уменьшается примерно в q раз (мы предполагаем, что частичные произведения по очереди складываются в параллельном накапливающем сумматоре, не учитываем возможность наличия одной неполной группы разрядов множителя и пренебрегаем временем, затрачиваемым вспомогательными сумматорами на формирование кратных множимого в начале операции). Это замечание относится и к методам, рассмотренным в § 4.3.

4.3. ИСПОЛЬЗОВАНИЕ ОТРИЦАТЕЛЬНЫХ ЧАСТИЧНЫХ ПРОИЗВЕДЕНИЙ

Этот метод обычно применяется в сочетании с предыдущим. Ниже рассмотрены два варианта использования вычитаний при умножении. В обоих случаях каждое частичное произведение соответствует умножению множимого на группу из q разрядов множителя. Основное отличие состоит в том, что при первом варианте расшифровка может начаться с любого конца множителя, а во втором варианте умножение производится с младших разрядов множителя. Существуют и другие отличия.

4.3.1. МЕТОД УМНОЖЕНИЯ НА ГРУППУ ИЗ q РАЗРЯДОВ МНОЖИТЕЛЯ С РАСШИФРОВКОЙ $q+1$ РАЗРЯДА МНОЖИТЕЛЯ

При использовании этого метода ускорения умножения очередная группа из q разрядов множителя дешифрируется вместе со старшим разрядом соседней младшей группы, который рассматривается как дополнительный младший разряд. Всем разрядам группы, кроме старшего и дополнительного, приписываются те же естественные веса, что и в методе, описанном в § 4.2. Старшему разряду группы приписывается вес -2^{q-1}, а дополнительному разряду — вес 1:



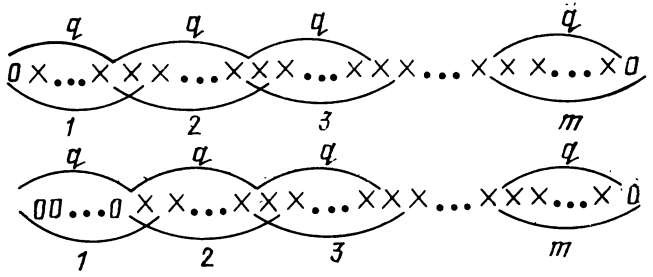
Таким образом, старший разряд каждой группы дешифрируется дважды: один раз — как старший разряд этой группы, второй — как дополнительный разряд соседней старшей группы.

Старший разряд самой старшей группы и дополнительный младший разряд самой младшей группы должны всегда быть нулями. Поэтому должно удовлетворяться двойное неравенство

$$(m-1)q \leqslant n \leqslant mq-1,$$

границы которого поясняются следующим образом (каждая нижняя дуга указывает одновременно дешиф-

рируемые разряды):



Количество групп цифр множителя (количество частичных произведений) при данном методе равно

$$m = \left]\frac{n+1}{q}\right[.$$

Сумма весов ненулевых цифр группы определяет коэффициент K в кратном KC, выбираемом в качестве

Таблица 4.3.1

Разряды группы	K	Разряды группы	K	Разряды группы	K	Разряды группы	K
0000	0	0100	2	1000	—4	1100	—2
0001	1	0101	3	1001	—3	1101	—1
0010	1	0110	3	1010	—3	1110	—1
0011	2	0111	4	1011	—2	1111	0

очередного частичного произведения. Величина K при этом принимает одно из значений

$$0, \pm 1, \pm 2, \pm 3, \ldots, \pm 2^{q-1},$$

т. е. количество кратных увеличивается на одно по сравнению с предыдущим методом, но зато количество дополнительных сумматоров для формирования кратных множимого уменьшается до $2^{q-2}-1$.

Правила расшифровки иллюстрируются табл. 4.3.1 ($q=3$).

4.3.2. МЕТОД УМНОЖЕНИЯ С МЛАДШИХ РАЗРЯДОВ МНОЖИТЕЛЯ

При расшифровке очередной группы разрядов множителя по этому методу производится анализ q очередных разрядов и одной двоичной цифры «переноса» из предыдущей (соседней младшей) группы в данную. Обозначим эту цифру буквой e. Веса, приписываемые отдельным разрядам, имеют при этом следующие значения:

$$\overbrace{\underset{2^{q-1}}{\times}\ \underset{2^{q-2}}{\times}\ \underset{2^{q-3}}{\times}\ \ldots\ \underset{2^{1}}{\times}\ \underset{2^{0}}{\times}}^{q\text{ разрядов}}\ \underset{2^{0}}{e}$$

Иными словами, разряды множителя имеют естественные веса, а вес цифры e равен единице. Коэффициент K при кратном множимого KC, используемом в качестве очередного частичного произведения, выбирается из 2^q значений

$$0,\ \pm 1,\ \pm 2,\ \pm 3,\ \ldots,\ \pm(2^{q-1}-1),\ 2^{q-1}$$

таким образом, чтобы выполнялось соотношение

$$K+2^q e'=S,$$

где e' — выбираемая одновременно с K цифра «переноса» в следующую (соседнюю старшую) группу разрядов множителя ($e'=0$ или $e'=1$), S — сумма весов ненулевых разрядов дешифрируемой группы цифр множителя вместе с весом ненулевой цифры e. Количество дополнительных сумматоров для формирования кратных множимого при этом, как и в предыдущем методе, равно $2^{q-2}-1$.

Одно из возможных правил расшифровки одной группы разрядов множителя для случая $q=3$ показано в табл. 4.3.2.

В работе [2] был предложен другой вариант расшифровки для случая $q=3$ (см. табл. 4.3.3).

Как видно, количество используемых кратных в обоих случаях равно 8, но вместо кратного — $2C$ появляется кратное $6C$. Можно предложить и другие варианты правила расшифровки.

Среди возможных вариантов некоторым преимуществом обладают те, в которых при равенстве нулю старшего разряда дешифрируемой группы вырабатывается

Таблица 4.3.2

Разряды группы	e	K	e'	Разряды группы	e	K	e'
000	0	0	0	000	1	1	0
001	0	1	0	001	1	2	0
010	0	2	0	010	1	3	0
011	0	3	0	011	1	4	0
100	0	4	0	100	1	—3	1
101	0	—3	1	101	1	—2	1
110	0	—2	1	110	1	—1	1
111	0	—1	1	111	1	0	1

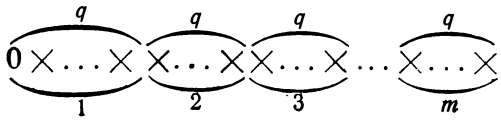
Таблица 4.3.3

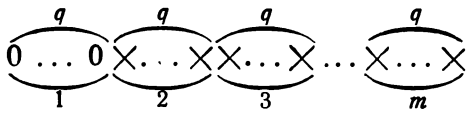
Разряды группы	e	K	e'	Разряды группы	e	K	e'
000	0	0	0	000	1	1	0
001	0	1	0	001	1	2	0
010	0	2	0	110	1	3	0
011	0	3	0	011	1	4	0
100	0	4	0	100	1	—3	1
101	0	—3	1	101	1	6	0
110	0	6	0	110	1	—1	1
111	0	—1	1	111	1	0	1

$e'=0$. Таковы, в частности, оба варианта, показанные в табл. 4.3.2, 4.3.3. Если при этом группы образовать таким образом, чтобы слева в старшей группе всегда стоял нуль, то можно тем самым избежать переноса из этой группы, т. е. избежать необходимости формирования еще одного частичного произведения. Поэтому параметры n, m, q должны быть связаны соотношением

$$(m-1)q \leqslant n \leqslant mq-1.$$

Правая и левая границы неравенства соответствуют следующим вариантам разбиения множителя на группы разрядов:





Как и в предыдущем методе, количество групп цифр множителя (количество частичных произведений) равно

$$m = \left]\frac{n+1}{q}\right[.$$

В табл. 4.3.4 сопоставлены параметры трех последних рассмотренных методов ускорения, при использовании которых множимое умножается сразу на группу, состоящую из q разрядов множителя. Цифры I, II, III обозначают: I — метод с использованием только положительных кратных множимого (§ 4.2), II, III — методы с использованием не только положительных, но и отрицательных кратных (соответственно п. 4.3.1 и 4.3.2).

Таблица 4.3.4

Характеристика метода	I	II	III
Требуемое количество кратных множимого	2^q	$2^q + 1$	2^q
Требуемое количество дополнительных сумматоров	$2^{q-1} - 1$	$2^{q-2} - 1$	$2^{q-2} - 1$
Соотношение между n, m, q	$m = \left]\frac{n}{q}\right[$	$m = \left]\frac{n+1}{q}\right[$	$m = \left]\frac{n+1}{q}\right[$

Для иллюстрации рассмотренных методов покажем, как при $q = 3$ выполняется всеми тремя методами умножение некоторого множимого C на множитель A, равный 0,101100011 ($n = 9$).

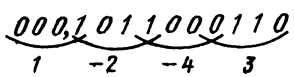
В первом случае разбиение на группы и расшифровка будут такие:

$$0,\ \underbrace{101}_{5}\ \underbrace{100}_{4}\ \underbrace{011}_{3}$$

и умножение фактически выполнится за 3 цикла следующим образом:

$$CA = 3C2^{-9} + 4C2^{-6} + 5C2^{-3}.$$

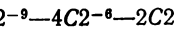
Во втором случае разбиение может быть таким:



и циклов будет четыре:

$$CA = 3C2^{-9} - 4C2^{-6} - 2C2^{-3} + 1C2^{0}.$$

В третьем случае разобьем множитель следующим образом:



и циклов будет тоже четыре

$$CA = 3C2^{-9} + 4C2^{-6} - 3C2^{-3} + 1C2^{0}.$$

Сравнивая достоинства и недостатки всех трех методов, следует отметить, что

при n, кратном q, умножение по методам II, III требует выполнения количества циклов большего на единицу, чем в случае использования метода I;

для реализации методов II, III требуется значительно меньшее количество дополнительных сумматоров, чем для реализации метода I;

количество кратных множимого для методов I, III на единицу меньше, чем для метода II;

методы I и II в отличие от метода III подходят для применения в матричных умножителях (см. § 4.4), так как при их использовании можно расшифровывать все группы разрядов множителя одновременно.

Таким образом, каждый из трех методов обладает некоторыми преимуществами, выгодно отличающими его перед хотя бы одним из двух остальных методов.

В качестве еще одной иллюстрации этой группы методов рассмотрим возможную аппаратную реализацию умножения с параметрами $n=8$, $q=3$, реализующего метод III (рис. 4.3.1). В качестве «прототипа» устройства выбран вариант «а» схемы неускоренного умножения (см. рис. 7.7.1,*а*, с. 337). В нашем случае (рис. 4.3.1) первый цикл умножения начинается с расшифровки трех младших разрядов множителя A, находящегося в регистре-сдвигателе. Если среди этих трех цифр есть хотя бы одна единица, то дешифратор Дш вырабатывает один из семи управляющих сигналов +1, —1, +2, —2, +3, —3, +4 (количество сигналов указано в соответствующих местах на рис. 4.3.1 в скобках) и сигнал переноса e' (сигнал e в первом цикле равен нулю). Выработанный управляющий сигнал пропускает сквозь коммутатор Км в 10-разрядный накапливающий сумматор-сдвигатель в качестве первого частичного произведения соответствующее кратное множимого C из регистра РгС или из сумматора См3C, который еще до начала первого цикла вычисляет и

хранит 10-разрядное утроенное множимое $3C$ (пронумеруем разряды этого числа от 1 до 10). Если, например, три младших разряда множителя равны 101, то вырабатывается и запоминается в триггере Т сигнал $e'=1$ и вырабатывается управляющий сигнал —3, который пропускает сквозь коммутатор Км число $3C$ в обратном коде (один из десяти разрядов коммутатора показан на рис. 4.3.2). Частичное

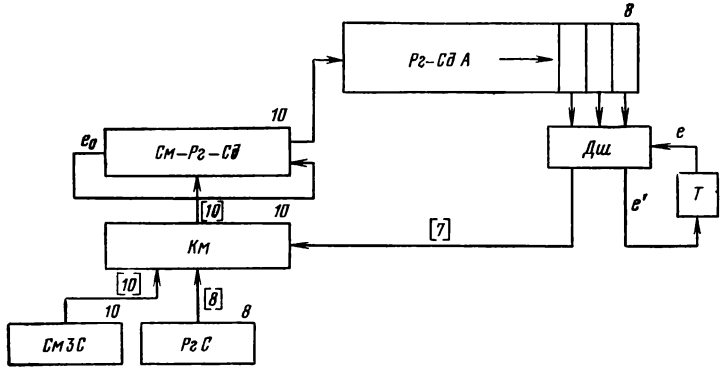


Рис. 4.3.1

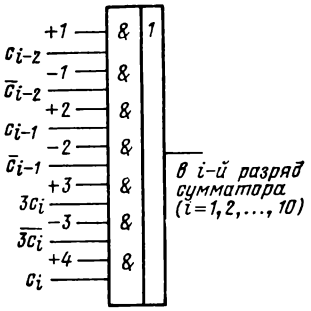


Рис. 4.3.2

произведение прибавляется к содержимому сумматора (при выполнении первого цикла это содержимое равно нулю) и сумма запоминается. В конце цикла производится сдвиг вправо на 3 разряда в сумматоре и в регистре множителя. При этом три младшие разряда произведения переходят из сумматора в освобождающиеся старшие разряды регистра множителя. Аналогично выполняются второй и третий циклы, но в конце третьего цикла сдвиги не производятся. В сумматоре при сложениях используется цепь кольцевого переноса e_0. Из других деталей можно отметить, что схема на рис. 4.3.2 слегка меняется для крайних разрядов коммутатора, так как нужно учесть, что $c_i=0$ при $i=9,10$; $c_{i-1}=0$ при $i=1,10$; $c_{i-2}=0$ при $i=1,2$.

4.4. АНАЛИЗ И МЕТОДИКА ПОСТРОЕНИЯ ОДНОТАКТНЫХ УМНОЖИТЕЛЕЙ

4.4.1. КЛАССИФИКАЦИЯ ОДНОТАКТНЫХ УМНОЖИТЕЛЕЙ

В начале этой главы уже говорилось, что современная вычислительная машина затрачивает значительную часть своего времени на выполнение умножения. В ря-

де разработок новых машин арифметическое устройство освобождается от множества тривиальных операций, что приводит к увеличению доли времени, затрачиваемого на умножение и деление. Арифметическое устройство такой машины тратит на эти две операции примерно половину своего времени. Тем не менее объем аппаратуры, имеющейся в машине для этих операций, редко бывает велик. В крупных вычислительных системах и ЦВМ часто возникает ситуация, в которой для улучшения общих экономических показателей системы, содержащей большую память, развитое внешнее оборудование и управление, выгодно увеличить затраты на умножение и деление даже выше того уровня, при котором прирост затрат дает равный прирост скоростей умножения и деления. Однотактные умножители (ОУ) относятся к средствам ускорения такого рода.

Основная особенность ОУ состоит в том, что выполнение умножения в нем представляет собой единый непрерывный сложный переходный процесс одновременного сложения всех частичных произведений, не разделяемый на более мелкие интервалы времени тактирующими сигналами, как это делается в умножителе любого другого типа.

Допустим, что нужно сформировать полное $2n$-разрядное произведение двух n-разрядных сомножителей и что в умножителе не используются дополнительные кратные множимого (об использовании таких кратных говорилось в § 4.2, 4.3). Тогда общее количество цифр во всех слагаемых (частичных произведениях) равно n^2. Позже мы увидим, что при изменении величины n количество аппаратуры в ОУ того или иного типа изменяется приблизительно пропорционально n^2. Поэтому применение ОУ названо в работе [3] методом второго порядка ускорения умножения.

Можно предложить [4] классификацию ОУ, основанную на разделении их на матричные ОУ и составные ОУ (рис. 4.4.1). Матричные ОУ [4—29] содержат приблизительно n^2 однотипных элементов (например, одноразрядных двоичных сумматоров), соединенных между собой тем или иным способом и складывающих все частичные произведения. Если используются дополнительные кратные множимого (см. § 4.2, 4.3), то количество суммирующих элементов соответственно уменьшается и в состав устройства включается дешифратор

множителя и узел формирования необходимых кратных множимого.

Матричные ОУ можно в свою очередь разделить на однородные и неоднородные. Однородные матричные ОУ относятся к классу устройств, часто называемых *итеративными сетями* (iterative arrays) и представляющих собой множество одинаковых элементов, образующих регулярную сеть [30]. Благодаря однородности структу-

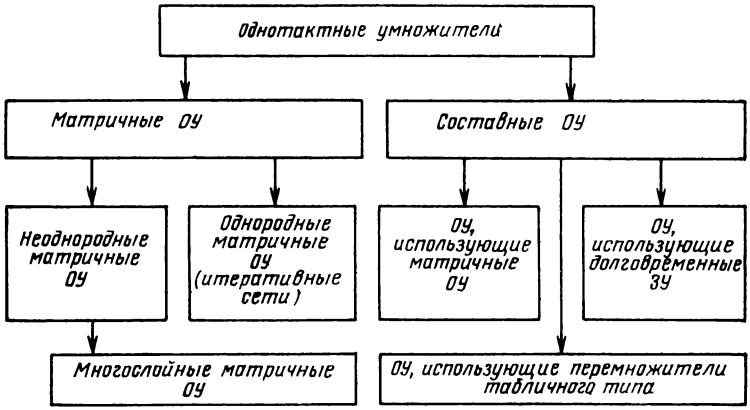


Рис. 4.4.1

ры и соблюдению принципа «ближнедействия» элементов применение таких устройств представляется весьма перспективным в больших и средних вычислительных машинах высокой производительности, создаваемых на базе современной технологии изготовления интегральных схем. В настоящее время разработано большое количество итеративных сетей, представляющих собой множительные, делительные [22—24, 31—34], суммирующие и другие вычислительные узлы; среди последних можно отметить схемы для вычисления квадратного корня [34—36], для умножения и деления двоичного числа на константу [37], для возведения в квадрат [38], для перевода числа из двоичной системы в десятичную [37] и т. п. Однородные матричные ОУ [1, 4—16, 22—24] также содержат однотипные элементы, соединенные регулярным образом. В неоднородных матричных ОУ [1, 4, 13—21] эта регулярность отсутствует.

Большую группу неоднородных матричных ОУ образуют так называемые многослойные построения [1, 4,

13—21], отличающиеся малым временем образования двухрядного кода произведения (см. п. 4.4.4).

Составные ОУ синтезируются из ОУ меньшей разрядности: матричных [4, 27], табличных или представляющих собой небольшие долговременные запоминающие устройства [25, 26], вычисляющих произведения отдельных частей множимого и множителя. Соединение между собой «элементарных» ОУ в составном ОУ может выполняться тем или иным способом и, в частности, так, как соединяются одноразрядные сумматоры в однородных или многослойных ОУ; полученные таким образом ОУ можно назвать макрооднородными, макромногослойными и т. д.

В настоящее время ОУ все чаще находят применение в арифметических устройствах цифровых машин. В ряде случаев одно и то же матричное устройство используется для выполнения не только умножения, но и деления [22—24]. По мере создания больших вычислительных систем, в которых удельный вес арифметического устройства невелик по сравнению с остальной аппаратурой, применение ОУ будет, вероятно, возрастать. Во многих случаях используются также неполные матрицы, складывающие одновременно несколько, но не все, частичные произведения [27, 39, 40]. Об устройствах такого рода речь уже шла (§ 4.1).

Однотактные умножители могут строиться, вообще говоря, из различных элементарных ячеек. Широкое распространение получили ОУ, основной типовой ячейкой в которых является одноразрядный двоичный сумматор на потенциальных элементах. Ниже мы будем иметь в виду ОУ, содержащие такие сумматоры.

В настоящее время известно большое количество схем одноразрядного сумматора. В последующих пунктах этой главы будет показано, что от выбора схемы одноразрядного сумматора может существенным образом зависеть быстродействие всего ОУ. Поэтому целесообразно рассмотреть возможные построения сумматора с точки зрения их влияния на свойства ОУ.

В арифметических устройствах, не использующих матричные схемы, время задержки сигнала в цепях переноса одноразрядных сумматоров намного сильнее влияет на быстродействие устройства, чем время задержки в цепях суммы. В устройствах, использующих матрицы сумматоров, общее время операции умножения

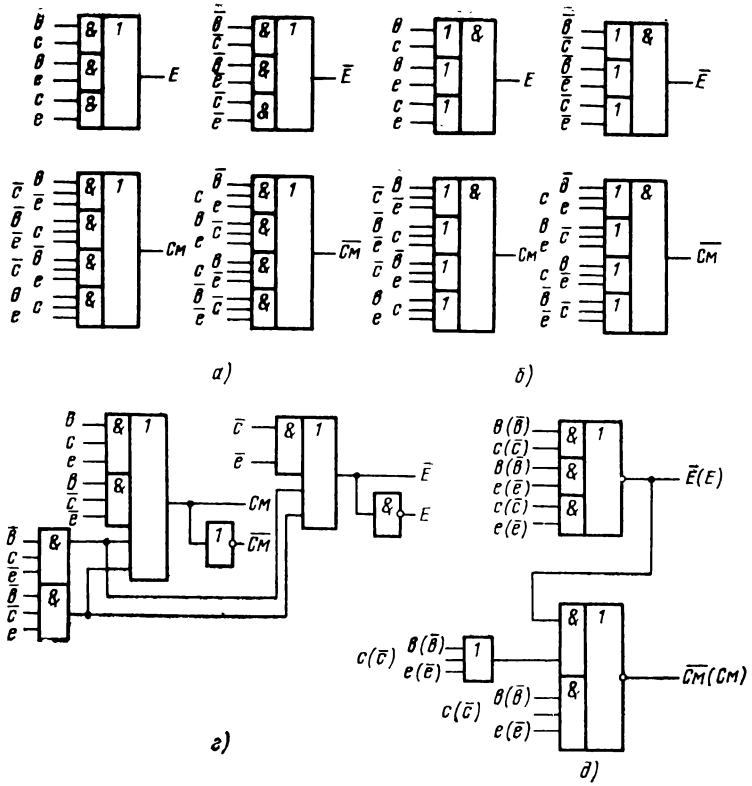
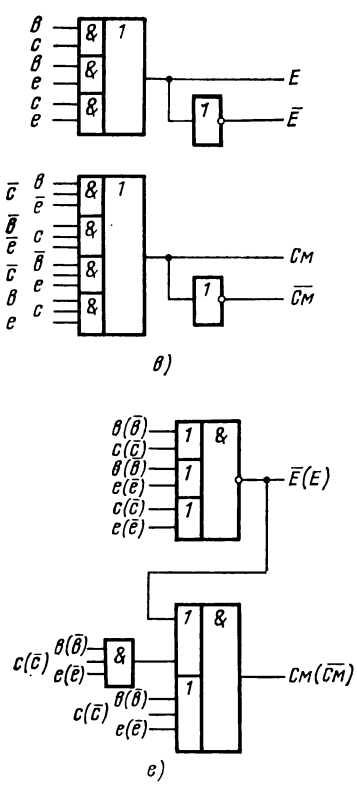


Рис. 4.4.2

(и других операций, в выполнении которых участвует матрица) существенным образом зависит и от быстродействия цепей суммы одноразрядных сумматоров. Для данной работы удобно ввести в рассмотрение отношение максимального времени задержки в цепи суммы к максимальному времени задержки в цепи переноса одноразрядного сумматора.

На рис. 4.4.2 показаны некоторые схемы одноразрядных комбинационных сумматоров. Из их рассмотрения видно, что максимальная задержка в цепи переноса τ_E, вообще говоря, не равна максимальной задержке в цепи суммы $\tau_{с\,м}$. Можно считать, что в общем случае $\tau_{с\,м}=k\tau_E$, где k — некоторая константа. Однако для большинства практических применений можно ограничиться рассмотрением двух основных случаев: $k=1$



и $k=2$. В первом из них мы будем называть сумматор «однотактным», во втором случае — «двухтактным».

Задержку τ_E будем называть «тактом» и считать, что она одинакова для всех сумматоров (в данном случае термин «такт» не имеет отношения к названию «однотактные умножители»). Величина $k=1$, как правило, соответствует сумматорам (рис. 4.4.2,*а*—*г*), в которых сигналы суммы и переноса вырабатываются независимо друг от друга, хотя входная часть этих двух схем может быть общей (рис. 4.4.2,*г*). Такие сумматоры требуют парафазных входных сигналов, и, следовательно, если мы хотим использовать их в матрице, то выходы этих сумматоров тоже должны быть парафазными. Случай $k=2$ соответствует сумматорам, в которых сигнал суммы вырабатывается при помощи сигнала переноса (рис. 4.4.2,*д*, *е*). Такие сумматоры не требуют парафазных входных сигналов и поэтому, как правило, более экономичны. Код сигнала переноса в этих схемах инверсен коду слагаемых, а в схеме 4.4.2,*д* инверсен по отношению ко входам и сигнал суммы. Схема двоичного одноразрядного сумматора обладает так называемым свойством *самодвойственности*, состоящим в том, что инвертирование всех входных сигналов приводит к инвертированию выходных сигналов. Это свойство отражено на рис. 4.4.2,*д*, *е* надписями в скобках и в дальнейшем будет использовано.

Основными достоинствами ОУ являются высокое быстродействие, простота схем управления, легкость настройки и выявления дефектов. Успехи микроэлектро-

ники и, в частности, развитие техники больших интегральных схем позволяют успешно бороться с основным недостатком этих устройств — большим объемом аппаратуры. Опыт развития советской и зарубежной вычислительной техники за последние годы показывает, что использование ОУ становится важным средством повышения производительности вычислительных средств.

4.4.2. АНАЛИЗ ПРОЦЕССОВ СУММИРОВАНИЯ ЧАСТИЧНЫХ ПРОИЗВЕДЕНИЙ В ОДНОРОДНЫХ МАТРИЧНЫХ ОУ

На рис. 4.4.3 в упрощенной форме показаны схемы 12-матричных ОУ четырех типов. Каждый кружок на рисунке изображает одноразрядный двоичный сумматор, сверху вниз направлены сигналы суммы, справа налево, справа вверх налево и справа вниз налево направлены сигналы переноса. В каждой из этих схем выдержан единый принцип построения: сумматоры, относящиеся к одному разряду, образуют вертикальную цепочку, кроме того регулярный характер имеют связи между соседними разрядами. Приведенные структуры являются, таким образом, итеративными сетями.

Часто с целью уменьшения количества аппаратуры совмещают применение рассматриваемого метода ускорения умножения с описанным ранее методом ускорения, состоящим в расшифровке групп, содержащих по q разрядов множителя, и в использовании предварительно вычисляемых кратных множимого. Поэтому на рис. 4.4.3 для каждого из четырех типов матриц приведены три схемы, соответствующие вариантам умножения, при которых каждое частичное произведение получается путем умножения множимого на q разрядов множителя ($q=1, 2, 3$). При этом предполагается, что при $q=2$ используются кратные 0, $1C$, $2C$, $3C$ (т. е. в арифметическом устройстве имеется дополнительный сумматор, вычисляющий утроенное множимое), а при $q=3$ используются кратные 0, $1C$, $2C$, $3C$, $4C$, $5C$, $6C$, $7C$ (т. е. имеются сумматоры для вычисления величин $3C$, $5C$, $7C$). При $q=2$ или $q=3$ могли бы быть использованы и другие кратные: например, при умножении на две цифры можно использовать кратные 0, $1C$, $2C$, $-1C$, $-2C$, а при умножении на три цифры — кратные 0, $1C$, $2C$, $3C$, $4C$, $-1C$, $-2C$, $-3C$, $-4C$. При этом умень-

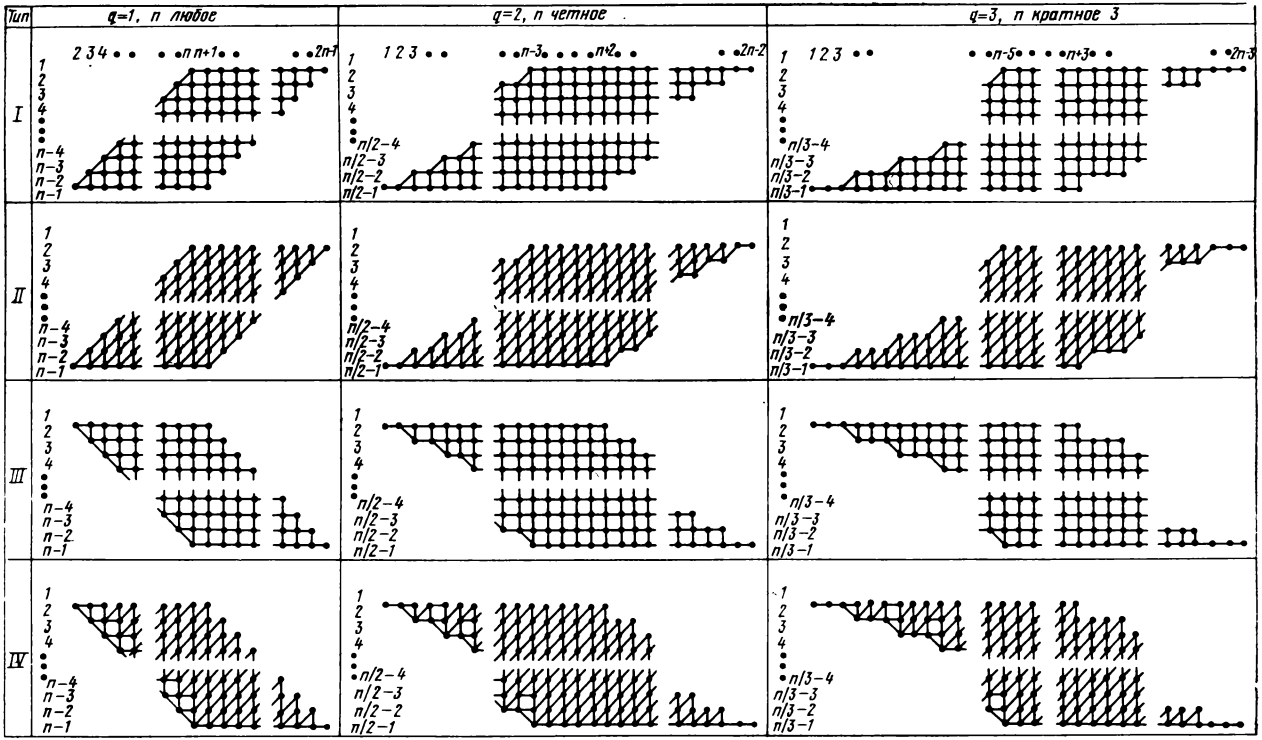

Тип
q=1, n любое
q=2, n четное
q=3, n кратное 3
I
II
III
IV
1 2 3 4 n n+1 2n-1
n-4 n-3 n-2 n-1
1 2 3 n-3 n+2 2n-2
n/2-4 n/2-3 n/2-2 n/2-1
1 2 3 n-5 n+3 2n-3
n/3-4 n/3-3 n/3-2 n/3-1

Рис. 4.4.3

шается количество дополнительных сумматоров, но появляется аппаратура, необходимая для ввода отрицательных частичных произведений. В основном же структура матрицы и ее свойства сохраняются.

Матричные умножители на рис. 4.4.3 изображены, как отмечено, в несколько упрощенном виде — не показаны, в частности, схемы, формирующие частичные произведения, не показаны входы, через которые поступают

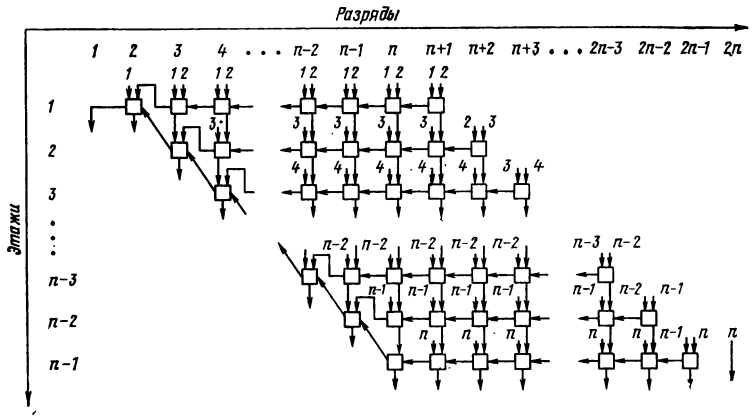


Рис. 4.4.4

частичные произведения, одинаковым образом изображены и сумматоры, и полусумматоры и т. д. Характер упрощений можно увидеть, сравнив схему типа III, показанную на рис. 4.4.3 в первой колонке, с этой же схемой, приведенной на рис. 4.4.4 в более подробном виде. В устройстве, показанном на рис. 4.4.4, складываются n частичных произведений, по n разрядов в каждом. Номера частичных произведений проставлены около стрелок, указывающих места ввода отдельных разрядов частичных произведений. Одноразрядные сумматоры и полусумматоры изображены в виде квадратов.

Схемы во второй и третьей колонках на рис. 4.4.3 показаны для n, кратного соответственно двум и трем. Если это условие не выполняется, то количество этажей сумматоров получается равным соответственно величинам $n/2-1$ и $n/3-1$, округленным до ближайшего большого целого. Структура устройства и его свойства при этом практически не изменяются.

Ни на рис. 4.4.3, ни на рис. 4.4.4 не показан узел формирования частичных произведений. В простейшем случае ($q=1$) этот узел содержит n^2 двухвходовых схем И, формирующих n^2 сигналов $a_i c_j$ ($i=1, 2, \ldots, n$; $j=1, 2, \ldots, n$), где a_i, c_j — разряды сомножителей. Выработанные (одновременно) сигналы являются двоичными цифрами, стоящими в разрядах частичных произведений (n произведений по n разрядов). Расставленные друг под другом в соответствии с весами разрядов, эти частичные произведения образуют матрицу, почти совпадающую по форме с матрицами, стоящими на рис. 4.4.3 в колонке $q=1$. Теперь каждый кружок изображает одну цифру одного частичного произведения. (Можно сказать, что при этом на рисунке не показано только младшее частичное произведение, занимающее разряды с $(n+1)$-го по $2n$-й включительно. Чтобы ОУ «заработало», остается только наложить матрицу частичных произведений на матрицу сумматоров, т. е. соединить выходы схем И с соответствующими свободными входами сумматоров. При этом, очевидно, цифры, находящиеся в одной колонке матрицы чисел, можно подавать на входы соответствующей колонки сумматоров в любом порядке, т. е. цифры в колонке можно менять местами.

Если $q>1$, то количество частичных произведений уменьшается в q раз (при n, кратном q), а каждое из них удлиняется с n до $n+q$ разрядов. При q, равном 2 или 3, ОУ приобретает форму, показанную в соответствующем месте рис. 4.4.3. Пример полной схемы ОУ типа II (ее можно также считать схемой типа IV) для случая $n=6$, $q=2$ приведен на рис. 4.4.5, на котором, кроме самой матрицы сумматоров, показаны также дополнительный сумматор, вырабатывающий утроенное множимое, и схемы формирования частичных произведений. Пустые прямоугольники на этом рисунке изображают одноразрядные сумматоры и полусумматоры. Устройство типа I ($q=2$) описано в работе [1], с. 451. Там же имеются полные схемы ОУ типа I, $q=1$ (с. 438), типа III, $q=1$ (с. 445), типа IV, $q=1$ (с. 448) и др.

Из рис. 4.4.3—4.4.5 видно, что на выходах ОУ формируется готовое $2n$-разрядное произведение.

На этих рисунках не показаны дополнительные инверторы, которые нужны для «фазировки» сигналов при использовании сумматоров, изображенных на

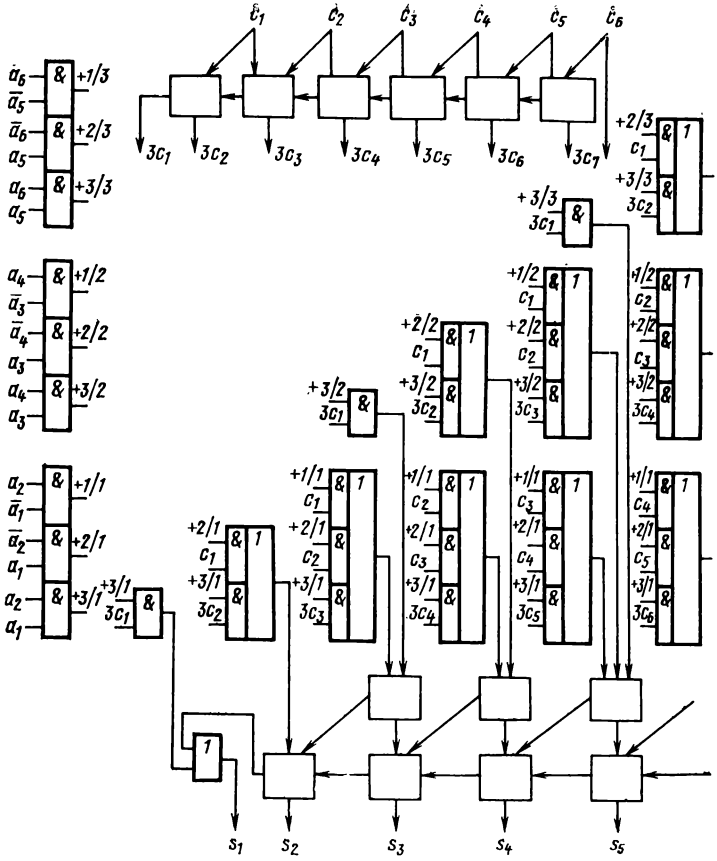


Рис. 4.4.5

рис. 4.4.2,*д*, *е*, так как в матрицах такого типа могут встречаться не только прямые, но и инверсные сигналы.

Рассмотрим условия, при которых возникает подобная необходимость. Для этого отметим прежде всего, что любая из структур, приведенных на рис. 4.4.3, может быть разделена на части, каждая из которых состоит из трех или четырех одноразрядных сумматоров, соединенных между собой «по кольцу» (рис. 4.4.6). Рассмотрим вначале кольцо из четырех сумматоров. Нетрудно убедиться в том, что, как показано на рис. 4.4.6,*а*, *б*, два верхних сумматора такого кольца должны быть построены по одной и той же схеме, соответствующей либо

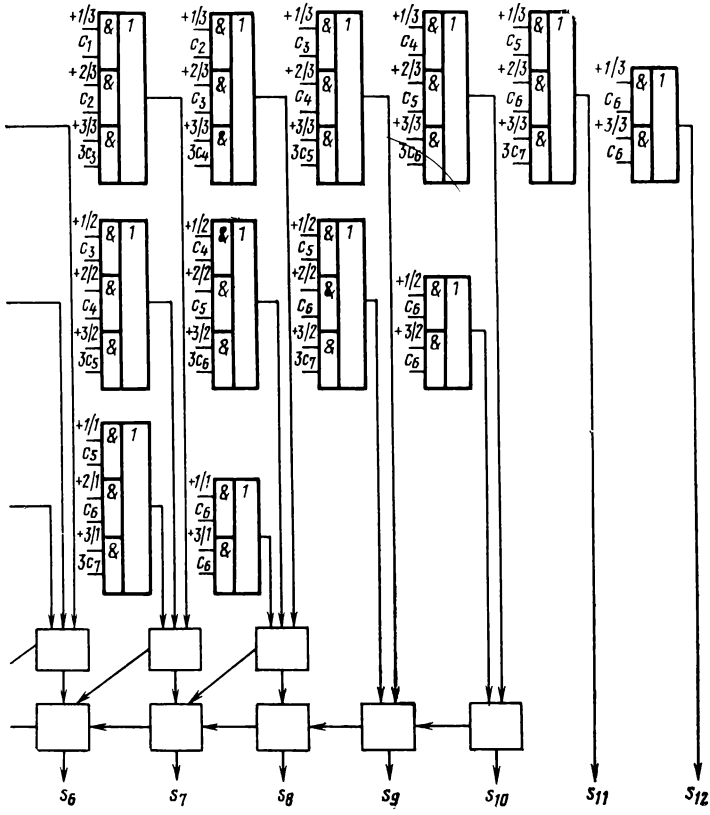


Рис. 4.4.5

рис. 4.4.2,*д*, либо рис. 4.4.2,*е* (для простоты сумматоры на рис. 4.4.6 обозначаются соответственно буквами *д* или *е*), а каждый из двух нижних сумматоров может быть построен по любой из этих двух схем. Таким образом, такое кольцо может быть построено на сумматорах либо типа *д*, либо типа *е*, либо на тех и других.

Однако кольцо из трех сумматоров может быть построено без дополнительных инверторов только в том случае, если в распоряжении разработчика имеются схемы типа *е* (см. рис. 4.4.6,*в*). Если же имеются только сумматоры типа *д*, то для «фазировки» в кольце должен

быть установлен дополнительный инвертор (рис. 4.4.6,*г*). На рисунке этот инвертор включен в цепь суммы, но, очевидно, вместо этого он может быть включен в любую из двух цепей переноса, образующих «треугольник» вместе с указанным сигналом суммы.

Следует отметить, что соображение относительно невозможности построения матрицы на сумматорах типа *д* без использования дополнительных инверторов относится не только к структурам, показанным на рис. 4.4.3, но также к любой матрице, в схеме которой можно выделить хотя бы один такой замкнутый контур, образованный одноразрядными сумматорами и соединяющими их сигналами суммы и переноса, в котором количество сумматоров было бы нечетным. Очевидно также, что такие контуры имеются практически в любой матрице. Однако их количество различно в разных типах матриц.

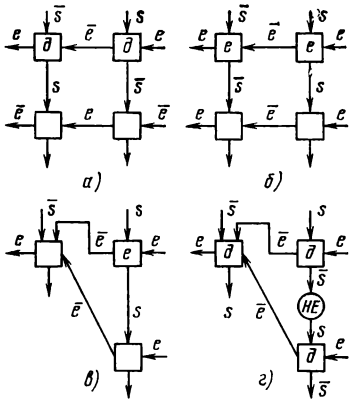


Рис. 4.4.6

Из рис. 4.4.3 можно заключить, что матрицы типа I и III содержат минимальное количество таких контуров («треугольников»). Для случая $q=2$ это количество равно $(n-4)/2$, а для матриц II и IV — соответственно $(3n-6)/2$ и $(5n-20)/2$; при $q=3$ матрицы I и III содержат по $(n-6)/3$ «треугольников» по сравнению с $(5n-18)/7$ и $(7n-36)/3$ для матриц II и IV. При $q=1$ матрицы I, II, III, IV, содержат соответственно $n-2$, $n-1$, $n-2$, $3n-8$ «треугольников».

Очевидно, что дополнительные инверторы должны устанавливаться таким образом, чтобы, во-первых, не нарушалась «фазировка» сигналов в смежных контурах и, во-вторых, по возможности не снижалось быстродействие матрицы. Непосредственный анализ показывает, что первое из этих двух требований может быть легко удовлетворено для всех схем, показанных на рис. 4.4.3, таким образом, чтобы количество дополнительных инверторов было равно числу «треугольников». Однако

далее будет показано, что для выполнения второго требования может понадобиться увеличение количества дополнительных инверторов.

Перейдем к анализу процессов установления решения в однородных матричных ОУ [9].

Выше уже отмечалось, что суммирование частичных произведений в ОУ представляет собой в общем случае сложный переходный процесс. Во время протекания этого процесса сигнал в некоторой точке устройства может несколько раз переключаться с одного уровня на другой, прежде чем принять окончательное, правильное значение. Наличие пересекающихся потоков сигналов ограничивает возможности дополнительного ускорения. В некоторых арифметических устройствах приходится также считаться и с тем, что кроме потерь времени на переходный процесс, возникающий после подачи частичных произведений, имеются потери времени на затухание процессов в матрице после снятия всех входных сигналов. Время затухания меньше времени суммирования, но и оно может быть, как будет показано в п. 4.4.3, достаточно большим.

Время выполнения умножения в матричном умножителе зависит от схемы матрицы, от параметров используемых элементов и от значений сомножителей. В настоящее время не существует простых методов точной оценки максимального времени работы в различных ОУ. Поэтому оценивать быстродействие матриц мы будет «сверху», предполагая, что переходный процесс протекает «наихудшим» образом. Сравнение скоростных свойств матриц, изображенных на рис. 4.4.3, можно провести по методике, основанной на двух следующих допущениях.

1) Будем считать, что все частичные произведения подаются в матрицу одновременно в момент времени $t=0$. (В действительности это условие не выполняется из-за разброса параметров элементов предыдущих узлов, задержек сигналов на проводниках и т. п. Кроме того, для матриц, использующих дополнительные кратные множимого, которые вырабатываются дополнительными сумматорами, это условие может не выполняться, так как сама выработка кратных отнимает некоторое время и младшие разряды кратных появляются раньше, чем старшие. В реальном устройстве указанные дополнительные сумматоры целесообразно делать ускорен-

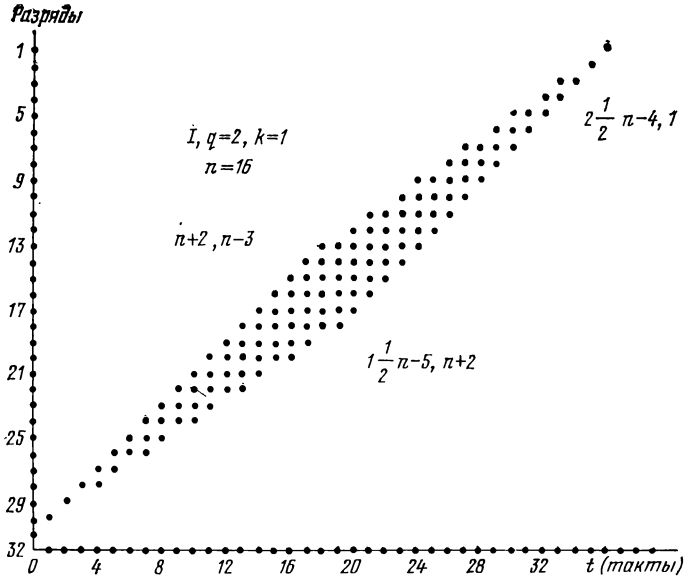


Рис. 4.4.7

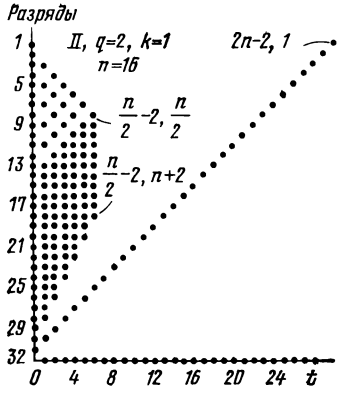


Рис. 4.4.8

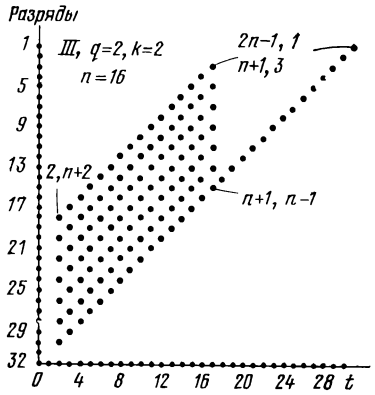


Рис. 4.4.9

ными, но лишь в такой степени, чтобы полное время работы матриц зависело только от ее структуры, но не от дополнительных сумматоров.)

2) Будем также считать, что из каждого одноразрядного «двухтактного» («однотактного») сумматора

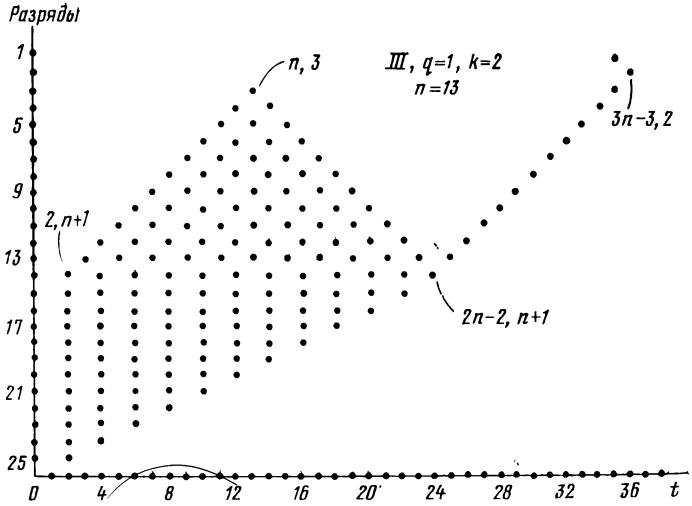


Рис. 4.4.10

правильный сигнал переноса выдается через «такт», а сумма — через два «такта» (через один «такт») после подачи на входы сумматора всех трех слагаемых.

По этой методике можно построить 24 временные диаграммы, соответствующие 12 структурам матриц, показанных на рис. 4.4.3, и двум значениям величины k ($k=1$ и $k=2$). На основе диаграмм были выведены общие выражения для координат характерных точек этих диаграмм.

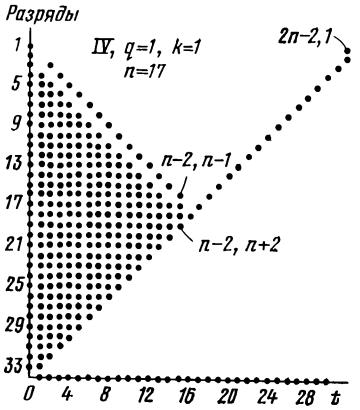


Рис. 4.4.11

На рис. 4.4.7—4.4.11 приведены пять из этих диаграмм. Точками отмечены моменты установления правильных сигналов на выходах сумм всех сумматоров матриц. Для наглядности показаны диаграммы, соответствующие конкретным значениям n, однако приведенные

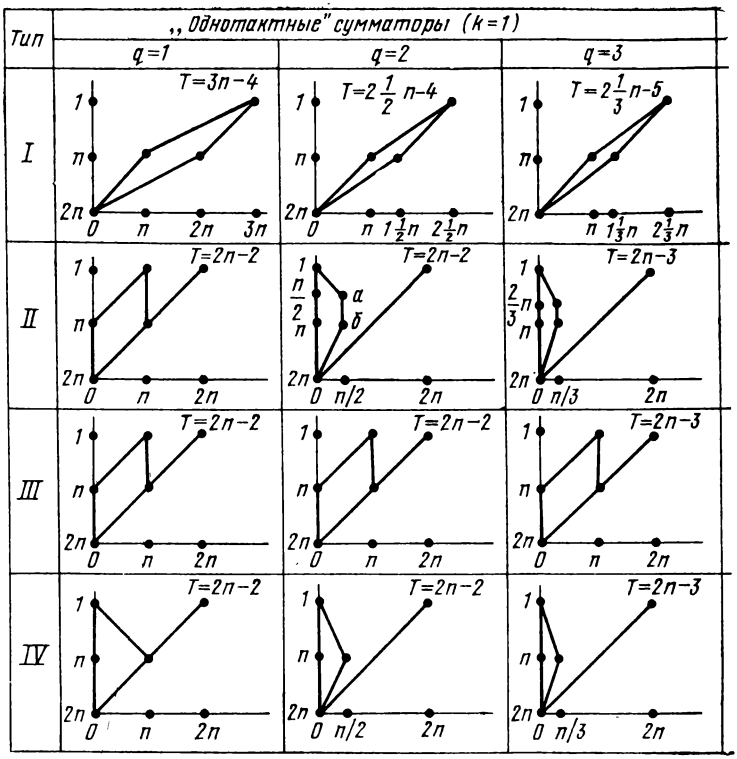


Рис. 4.4.12

на рисунках точные формулы для координат характерчых точек диаграмм соответствуют любому значению n (с тем, однако, ограничением, что n считается кратным q).

Полученные диаграммы соответствуют гипотетическому «наихудшему» с точки зрения быстродействия переходному процессу и позволяют оценивать «сверху» реальные процессы, происходящие в однородных матричных ОУ.

Так, например, из рис. 4.4.10 видно, что

1) перенос «медленно» распространяется вдоль каждого этажа до $(n+1)$-го разряда (пробег переносов задерживается поступающими сверху сигналами суммы), после чего пробег продолжается «быстро»;

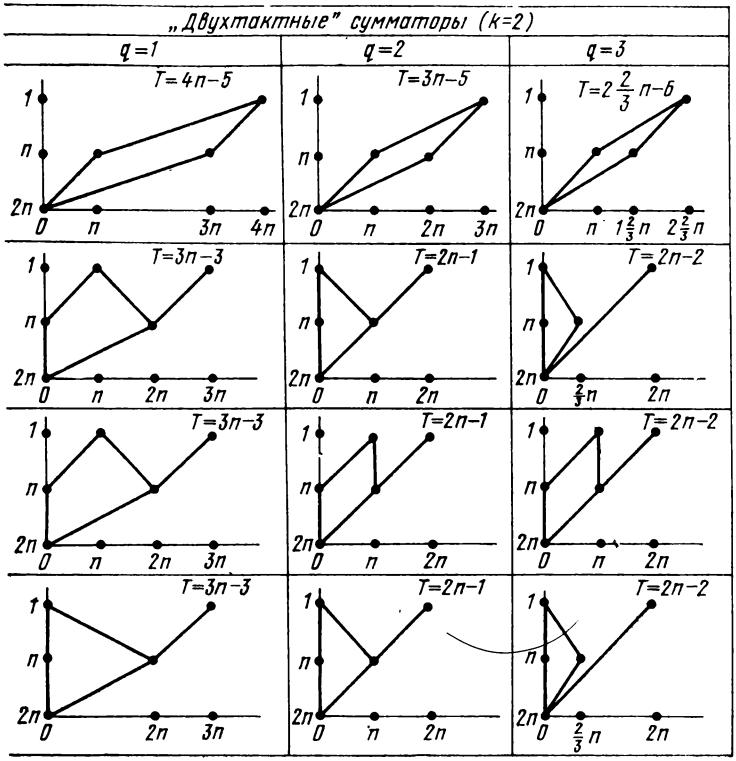


Рис. 4.4.12

2) полное время работы этой матрицы равно $3n$—3 «тактам»;

3) после $2n$—3 «тактов» переходный процесс затухает во всей схеме, за исключением сумматоров, расположенных во 2 ... n разрядах вдоль нижнего края. Это означает, что, построив схему ускоренного пробега переносов вдоль указанных сумматоров, можно сократить полное время работы устройства с величины $3n$—3 до величины порядка $2n$—2 «тактов». Построение схем ускорения в других местах матрицы с целью дальнейшего сокращения полного времени потребовало бы слишком больших аппаратных затрат. Из рисунка видно, что $t=2n$—3 «тактов» — это время формирования двухрядного кода произведения, т. е. время формирова-

ния двух чисел, сумма которых равна искомому произведению.

Все 24 временные диаграммы сведены в показанную на рис. 4.4.12 таблицу. Для того, чтобы выделить основные особенности диаграмм, были сделаны некоторые упрощения: участки диаграмм, которые соответствуют процессу установления сигналов в основной массе сумматоров, показаны в виде контуров; процессы пробега переносов вдоль нижнего края матрицы изображены отдельной линией. На осях координат показаны (с точностью до нескольких «тактов» и разрядов) координаты (время и номер разряда) характерных точек контура *), полное время работы матрицы T указано точно.

Рассмотрение полученных временных диаграмм позволяет сделать следующие выводы о свойствах и особенностях рассматриваемых однородных матричных ОУ.

1) Изменение типа структуры матрицы, типа одноразрядного сумматора или параметра q может существенно изменить временную диаграмму.

2) В ряде случаев можно ускорять пробег переносов вдоль всего нижнего края матрицы (например, в схемах II и IV при $q=3$), в других случаях это можно сделать только в старших n разрядах (в матрицах II, III, IV при $q=1$) и, наконец, для структуры I организовать ускорение практически невозможно.

3) Матрицы типа I при прочих равных условиях обладают меньшей скоростью, чем любые другие. Матрицы I практически невозможно ускорить, так как для этого пришлось бы ускорять пробеги во всех этажах.

Приведенные диаграммы позволяют произвести выбор оптимального варианта однородного матричного множительного устройства и схемы одноразрядного сумматора. Однако при этом должен быть также учтен ряд других обстоятельств, которые не могут быть проанализированы в общем виде из-за большого количества возможных ситуаций. Основные из этих обстоятельств следующие:

1) Используемая система элементов. Тип, быстродействие, надежность, нагрузочная способность, габариты, стоимость различных элементов, имеющихся в распо-

---

*) Можно считать, что координаты характерных точек указаны для $n \to \infty$.

ряжении проектировщика, сам факт наличия или отсутствия того или иного элемента могут влиять на выбор схемы матрицы.

2) Унификация конструктивных блоков, например больших интегральных схем, на которые «разбивается» матрица и все арифметическое устройство.

3) Необходимость введения в некоторых случаях в схему матрицы дополнительных инверторов для «фазировки» сигналов.

4) Использование матрицы при других операциях. Это обстоятельство имеет особенно большое значение. Если, например, на матрице должно выполняться деление при помощи метода без восстановления остатка, то матрицы типа I и II не могут быть применены, так как они не дают возможности осуществить сдвиг очередного остатка. Для реализации такого метода деления может быть использована матрица III или IV. Первая из них наилучшим образом подходит в том случае, когда очередной остаток должен получаться в однорядном коде, а другая — если остаток формируется в двухрядном коде. Если предполагается осуществлять деление при помощи одного из методов, использующих умножение (§ 5.3), то пригоден, вообще говоря, любой тип матрицы.

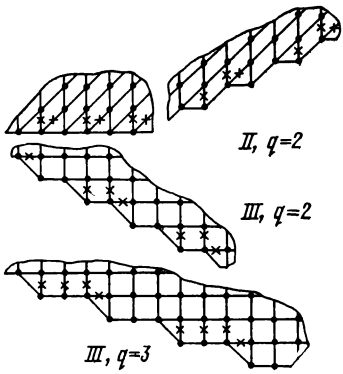


Рис. 4.4.13

Полученные временные диаграммы однородных матричных ОУ позволяют выбрать такие места установки дополнительных инверторов (необходимых для «фазировки» сигналов в случае использования одноразрядных сумматоров типа *д*), чтобы быстродействие матрицы не изменилось. Сопоставление схем и диаграмм позволяет сделать вывод, что во всех схемах, кроме трех (II, $q=2$; III, $q=2$, 3), можно установить инверторы таким образом, чтобы их количество было равно количеству «треугольных» контуров и при этом задержки, вносимые инверторами, не отражались на величине T. В трех отмеченных случаях дополнительные инверторы следует

устанавливать так, как это показано в упрощенном виде крестиками на рис. 4.4.13. Количество инверторов при этом должно превышать число «треугольников» (см. табл. 4.4.1), зато инверторы не располагаются на «критических» путях распространения сигналов.

Таблица 4.4.1

Тип матрицы	Количество „треугольных" контуров	Количество дополнительных инверторов
I, $q=1$	$n-2$	$n-2$
I, $q=2$	$\frac{n-4}{2}$	$\frac{n-4}{2}$
I, $q=3$	$\frac{n-6}{3}$	$\frac{n-6}{3}$
II, $q=1$	$n-1$	$n-1$
II, $q=2$	$\frac{3n-6}{2}$	$\left\lfloor \frac{7n-18}{4} \right\rfloor$
II, $q=3$	$\frac{5n-18}{7}$	$\frac{5n-18}{7}$
III, $q=1$	$n-2$	$n-2$
III, $q=2$	$\frac{n-4}{2}$	$\left[\frac{3n-12}{4}\right]$
III, $q=3$	$\frac{n-6}{3}$	$\frac{2n-12}{3}-2\left(\frac{n}{6}-\left[\frac{n}{6}\right]\right)$
IV, $q=1$	$3n-8$	$3n-8$
IV, $q=2$	$\frac{5n-20}{2}$	$\frac{5n-20}{2}$
IV, $q=3$	$\frac{7n-36}{3}$	$\frac{7n-36}{3}$

Из табл. 4.4.1 и временных диаграмм видно, в частности, что при $q=2$ и использовании сумматоров типа *д* матрица III, обладая таким же быстродействием, как матрицы II, IV, и лучшим быстродействием, чем схема I, содержит значительно меньше дополнительных инверторов. Матрицы III впервые были описаны, по-видимому, в работе [10].

4.4.3. АНАЛИЗ ПРОЦЕССОВ «ЗАТУХАНИЯ» В ОДНОРОДНЫХ МАТРИЧНЫХ ОУ

Если на матрице после умножения выполняется другая — более простая и более быстрая — операция (например, сложение), и если при этом пауза между концом умножения и началом следующей операции меньше, чем время «затухания» сигналов в матрице, которое имеет место после установления всех входных сигналов в матрице в нулевое состояние, то время затухания должно учитываться при расчете времени выполнения следующей операции. При необходимости должны применяться аппаратные средства защиты от такого влияния. Например, можно при помощи специального сигнала «отсоединять» выходы верхних этажей матрицы от нижнего этажа, в котором могут производиться операции сложения-вычитания.

Поэтому изучение процессов затухания в матричных ОУ, как и процессов установления решения, имеет определенную ценность. Задача анализа процессов затухания, по-видимому, впервые была поставлена в работе [41].

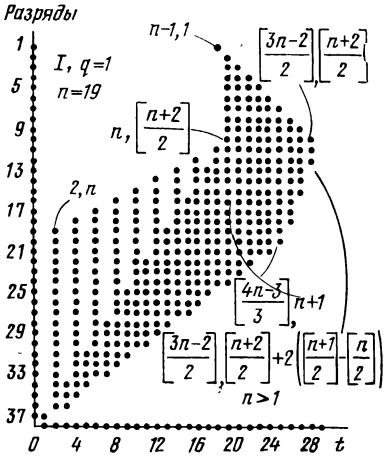


Рис. 4.4.14

Процессы затухания могут быть рассмотрены по методике, основанной на следующих допущениях.

1) Все частичные произведения становятся равными нулю одновременно в момент $t=0$.

2) Из каждого одноразрядного двоичного сумматора правильный сигнал переноса (нуль) выдается через «такт» после подачи на вход двух правильных (т. е. нулевых) сигналов слагаемых, а сигнал суммы (тоже нулевой) вырабатывается через «такт» после подачи всех трех (нулевых) слагаемых.

Правила эти едины для «однотактных» и «двухтактных» сумматоров. Справедливость данного утверждения следует из рассмотрения схем, приведенных на рис. 4.4.2.

По указанной методике были построены временные диаграммы для всех матриц, изображенных на рис. 4.4.3, и были выведены точные формулы для координат характерных точек этих диаграмм [41]. Одна из 12 диаграмм приведена на рис. 4.4.14. Она соответствует конкретному значению $n=19$, но формулы справедливы для любого $n>1$.

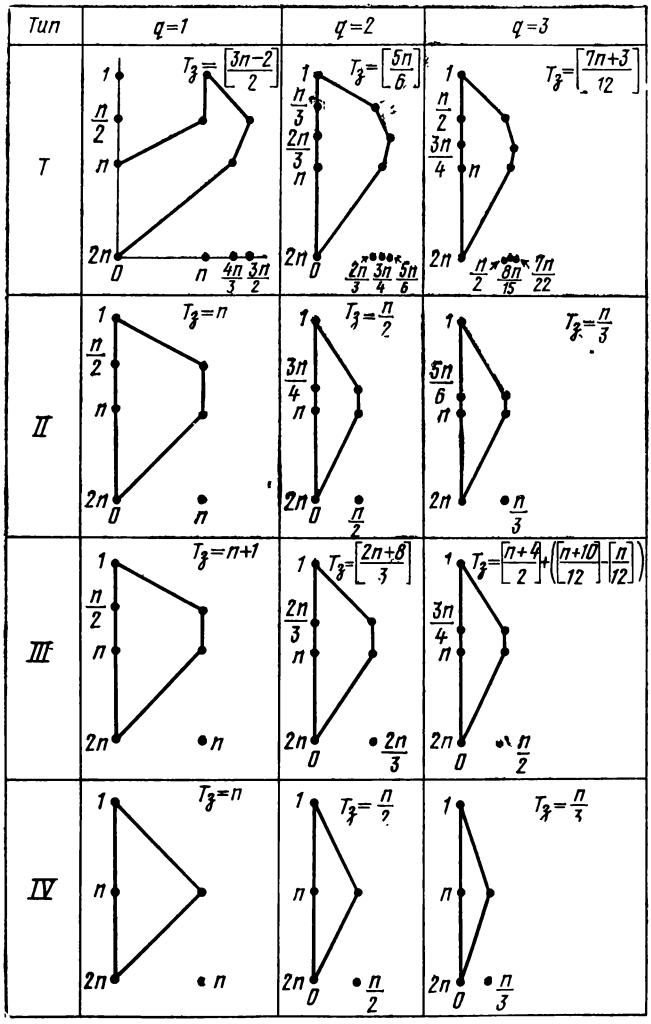


Рис. 4.4.15

Полученные диаграммы в упрощенном виде сведены на рис. 4.4.15 в общую таблицу, как это было сделано раньше для процессов установления решения. На рисунке указаны точные значения полного времени затухания $T_з$ и приближенные значения координат характерных точек (как и выше, точные формулы заменены приближенными лишь для наглядности).

Из рассмотрения временных диаграмм видно, что затухание в матрицах типа I происходит медленнее, чем для остальных трех типов.

При $q=1$ матрицы II—IV обладают практически одинаковой скоростью затухания, а при $q=2$, 3 в матрицах II и IV затухание происходит несколько быстрее, чем в матрице типа III. Видно также, что процесс затухания практически не может быть ускорен.

Диаграммы затухания, как и диаграммы установления решения, могут быть использованы для выбора типа однородной матрицы и схемы одноразрядного сумматора. Они позволяют выбрать необходимые меры защиты в тех случаях, когда наличие затухания приводит к снижению быстродействия арифметического устройства.

Следует отметить, что принятое в этой главе за единицу измерения времени время одного «такта» хотя и было определено как максимальное время задержки соответствующего элемента, в действительности может быть приравнено к величине, лишь немного превышающей среднее время срабатывания такого элемента. Это объясняется тем, что в длинной цепочке последовательно соединенных элементов вероятность большого отклонения суммарной задержки от величины, равной сумме средних задержек, весьма незначительна. Эта особенность «однопроходных» (однотактных, безцикловых) арифметических и иных устройств вообще и матриц сумматоров в частности обеспечивает, по существу, дополнительное ускорение их работы и повышение надежности [1, с. 442—443].

4.4.4. МНОГОСЛОЙНЫЕ ОДНОТАКТНЫЕ МАТРИЧНЫЕ УМНОЖИТЕЛИ

1°. Будем называть многослойным такой матричный ОУ, построенный на одноразрядных двоичных сумматорах и полусумматорах, в котором имеется хотя бы один одноразрядный сумматор или полусумматор, на входы которого поступают выходные сигналы суммы из двух

или трех других одноразрядных сумматоров. Иными словами, в многослойном матричном ОУ одноразрядные сумматоры, относящиеся к данному разряду устройства, не вытянуты в цепочку, как это было, в частности, в однородных структурах, рассмотренных в п. 4.4.1—4.4.3, а образуют некоторое ветвящееся «дерево».

Можно предложить очень много структур ОУ, которые будут соответствовать приведенному только что определению. Мы рассмотрим только некоторые, наиболее известные построения.

Разберем некоторые общие свойства, которыми обладает большинство многослойных построений, рассматриваемых в этой книге [46].

Будем считать, что ОУ состоит из ряда последовательно соединенных слоев: выходные сигналы i-го слоя являются входными сигналами $(i+1)$-го слоя ($i=1, 2, \ldots$). На входы каждого i-го слоя из предыдущего, $(i-1)$-го слоя поступает n_{i-1} чисел. Аппаратура i-го слоя обрабатывает (складывает) по некоторым одинаковым для каждого слоя правилам эти числа и вырабатывает n_i других чисел, поступающих на следующий, $(i+1)$-й слой. Сумма чисел, имеющихся на входах или на выходах любого слоя, равна одной и той же величине — произведению исходных сомножителей A и C.

В дальнейшем мы увидим, что во многих известных и рассмотренных ниже многослойных ОУ функция $n_i=f(n_{i-1})$ удовлетворяет следующим пяти условиям.

Условие 1. Любому целому положительному n_{i-1} соответствует единственное целое положительное n_i, т. е. функция $n_i=f(n_{i-1})$ — однозначная.

Условие 2.

$$n_i<n_{i-1} \text{ при } n_{i-1}>2 \quad (i=1, 2, \ldots)$$

(т. е. если на входы любого слоя поступает более двух чисел, то на выходах этого слоя получается меньше чисел, чем поступило на входы).

Условие 3.

$$0\leqslant f(n_{i-1}+1)-f(n_{i-1}) \quad (n_{i-1}=1, 2, \ldots)$$

(т. е. количество n_i чисел на выходах слоя не может убывать при увеличении количества n_{i-1} чисел на входах слоя)*).

*) Предполагается, что изменение количества чисел на входах слоя сопровождается соответствующей перестройкой аппаратуры слоя.

Условие 4.

$$f(n_{i-1}+1)-f(n_{i-1})\leqslant 1 \quad (n_{i-1}=1, 2, \ldots).$$

Условие 5. Для любого числа N, как бы велико оно ни было, существует такое n'_{i-1}, что $n_i>N$ при всех $n_{i-1}>n'_{i-1}$. Иными словами, $\lim\limits_{n_{i-1}\to\infty} n_i=\infty$.

В дальнейшем мы увидим, что указанные условия выполняются не только в многослойных ОУ, построенных из одноразрядных сумматоров, но и в ОУ, построенных из параллельных счетчиков *) (см. п. 4°, 5°, 9°, 10°, 11°, 12° данного параграфа), короткоразрядных параллельных сумматоров (п. 6.3.1) или из специальных преобразователей N-рядного кода в двухрядный (п. 6.3.2).

Рассмотрим несколько общих следствий, вытекающих из выполнения указанных пяти условий и относящихся, таким образом, ко всем только что перечисленным многослойным ОУ (а также ко всем ОУ, которые еще могут быть предложены и разработаны и в которых тоже выполняются перечисленные условия).

Следствие 1. Из условий 1, 2 следует, что количества n_1, n_2, ... чисел на выходах последовательных слоев образуют последовательность, члены которой строго убывают до первого из них, удовлетворяющего неравенству

$$n_i\leqslant 2.$$

Обозначим номер этого члена буквой M. Будем считать, что ОУ содержит M слоев **). Таким образом, исходные $n=n_0$ чисел последовательно преобразовываются («сворачиваются») M раз так, что удовлетворяются неравенства

$$n_0>n_1>n_2> \ldots >n_{M-1}>n_M;$$
$$n_i>2 \text{ при } i<M;\ n_M\leqslant 2.$$

---

*) Параллельным счетчиком (k, m) называется узел, складывающий k двоичных цифр одинакового веса (допустим, вес равен 1) и вырабатывающий их сумму в виде m-разрядного двоичного числа. Так как максимальное значение такого числа равно 2^m-1, то m есть наименьшее целое решение неравенства

$$2^m-1\geqslant k.$$

Частным случаем параллельного счетчика является счетчик (3,2), который представляет собой хорошо известный одноразрядный двоичный сумматор.

**) Мы полагаем, что $n_0>2$.

Следствие 2. Из условия 1 и следствия 1 следует, что любому n соответствует единственное M, т. е. что функция $M(n)$ — однозначная.

Следствие 3. Функция $M(n)$ — неубывающая, т. е. $M(n') \leqslant M(n'')$, если $n'' > n'$. Это следует из следствия 1 и условия 3. В самом деле, при увеличении n не может уменьшиться ни одна из величин n_1, n_2, ... и, следовательно, не может уменьшиться M.

Следствие 4. При увеличении n на единицу количество слоев либо не изменяется, либо увеличивается на единицу, т. е. $M(n+1) - M(n) \leqslant 1$. Это следует из следствия 1 и условий 3, 4.

Следствие 5. Количество слоев M может быть сколь угодно большим:

$$\lim_{n \to \infty} M = \infty.$$

(из следствия 1 и условия 5).

Следствие 6. Любому целому неотрицательному M соответствует некоторое максимальное значение n, которое будем обозначать в виде $n^{(M)}$. Иными словами, $n^{(M)}$ — это максимальное количество чисел, которые можно свернуть в двухрядный код при помощи M слоев. Следствие 6 следует из следствий 4 и 5*). Отметим, что $n^{(0)} = 2$.

Следствие 7. Из следствий 3 и 6 следует, что

$$n^{(T-1)} < n^{(T)} \quad (T = 1, 2, \ldots).$$

Таким образом, числа $n^{(0)}$, $n^{(1)}$, $n^{(2)}$, ... образуют возрастающую последовательность.

Следствие 8. Из следствий 3, 6, 7 следует, что при $n \geqslant 3$ $M(n) = T$, если $n^{(T-1)} < n \leqslant n^{(T)}$ $(T = 1, 2, \ldots)$. Это значит, что количество слоев ОУ при любом $n \geqslant 3$ можно определить, выбрав из последовательности чисел $n^{(0)}$, $n^{(1)}$, ... такие два соседних числа $n^{(T-1)}$, $n^{(T)}$, которые удовлетворяют приведенному неравенству. Тогда $M(n) = T$. Эту методику мы будем часто использовать в даль-

---

*) Выполнение условия 5 необходимо, так как если бы оно не выполнялось, т. е. если бы существовало такое X и такое $n_{i-1} = \tilde{n}$, что $n_i = X$ при всех $n_{i-1} \geqslant \tilde{n}$, то при всех $n_0 \geqslant \tilde{n}$ n_1 было равно X, а M было бы равно некоторой постоянной максимальной величине $\tilde{M}$ (т. е. следствие 5 не имело бы места). При этом $n^{(\tilde{M})}$ было бы равно бесконечности, а $n^{(M)}$ при $M > \tilde{M}$ не существовало бы (т. е. следствие 6 тоже не имело бы места).

нейшем. Если известно время τ_i срабатывания любого i-го слоя и если $\tau_1=\tau_2=\ldots=\tau_M=\tau$, то величина M определяет время работы всего устройства от момента подачи $n=n_0$ исходных чисел (частичных произведений) на входы первого слоя до получения двухрядного кода произведения; это время равно $M\tau$.

Следствие 9. Если $n_{i-1}=n^{(T)}$, то $n_i=n^{(T-1)}$. В самом деле, так как в данном случае n_i чисел сворачиваются $T-1$ слоями, то в силу следствия 8

$$n^{(T-2)}<n_i\leqslant n^{(T-1)}.$$

Если бы оказалось, что $n_i<n^{(T-1)}$, то при увеличении n_{i-1} на единицу n_i увеличилось бы не более чем на единицу (условие 4), и стало бы равно n'_i, неравенство

$$n^{(T-2)}<n'_i\leqslant n^{(T-1)}$$

выполнялось бы и поэтому n'_i чисел сворачивалось бы $T-1$ слоями. Но это означало бы, что $n^{(T)}+1$ чисел свернуто T слоями, что невозможно. Поэтому $n_i=n^{(T-1)}$.

Следствие 9, в частности, означает, что если $n_i=n^{(T-1)}$, то $\max(n_{i-1})=n^{(T)}$. Этой возможностью мы будем пользоваться для вычисления величин $n^{(T)}$.

Пользуясь следствием 8, можно написать следующую систему неравенств:

$$n^{(M-1)}<n_0\leqslant n^{(M)},\quad n^{(M-2)}<n_1\leqslant n^{(M-1)},\ \ldots,$$

$$n^{(M-i-1)}<n_i\leqslant n^{(M-i)},\quad \ldots,\ n^{(0)}<n_{M-1}\leqslant n^{(1)},$$

$$n_M=n^{(0)}=2.$$

Очевидно, что если мы некоторый i-й слой построим не в соответствии с функцией $n_i=f(n_{i-1})$, а некоторым другим способом, но так, чтобы при этом неравенство $n^{(M-i-1)}<n_i\leqslant n^{(M-i)}$ не нарушилось, то общее количество слоев M не изменится. В дальнейшем мы будем пользоваться этой возможностью для модификации рассматриваемых методов построения многослойных ОУ.

2°. Первое из многослойных ОУ, которое мы рассмотрим, организовано следующим образом [1, с. 454].

Разобъем множество, состоящее из n исходных слагаемых (частичных произведений), на непересекающиеся группы по два числа в группе. Количество таких групп (назовем их полными) равно $[n/2]$; кроме того, при нечетном n останется еще одна неполная группа, содержащая одно число. В каждой полной группе сложим два числа при помощи обычного параллельного двоичного сумматора с последовательным распространением переносов. Число, попавшее в неполную группу, преобразовывать не будем. Для преобразования понадобится, очевидно, $\lambda_1=[n/2]$ параллельных сумматоров, совокупность которых назовем первым слоем устройства. Выходами слоя будем считать $[n/2]$ чисел, получившихся на выходах сумматоров, и число, попавшее в неполную группу, которое, таким образом, проходит сквозь слой без преобразования. Количество ν_1 чисел, попавших в неполную группу, очевидно, равно $n-2[n/2]$. Видно, что ν_1 равно 0 или 1. Можно сказать, что в первом слое $n=n_0$ чисел при помощи λ_1 параллельных сумматоров преобразуется в

$$n_1=\lambda_1+\nu_1=\left[\frac{n_0}{2}\right]+n_0-2\left[\frac{n_0}{2}\right]=n_0-\left[\frac{n_0}{2}\right]$$

чисел. Аналогичным образом строится каждый i-й слой, т. е.

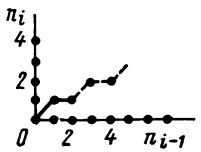
$$n_i=\lambda_i+\nu_i=n_{i-1}-\left[\frac{n_{i-1}}{2}\right]. \qquad (4.4.1)$$

Здесь λ_i — количество параллельных сумматоров в i-м слое.

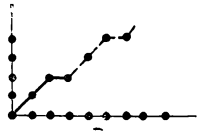
График функции $n_i=f(n_{i-1})$ приведен на рис. 4.4.16*а* (с. 203). Сплошной линией показана часть графика, которая потом периодически повторяется (штриховая линия). Из (4.4.1) и графика видно, что при данном методе построения многослойного ОУ выполняются все перечисленные выше условия (условия 1—5) и поэтому справедливы все девять следствий. Структура такого ОМУ описывается соотношениями

$$n_i=\lambda_i+\nu_i,\ \lambda_i=\left[\frac{n_{i-1}}{2}\right],$$

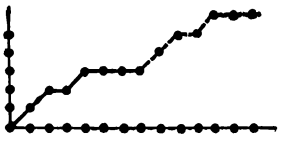
$$\nu_i=n_{i-1}-2\lambda_i,\quad n_M=2,\quad i=1, 2, \ldots, M. \qquad (4.4.2)$$



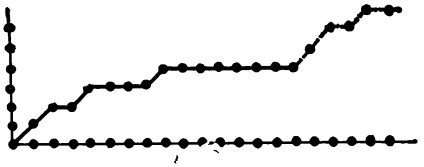
а) $n^{(T)}=2, 4, 8, 16, 32, 64$



б) $n^{(T)}=2, 3, 4, 6, 9, 13$

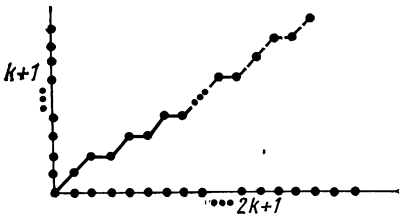


в) $n^{(T)}=2, 3, 7, 15, 35, 80$

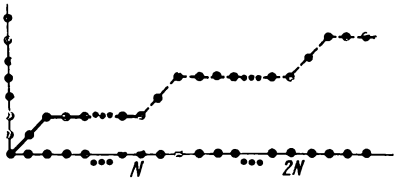


г) $n^{(T)}=2, 3, 7, 22, 78, 288$

д) $n^{(T)}=2, 3, 7, 127, 2^{127}-1$
$2^{2^{127}-1}-1$



е) $n^{(T)}=2, 3, 5, 9, 17, 32$ (при $k=8$)



ж) $n^{(T)}=2, 8, 32, 128, 512, 2048$
(при $N=8$)

Рис. 4.4.16. Типовой узел: а), б) — счетчики (3, 2); в) — счетчик (7, 3); г) — счетчик (15, 4); д) — счетчик (k, m); е) — k-разрядный сумматор; ж) преобразователь $N \longrightarrow 2$. $T=0, 1, 2, 3, 4, 5$.

Сумма двух чисел, получающихся на выходах последнего, M-го, слоя, как уже говорилось, равна произведению AC.

Из (4.4.1) видно также, что при данном n_i максимальное значение n_{i-1} равно $2n_i$. Поэтому в силу следствия 9

$$n^{(T)}=2n^{(T-1)} \quad (T=1, 2, \ldots).$$

Так как $n^{(0)}=2$, то $n^{(1)}=4$, $n^{(2)}=8$, ..., т. е.

$$n^{(M)}=2^{M+1} \quad (M=0, 1, \ldots). \qquad (4.4.3)$$

Первые шесть чисел $n^{(T)}$ приведены на рис. 4.4.16,*а*.

Так как для любого n есть такое T, что

$$n^{(T-1)}=2^T<n\leqslant 2^{T+1}=n^{(T)},$$

то в силу следствия 8 этому n соответствует

$$M=T.$$

Иными словами,

$$M=]\log_2 n[-1^{*}). \qquad (4.4.4)$$

Например, при $n=48$ $M=5$ (так как $n^{(4)}=32<n\leqslant 64=n^{(5)}$). Соотношения (4.4.3) и (4.4.4) полностью определяют прямую зависимость между количеством слагаемых n и числом слоев M и обратную зависимость между числом слоев M и максимальным количеством слагаемых $n^{(M)}$.

Поскольку каждый параллельный сумматор уменьшает количество складываемых чисел на единицу, то полное число этих сумматоров равно

$$\sum_{i=1}^{M} \lambda_i = n-2.$$

На рис. 4.4.17 показано расположение чисел на входах и выходах слоев ОУ рассмотренного типа для случая $n=12$, причем на рисунке отражена также процедура завершающего сложения двух чисел, после которого получается окончательное произведение. На рисунке каждая точка изображает двоичную цифру. Исходная матрица A чисел преобразуется первым слоем в матрицу B, после второго слоя получается матрица C, затем D и, наконец, результат E. Каждая рамка на рисунке окружает числа, подаваемые на входы одного парал-

*) $]d[$ обозначает наименьшее целое число, не меньшее чем d.

лельного сумматора. Горизонтальными линиями соединены выходные цифры каждого параллельного сумматора. Будем считать, что последний сумматор, преобразовывающий матрицу D, составляет дополнительный, $(M+1)$-й слой устройства. Из рисунка видно, что количество параллельных сумматоров равно 11, а количество одноразрядных сумматоров — 146. Видно также, что максимально полное время работы рассмотренного устройства равно

$$(M+1)\tau_{см}+(2n-2)\tau_E=]\log_2 n[\,\tau_{см}+(2n-2)\tau_E,$$

где $\tau_{см}$, τ_E — соответственно время образования цифры суммы и цифры переноса в одноразрядном сумматоре. В самом деле, второй слой начинает работать через $\tau_E+\tau_{см}$ после начала работы первого слоя (временем формирования частичных произведений пренебрегаем), третий слой — еще через $2\tau_E+\tau_{см}$, четвертый — еще через $4\tau_E+\tau_{см}$. К этому времени переносы фактически пройдут сквозь $1+2+4$ младших разрядов устройства (с $2n-1$-го до $2n-8$-го). В последнем слое переносы проходят разряды, оставшиеся до первого разряда. Кроме задержек, связанных с выработкой переносов, имеется еще задержка $\tau_{см}$ в каждом слое.

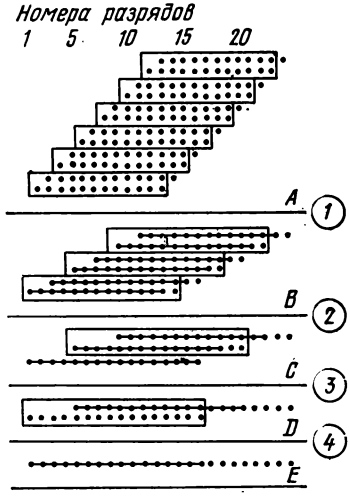


Рис. 4.4.17

Из рис. 4.4.17 видно, что данное ОУ (и рисунок) строились фактически по следующим правилам.

1) Исходная матрица слагаемых (частичных произведений) изображена в виде параллелограмма. Ряды цифр, получаемых на выходах каждого слоя, изображаются на рисунке друг под другом в том же порядке, что и рамки, отражающие соответствующие ряды аппаратуры (в данном случае — параллельные сумматоры) на входах слоя.

2) Количество n_i чисел на выходах каждого i-го слоя определяется функцией $n_i=f(n_{i-1})$, однако количество цифр на выходах одного разряда равно n_i только в нескольких средних разрядах слоя; в крайних разрядах из-за ограниченной длины слагаемых количество цифр на выходах меньше, чем n_i.

3) Можно сказать, что аппаратура (сумматоры) устанавливались сперва так, как если бы слагаемые были бесконечно длинными числами, а потом с учетом реальной разрядности слагаемых из схемы были убраны все «лишние» узлы (одноразрядные сумматоры), т. е. те части аппаратуры, которые никак не влияют на расположение значащих цифр на выходах слоев.

Можно заметить, что указанные правила однозначно определяют структуру, изображенную на рис. 4.4.17. Назовем такую структуру *«основной»*. В дальнейшем мы не раз будем встречаться с различными «основными» структурами ОУ, отличающимися между собой только тем, что вместо двоичных сумматоров в слоях будут устанавливаться некоторые другие типовые узлы.

Заметим также, что, изменив сформулированные правила, можно строить и другие структуры (при той же функции $n_i=f(n_{i-1})$). Например, если изъять первое из трех правил, то можно получить множество структур,

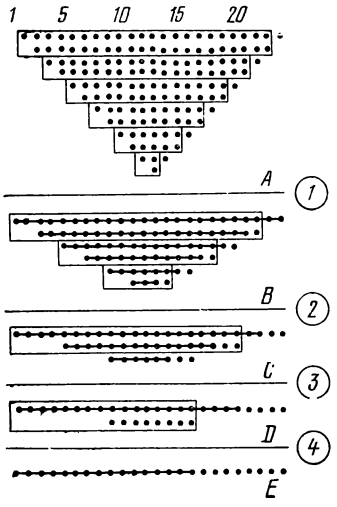


Рис. 4.4.18

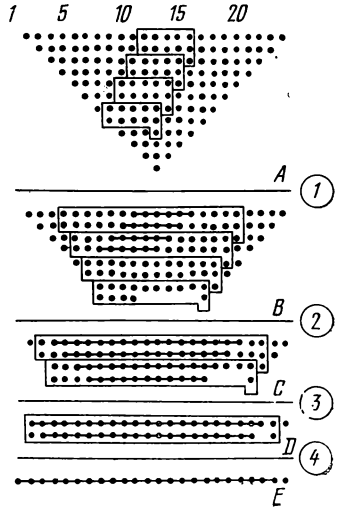


Рис. 4.4.19

одна из которых показана на рис. 4.4.18. Количество слоев M при этом не меняется.

3°. В соответствии с замечанием, сделанным в конце п. 1°, можно несколько видоизменить конструкцию только что рассмотренного устройства, сохранив то же самое количество слоев $M=]\log n_0[-1$. Попробуем, например, уменьшить до предела количество параллельных сумматоров в первом слое. Поскольку должно выполняться неравенство

$$n^{(M-2)}<n_1\leqslant n^{(M-1)}, \qquad (4.4.5)$$

то количество сумматоров $\lambda_1=[n_0/2]$ можно уменьшить до величины

$$\lambda'_1=n_0-n^{(M-1)}=n_0-2^M=n_0-2^{]\log_2 n_0[-1}.$$

Эти λ'_1 сумматоров складывают попарно $2\lambda'_1$ чисел, а остальные $n_0-2\lambda'_1$ чисел пропускаются сквозь слой без преобразования. Тогда на выходах первого слоя получается

$$n_1=\lambda'_1+(n_0-2\lambda'_1)=n^{(M-1)}=2^M$$

чисел, что соответствует правой границе неравенства (4.4.5). Остальные слои должны быть построены по старым правилам, т. е. $\lambda_i=[n_{i-1}/2]$ $(i=2, 3, \ldots, M)$. Очевидно, что при этом

$$n_2=\frac{n_1}{2}=n^{(M-2)}=2^{M-1}, \quad \lambda_2=n_2,$$

$$n_3=\frac{n_2}{2}=n^{(M-3)}=2^{(M-2)}, \quad \lambda_3=n_3,$$

$$\ldots\ldots\ldots\ldots\ldots\ldots$$

$$n_M=\frac{n_{M-1}}{2}=n^{(0)}=2, \quad \lambda_M=2.$$

Количество слоев M, как мы видим, сохраняется. Интересно, что количество параллельных сумматоров в данном случае тоже не меняется:

$$(n_0-2^M)+2^{M-1}+2^{M-2}+\ldots+2^1=n_0-2.$$

На рис. 4.4.19 отражена такая модифицированная структура для случая $n=12$. Кроме того, что $n_1=n^{(3)}=8$ (в отличие от $n_1=6$ на рис. 4.4.17), $n_2=4$, $n_3=2$, из

рис. 4.4.19 виден еще ряд особенностей новой структуры. Правила, по которым строилась новая структура, можно сформулировать следующим образом.

1) Аппаратура устанавливается так, чтобы количество цифр n^j_i на выходах каждого j-го разряда каждого i-го слоя удовлетворяло неравенству $n^j_i \leqslant n^{(M-i)}$.

2) В каждом слое устанавливаются только такие узлы (в данном случае — одноразрядные сумматоры), изъятие любого из которых приведет к невыполнению требования $n^j_i \leqslant n^{(M-i)}$.

3) Узлы в слоях используются «максимальным» образом (задействуется как можно больше входов этих узлов; на рис. 4.4.19 показано, в частности, что используется даже вход переноса младшего разряда параллельного сумматора).

Устройство, построенное по указанным правилам, будем называть «экономичным». Из рис. 4.4.19 видно, что в данном случае «экономичное» ОУ содержит столько же слоев (4), что и основное, столько же параллельных сумматоров (11), но количество одноразрядных сумматоров существенно уменьшилось (132 вместо 146). Мало того, уменьшилось и время работы устройства. Из рисунка видно, что теперь оно равно

$$(2n-2)\tau_E \text{ при } \tau_E \geqslant \tau_{см},$$

$$(2n-4)\tau_E + 2\tau_{см} \text{ при } \tau_E < \tau_{см}.$$

«Экономичные» структуры мы будем встречать и в дальнейшем. В силу следствия 8 «экономичную» модификацию можно построить для любого ОУ, удовлетворяющего условиям 1—5.

Можно заметить, что сформулированные правила построения «экономичной» схемы допускают различные построения, т. е. не определяют структуру ОУ однозначно.

Кроме основной и «экономичной», можно было бы предложить множество других модификаций, но мы этого делать не будем.

В табл. 4.4.2 в качестве еще одного примера приведены величины n_i и λ_i основной и «экономичной» структуры ОУ рассмотренного типа для случая $n_0 = 48$.

Основной недостаток обеих модификаций рассмотренного устройства состоит в том, что в них нецелесообразно делать ускоренным сумматор последнего, $M+1$-го

слоя, так как распространение переноса в нем все равно задерживается сумматорами предыдущих слоев. Аналогичное положение мы наблюдали в однородных матричных ОУ типа I (см. рис. 4.4.7).

4°. Этот недостаток объясняется тем, что внутри каждого слоя происходит распространение переносов от младших разрядов к старшим. В дальнейшем мы рас-

Таблица 4.4.2

i	Основное ОУ		„Экономичное" ОУ		i	Основное ОУ		„Экономичное" ОУ	
	n_i	λ_i	n_i	λ_i		n_i	λ_i	n_i	λ_i
0	48		48		3	6	6	8	8
1	24	24	32	16	4	3	3	4	4
2	12	12	16	16	5	2	1	2	2

смотрим ряд структур ОУ, в которых будет достигаться высокое быстродействие благодаря тому, что процессы распространения переносов внутри слоев отсутствуют. Но раньше опишем структуру, которую можно считать однослойной и которая в определенном смысле является исходной для всех последующих модификаций. Это ОУ представляет собой цепочку из $2n-1$ последовательно соединенных параллельных счетчиков (k, m). В каждом разряде устройства находится один параллельный счетчик, который складывает все цифры частичных произведений, относящиеся к данному разряду (т. е. до n цифр), и, кроме того, переносы, поступающие из некоторых счетчиков, находящихся в более младших разрядах устройства [21]. Такое ОУ для случая $n_0=12$ иллюстрируется рис. 4.4.20, 4.4.21, на первом из которых показаны несколько младших разрядов устройства, а на втором — расположение чисел на входах и выходах счетчиков. На рис. 4.4.21 выходы каждого счетчика соединены наклонной линией. Из рисунка видно, что самый сложный счетчик — (15,4) — стоит в 13-м разряде. Общее количество счетчиков равно $2n-1$.

Устройства такого типа имеют простую структуру, но обладают существенными недостатками: параллельные счетчики с большим числом входов представляют собой сложные схемы и распространение переносов вдоль цепочки счетчиков отнимает много времени.

5°. Теперь перейдем к многослойным структурам, в которых нет распространения переносов внутри слоев. Изменим предыдущее устройство [21]. Будем формировать произведение в два этапа — вначале получим двухрядный код произведения, т. е. два числа, сумма которых равна искомому произведению, а затем сложим эти два числа на ускоренном сумматоре.

В каждом из M слоев устройства, формирующего двухрядный код произведения, в одном разряде будет стоять не более одного параллельного счетчика (k, m). Внутри слоя связей между счетчиками не будет. Слои соединим, как и раньше, последовательно. Если на вход данного слоя в некотором разряде поступает только одна цифра, то она будет проходить сквозь слой без преобразования (две цифры в одном разряде также не будут преобразовываться, если в каждом из более младших разрядов на входе этого слоя тоже не будет более двух цифр).

Такое устройство для $n=12$ иллюстрируется рис. 4.4.22, на котором, как и на рис. 4.4.21, наклонные линии соединяют выходные цифры отдельных параллельных счетчиков. Из рис. 4.4.22 видно, что устройство содержит всего 3 слоя; матрица A преобразуется в матрицу B, затем B — в C и, наконец, C — в D. Видно, что

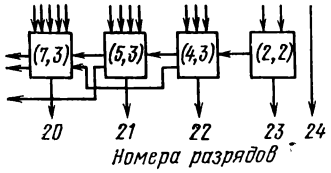


Рис. 4.4.20

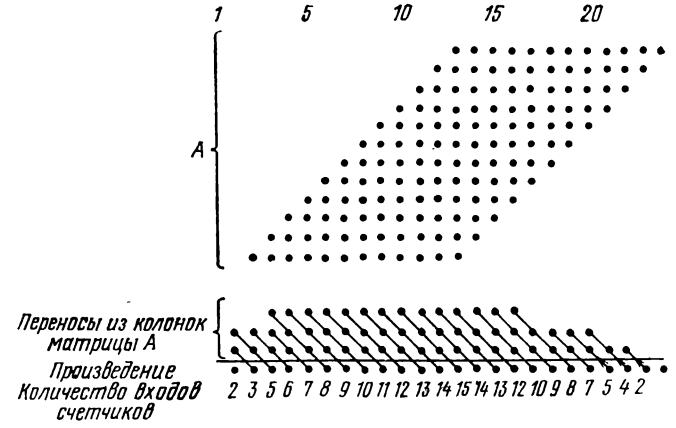


Рис. 4.4.21

максимальное количество входов одного счетчика в таком устройстве равно n.

Из описания способа построения слоев следует, что данному n_{i-1} соответствует n_i, являющееся наименьшим целым числом, удовлетворяющим неравенству

$$2^{n_i} - 1 \geqslant n_{i-1}.$$

Отсюда следует, что

$$n_i = [\log_2 n_{i-1}] + 1. \qquad (4.4.7)$$

График функции $n_i = f(n_{i-1})$ показан на рис. 4.4.16*д* (с. 203). Из описания этой зависимости видно, что и в данном случае условия 1—5 выполняются. Так как $\max(n_{i-1}) = n^{(T)}$ при $n_i = n^{(T-1)}$ (следствие 9), то из (4.4.7) выводится:

$$n^{(T-1)} = \log_2 (n^{(T)} + 1),$$

т. е.

$$n^{(T)} = 2^{n^{(T-1)}} - 1. \qquad (4.4.8)$$

Вычисленные при помощи соотношения (4.4.8) несколько первых величин $n^{(T)}$ приведены на рис. 4.4.16*д*.

В силу следствия 8

$$M = T \text{ при } n^{(T-1)} < n \leqslant n^{(T)}.$$

Например, $M = 3$ при $n = 48$, так как

$$n^{(2)} = 7 < n \leqslant 127 = n^{(3)}.$$

Последовательность чисел n_i для этого примера имеет такой вид: 48, 6, 3, 2.

Видно, что при использовании данного метода $n^{(T)}$ стремительно увеличивается с ростом T. Для реальных значений n количество слоев не превышает величины 2—3.

Следует отметить, что до появления работы [21] этот метод сложения множества чисел был предложен в [42] в более общем виде для систем счисле-

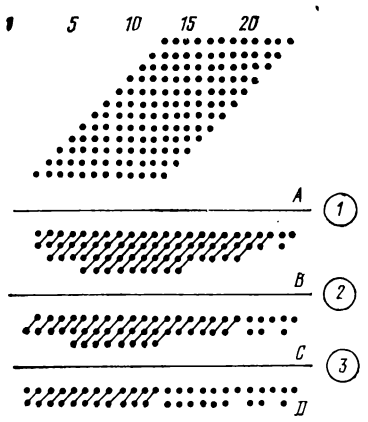


Рис. 4.4.22

ния с произвольным основанием r. Для $r=2$ в [42] была приведена таблица величин $n^{(T)}$. Эта же таблица в [21] содержала ряд неточностей: в качестве $n^{(3)}$, $n^{(4)}$, $n^{(5)}$ вместо 127, $2^{127}-1, 2^{2^{127}-1}-1$ было указано: 15, 32, 64.

6°. Только что описанный метод обеспечивает весьма малое число слоев, но для использования метода нужны параллельные счетчики, имеющие до n входов. При больших n такие узлы громоздки. Поэтому целесообразно рассмотреть методы, позволяющие использовать параллельные счетчики (k, m) с ограниченными k.

Одной из самых первых работ в этой области была работа [42] В. М. Храпченко, в которой описан способ ускоренного суммирования большого количества r-ичных чисел при помощи одинаковых параллельных счетчиков (k, m), каждый из которых складывает k r-ичных цифр одинакового веса. Этот способ создан независимо от работ [19, 43] и представляет собой обобщение идеи, выдвинутой в них. Многослойные матричные ОУ, построенные на счетчиках (k, m), с различными k, исследовались также в работах [17, 21] и в ряде более поздних публикаций.

Начнем со случая $k=2$, $r=2$ (в дальнейшем в этом параграфе мы будем говорить только о суммировании двоичных чисел). Использование счетчиков (2, 2), т. е. двоичных полусумматоров, при $n=3$ и $n=4$ показано на рис. 4.4.23, 4.4.24. Видно, что при $n=3$ устройство содержит один слой, а при $n=4$ — четыре слоя. Как и в предыдущем случае (рис. 4.4.22), внутри каждого слоя устройства, формирующего двухрядный код произведения, связей между счетчиками нет. Отличие состоит в том, что теперь в одном разряде одного слоя могут находиться несколько счетчиков (при $n=4$ в 5-м разряде 1-го слоя и в 4-м разряде 2-го слоя стоят по два счетчика).

Рассматривая структуры при бóльших n, можно убедиться в том, что количество слоев в устройстве, построенном на счетчиках (2, 2) по этому методу, примерно равно $3{,}7n-10{,}4$ *). Такие устройства, конечно, слишком медленны.

7°. Перейдем к случаю $k=3$. К настоящему времени предложено значительное количество методов построе-

*) Погрешность этой эмпирической формулы при $n\leqslant 80$ не превышает $\pm 1{,}8$. Формула $M=2^{n-2}$ из [21] неверна.

ния многослойных ОУ на счетчиках (3, 2), т. е. на одноразрядных двоичных сумматорах. Рассмотрим одно из самых первых предложений [17, 18].

На рис. 4.4.25, взятом из работы [21], показано расположение чисел на входах и выходах каждого слоя предложенного устройства для случая $n=12$. Принцип построения такого ОУ состоит в следующем.

В первом слое устройства расположены $\lambda_1=[n/3]$ рядов одноразрядных двоичных сумматоров. Соответственно n исходных слагаемых делятся на непересека-

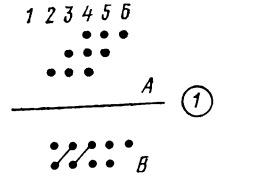


Рис. 4.4.23

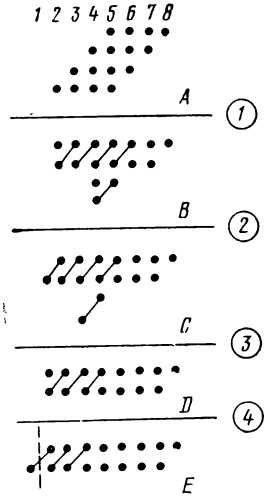


Рис. 4.4.24

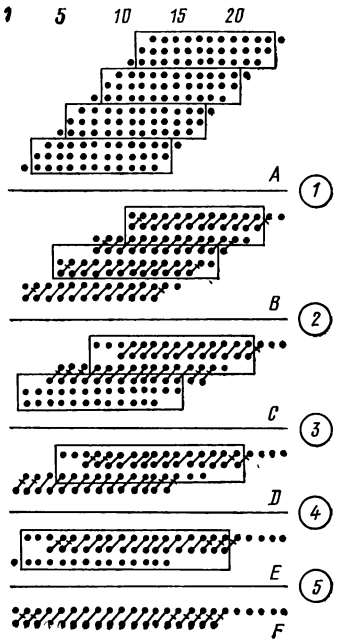


Рис. 4.4.25

ющиеся группы: $\lambda_1=[n/3]$ полных групп по 3 числа каждая и одна неполная группа, содержащая $\nu_1=n-3[n/3]$ чисел ($\nu_1=0$, 1 или 2). Каждый из λ_1 рядов сумматоров складывает 3 слагаемых соответствующей полной группы (каждая такая тройка чисел на рисунке обведена рамкой) и вырабатывает два числа — ряд по-

разрядных сумм и ряд поразрядных переносов, т. е. преобразовывает трехрядный код в двухрядный. На выходах первого слоя образуется n_1-рядный код, причем

$$n_1=2\lambda_1+\nu_1=2[n_0/3]+n_0-3[n_0/3]=n_0-[n_0/3]$$

(ν_1 слагаемых неполной группы проходят сквозь слой без преобразования). На рис. 4.4.25 $\lambda_1=4$, $\nu_1=0$, $n_1=8$. Короткими черточками на этом и следующем рисунке перечеркнуты линии, соединяющие выходные цифры тех одноразрядных сумматоров, которые складывают по две цифры и поэтому могут быть заменены полусумматорами.

Аналогичным образом во втором слое располагаются $\lambda_2=[n_1/3]$ рядов одноразрядных сумматоров. Второй слой заменяет n_1-рядный код n_2-рядным, причем

$$n_2=2\lambda_2+\nu_2,$$

где $\nu_2=n_1-3\lambda_2$. На рис. 4.4.25 $\lambda_2=2$, $\nu_2=2$, $n_2=6$. Таким же способом строится каждый i-й слой, т. е.

$$n_i=2\lambda_i+\nu_i=2[n_{i-1}/3]+(n_{i-1}-$$
$$-3[n_{i-1}/3])=n_{i-1}-[n_{i-1}/3]. \qquad (4.4.9)$$

График функции $n_i=f(n_{i-1})$ приведен на рис. 4.4.16*б* (с. 203). Из графика и из (4.4.9) видно, что условия 1—5 выполняются. Структура такого ОУ описывается соотношениями:

$$n_i=2\lambda_i+\nu_i,\ \lambda_i=[n_{i-1}/3],$$
$$\nu_i=n_{i-1}-3\lambda_i,\ n_M=2,\ i=1,\ 2,\ \ldots,\ M.$$

При $n_i=n^{(T-1)}$ максимальная величина $n_{i-1}=n^{(T)}$ получается, очевидно, в том случае, когда слой содержит $\lambda=[n_i/2]$ рядов сумматоров, т. е. каждой паре чисел на выходах слоя соответствует 3 числа на входах слоя. При этом сквозь слой без преобразования проходят $\nu=n_i-2\lambda$ чисел (ν равно 0 или 1). Поэтому

$$n^{(T)}=3\lambda+\nu=3\left[\frac{n^{(T-1)}}{2}\right]+\left(n^{(T-1)}-2\left[\frac{n^{(T-1)}}{2}\right]\right)=$$
$$=n^{(T-1)}+\left[\frac{n^{(T-1)}}{2}\right]. \qquad (4.4.10)$$

С помощью этого соотношения можно рассчитывать последовательность величин $n^{(T)}$; несколько первых из них приведены на рис. 4.4,16,*б*.

Как и раньше, $M=T$ при

$$n^{(T-1)}<n\leqslant n^{(T)}. \qquad (4.4.11)$$

Например, $M=9$ при $n=48$, так как

$$n^{(8)}=42<n\leqslant 63=n^{(9)}.$$

Последовательность чисел n_i для этого примера можно определить при помощи формулы (4.4.9):

48, 32, 22, 15, 10, 7, 5, 4, 3, 2.

В работе [44] указано, что при использовании данного метода

$$M=\left]\log_{\frac{3}{2}}\frac{n}{2}\right[\qquad (4.4.12)$$

или

$$M=\left]\log_{\frac{3}{2}}\frac{n}{2}\right[+1^{*}). \qquad (4.4.13)$$

Сравнивая (4.4.12), (4.4.13) с (4.4.4), мы видим, что число слоев существенно увеличивается, но следует учитывать, что задержка в каждом слое теперь равна лишь времени срабатывания одного одноразрядного сумматора.

Относительно структуры ОУ, соответствующей рис. 4.4.25, можно отметить, что она относится к классу «основных» структур, так как построена по правилам, приведенным в конце п. 2° данного параграфа.

8°. Опишем «экономичную» модификацию только что рассмотренного устройства [21]. В соответствии с правилом, изложенным в 3°, во-первых, поставим в первом слое не $\lambda_1=[n_0/3]$ рядов сумматоров, а $\lambda'_1=n_0-n^{(M-1)}$ рядов. Тогда

$$n_1=2\lambda'_1+(n_0-3\lambda'_1)=n^{(M-1)}.$$

В каждом из остальных слоев, как и в предыдущем устройстве, количество рядов сумматоров будет равно $\lambda_i=[n_{i-1}/3]$. При этом, как уже отмечалось, $n_2=n^{(M-2)}$, $n_3=n^{(M-3)}$, ..., $n_M=n^{(0)}=2$. Во-вторых, каждый ряд сумматоров в каждом слое будем максимально укорачивать с обеих сторон (до тех пор, пока дальнейшее укорачивание будет приводить к тому, что n_i будет получаться большим, чем $n^{(M-i)}$). Рис. 4.4.26 отражает структуру

*) $]d[$ обозначает наименьшее целое число, не меньшее чем d.

такого «экономичного» ОУ ($n=12$, как и на рис. 4.4.25). Из рисунка видно, что теперь в первом слое устройства устанавливаются только те одноразрядные сумматоры, при отсутствии любого из которых количество чисел n_1 на выходах слоя было больше, чем $n^{(M-1)}=n^{(4)}=9$. При этом одноразрядные сумматоры заменяются полусумматорами (отмечены черточками) там, где это возможно. Аналогичные меры приняты и в остальных слоях. Сравнивая последние два рисунка, можно увидеть, что количество слоев в «экономичном» устройстве такое же, как в предыдущем, а объем аппаратуры меньше.

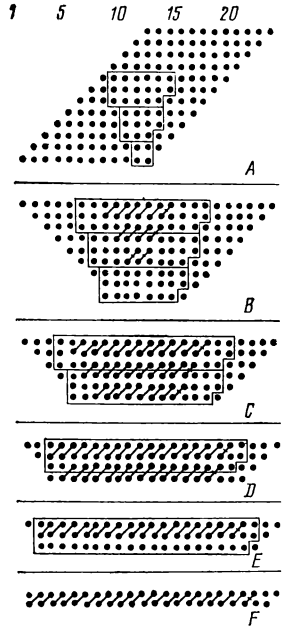


Рис. 4.4.26

Можно показать, что «экономичное» ОУ этого типа содержит $(n-1)(n-2)$ одноразрядных сумматоров и полусумматоров. Эта величина минимальна для любого ОУ, построенного из таких узлов и формирующего двухрядный код произведения. В данном случае (рис. 4.4.26) это количество равно $11\cdot 10=110$, т. е. на 26 меньше, чем в предыдущем устройстве.

Следует отметить, что до появления работы [21] практически такой же метод построения многослойных ОУ на счетчиках (3, 2) был описан в работе [17]. Основное отличие состояло в том, что в [17] рассматривалось «усеченное» устройство, содержащее n основных и несколько дополнительных разрядов.

9°. Допустим теперь, что в распоряжении разработчика имеются счетчики (7, 3) и (3, 2)*) [21]. Каждый слой будем строить способом, который в определенной

*) Наличие счетчика (3,2) при этом не меняет существа задачи, так как он может быть заменен счетчиком (7,3); вообще любой счетчик (k_1, m_1) может быть заменен счетчиком (k_2, m_2) при $k_2 \geqslant k_1$ (на k_2-k_1 лишних входов подаются нули, но, конечно, счетчик (k_1, m_1) проще, чем (k_2, m_2)).

степени аналогичен предыдущему. В каждом i-м слое поставим $\lambda_i=[n_{i-1}/7]$ рядов счетчиков (7, 3). Один такой ряд преобразовывает 7-рядный код в 3-рядный. Остальные $n_{i-1}-7\lambda_i$ входных чисел i-го слоя либо проходят сквозь слой без преобразования (при $n_{i-1}-7\lambda_i\leqslant 2$), либо преобразовываются еще одним рядом счетчиков (при $n_{i-1}-7\lambda_i\geqslant 3$). Очевидно, что при $n_{i-1}-7\lambda_i=3$ на выходах этого дополнительного ряда счетчиков образуется 2-рядный, а при $n_{i-1}-7\lambda_i>3$ получается 3-рядный код.

График функции $n_i=f(n_{i-1})$ приведен на рис. 4.4.16*в* (с. 203). Из описания этой функции следует, что условия 1—5 выполняются. Структура ОУ описывается соотношениями

$$n_i=3\lambda_i+\nu_i,\ \lambda_i=[n_{i-1}/7],$$

$$\nu_i=\begin{cases} n_{i-1}-7\lambda_i & \text{при } n_{i-1}-7\lambda_i\leqslant 2,\\ 2 & \text{при } n_{i-1}-7\lambda_i=3,\\ 3 & \text{при } n_{i-1}-7\lambda_i>3,\end{cases} \qquad (4.4.14)$$

$$n_M=2,\ i=1,\ 2,\ \ldots,\ M.$$

В соответствии со следствием 9 максимальная величина $n_{i-1}=n^{(T)}$ при $n_i=n^{(T-1)}$ получается, очевидно, в том случае, когда слой содержит $\lambda'=[n_i/3]$ рядов счетчиков (7, 3) и $\lambda''=[(n_i-3\lambda')/2]$ рядов счетчиков (3, 2) (λ'' равно 0 или 1), а $\nu=n_i-3\lambda'-2\lambda''$ чисел проходит сквозь слой без преобразования (ν равно 0 или 1). Иными словами, каждой тройке чисел на выходах слоя соответствует 7 чисел на его входах; если количество $n_i-3\lambda'$ оставшихся чисел равно 2, то им соответствует еще 3 числа на входах слоя, а если $n_i-3\lambda'=1$, то соответствующее число прошло сквозь слой без преобразования. Таким образом,

$$n^{(T)}=7\lambda'+3\lambda''+\nu,\ \lambda'=[n^{(T-1)}/3],$$

$$\lambda''=[(n^{(T-1)}-3\lambda')/2],\ \nu=n^{(T-1)}-$$

$$-3\lambda'-2\lambda'',\ T=1,\ 2,\ \ldots \qquad (4.4.15)$$

Первые несколько чисел $n^{(T)}$, полученные с помощью этих соотношений, приведены на рис. 4.4.16*в*.

Как и раньше, $M=T$ при $n^{(T-1)}<n\leqslant n^{(T)}$. Например, $M=5$ при $n=48$. Последовательность чисел n_i для этого примера можно получить при помощи (4.4.14):

48, 21, 9, 5, 3, 2.

Сравнивая величины $n^{(T)}$ на рис. 4.4.16*в* и *б* (с. 203), видим, что уже при $n>4$ количество слоев в новом устройстве меньше.

Заметим, что можно было бы в каждом i-м слое ставить только $\lambda_i=[n_{i-1}/7]$ рядов счетчиков (7, 3), а остальные $n_i-7[n_{i-1}/7]$ чисел пропускать сквозь слой без преобразования. Для структуры такого типа $n^{(T)}=7[n^{(T-1)}/3]+(n^{(T-1)}-3[n^{(T-1)}/3])$ и поэтому последовательность чисел $n^{(0)}$, $n^{(1)}$, ... изменяется*). В новой структуре не выполняется условие 3, т. е. n_i может убывать при росте n_{i-1}, из-за чего количество слоев M может быть больше, чем в только что описанной структуре, а объем аппаратуры — меньше. Подобные построения можно предложить и при использовании других типовых узлов — счетчиков (15, 4) (см. п. 11° этого параграфа), k-разрядных сумматоров (п. 6.3.1), преобразователей $N\to 2$ (п. 6.3.2) и др. Такие построения на преобразователях $N\to 2$ рассмотрены в работе [45].

10°. Аналогично тому, как это делалось в п. 3°, 8° данного параграфа, можно построить «экономичное» многослойное ОУ на счетчиках (7, 3), (3, 2). В таком устройстве после первого слоя получается $n_1=n^{(M-1)}$ чисел, после второго — $n_2=n^{(M-2)}$ и т. д. Например, если $n=48$, то $n_1=35$, $n_2=15$, $n_3=7$, $n_4=3$, $n_5=2$.

Основное и модифицированное («экономичное») ОУ для случая $n=12$ иллюстрируются соответственно рис. 4.4.27 и 4.4.28. Видно, что количество аппаратуры во втором устройстве меньше.

11°. Теперь допустим, что многослойное ОУ строится из счетчиков (15,4), (7,3), (3,2). Рассуждения, аналогичные предыдущим, приводят к формулам

$$\left.\begin{aligned} &n_i=4\lambda_i+\nu_i,\quad \lambda_i=[n_{i-1}/15],\\ &\nu_i=\begin{cases} n_{i-1}-15\lambda_i & \text{при } n_{i-1}-15\lambda_i\leqslant 2,\\ 2 & \text{при } n_{i-1}-15\lambda_i=3,\\ 3 & \text{при } 3<n_{i-1}-15\lambda_i\leqslant 7,\\ 4 & \text{при } n_{i-1}-15\lambda_i>7,\end{cases}\\ &n_M=2,\ i=1,\ 2,\ \ldots,\ M,\end{aligned}\right\} \qquad (4.4.16)$$

*) Возможно, что в работе [21] имелся в виду именно такой способ построения ОУ, так как в ней вместо $n^{(5)}=80$ указана величина 79.

$$n^{(T)}=15\lambda'+7\lambda''+3\lambda'''+\nu,\quad \lambda'=\left[\frac{n^{(T-1)}}{4}\right],$$

$$\lambda''=\left[\frac{n^{(T-1)}-4\lambda'}{3}\right],\quad \lambda'''=\left[\frac{n^{(T-1)}-4\lambda'-3\lambda''}{2}\right],$$

$$\nu=n^{(T-1)}-4\lambda'-3\lambda''-2\lambda''',\quad T=1,\ 2,\ldots \qquad (4.4.17)$$

График функции $n_i=f(n_{i-1})$ и первые несколько величин $n^{(T)}$ показаны на рис. 4.4.16*г* (с. 203) *). Условия 1—5 выполняются. Из рис. 4.4.16*г* видно, что при $16\leqslant n\leqslant 22$, $36\leqslant n\leqslant 78$, $n\geqslant 81$ количество слоев в этом

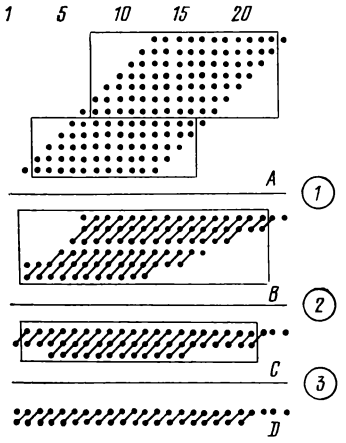


Рис. 4.4.27

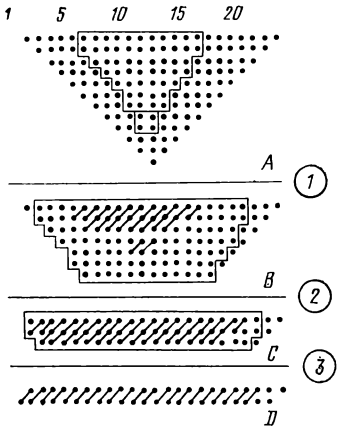


Рис. 4.4.28

устройстве меньше, чем в предыдущем, построенном на счетчиках (7, 3). Например, при $n=48$ число слоев равно 4 (так как $22=n^{(3)}<48<n^{(4)}=78$).

Из (4.4.16) для $n_0=48$ получаем все остальные величины n_i: $n_1=14$, $n_2=4$, $n_3=3$, $n_4=2$.

12°. При использовании счетчиков (15, 4), как и в других случаях, можно строить «экономичные» ОУ, в которых

$$n_i=n^{(M-i)} \qquad (i=1,\ 2,\ \ldots,\ M).$$

*) Соотношения (4.4.16), (4.4.17) приводятся здесь впервые. В работе [21] в качестве $n^{(3)}$, $n^{(4)}$, $n^{(5)}$ ошибочно указано 21, 61, 226.

В п. 6.3.1 и 6.3.2 шестой главы будут рассмотрены еще два метода построения многослойных матричных ОУ. По первому из этих методов умножитель составляется из больших интегральных схем (БИС), каждая из которых является k-разрядным двоичным параллельным сумматором. Второй метод предусматривает использование БИС, каждый из которых представляет собой малоразрядный преобразователь N-рядного кода в двухрядный. Несмотря на такую разницу в элементной

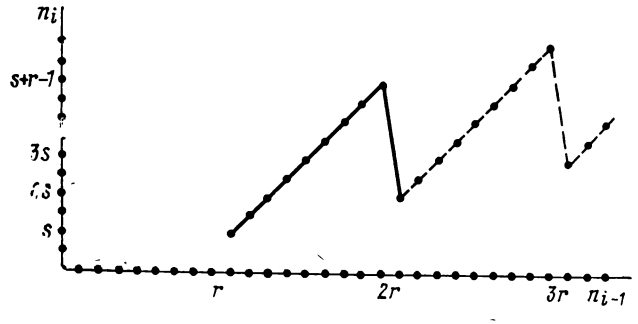


Рис. 4.4.29

базе, структуры ОУ в обоих этих случаях родственны структурам, рассмотренным в данном параграфе. Использование k-разрядных сумматоров и преобразователей $N \to 2$ отражено на рис. 4.4.16*ж* (с. 203).

Кроме методов, отображенных на рис. 4.4.16, существуют и другие. Например, в [47] рассмотрены многослойные ОУ, в которых типовым узлом является сумматор, складывающий r k-разрядных двоичных чисел и вырабатывающий их сумму в виде одного d-разрядного числа (подобный узел можно выполнить в виде постоянного запоминающего устройства, содержащего 2^{rk} чисел по d разрядов каждое). Предлагается выбирать такие r и k, чтобы d было кратно k. Цепочка (ряд) таких сумматоров преобразовывает группу из r многоразрядных чисел в s чисел, причем

$$d=sk, \qquad r=2^{(s-1)k}+2^{(s-2)k}+\ \ldots\ +1.$$

График $n_i=f(n_{i-1})$ для такого ОУ показан на рис. 4.4.29 (выбран случай $r=9$, $k=3$, $d=6$, $s=2$). Видно, что условие 3 не выполняется, если $r-1>s$ (оно

выполняется только при $r=3$, $k=1$, $d=2$, $s=2$, что соответствует методу, отраженному на рис. 4.4.16*б* (с. 203) и являющемуся, таким образом, частным случаем методов, рассмотренных в [47]).

5. БЫСТРОДЕЙСТВУЮЩИЕ СИНХРОННЫЕ УСТРОЙСТВА ДЛЯ ДЕЛЕНИЯ

Методы ускорения деления до недавнего времени были разработаны меньше, чем методы ускорения сложения, вычитания и умножения. По-видимому, это объясняется, с одной стороны, «последовательным» характером процедуры обычного деления (очередная цифра частного не может быть выбрана и новый остаток не может быть вычислен прежде, чем будет получен и исследован предыдущий остаток) и, с другой стороны, тем, что деление — сравнительно редкая операция. В некоторых ЦВМ непосредственное выполнение деления «обходится» при помощи стандартных подпрограмм, включающих умножение и сложение-вычитание. Однако в настоящее время деление включается в список операций почти всех современных универсальных машин и необходимость ускорения деления следует уже из самого существования весьма эффективных методов ускорения умножения: неускоренная операция деления теряет смысл при ускоренных сложениях-вычитаниях и умножениях, так как в этом случае подпрограмма деления может обеспечить большую скорость, чем аппаратное решение. Нужно отметить, что за последние годы появился ряд новых мощных синхронных методов ускорения деления и разработка новых методов продолжается. Основные из известных методов рассмотрены в этой главе.

Синхронные методы ускорения деления удобно разделить на две группы: ускорение деления, в каждом цикле которого выбирается одна или несколько цифр частного и вычисляется новый частичный остаток*), и

*) В дальнейшем будем называть несдвинутый частичный остаток просто остатком и обозначать как B_i, делимое и делитель будем обозначать соответственно буквами A и C.

передается результат вычитания, во втором (D=1) — цифра уменьшаемого (разряд предыдущего остатка). Вся схема состоит из ярусов (рис. 5.1.2), в каждом из которых производится вычитание делителя C из удвоенного предыдущего остатка, поступающего с выходов предыдущего яруса. Вдоль яруса при этом распространяются сигналы P, G займа (см. рис. 5.1.1 и 5.1.2).

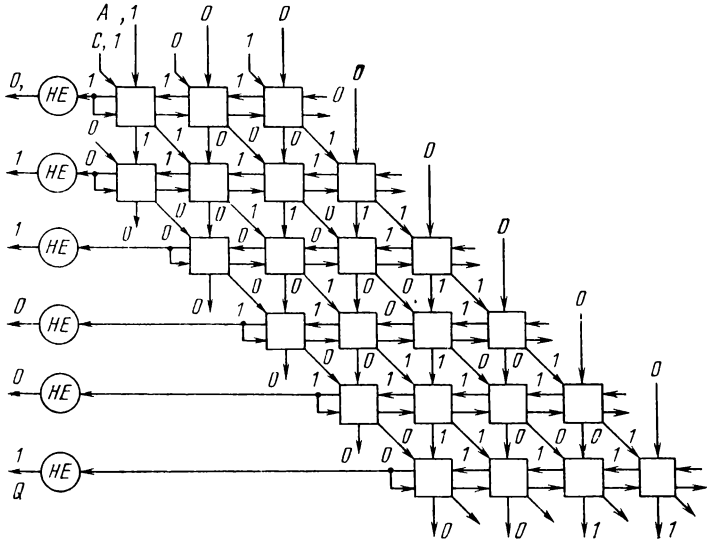


Рис. 5.1.2

Сигнал займа в старшем разряде яруса определяет знак разности $2B_i$—C (и очередную цифру частного) и управляет выработкой сигналов S во всех элементах данного яруса. На рис. 5.1.2 «расписан» пример деления A=0,100 на C=0,101, при котором получено частное Q=0,11001 и остаток B_6=0,11; точное соотношение между A, C, Q, B_6 в этом устройстве имеет вид

$$\frac{A}{C} = Q + \frac{B_6}{C2^5}.$$

Ускорение деления в таком устройстве достигается не усложнением алгоритма, а за счет использования

быстродействующих миниатюрных элементов, соединенных короткими регулярными связями *).

5.2. МЕТОДЫ СТЕФАНЕЛЛИ

Основу методов деления, предложенных Стефанелли [5], составляют несколько алгоритмов получения обратной величины $1/C$ двоичного делителя C, расположенного в диапазоне между $1/2$ и 1; эти методы могут быть расширены и для вычисления частного от деления двух чисел.

Формирование частного $Q \approx 1/C$ производится в два этапа — вначале частное вырабатывается в виде двоичного числа с избыточным представлением цифр, затем оно преобразовывается в простую двоичную форму.

В избыточном двоичном числе

$$X = x_1 2^{-1} + x_2 2^{-2} + \ldots$$

в качестве цифр x_i используются, вообще говоря, любые положительные или отрицательные целые числа.

Основная процедура формирования обратной величины делителя

$$C = c_1 2^{-1} + c_2 2^{-2} + \ldots + c_n 2^{-n}$$

предусматривает получение Q в избыточной форме

$$Q = a_0 2^0 + a_1 2^{-1} + a_2 2^{-2} + \ldots$$

Произведение CQ должно быть равно, очевидно, единице. Будем полагать, что оно равно двоичному числу 0,111 ... Располагая в виде матрицы частичные произведения множимого C на отдельные разряды a_i множителя Q, увидим в первой колонке этой матрицы член $c_1 a_0$, во второй — члены $c_2 a_0$, $c_1 a_1$, в третьей — $c_3 a_0$, $c_2 a_1$, $c_1 a_2$ и т. д. Основная идея метода состоит в том, что избыточные цифры a_i подбираются таким образом, чтобы сумма членов, стоящих в одной колонке, в точности равнялась единице, т. е. очередному разряду произве-

*) Строго говоря, данный метод является частным случаем деления с восстановлением остатка — делением «с незавершением» (non — performing division), при котором вычитание доводится до конца только в случае положительной разности $2B_i - C$; в противном случае вычитание прерывается и производится сдвиг остатка (см. § 7.8).

дения CQ. Легко показать, что при этом величины a_i определяются соотношениями

$$
\begin{aligned}
a_0 &= 1,\\
a_1 &= 1 - c_2,\\
a_2 &= 1 - (c_2 a_1 + c_3),\\
a_3 &= 1 - (c_2 a_2 + c_3 a_1 + c_4),\\
&\ldots\ldots\ldots\ldots\ldots\ldots\\
a_i &= 1 - (c_2 a_{i-1} + c_3 a_{i-2} + \ldots + c_i a_1 + c_{i+1}).
\end{aligned}
$$

В работе [5] описано матричное устройство, вычисляющее цифры a_i по этим формулам. Устройство это весьма громоздкое, что вызвано, в первую очередь, большим диапазоном возможных значений цифр a_i. Например, при 12-разрядном делителе C $-21 \leqslant a_9 \leqslant 6$, $-28 \leqslant a_{14} \leqslant 232$, $-80 \leqslant a_{18} \leqslant 1579$. Далее в работе [5] предложено последовательно три модификации метода, направленные на уменьшение диапазона значений цифр a_i и упрощение устройства. В последней модификации цифры a_i не выходят из диапазона $-3 \ldots +3$, что значительно упрощает вычисление величин a_i и преобразование Q в обычную двоичную форму.

5.3. ВЫПОЛНЕНИЕ ДЕЛЕНИЯ ЧЕРЕЗ УМНОЖЕНИЕ

В аппаратуре арифметических устройств современных ЦВМ часто реализуются методы деления, использующие те или иные итерационные формулы, включающие в себя умножение [6—12]. В таких арифметических устройствах частное или обратная величина делителя находятся путем последовательных приближений. В основе большинства из этих методов лежит известная итерационная формула

$$y_{i+1} = y_i(2 - y_i C), \quad i = 0, 1, 2, \ldots, \qquad (5.3.1)$$

используемая для приближенного вычисления величины $1/C$. Здесь $y_0, y_1, y_2, \ldots$ — последовательные приближения величины $1/C$. Известно, что если первое приближение y_0 не выходит из диапазона $0 < y_0 < 2/C$, то $y_i \to 1/C$ при $i \to \infty$. Чтобы ограничить сверху величины y_i (в том числе и y_0), ограничивают снизу величину C. Обычно полагают, что $1/2 \leqslant C < 1$.

Пусть δ_i — относительная погрешность величины y_i:

$$y_i = \frac{1}{C}(1 - \delta_i).$$

Подставив это выражение в (5.3.1), получим:

$$y_{i+1} = \frac{1}{C}(1 - \delta^2{}_i),$$

что означает удвоение количества верных разрядов после каждой итерации.

Количество итераций, которое нужно выполнить для получения необходимой точности результата, очевидно, сильно зависит от выбора первого приближения. Поэтому выбору величины y_0 уделяется большое внимание. Существует несколько методов выбора y_0 [9]. Наибольшую точность (до 8—9 верных двоичных знаков в y_0) обеспечивают табличные методы [9, 11, 12].

Момент окончания деления в арифметическом устройстве определяется не путем сравнения разности $|y_{i+1} - y_i|$ с некоторым допуском, как это бывает при выполнении деления по подпрограмме, а путем подсчета количества итераций. Например, если y_0 содержит 8 верных разрядов ($|\delta_0| \leqslant 2^{-8}$), то для получения, скажем, 15-разрядного частного выполняют одну итерацию, для получения 30-разрядного частного нужны две итерации и т. д.

Известно большое количество различных машинных алгоритмов для реализации итерационного процесса, основанного на формуле (5.3.1). Некоторые из этих алгоритмов с целью повышения скорости сходимости итераций предусматривают те или иные изменения основной формулы (5.3.1) (см., например, [13], с. 535—536). Здесь будут кратко рассмотрены четыре наиболее известных алгоритма.

Процедуры, выполняемые по первому из этих алгоритмов, буквально следуют самому написанию соотношения (5.3.1):

$$y_0 C,$$
$$2 - y_0 C,$$
$$y_1 = y_0 (2 - y_0 C),$$
$$y_1 C,$$
$$2 - y_1 C,$$

$$y_2 = y_1(2 - y_1C),$$
$$\cdots\cdots\cdots$$
$$y_iC,$$
$$2 - y_iC,$$
$$y_{i+1} = y_i(2 - y_iC).$$

При этом каждый цикл итерационного процесса состоит из трех последовательно выполняемых действий: умножения, взятия дополнения и еще одного умножения.

Для удобства изложения введем обозначения:

$$x_i = y_{i-1}C, \qquad R_i = 2 - x_i.$$

Одна (i-я) итерация тогда может быть описана соотношениями

$$x_i = y_{i-1}C, \quad R_i = 2 - x_i, \quad y_i = y_{i-1}R_i. \tag{5.3.2}$$

При $i \to \infty$ величины x_i, R_i стремятся к 1, а y_i, как было отмечено, стремится к $1/C$.

При втором алгоритме i-я итерация описывается соотношениями [6, 10]

$$x_i = x_{i-1}R_{i-1}, \quad R_i = 2 - x_i,$$
$$y_i = y_{i-1}R_i \quad (i = 2, 3, \ldots). \tag{5.3.3}$$

Первая итерация производится несколько иначе, а именно так, как при первом алгоритме:

$$x_1 = y_0C, \quad R_1 = 2 - x_1, \quad y_1 = y_0R_1.$$

Величины x_i, R_i, y_i при этих двух алгоритмах имеют совершенно одинаковые числовые значения (если пренебречь аппаратными погрешностями), в чем нетрудно убедиться, преобразовывая соотношения (5.3.2):

$$x_i = y_{i-1}C = Cy_{i-2}R_{i-1} = x_{i-1}R_{i-1}$$

(приходим к соотношению, используемому в формулах (5.3.3)). Отличие второго алгоритма от первого состоит лишь в том, что x_i вычисляется путем умножения x_{i-1} на R_{i-1} вместо использования эквивалентной формулы $x_i = y_{i-1}C$. Однако это отличие существенно влияет на аппаратную реализацию, так как после получения R_i величины y_i и x_{i+1} могут вычисляться одновременно.

Итерация при третьем алгоритме [8, 10, 11] выполняется по тем же соотношениям (5.3.3), что и при втором (для $i=2, 3, \ldots$). Отличие состоит в первой итерации:

$$x_1=y_0C, \quad R_1=2-x_1, \quad y_1=y_0AR_1,$$

где A — делимое. Величины x_i, R_i при этом совпадают с аналогичными величинами, получающимися при первых двух алгоритмах (если опять-таки пренебречь аппаратными погрешностями умножений и вычитаний). Но y_i в этом случае стремится к частному A/C. Третий алгоритм был реализован в арифметическом устройстве машины ИБМ 360/91 [11].

Четвертый алгоритм был разработан в работе [7]. Очередная, i-я, итерация в этом случае состоит в вычислении величин

$$w_i=w^2_{i-1}, \quad r_i=1+w_i, \quad \varphi_i=\varphi_{i-1}r_i \quad (i=2, 3, \ldots). \quad (5.3.4)$$

Первая итерация описывается соотношениями

$$w_1=1-y_0C, \quad r_1=1+w_1, \quad \varphi_1=y_0Ar_1.$$

Если пренебречь аппаратными погрешностями умножений и вычитаний, то третий и четвертый алгоритмы практически совпадают, так как

$$x_i=1-w_i, \quad R_i=r_i, \quad y_i=\varphi_i \quad (i=1, 2, \ldots).$$

Однако наличие аппаратных погрешностей приводит к тому, что вычисления по (5.3.3) и (5.3.4) протекают не одинаково. Например, в первом случае величина $R_i=2-x_i=2-x_{i-1}R_{i-1}$ может быть меньше единицы, а во втором случае $r_i=1+w_i=1+w^2_{i-1}$ не может быть меньше единицы.

Процессы накопления аппаратных и методических погрешностей для описанных в этом параграфе методов деления имеют довольно сложный характер и протекают при использовании различных алгоритмов по-разному. В настоящее время, по-видимому, не существует методов точного анализа этих процессов. Поэтому на практике обычно пользуются сравнительно грубыми оценками погрешностей. Эти оценки основаны на предположении, что аппаратные погрешности разных действий (например, погрешность умножения $y_{i-1}R_i$ в первом алгоритме и погрешность выполняемого вслед за ним умно-

жения y_iC) являются независимыми случайными величинами, что, вообще говоря, неверно. Погрешность после i-й итерации, т. е. разность между y_i и искомым частным, оценивается обычно «сверху» в предположении, что аппаратные погрешности всех промежуточных действий, выполненных после начала деления, имели наиболее «невыгодные» знаки и абсолютные величины.

Поскольку при каждом делении по любому из рассмотренных здесь методов нужно выполнять несколько (обычно 4—5) умножений, то эти методы особенно эффективны в арифметических устройствах с быстрыми (например, матричными) умножителями.

5.4. УСКОРЕННОЕ ДЕЛЕНИЕ БЕЗ ВОССТАНОВЛЕНИЯ ОСТАТКА

В этом параграфе рассматриваются различные синхронные методы ускоренного деления без восстановления остатка; описанный в § 7.8 метод деления без восстановления остатка является частным случаем рассматриваемой здесь группы методов.

5.4.1. ОБЩЕЕ ОПИСАНИЕ ДЕЛЕНИЯ БЕЗ ВОССТАНОВЛЕНИЯ ОСТАТКА

В общем виде синхронный метод деления без восстановления остатка можно описать следующим образом. Операция состоит из повторяющихся однотипных циклов, из которых только первый и последний несколько отличаются от остальных. Во время очередного $i+1$-го цикла деления в устройстве вычисляется очередной остаток B_{i+1}:

$$B_{i+1}=B_ir-a_{i+1}C. \qquad (5.4.1)$$

Здесь $i+1=1, 2, 3, \ldots$ — номер очередного цикла деления; B_i — предыдущий остаток; r — основание системы счисления, в которой производится деление *); C — делитель; a_{i+1} — очередная цифра частного.

Из (5.4.1) следует:

$$B_0/C=\sum_{i=1}^{k} a_ir^{-i}+B_k/r^kC. \qquad (5.4.2)$$

*) Величину r можно также определить как основание системы счисления, в которой производится предварительное формирование частного.

Здесь k обозначает полное число циклов деления. Таким образом, начальным остатком, т. е. первым из остатков B_i, к которым применяется формула (5.4.1) для вычисления B_{i+1}, является остаток B_0. Этот начальный остаток в общем случае отличается от делимого некоторым постоянным положительным множителем h:

$$\frac{A}{C}=\frac{hB_0}{C}=h\sum_{i=1}^{k}a_i r^{-i}+h\frac{B_k}{r^k C}. \qquad (5.4.3)$$

Сумма $h\sum_{i=1}^{k}a_i r^{-i}$, полученная в процессе выполнения операций в арифметическом устройстве, обычно воспринимается как результат деления, а остаточный член hB_k/r^kC составляет, таким образом, погрешность операции.

Каждый $(i+1)$-й цикл деления состоит в выборе a_{i+1} и вычислении B_{i+1}. Выбор цифры a_{i+1} осуществляется в устройстве по некоторым правилам, включающим, в общем случае, анализ величин C, B_i и некоторые операции над ними. Остаток B_{i+1}, как уже отмечено, вычисляется в соответствии с соотношением (5.4.1). Кроме того, производится «присоединение» очередной цифры частного a_{i+1} к образованной в предыдущих циклах сумме $\sum_{t=1}^{i}a_t r^{-t}$, т. е. формируется величина

$$\sum_{t=1}^{i}a_t r^{-t}+a_{i+1}r^{-i-1}. \qquad (5.4.4)$$

Введем обозначение

$$Q=\sum_{i=1}^{k}a_i r^{-i}. \qquad (5.4.5)$$

Величина Q с точностью до остаточного члена B_k/r^kC равна B_0/C. Величина hQ с точностью до hB_k/r^kC равна искомому частному A/C:

$$A/C=hQ+hB_k/r^kC. \qquad (5.4.6)$$

Цифры a_i выбираются из некоторого множества (набора) ξ допустимых значений цифр:

$$\xi=\{\xi_1, \xi_2, \ldots, \xi_m\}. \tag{5.4.7}$$

Для определенности будем считать, что

$$\xi_1<\xi_2< \ldots <\xi_m. \tag{5.4.8}$$

Следует отметить, что соотношение, служащее для вычисления B_{i+1}, могло бы быть записано в другой форме. Например:

$$B_{i+1}=rB_i-a_iC$$

или

$$B_{i+1}=r(B_i-a_{i+1}C),$$

или

$$B_{i+1}=r(B_i-a_iC)$$

и т. д. При этом изменяется форма записи соотношений (5.4.2), (5.4.3) и ряда других формул. Однако эти преобразования формул означают только изменение наименований переменных и констант и не меняют существа метода давления и его аппаратной реализации.

Набор величин ξ_j ($j=1, 2, \ldots, m$) и основание r являются основными параметрами метода деления без восстановления остатка. Они в значительной степени определяют быстродействие и аппаратную сложность конкретного варианта метода.

Как видно из (5.4.5), основанием системы счисления, в которой формируется частное Q, является число r, а в качестве цифр a_i в r-ичных разрядах числа Q используются величины $\xi_1, \xi_2, \ldots, \xi_m$.

Будем для общности считать, что эта система счисления отличается от системы, в которой представляются все числа в машине (и, в частности, делимое и делитель). Положим, что в машине используется обычная p-ичная система с основанием p и цифрами 0, 1, 2, ..., $p-1$.

Полученное частное Q должно быть преобразовано из r-ичной формы (5.4.5) в p-ичную форму

$$Q=\Sigma d_i p^{-i}, \tag{5.4.9}$$

в которой цифры d_i выбираются из набора

$$d=\{0, 1, \ldots, p-1\}. \tag{5.4.10}$$

Это преобразование может быть осуществлено в конце операции, однако возможно и преобразование каждого частичного частного $\sum_{t=1}^{i} a_t r^{-t}$ на каждом i-м цикле ($i=1, 2, \ldots, k$).

Можно отметить, что соотношения (5.4.1)—(5.4.3) имеют место и при синхронном делении с восстановлением остатка. Характерным признаком деления без восстановления остатка является именно необходимость преобразования частного из формы (5.4.5) в форму (5.4.9). В отличие от этого, при делении с восстановлением остатка цифры частного сразу выбираются из набора (5.4.10).

Поскольку учет знаков делимого и делителя и учет формы представления отрицательных чисел в машине (прямой или дополнительный код) не представляет принципиальных трудностей для методов деления без восстановления остатка, то для простоты изложения будем далее в этой главе считать, что A и C — неотрицательные числа *).

Первый цикл деления может отличаться от последующих наличием подготовительной операции формирования величины $B_0=A/h$, а также описанными в последующих разделах главы начальными преобразованиями исходных значений делимого и делителя.

Последний цикл деления может тем или иным образом отличаться от предыдущих в зависимости от требований, предъявляемых в данном устройстве к последнему остатку hB_k. В простейшем случае остаток B_k просто не вычисляется.

Каждый остаток B_i не должен выходить из некоторого диапазона значений. Это требуется, во-первых, для того, чтобы остаточный член в (5.4.6) был бы меньше допустимой погрешности деления, т. е. чтобы этот член при достаточно большом k становился достаточно малым, и, во-вторых, для того, чтобы каждый остаток B_i мог быть сформирован в некотором реальном устройстве. Таким образом, должно соблюдаться неравенство

$$B_{\min} \leqslant B_i \leqslant B_{\max}, \qquad (5.4.11)$$

*) В действительности представление отрицательных делимых и делителей в дополнительном коде сказывается на узлах арифметического устройства, выполняющих нормализацию делителя, выбор очередной цифры a_{i+1}, преобразование частного и др.

где B_{max} и B_{min} — границы диапазона допустимых значений остатка. Отметим, что должно выполняться требование

$$\left.\begin{aligned} B_{max} &\leqslant \xi_m C/(r-1), \\ B_{min} &\geqslant \xi_1 C/(r-1). \end{aligned}\right\} \qquad (5.4.12)$$

В самом деле, если, например, $B_0=\xi_m C/(r-1)+|\delta|$, то даже при $a_1=\xi_m$ получается $B_1=B_0 r-\xi_m C=\xi_m C/(r-1)+|\delta| r$. Выбирая далее $a_2=a_3=\ldots=a_k=\xi_m$, получаем $B_k=B_0+|\delta| r^k$, т. е. остатки B_i будут непрерывно возрастать, остаточный член в (5.4.3) будет равен $h\frac{B_k}{r^k C}=h\frac{B_0}{r^k C}+h\frac{|\delta|}{C}$ и при сколько-нибудь большом $|\delta|$ величиной $hB_k/(r^k C)$ нельзя будет пренебречь. Если же в качестве a_i выбирать не ξ_m, а другие цифры набора, то остатки будут возрастать еще быстрее. Аналогичным образом остатки B_{i+1}, B_{i+2}, ... начинают непрерывно возрастать по абсолютной величине, если $B_i<\xi_1 C/(r-1)$.

Диапазоны допустимых значений величин делимого A, делителя C и частного A/C зависят от принятой в машине формы представления чисел и в общем случае заданы неравенствами

$$A_{min}\leqslant A\leqslant A_{max}, \qquad (5.4.13)$$

$$C_{min}\leqslant C\leqslant C_{max}, \qquad (5.4.14)$$

$$(A/C)_{min}\leqslant A/C\leqslant (A/C)_{max}, \qquad (5.4.15)$$

где A_{min}, A_{max}, C_{min}, C_{max}, $(A/C)_{min}$, $(A/C)_{max}$ — константы.

5.4.2. КЛАССИЧЕСКИЙ МЕТОД ДЕЛЕНИЯ

Будем называть классическим обычный метод деления без восстановления остатка [14]. Деление по этому методу производится в r-ичной системе счисления с использованием следующего набора допустимых величин цифр частного:

$$\xi=\{-(r-1), -(r-2), \ldots, -2, -1, 1, 2, \ldots, r-1\}. \qquad (5.4.16)$$

Границами диапазона (5.4.11) допустимых значений остатков для классического метода являются величины

$$B_{\max}=\xi_m C/(r-1),$$

$$B_{\min}=\xi_1 C/(r-1), \qquad (5.4.17)$$

т. е. величины $+C$, $-C$. Таким образом,

$$-C\leqslant B_i\leqslant C. \qquad (5.4.18)$$

Правила выбора очередной цифры a_{i+1} для любого из описываемых в § 5.4 методов деления могут, как будет показано ниже, несколько видоизменяться. В частности, для метода, использующего набор (5.4.16), можно предложить различные правила. Наиболее известное из них сформулировано в работе [14, с. 42] в следующем виде: если $B_i\geqslant 0 (B_i<0)$, то из $B_i r$ многократно вычитают (прибавляют) делитель C до тех пор, пока знак этой разности (суммы) станет противоположным знаку величины B_i *). Цифра a_{i+1} при этом по абсолютной величине равна числу произведенных вычитаний (сложений) и имеет знак плюс (минус).

Покажем, что если B_i удовлетворяет неравенству (5.4.18), то приведенное правило выбора цифры a_{i+1} обеспечивает его выполнение и для следующего остатка B_{i+1}. Запишем величину rB_i в виде $rB_i=(M+\varepsilon)C$, где M — целое $(-r\leqslant M\leqslant r)$, а ε удовлетворяет неравенству $0\leqslant\varepsilon<1$ (при $M=r$ следует считать, как этого требует (5.4.18), что $\varepsilon=0$). В табл. 5.4.1 показана зависимость a_i и B_{i+1} от M, реализуемая приведенным правилом выбора a_{i+1}.

Из таблицы видно, что новый остаток B_{i+1} находится в том же диапазоне (5.4.18), что и B_i (здесь и ниже предполагается, что если $rB_i-\rho C=0$, где ρC — кратное делителя C, то эта разность имеет знак плюс).

Допустим, что числа в машине представлены в обычной r-ичной системе, использующей набор цифр

$$d=\{0, 1, 2, \ldots, r-1\}, \qquad (5.4.19)$$

---

*) Более точная формулировка должна была бы быть следующей: если $B_i\geqslant 0$ $(B_i<0)$, то из $B_i r$ многократного вычитают (прибавляют) делитель C до тех пор, пока знак этой разности (суммы) станет противоположен знаку величины B_i или пока количество произведенных сложений (вычитаний) не сделается равным $r-1$.

Таблица 5.4.1

M	a_{i+1}	B_{i+1}
r	$r-1$	$rC-(r-1)C=C$
$r-1$	$r-1$	$(r-1+\varepsilon)C-$ $-(r-1)C=\varepsilon C$
$0, 1, 2, \ldots, r-2$	$M+1$	$(M+\varepsilon)C-$ $-(M+1)C=-C(1-\varepsilon)$
$-(r-1), -(r-2), \ldots$ $\ldots, -2, -1$	M	$(M+\varepsilon)C-MC=\varepsilon C$
$-r$	$-(r-1)$	$(-r+\varepsilon)C+(r-1)C=$ $=-C(1-\varepsilon)$

и покажем, что можно сравнительно просто преобразовывать сумму $\sum_{t=1}^{i+1} a_t r^{-t}$ в эту систему счисления на каждом $(i+1)$-м цикле деления.

Пусть число $\sum_{t=1}^{i+1} a_t r^{-t}$, имеющее в системе с набором (5.4.16) вид

$$0,\ a_1 a_2 \ldots a_i a_{i+1},$$

после преобразования в систему с набором (5.4.19) выглядит так:

$$0,\ d_1 d_2 \ldots d_i d'_{i+1},$$

т. е. равно $\sum_{t=1}^{i} d_i r^{-i} + d'_{i+1} r^{-i-1}$. Так как $B_0 \geqslant 0$, то $a_1 \geqslant$ $\geqslant 1$ и поэтому после первого цикла деления число $0, a_1$ в системе счисления, использующей набор (5.4.16), и это же число $0, d'_1$ в системе, использующей набор (5.4.19), записываются одинаково, т. е. $d'_1 = a_1$. После второго цикла числа $0, a_1 a_2$ и $0, d_1 d'_2$ также записываются одинаково, если $B_1 \geqslant 0$, так как при этом $a_2 > 0$. Если же $B_1 < 0$, то $a_2 < 0$ и $d_1 = d'_1 - 1$, $d'_2 = r + a_2$.

Продолжая аналогичные рассуждения, приходим к тому, что если $B_i \geqslant 0$, то $a_{i+1} > 0$, и преобразованное частное после $(i+1)$-го цикла имеет вид

$$0,\ d_1 d_2 \ldots d_{i-1} d_i d'_{i+1} =$$
$$= 0,\ d_1 d_2 \ldots d_{i-1} d'_i a_{i+1},$$

т. е. в этом случае $d_i = d'_i$, $d'_{i+1} = a_{i+1}$. Если же $B_i < 0$, то $a_{i+1} < 0$ и преобразованное частное записывается в виде

$$0,\ d_1 d_2 \ldots d_{i-1} (d'_i - 1)\ (r + a_{i+1}),$$

т. е. $d_i = d'_i - 1$, $d'_{i+1} = r + a_{i+1}$. Таким образом, для вычисления суммы (5.4.4) с одновременным преобразованием частного необходимо иметь, кроме сдвигового регистра, только двухразрядную r-ичную вычитающую схему.

Особый интерес представляет случай $r = 2$, соответствующий рассмотренному в § 7.8 широко распространенному двоичному делению без восстановления остатка. Особенностью этого метода является простота выбора цифры a_{i+1}, вычисления нового остатка B_{i+1} и преобразования частного:

если $B_i \geqslant 0$, то $a_{i+1} = 1$, $d_i = d'_i = 1$, $d'_{i+1} = a_{i+1} = 1$,

если $B_i < 0$, то $a_{i+1} = -1$, $d_i = d'_i - 1 = 0$, $d'_{i+1} = r + a_{i+1} = 1$.

Если считать, что $(i+1)$-м циклом называется совокупность следующих действий:

а) выбор a_{i+1} по знаку B_i,

б) выбор d_i и запись d_i в качестве окончательной цифры частного,

в) вычисление $B_{i+1} = 2B_i - a_{i+1}C$, то следует сказать, что всего при делении выполняется $k+1$ цикл, но первый и последний из них — неполные: на первом цикле только вычисляется B_1 (a_1 заведомо равно 1, d_0 не выбирается), на последнем цикле только выбирается d_k (так как a_{k+1} и B_{k+1} не нужны). После завершения деления точные соотношения между делимым, делителем, полученным частным и последним остатком B_k имеют следующий вид:

$$\text{если } B_k \geqslant 0, \text{ то } \frac{A}{C} = h\frac{B_0}{C} = h\sum_{i=1}^{k} d_i 2^{-i} + h\frac{B_k}{2^k C},$$

$$\text{если } B_k < 0, \text{ то } \frac{A}{C} = h\frac{B_0}{C} = h\sum_{i=1}^{k} d_i 2^{-i} + h\frac{B_k + C}{2^k C}.$$

В примерах деления, помещенных в § 7.8, константа h была равна 2.

Ускорение обычного деления без восстановления остатка ($r=2$), как и ускорение деления с восстановлением остатка (см. § 5.1), может быть достигнуто путем выполнения устройства для деления в виде итеративной сети.

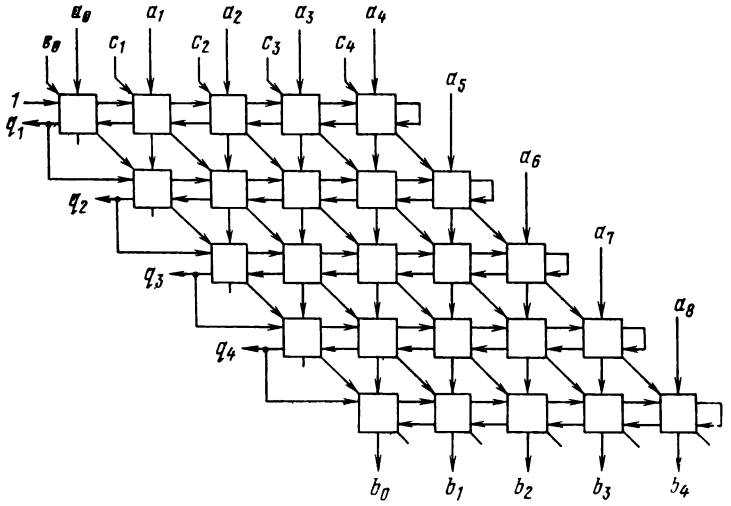


Рис. 5.4.1

На рис. 5.4.1 показана одна из простейших модификаций такой сети, составленной из одинаковых элементов. Каждый элемент (рис. 5.4.2) содержит комбинационный одноразрядный двоичный сумматор См и элемент ИСЛЮЧАЮЩЕЕ ИЛИ, реализующий функцию $c\bar{q}+\bar{c}q$ и управляющий вводом делителя в сумматор. Восьмиразрядное делимое $A=a_0,\ a_1a_2a_3a_4a_5a_6a_7a_8$, четырехразрядный делитель $C=c_0,\ c_1c_2c_3c_4$ и каждый остаток B_i (в частности, последний остаток $B_5=b_0,\ b_1b_2b_3b_4$) представлены в дополнительном коде. Предполагается, что делимое и делитель являются положительными нормализованными дробями (т. е. $a_0=c_0=0$, $a_1=c_1=1$). Так как $1/2\leqslant A<1$, $1/2\leqslant C<1$, то частное $Q=q_1,\ q_2q_3q_4$ — положительное число, лежащее в диапазоне $1/2<Q<2$, т. е. q_1 — целая часть этого числа. Каждая цифра част-

ного q_i является управляющим сигналом для следующей строки матрицы, определяющим, какую операцию — сложение или вычитание — нужно выполнять в этой строке. Вычитание, как видно из рис. 5.4.1, 5.4.2, выполняется путем формирования дополнительного кода делителя.

Ускорение деления в таком устройстве, как и в ранее разобранном устройстве (рис. 5.1.2), достигается не за счет усовершенствования алгоритма операции, а путем использования быстродействующих миниатюрных элементов, соединенных короткими проводниками.

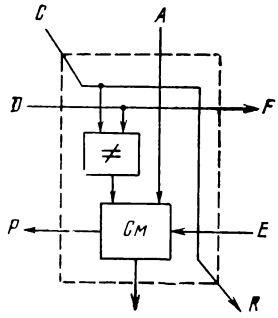


Рис. 5.4.2

Теперь рассмотрим модификацию только что описанной схемы, позволяющую добиться дополнительного увеличения скорости деления [4]. Основная идея модификации состоит в исключении времени распространения переноса вдоль каждой строки матрицы. С этой целью, во-первых, каждый остаток B_i в каждой строке будет вычисляться в виде

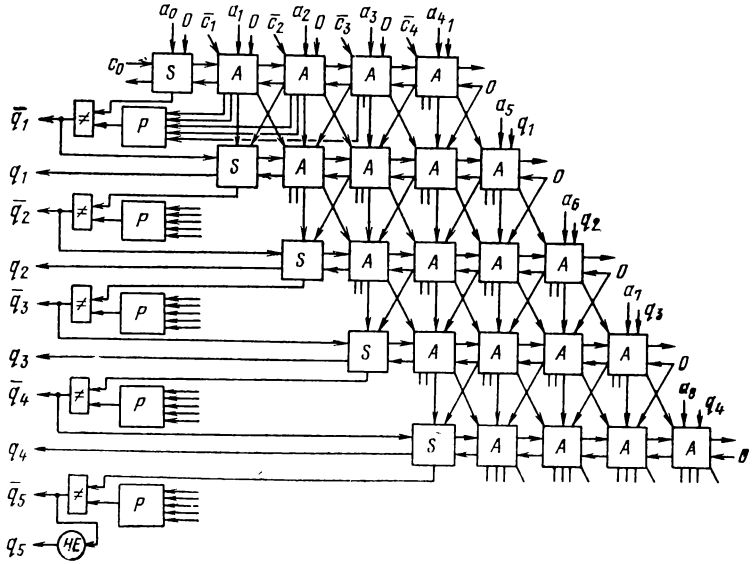


Рис. 5.4.3

двух чисел: числа S, составленного из поразрядных сумм s_j, и числа E, составленного из поразрядных переносов e_j (сумма этих двух чисел равна остатку B_i), и, во-вторых, перенос в знаковый разряд строки, очередной разряд частного и управляющий сигнал для следующей строки (сложить или вычесть) будут формироваться при помощи схемы опережения переносов.

На рис. 5.4.3 показана модифицированная итеративная сеть для 4-разрядного делителя и 8-разрядного делимого. Сеть использует ячейки трех типов (A, S, P) и небольшую дополнительную логику.

Действия, выполняемые на i-й строке матрицы (i==1, 2, ...), соответствуют действиям, производимым во время очередного i-го цикла обычного деления без восстановления остатка: если q_{i-1}=1 (0), то делитель C вычитается (прибавляется) из очередного остатка $2B_{i-1}$. В данном случае эта процедура выполняется следующим образом:

$$\begin{array}{l} 2S = s_0, s_1 s_2 \ldots s_{n-2} \; s_{n-1} \; a_{n+i} \\ 2E = e_0, e_1 e_2 \ldots e_{n-2} \; 0 \; q_{i-1} \end{array} \Big\} 2B_{i-1}$$

$$\pm c_0, c_1 c_2 \ldots c_{n-2} \; c_{n-1} c_n$$

$$\begin{array}{l} S' = s'_0, s'_1 s'_2 \ldots s'_{n-2} \; s'_{n-1} s'_n \\ E' = e'_0, e'_1 e'_2 \ldots e'_{n-2} e'_{n-1} 0 \end{array} \Big\} B_i .$$

Здесь n — разрядность делителя.

Делитель вычитается путем прибавления дополнительного кода числа $-C$; дополнительный код получается инвертированием всех цифр c_j и прибавлением единицы (q_{i-1}=1) в младший разряд.

Ячейки P (рис. 5.4.4) содержат логику для опережающего определения переноса в знаковый (нулевой) разряд очередного остатка. Схема опережения в i-й строке должна выработать сигнал

$$P = D_1 + R_1 D_2 + R_1 R_2 D_3 + \ldots + R_1 R_2 \ldots R_{n-2} D_{n-1},$$

где $D_j = s'_j e'_j$, $R_j = s'_j + e'_j$ — вспомогательные разрядные функции возникновения переноса и разрешения распространения переноса. Очевидно, что эта часть схемы зависит от длины операндов n; для большой длины n необходимо использовать многоступенчатую схему опережения переносов (на рис. 5.4.4 показана ячейка, при использовании которой схема опережения будет одно-

ступенчатой только при $n \leqslant 3$). Ячейка P в i-й строке матрицы вырабатывает перенос в нулевой разряд, получающийся при сложении чисел

$$\begin{array}{r} s'_1 s'_2 \ldots s'_{n-1} \\ + \\ e'_1 e'_2 \ldots e'_{n-1} \end{array}$$

Ячейка S (рис. 5.4.4) i-й строки суммирует цифры s_0, e_0, c_0 (эта сумма равна 0 или 1 в зависимости от того, четно или нечетно количество единиц среди цифр s_0, e_0, c_0); полученная сумма затем прибавляется к переносу

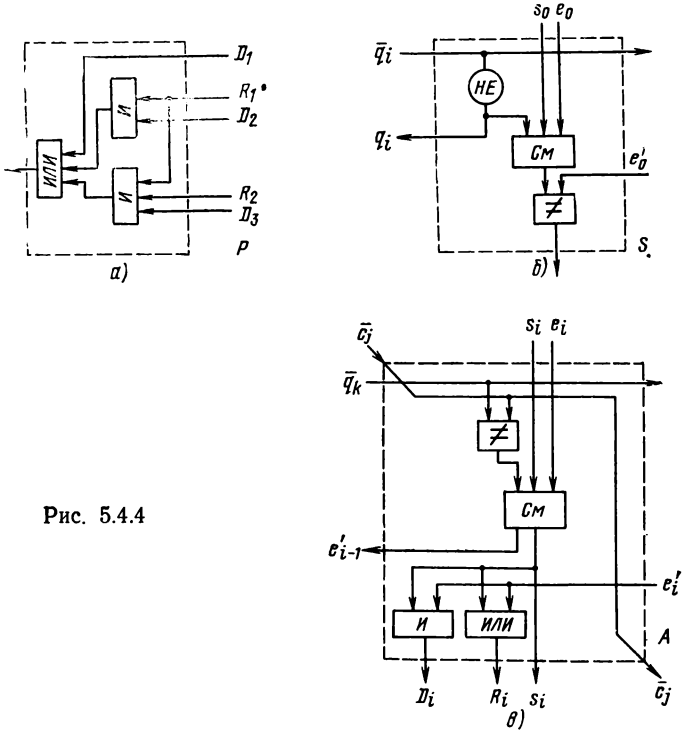


Рис. 5.4.4

e'_0, полученному при сложении с запоминанием переносов в соседнем элементе A (структура такого элемента показана на рис. 5.4.4). Наконец, результат суммируется с переносом, полученным в элементе P. Окончательная сумма является знаковым разрядом остатка B_i, т. е.

величиной $\tilde{q}_i$. Сама цифра q_i формируется инвертором в ячейке S следующей строки.

Между делимым, делителем, частным и последним остатком (в данном случае — остатком B_5) соблюдается, как обычно при делении без восстановления остатка, соотношение

$$A/C=Q+B_5/2^4C \quad (\text{при } B_5\geqslant 0),$$

$$A/C=Q+(B_5+C)/2^4C \quad (\text{при } B_5<0).$$

Следует заметить, что использование только одной цепи опережения переносов в каждой строке обычной матрицы (рис. 5.4.1) не привело бы к таким результатам; понадобилось бы ускорение переносов во всех разрядах остатка, на что потребовалось бы большое количество дополнительной аппаратуры. Именно сочетание формирования остатка в двухрядном коде и получение опережающего переноса в знаковый разряд позволяет достичь значительного ускорения деления при умеренных затратах. Из рис. 5.4.3, 5.4.4 видно, что общее время операции равно

$$m(\tau_A+\tau_P+\tau_{\text{ИИ}}),$$

где m — разрядность частного; τ_A — время задержки одной ячейки A; τ_P — время срабатывания элемента P (здесь предполагается, что схема опережения одноступенчатая); $\tau_{\text{ИИ}}$ — время задержки элемента ИСКЛЮЧАЮЩЕЕ ИЛИ, вырабатывающего цифру $\tilde{q}_i$.

5.4.3. ГРАФОАНАЛИТИЧЕСКИЙ МЕТОД АНАЛИЗА ПРОЦЕССОВ ДЕЛЕНИЯ

Прежде чем рассматривать методы деления без восстановления остатка, использующие наборы, отличные от классического набора (5.4.16), введем в рассмотрение графические изображения зон ξ_j на плоскости B_iC [15, 17]. Эти изображения позволят нам проанализировать связь между величиной r и структурой схемы выбора цифры a_{i+1}.

Будем считать, что границами диапазона (5.4.11) допустимых значений остатков B_i являются величины (5.4.17), т. е. что

$$\frac{\xi_1 C}{r-1}\leqslant B_i\leqslant\frac{\xi_m C}{r-1} \quad (i=1,\ 2,\ \ldots,\ k). \qquad (5.4.20)$$

Это означает, что при данных B_i и C можно в качестве a_{i+1} выбрать некоторую цифру ξ_j из набора ξ, если

$$\frac{\xi_1 C}{r-1} \leqslant B_i r - \xi_j C \leqslant \frac{\xi_m C}{r-1},$$

т. е. если

$$\frac{\xi_1 C}{r(r-1)} + \frac{\xi_j C}{r} \leqslant B_i \leqslant \frac{\xi_m C}{r(r-1)} + \frac{\xi_j C}{r} \qquad (5.4.21)$$

или

$$\frac{\xi_m C}{r-1} - \frac{(\xi_m - \xi_j) C}{r} - \frac{(\xi_m - \xi_1) C}{r(r-1)} \leqslant B_i \leqslant \frac{\xi_m C}{r-1} - \frac{(\xi_m - \xi_j) C}{r}. \qquad (5.4.22)$$

Неравенству (5.4.22) соответствует рис. 5.4.5, на котором на плоскости $B_i C$ заштрихована зона значений величин B_i, C, при которых удовлетворяется (5.4.22). На рисунке отражен некоторый конкретный случай, для которого

$$C_{\min} > 0,\ \xi_1 < 0,\ \xi_j < \xi_m,\ \frac{\xi_m C}{r-1} - \frac{(\xi_m - \xi_j) C}{r} - \frac{(\xi_m - \xi_1) C}{r(r-1)} > 0.$$

Тем не менее, нижеследующее изложение имеет общий характер.

Нетрудно усмотреть соответствие между рис. 5.4.5 и слагаемыми в левой и правой частях двойного неравенства (5.4.22); слагаемое $\frac{(\xi_m - \xi_1) C}{r(r-1)}$ соответствует ширине зоны цифры ξ_j (видно, что эта ширина не зависит от самой величины ξ_j), слагаемое $\frac{(\xi_m - \xi_j) C}{r}$ соответствует расстоянию по горизонтали между правым краем зоны и прямой GL.

Итак, если точка с координатами B_i, C попадает на рис. 5.4.5 в заштрихованную зону, соответствующую некоторой цифре ξ_j, то в качестве a_{i+1} можно выбрать эту цифру. Очевидно, что для того, чтобы процесс деления мог быть реализован при помощи некоторого набора ξ

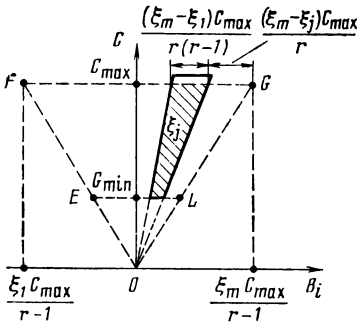


Рис. 5.4.5

цифр ξ_j, каждая точка внутри четырехугольника $EFGL$ на рис. 5.4.5 должна входить в зону хотя бы одной из цифр набора. Для этого необходимо, чтобы на плоскости B_iC зоны соседних цифр набора (5.4.7) перекрывались или хотя бы касались друг друга, т. е. сдвиг между зонами каждых двух соседних цифр ξ_j и ξ_{j+1} должен быть меньше ширины зоны или равен ей. Так как сдвиг между указанными зонами (рис. 5.4.6) равен $\frac{(\xi_{j+1}-\xi_j)C}{r}$, то должно выполняться требование

$$(\xi_{j+1}-\xi_j)\,C/r \leqslant (\xi_m-\xi_1)\,C/r\,(r-1),$$

т. е.

$$\xi_{j+1}-\xi_j \leqslant (\xi_m-\xi_1)/(r-1) \quad (j=1, 2, \ldots, m-1). \quad (5.4.23)$$

Из (5.4.23) можно вывести требование $m \geqslant r$. В самом деле,

$$\xi_2-\xi_1 \leqslant (\xi_m-\xi_1)/(r-1),$$
$$\xi_3-\xi_2 \leqslant (\xi_m-\xi_1)/(r-1),$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$\xi_m-\xi_{m-1} \leqslant (\xi_m-\xi_1)/(r-1).$$

Складывая порознь левые и правые части всех этих неравенств, получим:

$$\xi_m-\xi_1 \leqslant (m-1)\,(\xi_m-\xi_1)/(r-1),$$

откуда следует, что

$$m \geqslant r. \quad (5.4.24)$$

На рис. 5.4.7 показаны зоны цифр набора (5.4.16), используемого при классическом делении (величины ξ_j указаны в верхней части зон). Считается, что $C_{min}=0$. Из рисунка видно, что зоны цифр перекрываются таким образом, что в случаях

$$C(r-1)/r < B_i \leqslant C, \quad 0 < B_i < C/r,$$
$$-C/r < B_i < 0, \quad -C \leqslant B_i < -C(r-1)/r$$

должна выбираться определенная цифра набора, а именно цифра (соответственно) r—1, 1, —1, —(r—1). Для случаев $B_i=\rho C/r$ ($\rho=\pm 2, \pm 3, \ldots, \pm(r-2)$), соответствующих границам зон, может быть выбрана одна из трех цифр: $\pm(\rho-1)$, $\pm\rho$, $\pm(\rho+1)$. Во всех остальных случаях может быть выбрана одна из двух возможных цифр, так как все соответствующие участки плоскости B_iC накрываются двумя зонами. Какая именно из

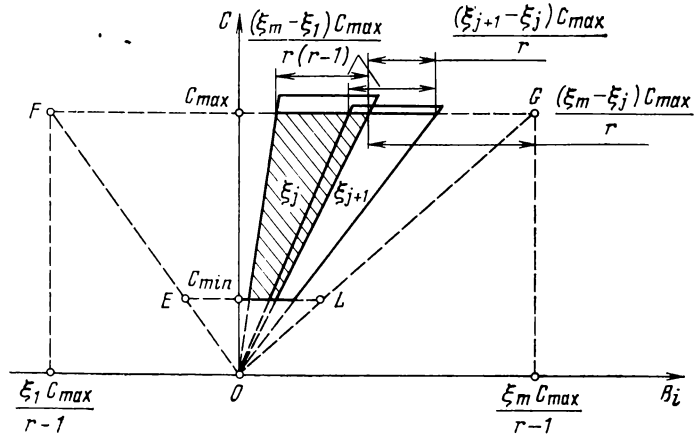


Рис. 5.4.6

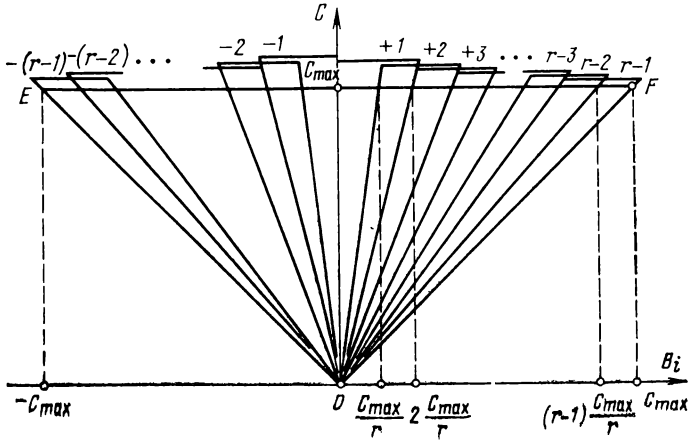


Рис. 5.4.7

этих цифр выбирается, зависит от принятого правила выбора цифр a_i.

Из рис. 5.4.7 видна также характерная особенность набора (5.4.16), заключающаяся в том, что зоны двух средних цифр (+1 и —1) не перекрываются, а только касаются друг друга. Поэтому при выборе a_{i+1} необходимо знать знак остатка B_i и, следовательно, остаток должен вычисляться в арифметическом устройстве в виде одного числа, т. е. однорядного кода (схема на рис. 5.4.3 является исключением).

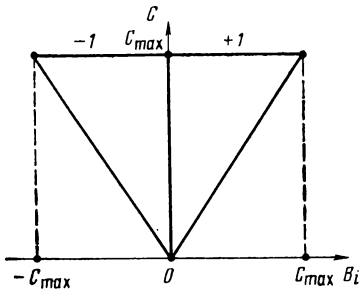


Рис. 5.4.8

На рис. 5.4.8 показаны зоны для частного случая набора (5.4.16), соответствующего обычному двоичному делению без восстановления остатка ($r=2$). В этом случае зоны цифр ξ_j не перекрываются (рис. 5.4.8).

Наличие перекрытия зон цифр ξ_j на плоскости B_iC отражает избыточность набора ξ, которая приводит к возможности неоднозначного представления чисел в системе счисления, использующей цифры ξ_j. Эта возможность, как будет показано в следующем параграфе, позволяет в определенных случаях существенно повысить скорость деления.

5.4.4. ДЕЛЕНИЕ С ИСПОЛЬЗОВАНИЕМ СИММЕТРИЧНОГО НАБОРА ЦЕЛЫХ ЧИСЕЛ, ВКЛЮЧАЮЩЕГО НУЛЬ

Для представления цифр формируемого частного в методах деления без восстановления остатка до недавнего времени, кроме классического набора (5.4.16), применялся еще только один вид наборов ξ:

$$\xi=\{-q, -(q-1), \ldots, -1, 0, 1, \ldots, q-1, q\}, \quad (5.4.25)$$

где q — целое и удовлетворяет неравенству

$$\frac{r-1}{2} \leqslant q \leqslant r-1.$$

Первым из методов деления, использующих наборы этого вида, был метод [18—20], при котором

$$r=2,\ \xi=\{-1,\ 0,\ +1\}. \tag{5.4.26}$$

Кроме того, был описан метод деления [20, 21] с параметрами

$$r=4,\ \xi=\{-2,\ -1,\ 0,\ 1,\ 2\}, \tag{5.4.27}$$

метод [22] с параметрами

$$r=4,\ \xi=\{-3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3\} \tag{5.4.28}$$

и метод [20] с параметрами

$$r=10,\ \xi=\{-7,\ -6,\ -5,\ -4,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7\}. \tag{5.4.29}$$

В общем виде набор вида (5.4.25) был описан в работе [20].

Наборы (5.4.25) при $q>(r-1)/2$ являются избыточными. В работах [18, 19] был предложен метод использования этой избыточности для набора (5.4.26), заклю-

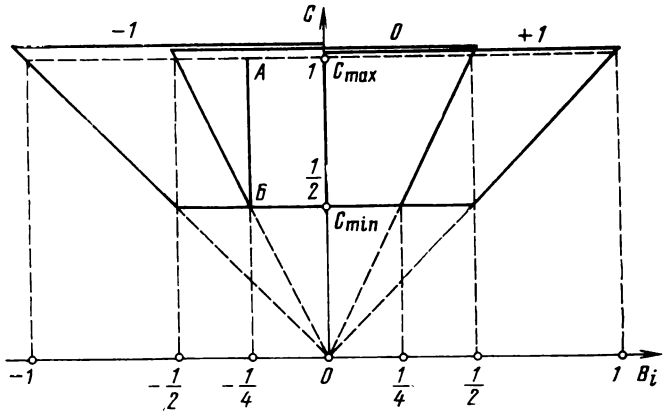


Рис. 5.4.9

чающийся в том, что остаток B_{i+1} стал вычисляться по формуле (5.4.1) в виде двухрядного кода на сумматоре с запоминанием переносов, т. е. в виде ряда поразрядных сумм и ряда поразрядных переносов. Это резко сокращает время одного цикла и, следовательно, время всего деления.

На рис. 5.4.9 показано расположение зон для набора (5.4.26). Избыточность набора приводит к перекрытию зон соседних цифр и позволяет производить выбор a_{i+1} на основе простого правила прямого анализа старших разрядов числа B_i. При нормализованном делителе $(1/2 \leqslant C < 1)$ и остатке, лежащем в диапазоне $-C \leqslant B_i \leqslant C$, для правильного выбора цифры a_{i+1} достаточно знание всего лишь четырех старших разрядов двухрядного кода остатка B_i. Если сложить два указанных четырехразрядных числа, то по полученной четырехразрядной сумме B^*_i можно выбрать цифру a_{i+1}, пользуясь простой таблицей (табл. 5.4.2).

Таблица 5.4.2

B^*_i	a_{i+1}	B^*_i	a_{i+1}
0, XXX 1, 11X	1 0	1, 0XX 1, X0X	—1 —1

Крестиками в таблице обозначены разряды, значения цифр в которых могут быть произвольными. Справедливость правила, отраженного в таблице, подтверждается рис. 5.4.9, из которого видно, что поскольку

$$0 \leqslant B_i - B^*_i < \frac{1}{4},$$

то выбор, например, $a_{i+1}=0$ при $B^*_i=1{,}110$ (т. е. $-1/4$) не может привести к неправильному результату деления, так как все остатки

$$-\frac{1}{4}+0 \leqslant B_i \leqslant -\frac{1}{4}+\frac{1}{4}$$

независимо от величины C лежат в зоне цифры $\xi_2=0$.

Сплошными вертикальными линиями на этом рисунке отделены зоны действия дешифраторов, выбирающих цифру a_{i+1} (при этом следует считать, что по оси абсцисс откладывается величина B^*_i): слева от прямой АБ выбирается $a_{i+1}=-1$, между АБ и осью ординат выбирается $a_{i+1}=0$, справа от оси ординат выбирается $a_{i+1}=+1$. Этим подтверждается очевидное соображение о том, что границы зон действия дешифраторов должны проходить внутри участков перекрытия зон соответствующих цифр ξ_j.

На рис. 5.4.10 показаны зоны цифр ξ_j для еще одного из упомянутых наборов — (5.4.27). Из рисунка видно,

что если a_{i+1} выбирается путем прямого анализа остатка и делителя, то дешифраторы цифр в этом случае должны быть значительно более сложными. Во-первых, видно, что кроме остатка B_i следует анализировать и делитель C, поскольку внутри участка перекрытия зон цифр 1 и 2 нельзя провести вертикальную прямую от прямой EL до прямой FG. Во-вторых, видно, что должно анализироваться сравнительно большое количество разрядов

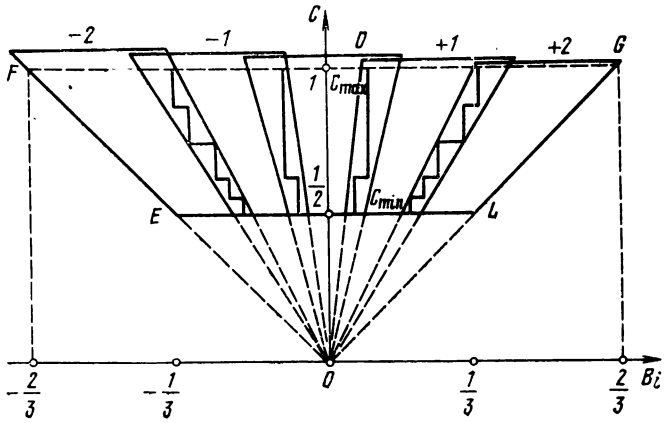


Рис. 5.4.10

чисел B_i и C. В самом деле, допустим для простоты, что B_i вычисляется в виде однорядного кода. Дешифратор любой цифры ξ_j может быть представлен как несколько схем И, выходы которых логически суммируются в схеме ИЛИ. На каждую схему И подаются несколько разрядов чисел B_i и C. При этом каждой такой схеме И на плоскости B_iC соответствует один или несколько прямоугольников, стороны которых параллельны осям координат. Прямоугольники, соответствующие всем схемам И дешифратора некоторой цифры ξ_j, должны полностью покрывать на плоскости B_iC участок, правая и левая границы которого при этом оказываются составленными из «лесенок», которые должны быть расположены внутри участков перекрытия зоны цифры ξ_j с двумя соседними цифрами набора ξ. На рис. 5.4.10 показано одно из возможных разбиений плоскости на зоны действия дешифраторов. Очевидно, что дешифратор получается тем сложнее, чем больше «ступенек» содержат

границы его зоны. Отсюда следует, что дешифратор тем сложнее, чем уже участки перекрытия зоны цифры ξ_j с зонами ξ_{j-1} и ξ_{j+1} и чем больше наклон участков перекрытия.

Для набора (5.4.25) при данных r, q максимальные наклоны имеют участки перекрытия зон крайних пар цифр (ξ_1 и ξ_2, ξ_{m-1} и ξ_m). Ширина участка перекрытия для набора (5.4.25), как это видно из рис. 5.4.6, равна

$$\frac{(\xi_m - \xi_1)C}{r(r-1)} - \frac{(\xi_{j+1} - \xi_j)C}{r} = \frac{2q - r + 1}{r(r-1)} C.$$

При данном r эта величина тем больше, чем больше q. Максимальное значение (при $q=r-1$) равно C/r. Но даже при максимальном q количество ступеней в лесенках быстро растет, если переходить к большим значениям r. Это следует из уменьшения ширины участка перекрытия, равной C/r; ширина всего диапазона остатков при $q=r-1$ и изменении r остается постоянной:

$$\frac{\xi_m C_{\max}}{r-1} = -\frac{\xi_1 C_{\max}}{r-1} = C_{\max}.$$

Нетрудно показать, что и для других типов наборов ξ при использовании метода выбора a_{i+1} путем прямой расшифровки B_i и C дешифраторы усложняются (при прочих равных условиях) с ростом r. Количество m дешифраторов тоже, естественно, растет, так как $m \geqslant r$.

В работе [23] был предложен метод деления с параметрами $r=2$, $\xi=\{-2, -1, 0, +1, +2\}$, в котором остаток B_{i+1} вычисляется в виде двухрядного кода, однако эти два ряда составляются не поразрядными суммами и переносами обычного двоичного сумматора. Остаток B_{i+1} формируется в избыточной двоичной системе счисления ($r=2$) с допустимыми значениями цифр -1, $0+1$. Остаток вычисляется с высокой скоростью, так как при этом отсутствует процесс распространения переносов. Но из-за увеличения количества допустимых цифр в наборе ξ и связанного с этим усложнения входных схем сумматора этот метод, по-видимому, не имеет преимуществ по сравнению с методами, описанными в [18, 19].

Еще один способ выбора цифры a_{i+1} был предложен в работе [20]. Он основан на анализе знаков приближенно вычисляемых разностей

$$B_i - M_j C \ (j=1, 2, \ldots, m-1),$$

где M_j — некоторые константы. Например, для набора (5.4.27) вычисляются разности

$$B_i + \frac{3}{8}C,\ B_i + \frac{1}{8}C,\ B_i - \frac{1}{8}C,\ B_i - \frac{3}{8}C. \quad (5.4.30)$$

На рис. 5.4.11 на плоскости B_iC проведены прямые линии, уравнения которых соответствуют разностям (5.4.30), вычисляемым для выбора цифры частного при

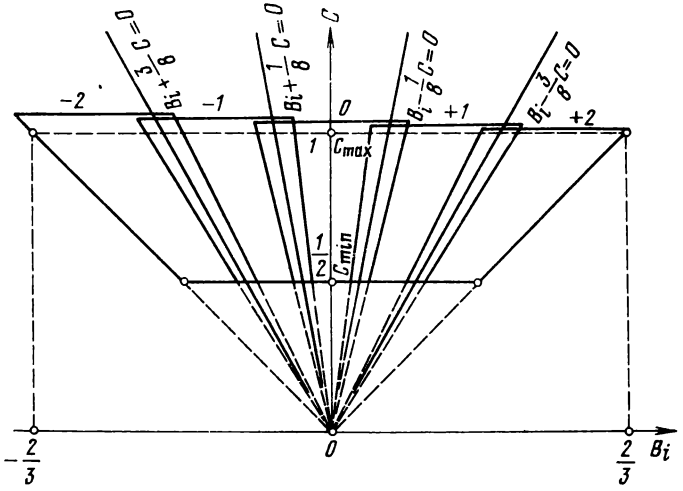


Рис. 5.4.11

использовании набора (5.4.27). Знак разности указывает на то, с какой стороны от соответствующей прямой линии расположена точка с координатами B_i, C. Очевидно, что количество констант M_j, действительно, должно быть равно $m-1$, каждая прямая $B_i - M_jC$ должна проходить внутри участка перекрытия зон цифр ξ_j и ξ_{j+1}, а сами цифры a_{i+1} должны при этом выбираться по правилу (полагаем, что $M_1 < M_2 < \ldots < M_{m-1}$):

$$a_{i+1} = \xi_1, \text{ если } B_i - M_1C < 0,$$

$$a_{i+1} = \xi_m, \text{ если } B_i - M_{m-1}C \geqslant 0,$$

$$a_{i+1} = \xi_j, \text{ если } \begin{cases} B_i - M_jC < 0, \\ B_i - M_{j-1}C \geqslant 0. \end{cases}$$

Из рисунка также следует, что ошибка вычисления разности $B_i - M_j C$ не должна превышать расстояния между прямой $B_i - M_j C = 0$ и границей участка перекрытия двух зон, внутри которой проходит эта прямая.

Поскольку $m \geqslant r$, то в общем с ростом r возрастает количество разностей $B_i - M_j C$; при этом соответственно растут аппаратные трудности и время, требуемое для выбора очередной r-ичной цифры частного.

В [24] был предложен метод, позволяющий добиться дополнительного ускорения деления без восстановления остатка в случае использования набора (5.4.26) при вычислении B_{i+1} в виде двухрядного кода. Предложено использовать матрицу одноразрядных сумматоров, в каждом этаже которой определяется очередная цифра a_{i+1} и вычисляется новый остаток B_{i+1}. Выигрыш в быстродействии при этом может быть весьма большим. В самом деле, при использовании одного n-разрядного сумматора для вычисления двухрядного кода B_{i+1} передача предыдущего остатка с выходов сумматора на его входы (с одновременным сдвигом на один разряд влево) при помощи триггерных регистров или линий задержки может отнять не меньше времени, чем само вычисление нового остатка. Матрица сумматоров в таком устройстве может служить и для выполнения умножения.

При небольших величинах r (например, при $r \leqslant 4$) и двухрядном представлении остатка матрицы сумматоров могут быть, очевидно, использованы и для деления с другими наборами допустимых значений цифр частного.

Из соотношения (5.4.2) следует, что увеличение величины r означает, по существу, пропорциональное увеличение скорости деления (в предположении, что время цикла остается неизменным). Например, при $r=2$, 4, 8, 16 за один цикл определяется соответственно сразу одна, две, три, четыре двоичные цифры частного.

Основным и, по существу, единственным препятствием к увеличению используемой при делении величины r является аппаратная сложность (либо большие затраты времени) выбора очередной цифры a_{i+1}. В этом и предыдущем разделах были описаны три основных способа выбора a_{i+1}:

1) многократное вычитание (сложение) C из rB_i (для классического деления);

2) приближенное вычисление вспомогательных разностей типа $B_i - M_j C$;

3) прямая расшифровка величин B_i и C.

Значительное увеличение потерь времени или аппаратное усложнение схемы выбора a_{i+1} с ростом r при использовании двух первых способов очевидны, поскольку с ростом r увеличивается количество цифр в наборах (5.4.16), (5.4.25) и, следовательно, для обоих способов растет число процедур сложения-вычитания. Об увеличении количества и усложнении дешифраторов выбора a_{i+1} при прямом анализе B_i и C тоже уже говорилось.

В следующем параграфе этой главы будут рассматриваться методы деления с большими значениями r. В этих методах при выборе a_{i+1} будет производиться прямой анализ B_i и C. Следует отметить, что если остаток B_i формируется в виде двухрядного кода, то «лесенка» дешифратора не должна подходить на плоскости $B_i C$ к правой границе участка перекрытия ближе, чем на расстояние, соответствующее максимальной разности между анализируемой частью B^*_i остатка и точным значением остатка B_i. Эта разность поэтому не должна превышать ширины участка перекрытия и даже должна быть существенно меньше ее, так как в противном случае количество «ступенек» и связанная с ним сложность дешифратора могут недопустимо возрасти. При больших r усложняются дешифраторы цифр (даже при однорядном коде остатка). Наличие двухрядного кода остатка в силу указанного обстоятельства еще больше усложняет эти дешифраторы, так как приводит к дополнительному увеличению ступеней в «лесенках». С другой стороны, если на входы дешифраторов поступают не цифры двухрядного кода, а выходы дополнительного сумматора, складывающего несколько старших разрядов двухрядного кода (это делается для упрощения дешифраторов), то требование уменьшить максимальную разность между B^*_i и B_i приводит к необходимости увеличения длины этого сумматора, что, в свою очередь, снижает выигрыш в быстродействии. Поэтому, а также для облегчения дальнейшего изложения, будем полагать, что B_i вычисляется в виде однорядного кода, хотя излагаемые далее методы применимы и в случае двухрядного кода остатка.

5.4.5. ОБОБЩЕННЫЙ МЕТОД ДЕЛЕНИЯ БЕЗ ВОССТАНОВЛЕНИЯ ОСТАТКА И ЕГО ИССЛЕДОВАНИЕ

Здесь описывается предложенный в 1966 г. синхронный метод деления [17], основной особенностью которого является возможность практического использования больших значений r при обеспечении высокой скорости выбора цифр частного a_{i+1}. Этот метод позволил, в частности, создать арифметическое устройство, производящее деление в системе счисления с основанием $r=16$, т. е. определяющее за один цикл сразу 4 двоичных разряда частного [15].

В предыдущем разделе было показано, что попытка серьезно увеличить величину r приводит к значительному усложнению схемы выбора цифры a_{i+1}. Покажем, что способ выбора a_{i+1}, основанный на прямой расшифровке B_i и C, может тем не менее позволить перейти к значительно большим величинам r, если при этом использовать другие наборы ξ.

Такая возможность основана на следующей идее. Поскольку при использовании набора с равноотстоящими цифрами ξ_j дешифраторы отдельных цифр ξ_j усложняются по мере увеличения абсолютной величины ξ_j из-за увеличения наклона границ зоны этой цифры на плоскости B_iC, то следует перейти к таким наборам, в которых перекрытие зон соседних цифр увеличивается по мере роста абсолютных значений этих цифр. При этом увеличение наклона будет в значительной степени компенсироваться увеличением перекрытия, так что количество ступенек в «лесенках» не будет быстро возрастать. Иными словами, нужно перейти от наборов (5.4.16), (5.4.25), содержащих только равноотстоящие числа, к наборам, содержащим неравноотстоящие числа. Возможность такого решения скрыта в неравенстве (5.4.23), являющемся единственным ограничением, наложенным на разности между соседними цифрами набора ξ, и позволяющем при данных ξ_1 и ξ_m включать в набор ξ любые действительные числа, удовлетворяющие этому неравенству.

Но, с другой стороны, расширение участков перекрытия ведет к увеличению числа цифр в наборе ξ, что увеличивает количество вырабатываемых кратных делителя и усложняет входные схемы сумматоров, вычисляю-

щих B_{i+1}. Оптимум (при котором суммарная аппаратура схемы выбора a_{i+1}, схемы вычисления B_{i+1} и схемы преобразования частного минимальна) зависит от используемой системы элементов. Поэтому здесь можно ограничиться лишь общим соображением о том, что при реализации этого метода перекрытия зон *средних* цифр набора ξ должны быть, по-видимому, меньше, чем для набора (5.4.16), так как дешифраторы этих цифр весьма просты и их можно несколько усложнить, получив в качестве компенсации уменьшение количества цифр ξ_j, а для *крайних* цифр набора перекрытия должны быть больше, чем для (5.4.16), хотя при этом и увеличивается число цифр в наборе ξ.

Прежде чем перейти к рассмотрению конкретных наборов ξ, выясним ограничения, накладываемые на крайние цифры этих наборов, т. е. на величины ξ_1 и ξ_m.

Отметим, что частное B_0/C находится в диапазоне, зависящем от h и от диапазона (5.4.15) частного A/C:

$$\frac{1}{h}\left(\frac{A}{C}\right)_{\min}=\left(\frac{B_0}{C}\right)_{\min}\leqslant\frac{B_0}{C}\leqslant\left(\frac{B_0}{C}\right)_{\max}=\frac{1}{h}\left(\frac{A}{C}\right)_{\max}. \tag{5.4.31}$$

С другой стороны, B_0 должно удовлетворять неравенству (5.4.20). Подставив $A/h=B_0$ в (5.4.20), получим требование

$$\xi_1 C/(r-1)\leqslant A/h\leqslant\xi_m C/(r-1), \tag{5.4.32}$$

которое совместно с (5.4.15) определяет соотношение между константами h, ξ_1, ξ_m.

Рассмотрим сначала случай, когда $(A/C)_{\min}=0$, что имеет место при представлении чисел в форме с фиксированной запятой. Из (5.4.32) и (5.4.15) следует, что при этом величины ξ_1, ξ_m должны удовлетворять требованиям

$$\xi_1\leqslant 0, \tag{5.4.33}$$

$$\xi_m\geqslant(A/C)_{\max}(r-1)/h. \tag{5.4.34}$$

Конечно, можно считать, что ξ_1 и ξ_m заданы (причем $\xi_1\leqslant 0$) и вместо неравенства (5.4.34) вывести из (5.4.32) и (5.4.15) неравенство, накладывающее ограничение на константу h:

$$h\geqslant(A/C)_{\max}(r-1)/\xi_m. \tag{5.4.35}$$

Теперь рассмотрим случай, когда $(A/C)_{\min}>0$, что соответствует представлению чисел в форме с плавающей запятой. Из (5.4.32) и (5.4.15) следует, что при этом может быть выбрана как величина $\xi_1\leqslant 0$, так и $\xi_1>0$. Если $\xi_1\leqslant 0$, то ξ_m должна при проектировании выбираться на основе требования (5.4.34) или, что то же, h должна выбираться, исходя из (5.4.35). Если $\xi_1>0$, то из (5.4.32) и (5.4.15) выводится требование

$$(A/C)_{\max}(r-1)/\xi_m\leqslant h\leqslant \leqslant (A/C)_{\min}(r-1)/\xi_1. \qquad (5.4.36)$$

Можно считать, что (5.4.36) указывает допустимые границы для h при заданных ξ_1 и ξ_m. Можно, наоборот считать, что задана величина h, а параметры ξ_1 и ξ_m выбираются с учетом требований

$$\xi_1\leqslant (A/C)_{\min}(r-1)/h,$$
$$\xi_m\geqslant (A/C)_{\max}(r-1)/h. \qquad (5.4.37)$$

Из (5.4.36) видно также, что при $\xi_1>0$ независимо от величины h должно выполняться требование

$$\xi_m/\xi_1\geqslant (A/C)_{\max}/(A/C)_{\min}. \qquad (5.4.38)$$

Проиллюстрируем смысл неравенств (5.4.37) и (5.4.38) на примере деления мантисс A и C чисел, представленных в форме с плавающей запятой. В этом случае

$$N\leqslant A\leqslant M, \quad N\leqslant C\leqslant M, \qquad (5.4.39)$$

где M и N — константы, причем $N>0$. При этом

$$N/M\leqslant A/C\leqslant M/N, \qquad (5.4.40)$$

т. е. требование (5.4.38) в данном случае имеет вид

$$\xi_m/\xi_1\geqslant (M/N)^2.$$

На рис. 5.4.12 заштрихована зона $PRST$ допустимых значений A и C, а на рис. 5.4.13 показана зона $EFGL$ допустимых значений величин C, B_i. При

$$\xi_1=(N/M)(r-1)/h, \quad \xi_m=(M/N)(r-1)/h,$$

т. е. в случае «граничного» выполнения требований (5.4.37), (5.4.38), квадрат $PRST$ после подстановки $B_0=A/h$ преобразуется в заштрихованный прямоугольник на плоскости B_iC. Из рисунков видно, что нарушение любого из неравенств (5.4.37) приводит к тому, что заштрихованный прямоугольник на рис. 5.4.13 начинает выходить за пределы зоны допустимых значений.

Следует отметить, что оба использованных подхода (при одном считалось, что заданы ξ_1 и ξ_m, а при другом, — что задано h) отличаются только методологически.

Рассматривая зависимость между h и ξ_1, ξ_m, следует отметить, что принятые соотношения (5.4.1), (5.4.3) допускают неоднозначное

описание одного и того же метода деления. В самом деле, процесс деления делимого A на делитель C с использованием набора ξ цифр ξ_j при определенной константе h и остатках B_i, лежащих в диапазоне (5.4.20), описываемый соотношениями (5.4.1)—(5.4.3), (5.4.7), (5.4.8), (5.4.11), (5.4.13)—(5.4.15), (5.4.20), будет по-прежнему описываться этими соотношениями, если вместо h, ξ_j, B_i (в том числе B_0) будем рассматривать величины h/φ, $\varphi\xi_j$, φB_i, где φ — любая

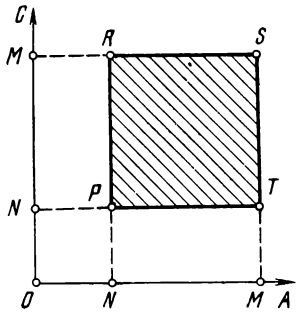


Рис. 5.4.12

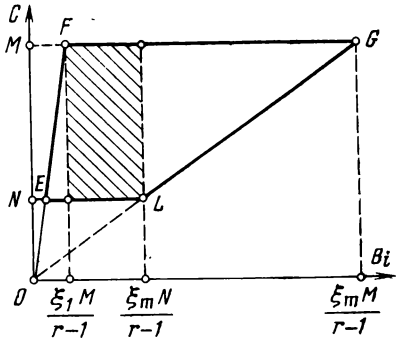


Рис. 5.4.13

положительная константа. Так, например, метод деления, определяемый соотношениями

$$r=4;\ \xi=\left\{-2,\ -1\frac{1}{8},\ 0,\ 1\frac{1}{8},\ 2\right\};$$

$$0\leqslant A\leqslant 1;\ \ \frac{1}{2}\leqslant C\leqslant 1;\ \ -\frac{2}{3}C\leqslant B_i\leqslant\frac{2}{3}C;\ \ h=4,$$

и метод

$$r=4;\ \xi=\left\{-4,\ -2\frac{1}{4},\ 0,\ 2\frac{1}{4},\ 4\right\};$$

$$0\leqslant A\leqslant 1;\ \ \frac{1}{2}\leqslant C\leqslant 1;\ \ -\frac{4}{3}C\leqslant B_i\leqslant\frac{4}{3}C;\ \ h=2$$

это один и тот же метод, реализуемый одной и той же аппаратурой.

Чтобы добиться однозначности описания, будем в дальнейшем записывать набор ξ, как это по существу сделано для наборов (5.4.16), (5.4.25), таким образом (т. е. подбирать такую константу φ), чтобы, во-первых, все величины ξ_j оказались целыми (это возможно, поскольку мы имеем дело с рациональными числами ξ_j) и, во-вторых, чтобы наибольший общий делитель чисел ξ_j был равен единице. Следовательно, только что приведенный пример следует записать так:

$$r=4;\ \xi=\{-16,\ -9,\ 0,\ 9,\ 16\};$$

$$0 \leqslant A \leqslant 1; \quad \frac{1}{2} \leqslant C \leqslant 1; \quad -\frac{16}{3} C \leqslant B_i \leqslant \frac{16}{3} C; \quad h = \frac{1}{2}. \tag{5.4.41}$$

Перейдем теперь к рассмотрению некоторых наборов ξ с переменным «шагом» между цифрами ξ_j. Набор (5.4.41) является первым из таких примеров. Располо-

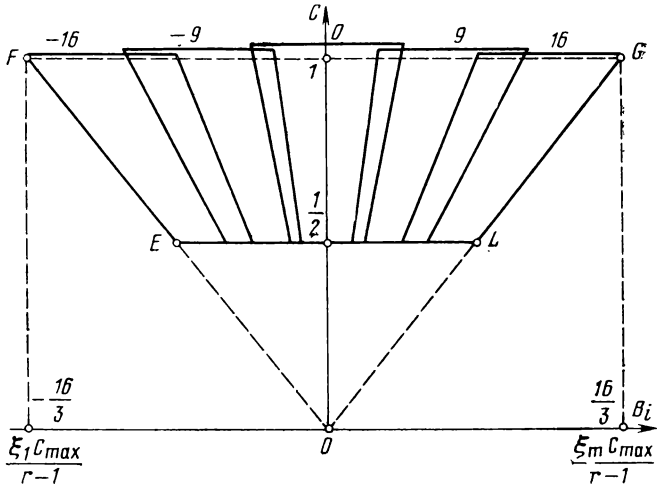


Рис. 5.4.14

жение зон цифр на плоскости B_iC для этого набора показано на рис. 5.4.14. Из рисунка видно, что благодаря переменному «шагу» между цифрами ξ_j ширина участка перекрытия между зонами 9 и 16 шире, чем между зонами 0 и 9. Сравнив рис. 5.4.14 и 5.4.10, увидим, что перекрытие зон 0 и 9 меньше, чем 0 и 1, но зато перекрытие крайних зон 9 и 16 больше, чем 1 и 2.

Другой пример симметричного набора с неравномерным расположением цифр приведен на рис. 5.4.15 для случая $r=16$:

$$\xi = \{-30, -29, -28, -27, -26, -25, -24, -23, -22, -21, -19, -17, -15, -13, -11, -8, -5, -2, 2, 5, 8, 11, 13, 15, 17, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30\}. \tag{5.4.42}$$

Зоны отрицательных цифр для упрощения рисунка показаны не полностью.

На этом и на некоторых последующих рисунках в кружках указаны количества ступеней в идеальных «лесенках», разделяющих зоны соседних цифр набора. Под идеальной условно понимается такая лесенка, все углы которой касаются границ участка перекрытия, т. е.

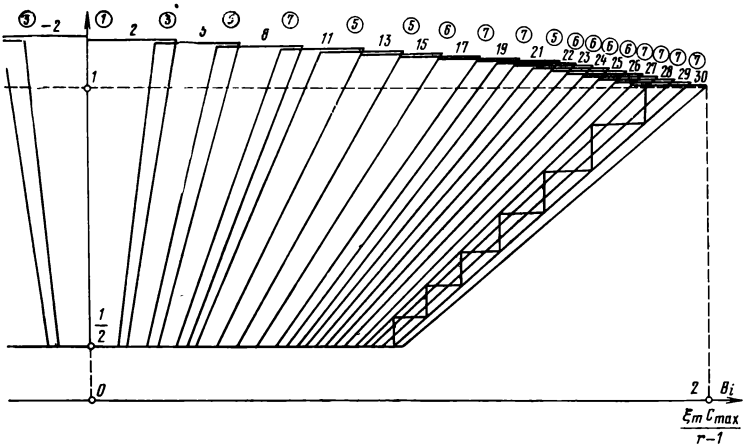


Рис. 5.4.15

«лесенка», имеющая минимальное количество ступеней. Эти величины позволяют оценивать сложность соответствующих дешифраторов, выбирающих цифры частного *).

Максимально допустимое количество ступеней в «лесенке» зависит от системы элементов, от требуемого быстродействия и других обстоятельств. Допустим, например, что в идеальных «лесенках» должно быть не более 5 ступеней.

Несмотря на значительное перекрытие зон крайних цифр набора (5.4.42), количество ступеней в идеальных «лесенках» все же велико — до 7 ступеней. Из

*) Можно отметить, что после появления работы [17] предложения об использовании графических построений зон цифр частного на плоскости B_iC и об оценке аппаратных затрат на схему выбора цифр частного по количеству ступеней в «лесенках» (правда, только для наборов (5.4.25)) были сделаны также в работах [22, 25, 16].

рис. 5.4.15 видно, что дальнейшее увеличение перекрытия зон не может дать существенного уменьшения количества ступеней. В самом деле. как бы ни была мала, например, разность $\xi_m - \xi_{m-1}$, все равно «лесенка» между зонами этих двух цифр будет содержать не менее 6 ступеней. Но и 6 ступеней, как было условлено выше, слишком много. Рассмотрим способ борьбы с этой трудностью.

Для дальнейшего изложения желательно более точная формулировка правила изображения «лесенки», т. е. границы между зонами действия дешифраторов двух цифр набора ξ на плоскости B_iC. Уточнение касается точек, принадлежащих непосредственно самим отрезкам прямых, составляющих «лесенку». Правило поясняется рис. 5.4.16. Стрелками показано, к какой зоне—правой или левой—считаются относящимися точки, расположенные на отрезках «лесенок». Точки плоскости, расположенные вне самой «лесенки» справа или слева от нее, относятся, естественно, к правой и левой зонам. Как было указано выше, «лесенка», разделяющая зоны действия дешифраторов двух соседних цифр ξ_j, ξ_{j+1} набора ξ, должна располагаться внутри участка перекрытия зон этих двух цифр, может касаться границ участка перекрытия, но не должна выходить из них.

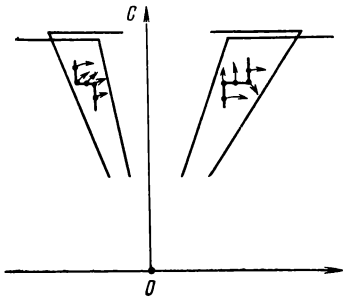


Рис. 5.4.16

В дальнейшем, при изображении зон цифр в других координатах будем придерживаться этого же правила.

Теперь покажем, как можно упростить дешифраторы, выбирающие цифры a_i. Пусть на плоскости B_iC изображена (рис. 5.4.17) зона $EFGL$ допустимых остатков и делителей. Границы зон отдельных цифр ξ_j являются прямыми линиями, уравнения которых записываются в виде

$$B_i = KC,$$

где

$$K = \frac{\xi_m}{r(r-1)} + \frac{\xi_j}{r}$$

для правой границы зоны ξ_j и

$$K = \frac{\xi_1}{r(r-1)} + \frac{\xi_j}{r}$$

для левой границы.

Изобразим всю эту систему линий в новой системе координат, в которой по осям отложены величины B_i—HC и C (рис. 5.4.18), где H — положительная константа. На плоскости B_iC (см. рис. 5.4.17) величине H соответствует прямая $0C'$ с уравнением

$$B_i - HC = 0.$$

Четырехугольник $EFGL$ «переходит» с плоскости B_iC на плоскость B_i—HC, C таким образом, что отрезки FG и EL сохраняют свою длину. То же самое происходит с зоной любой цифры ξ_j: не изменяется ни ее ширина (при данном C), ни ширина участков перекрытий. Но наклоны границ всех зон изменяются. В частности, наклоны тех зон, которые на рис. 5.4.17 были расположены справа от $0C'$ или слева от $0C'$, но ближе к $0C'$, чем к $0C$,

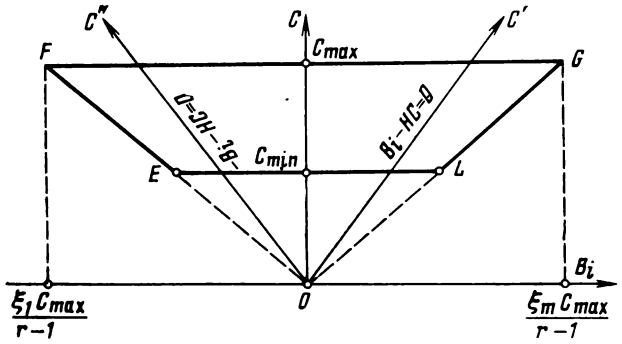


Рис. 5.4.17

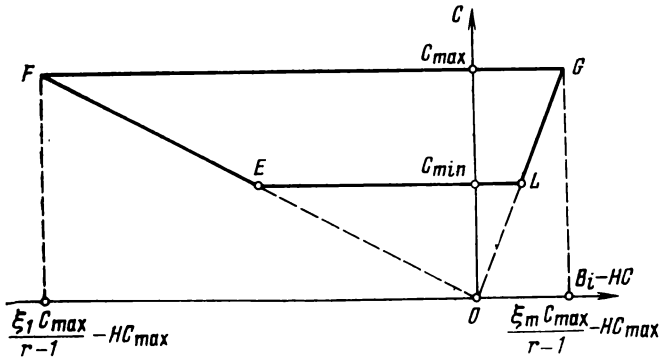


Рис. 5.4.18

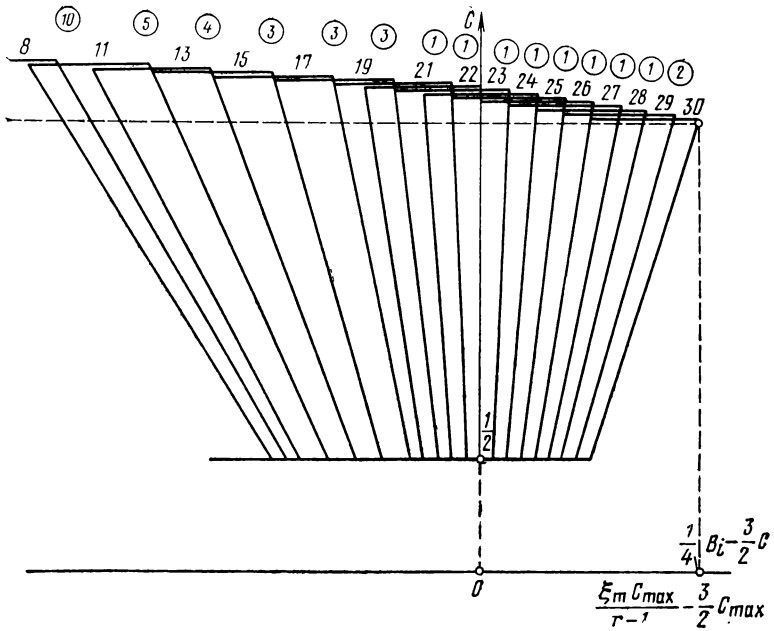


Рис. 5.4.19

изменяются таким образом, что новые «лесенки», проведенные для этих зон в новой системе координат, имеют меньшее количество ступеней, чем в старой системе (предполагается, что каждый из отрезков прямых, составляющих «лесенку», параллелен одной из осей координат). Аналогичным образом можно на рис. 5.4.17 слева от $0C$ так провести другую прямую $0C''$ с уравнением

$$-B_i - HC = 0,$$

чтобы в полученной третьей системе координат $-B_i - HC$, C левые зоны цифр ξ_j заняли выгодное положение.

Для примера на рис. 5.4.19 показаны в системе координат $B_i - \frac{2}{3}C$, C ранее приведенные на рис. 5.4.15 зоны крайних цифр набора (5.4.42). Из рисунка видно, что количество ступеней в идеальных «лесенках» для цифр 13, 15, 17, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 резко сократилось. Точно такое же положение на пло-

скости $-B_i-\frac{3}{2}C$, C занимают зоны отрицательных цифр —13, —15, —17, —19, —21, —22, —23, —24, —25, —26, —27, —28, —29, —30.

Из сказанного следует, что если в устройстве для деления, использующем набор (5.4.42), кроме остатка B_i, вычислять разности $B_i-\frac{3}{2}C$, $-B_i-\frac{3}{2}C$ и построить дешифраторы, выбирающие цифры 13, 15, 17, ..., 30 путем расшифровки старших разрядов величин $B_i-\frac{3}{2}C$, C, цифры —13, —15, —17, ..., —30 — путем расшифровки величин $-B_i-\frac{3}{2}C$, C и цифры —8, —5, —2, 2, 5, 8 — путем обычной расшифровки B_i и C, то дешифраторы крайних цифр набора значительно упростятся *).

Так как теперь ступеней в «лесенках» получается совсем мало, то целесообразно уменьшить количество цифр в наборе ξ, что в ряде случаев может дать существенный выигрыш в аппаратуре. Рассматриваемый набор (5.4.42) можно заменить, например, набором

$$\xi=\{-30, -27, -24, -21, -18, -16, -14, -12, -11, -9, -7, -5, -2, 2, 5, 7, 9, 11, 12, 14, 16, 18, 21, 24, 27, 30\}, \qquad (5.4.43)$$

расположение зон которого на плоскостях B_iC; $B_i-\frac{3}{2}C$, C; $-B_i-\frac{3}{2}C$, C показано на рис. 5.4.20 и 5.4.21, причем на рис. 5.4.21 наложены друг на друга два одинаковых рисунка — один для зон положительных цифр в координатах $B_i-\frac{3}{2}C$, C, другой — для зон отрицательных цифр в координатах $-B_i-\frac{3}{2}C$, C.

Цифры —11, —9, —7, —5, —2, 2, 5, 7, 9, 11 можно выбирать путем прямой расшифровки величин B_i и C, цифры 14, 16, 18, 21, 24, 27, 30 — путем анализа B_i —

*) Для цифр +11, —11 можно не строить дешифраторов, подобных остальным, а выбирать эти цифры по следующему простому правилу: $a_{i+1}=+11$ (—11), если $B_i\geqslant 0$ ($B_i<0$) и при этом не выбрана ни одна другая цифра набора ξ.

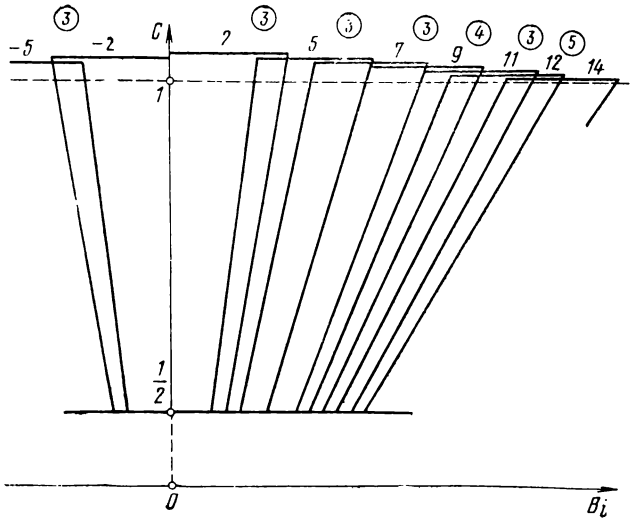


Рис. 5.4.20

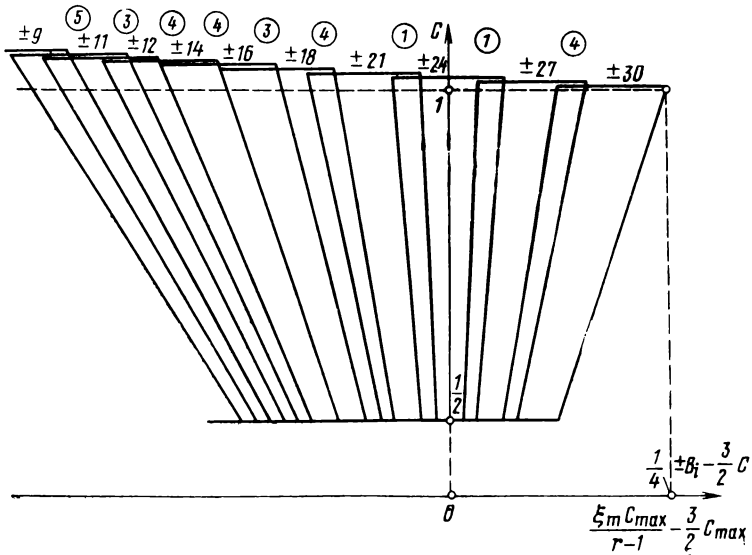


Рис. 5.4.21

$-\frac{3}{2}C$ и C и цифры $-\dot{3}\dot{0}$, $-\dot{2}\dot{7}$, $-\dot{2}\dot{4}$, $-\dot{2}\dot{1}$, $-\dot{1}\dot{8}$, $-\dot{1}\dot{6}$, -14 — путем анализа $-B_i-\frac{3}{2}C$ и C. Из рисунков видно, что при этом все идеальные «лесенки» содержат не более четырех ступеней. Цифры -12, и $+12$ можно выбирать по правилу, аналогичному тому, которое было приведено выше для цифр $+11$, -11 набора (5.4.42).

Рассмотрим вопрос о количестве двоичных разрядов в дополнительных сумматорах, вычисляющих разность типа $\pm B_i-HC$. Нет необходимости в том, чтобы эти разности вычислялись точно. В указанных сумматорах достаточно обрабатывать величины, представленные несколькими старшими разрядами чисел B_i, HC. Однако при этом должны быть соответствующим образом изменены правила построения дешифраторов цифр ξ_j и правила проведения «лесенок» внутри участков перекрытия зон соседних цифр набора.

Пусть 2^l и 2^{-l} — веса старшего и младшего значащих двоичных разрядов сумматора, вычисляющего разность типа $\pm B_i-HC$. Будем считать, что отрицательные числа — HC подаются на входы сумматора в обратном коде и что цепь кольцевого переноса в этом сумматоре отсутствует. Условимся также, что положительные и отрицательные числа на выходе сумматора представляются соответственно в прямом и дополнительном коде (дополнение до 2^{l+1}). Приближенное число $\pm B_i-HC$, получаемое на выходах сумматора, будем в отличие от точного числа $\pm B_i-HC$ обозначать в виде $(\pm B_i-HC)^*$.

Величины B_i и HC можно представить в следующей форме:

$$B_i=K_1 2^{-l}+\beta,\quad HC=K_2 2^{-l}+\gamma,$$

где K_1 и K_2 — целые, причем $K_2>0$, а β и γ удовлетворяют неравенствам

$$0\leqslant\beta<2^l,\quad 0\leqslant\gamma<2^l.$$

Вначале рассмотрим вычисление разности B_i-HC. Поскольку эта разность используется для выбора a_{i+1} только при $B_i>0$, то можно считать, что число B_i положительно и представлено в прямом коде. Будем также считать, что число B_i находится в некотором регистре B. Старшие разряды числа B_i поступают из регистра B в сумматор B_i-HC также в прямом коде. Точная разность B_i-HC равна $B_i-HC=(K_1-K_2)2^{-l}+(\beta-\gamma)$. В сумматоре при этом складываются числа $K_1 2^{-l}$ и $(2^{l+1}-K_2 2^{-l}-2^{-l})$ *).

Если $K_1>K_2$, то на выходах сумматора получается неотрицательное число $(B_i-HC)^*=(K_1-K_2)2^{-l}-2^{-l}$.

---

*) Так как максимальное значение B_i равно $\xi_m C_{\max}(r-1)$, то должно удовлетворяться требование

$$2^{l+1}>\xi_m C_{\max}/(r-1)\geqslant 2^l.$$

Если $K_1 \leqslant K_2$, то на выходах сумматора получается число

$$2^{t+1} - (K_2 - K_1) 2^{-l} - 2^{-l},$$

что соответствует представлению в дополнительном коде отрицательного числа

$$(B_i - HC)^* = (K_1 - K_2) 2^{-l} - 2^{-l}.$$

В обоих случаях

$$0 < B_i - HC - (B_i - HC)^* = 2^{-l} + (\beta - \gamma) < 2 \cdot 2^{-l}. \qquad (5.4.44)$$

Следовательно, число $B_i - HC$ лежит на числовой оси правее числа $(B_i - HC)^*$ на расстоянии, меньшем, чем цена двух единиц младшего разряда сумматора $B_i - HC$.

Теперь перейдем к сумматору $-B_i - HC$. Поскольку этот сумматор используется при $B_i < 0$, то будем считать, что $K_1 < 0$. Точная разность $-B_i - HC$ равна

$$-B_i - HC = (-K_1 - K_2) 2^{-l} - (\beta + \gamma).$$

Допустим также, что число B_i представлено в регистре B в дополнительном коде:

$$2^{t+1} - |B_i| = 2^{t+1} + K_1 2^{-l} + \beta$$

и что старшие разряды регистра B инвертируются перед передачей в сумматор $-B_i - HC$.

В сумматоре складываются числа

$$(2^{t+1} - 2^{t+1} - K_1 2^{-l} - 2^{-l}) = (-K_1 2^{-l} - 2^{-l}) \quad \text{и} \quad (2^{t+1} - K_2 2^{-l} - 2^{-l})\ ^*).$$

Если $-K_1 \geqslant K_2 + 2$, то на выходах сумматора получается неотрицательное число

$$(-B_i - HC)^* = (-K_1 - K_2) 2^{-l} - 2 \cdot 2^{-l}.$$

Если $K_1 \leqslant K_2 + 1$, то результат сложения равен

$$2^{t+1} - (K_1 + K_2) 2^{-l} - 2 \cdot 2^{-l},$$

что соответствует представлению в дополнительном коде отрицательного числа

$$(-B_i - HC)^* = (-K_1 - K_2) 2^{-l} - 2 \cdot 2^{-l}.$$

В обоих случаях

$$0 < B_i - HC - (B_i - HC)^* = 2 \cdot 2^{-l} - (\beta + \gamma) \leqslant 2 \cdot 2^{-l},$$

т. е. опять число $-B_i - HC$ лежит на числовой оси правее числа $(-B_i - HC)^*$ на расстоянии, не превышающем цены двух единиц младшего разряда сумматора $-B_i - HC$.

Из сказанного, а также из рис. 5.4.16 вытекает следующее правило построения «лесенок» внутри участков перекрытия зон цифр ξ_j на плоскости $|B_i| - HC$, C: «лесенки» не должны переходить левую границу зоны перекрытия и не должны подходить к правой границе

---

*) Так как максимальное значение $|B_i|$ при $B_i < 0$ равно $|\xi_1| C_{\max}/(r-1)$, то $2^{t+1} > |\xi_1| C_{\max}/(r-1) \geqslant 2^t$.

ближе, чем на расстояние, равное цене одной единицы младшего разряда сумматора $\pm B_i$—HC.

При необходимости (например, при малых перекрытиях зон цифр ξ_j) можно вычислять несколько разностей типа $\pm B_i$—HC с различными константами H; каждая из этих разностей может использоваться для построения дешифраторов определенной группы цифр ξ_j.

Следует отметить одну тонкость в толкозании графического изображения зон цифр и зон действия дешифраторов в координатах $|B_i|$—HC, C: границы зон цифр изображаются в предположении, что по оси абцисс откладывается точная разность $|B_i|$—HC, а границы зон действия дешифраторов строятся в предположении, что по оси абсцисс указывается величина, получающаяся на выходах сумматора $|B_i|$—HC, т. е. величина $(|B_i|—HC)$*.

Обычно узлы устройства для деления значительно упрощаются при симметричном наборе ξ (т. е. если $\xi_j = -\xi_{m-j+1}$; $j=1, 2, \ldots, m/2$ при четном m; $j=1, 2, \ldots, (m-1)/2$ при нечетном m). В частности, на пару симметричных цифр ξ_j и ξ_{m-j+1} может быть построен один общий дешифратор, как это показано на рис. 5.4.22. Сигналы $+B$, $-B$ являются выходными сигналами знакового разряда регистра B, т. е. $+B=1$ при $B_i \geqslant 0$, а $-B=1$ при $B_i<0$. Если набор ξ симметричен, то упрощается также узел вычисления B_{i+1} и узел преобразования частного.

Описанное в этом разделе использование приближенных разностей типа $\pm B_i$—HC существенно отличается от использования аналогичных разностей, описанного в работе [20] (см. п. 5.4.4). Там приближенные разности вычислялись только для определения знаков этих разностей и количество вычисляемых разностей быстро росло с увеличением m. В нашем случае разность $(\pm B_i$—$HC)$ * нужна для построения дешифраторов; поэтому используется не только ее знак, но и величина. Количество разностей невелико; даже для $m=20$—30 достаточно вычислить две разности типа $\pm B_i$—HC.

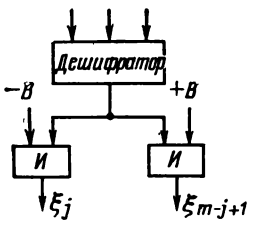


Рис. 5.4.22

Рассмотрим кратко проблему обнаружения переполнения частного, которая возникает при делении чисел с фиксированной запятой.

До сих пор в этой главе мы считали, что делимое A и делитель C находятся в диапазонах (5.4.13) и (5.4.14),

причем $C_{\min}>0$. Однако при делении чисел с фиксированной запятой истинные делимое A^* и делитель C^* расположены в диапазоне

$$0\leqslant A^*\leqslant R,\ 0\leqslant C^*\leqslant R \qquad (5.4.45)$$

и поэтому перед началом деления A^* и C^* должны быть преобразованы в A и C. При этом должно сохраниться отношение этих двух величин.

Сигнал переполнения должен вырабатываться, если частное превышает величину R. Это может произойти как при $C^*=0$, так и при $C^*\neq 0$. Первая ситуация не вызывает каких-либо аппаратных трудностей. Рассмотри внимательнее случай $C^*\neq 0$.

Самой простой реализацией процедуры преобразования A^* и C^* в A и C является нормализация делителя C^* и такой же сдвиг влево делимого A^*:

$$A=p^fA^*, \qquad C=p^fC^*. \qquad (5.4.46)$$

Здесь p — основание системы счисления, используемой для представления чисел в машине; f — количество нулей перед старшей значащей цифрой в n-разрядном числе C^*. Очевидно, что если $C^*\neq 0$, то

$$0\leqslant f\leqslant n-1. \qquad (5.4.47)$$

Будем считать, что A^* и C^* преобразуются в A и C при помощи нормализации.

Переполнение можно обнаруживать различными способами. Например, можно использовать следующий прием. Так как $C\leqslant R$, то будем сравнивать A и R^2. Если

$$A>R^2, \qquad (5.4.48)$$

то будем считать, что имеет место переполнение. Но если $A\leqslant R^2$, то это еще не означает, что переполнение отсутствует.

Потребуем, чтобы при $A\leqslant R^2$ автоматически обеспечивалось выполнение неравенства

$$B_0\leqslant \xi_m C_{\min}/(r-1). \qquad (5.4.49)$$

Поскольку при делении с фиксированной запятой $(A/C)_{\min}=0$, то $\xi_1\leqslant 0$ и поэтому выполнение неравенства (5.4.49) означает, что точка с координатами B_0, C попадает на плоскости B_iC в зону допустимых значений.

Так как теперь мы считаем, что $A \leqslant R^2$, то, подставив $B_0 = A/h = R^2/h$ в (5.4.49), получим требование

$$h \geqslant R^2(r-1)/C_{\min}\xi_m. \qquad (5.4.50)$$

На рис. 5.4.23 заштрихован прямоугольник на плоскости AC, в котором может находиться точка с координатами A, C, и прямоугольник на плоскости B_iC, в который может попасть точка B_0, C при

$$h = R^2(r-1)/C_{\min}\xi_m.$$

Таким образом, константу h следует выбирать с учетом требования (5.4.50). При этом деление будет правильно выполняться все k циклов (т. е. все B_i будут на-

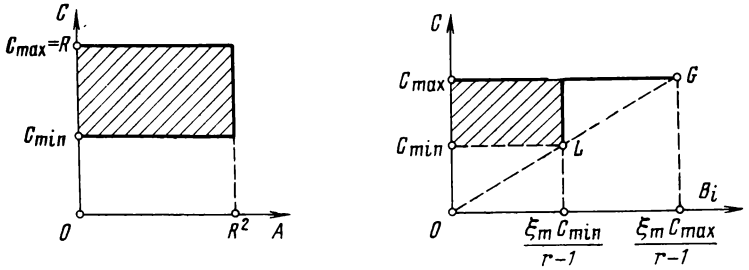


Рис. 5.4.23

ходиться в диапазоне допустимых значений), после чего останется только проверить неравенство

$$hQ > R, \qquad (5.4.51)$$

соблюдение которого следует расценивать как признак переполнения.

В п. 5.4.6 будет описан пример применения такого способа обнаружения переполнения.

5.4.6. ПРИМЕР РЕАЛИЗАЦИИ ОБОБЩЕННОГО МЕТОДА ДЕЛЕНИЯ БЕЗ ВОССТАНОВЛЕНИЯ ОСТАТКА

Рассмотрим кратко структуру устройства для деления, построенного на основе результатов, изложенных в данной главе.

Метод деления, реализованный в указанном арифметическом устройстве, и само устройство обладают следующими параметрами [15, 17]: $p=2$, $n=28$, запятая фиксирована перед старшим разрядом, $r=16$, $h=1$, $k=7$,

$$\xi = \{-30, -29, -28, -27, -26, -24, -22, -19, -16, -13, -10, -8, -5, -2, 2, 5, 8, 10, 13, 16, 19, 22, 24, 26, 27, 28, 29, 30\}. \qquad (5.4.52)$$

Очевидно, что $R=1-2^{-28}$ и поэтому исходные делимое A^* и делитель C^* расположены в диапазонах

$$0\leqslant A^*\leqslant 1-2^{-28},\quad 0\leqslant C^*\leqslant 1-2^{-28}$$

(случай $C^*=0$ здесь не рассматривается).

После нормализации чисел A^* и C^* величины A и C расположены в диапазонах

$$0\leqslant A\leqslant 2^{+27}-2^{-1},\quad 2^{-1}\leqslant C\leqslant 1-2^{-28},$$

т. е. $C_{min}=2^{-1}$, $C_{max}=1-2^{-28}$.

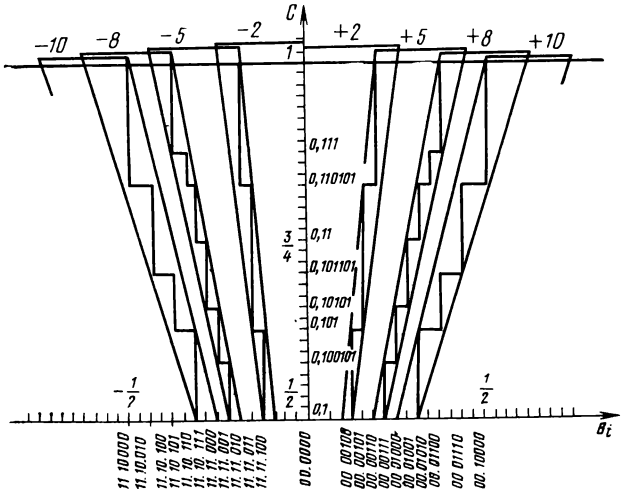


Рис. 5.4.24

Если $A\geqslant 1$, т. е. если $A>R^2+2^{-28}$, то вырабатывается сигнал переполнения. Очевидно, что при этом $A^*/C^*>1$. Если же

$$0\leqslant A\leqslant 1-2^{-28},\quad 2^{-1}\leqslant C\leqslant 1-2^{-28},$$

то деление идет верно все 7 циклов, так как выбранная величина h обеспечивает выполнение неравенства (5.4.49).

В конце операции деления производится проверка неравенства (5.4.51), имеющего в данном случае вид

$$hQ\geqslant 1.$$

Если $hQ\geqslant 1$, то тоже вырабатывается признак переполнения.

Диапазон (5.4.20) допустимых значений остатков B_i имеет вид

$$-2C\leqslant B_i\leqslant 2C.$$

Средние цифры набора ξ (цифры —8, —5, —2, 2, 5, 8) выбираются при помощи дешифраторов, анализирующих старшие разряды величин B_i, C. Расположение зон этих цифр и зон действия соответствующих дешифраторов на плоскости B_iC показано на рис. 5.4.24.

Для выбора крайних цифр набора ξ (цифры —30, —29, —28, —27, —26, —24, —22, —19, —16, —13, 13, 16, 19, 22, 24, 26, 27, 28, 29, 30) в устройстве применены два дополнительных сумматора, вычисляющих разности типа $\pm B_i - HC$, а именно, разности

$$B_i - C, \quad -B_i - C.$$

Расположение зон крайних цифр набора (5.4.52) и зон действия соответствующих дешифраторов на плоскости $|B_i| - C$, C показано на рис. 5.4.25. Дешифраторы крайних цифр построены в соответствии с рис. 5.4.22, т. е. попарно объединены.

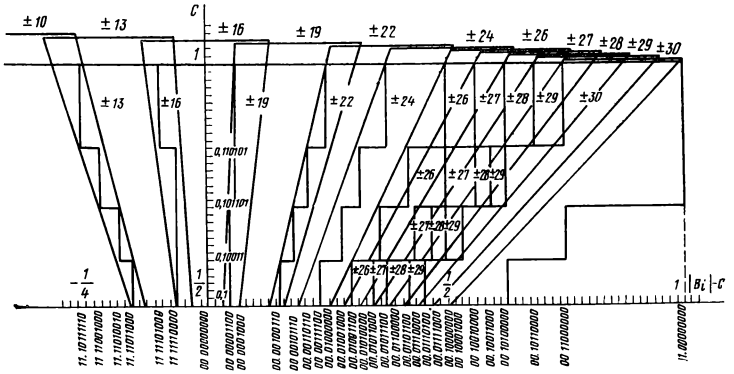


Рис. 5.4.25

Сдвоенный дополнительный сумматор $|B_i| - C$ содержит 9 разрядов; веса старшего и младшего разрядов этого сумматора соответственно равны 1 и 2^{-8}.

Поэтому «лесенки» дешифраторов на рис. 5.4.25 не должны подходить (и не подходят) к правой границе участка перекрытия зон ближе чем на 2^{-8}.

Цифра —10 (+10) выбирается автоматически, если $B_i < 0$ ($B_i \geqslant 0$) и если при этом не выбрана ни одна другая цифра набора ξ. Дешифраторы цифр ξ_j, как и все остальные узлы арифметического устройства, построены на потенциальной системе элементов, содержащей инвертор НЕ и формирователь (усилитель) Ф, выполненные по схеме транзисторного переключателя тока, эмиттерные повторители и диодные логические схемы И, ИЛИ. На рис. 5.4.26 в качестве примера показана схема выбора цифры $\xi_{12} = -8$. Можно проследить соответствие этой схемы зоне, показанной на рис. 5.4.24. Разряды чисел пронумерованы слева направо: 0 (вес 1), 1 (вес 2^{-1}), 2 (вес 2^{-2}) и т. д. Сигнал $-B$ является выходом знакового регистра B и равен 1 при $B_i < 0$. Вспомогательные сигналы $Д_1$, $Д_2$, $Д_3$ вырабатываются отдельными схемами таким образом, что

$$Д_1 = 1 \text{ при } 1/2 \leqslant C < 37/64,$$

$$Д_2 = 1 \text{ при } 45/64 \leqslant C < 53/64,$$

$$Д_3 = 1 \text{ при } 53/64 \leqslant C < 1.$$

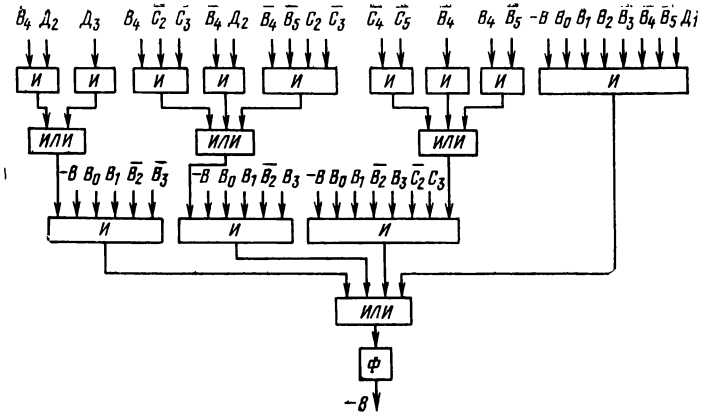


Рис. 5.4.26

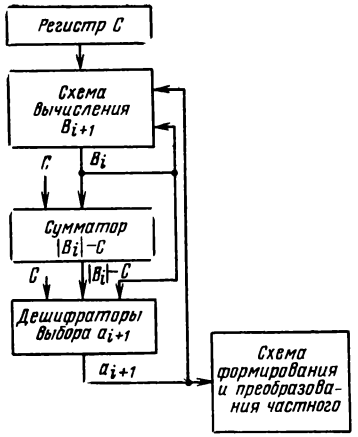


Рис. 5.4.27

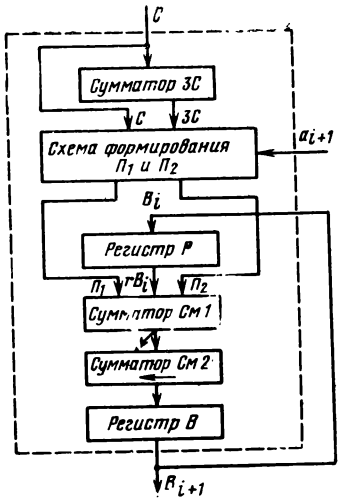


Рис. 5.4.28

Дешифраторы цифр частного содержат по 22—80 диодов. Самые простые схемы имеют дешифраторы +2, —2 (по 22 диода), ±24 (24 диода), самые сложные схемы — дешифраторы +5, —5 (по 80 диодов).

Блок-схема устройства для деления показана на рис. 5.4.27, а входящая в нее схема вычисления очередного остатка B_{i+1} — на рис. 5.4.28.

Значения кратных Π_1 и Π_2 делителя C, выбираемые из набора

$$\Pi_1 = \{\pm 32C,\ \pm 24C,\ \pm 16C,\ \pm 8C,\ 0\}$$

$$\Pi_2 = \{\pm 4C,\ \pm 3C,\ \pm 2C,\ 0\}$$

по правилам, определяемым табл. 5.4.3, обеспечивают выполнение соотношения

$$\Pi_1 + \Pi_2 = -a_{i+1}C.$$

Как видно из таблицы, каждое из кратных Π_1 или Π_2 формируется из сдвинутого на нужное количество разрядов числа C или числа $3C$, вырабатываемого вспомогательным сумматором $3C$.

Таблица 5.4.3

a_{i+1}	Π_1	Π_2	a_{i+1}	Π_1	Π_2
± 2	0	$\mp 2C$	± 22	$\mp 24C$	$\pm 2C$
± 5	$\mp 8C$	$\pm 3C$	± 24	$\mp 24C$	0
± 8	$\mp 8C$	0	± 26	$\mp 24C$	$\mp 2C$
± 10	$\mp 8C$	$\mp 2C$	± 27	$\mp 24C$	$\mp 3C$
± 13	$\mp 16C$	$\pm 3C$	± 28	$\mp 24C$	$\mp 4C$
± 16	$\mp 16C$	0	± 29	$\mp 32C$	$\pm 3C$
± 19	$\mp 16C$	$\mp 3C$	± 30	$\mp 32C$	$\pm 2C$

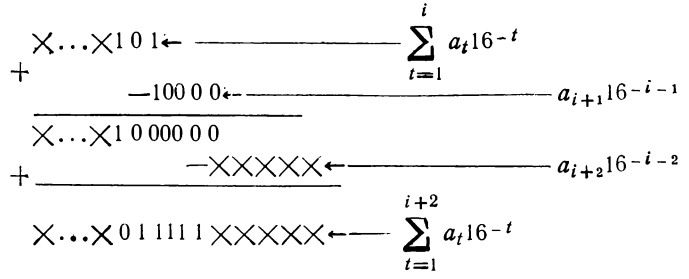
В сумматоре См 2 организован ускоренный пробег переносов. Схема преобразования частного Q в двоичную форму представляет собой по существу семиразрядный двоичный сумматор. Величины ξ_j набора (5.4.52) являются пятиразрядными двоичными числами. Прямой перебор всех возможных случаев показывает, что при сложении двух двоичных чисел

$$\sum_{t=1}^{i} a_t 16^{-t} + a_{i+1}16^{-i-1}$$

пробег переноса не превышает трех разрядов, а пробег займа — семи разрядов. Трехразрядный перенос получается, например, если a_i равно 19 или 27, а $a_{i+1} \geqslant 16$:

$$\begin{array}{l} \times\ldots\times\times 011 \longleftarrow \sum\limits_{t=1}^{i} a_t 16^{-t} \\ + \\ \qquad 1\times\times\times\times \longleftarrow a_{i+1}16^{-i-1} \\ \hline \times\ldots\times\times 100\times\times\times\times \longleftarrow \sum\limits_{t=1}^{i+1} a_t 16^{-t} \end{array}$$

Семиразрядный пробег займа имеет место, если $a_i=5{,}13$ или 29, $a_{i+1}=-16$, $a_{i+2}<0$:



Семиразрядный сумматор в схеме преобразования частного построен с использованием параллельно-параллельной логики, т. е. в схеме нет сигналов переноса или займа, и время образования всех семи разрядов суммы равно времени задержки одного активного элемента.

Следует отметить удачное совмещение в созданном арифметическом устройстве операций умножения и деления. Умножение производится за 5 циклов, во время каждого из которых к текущей сумме частичных произведений прибавляется произведение множимого на 6 очередных разрядов множителя A. При этом используются те же сумматоры См 1, См 2, $3C$ (рис. 5.4.28), те же кратные $П_1$, $П_2$ множимого C (при операции умножения используются еще два значения кратного $П_2$, равные $\pm C$), те же регистры B и P. Аппаратура, установленная в арифметическом устройстве только для операции деления (схема преобразования частного, сумматоры $\pm B_i$—C, дешифраторы выбора цифры a_{i+1}) и не используемая при других операциях (умножение, сложение, вычитание, сдвиги, логические операции), составляет лишь 10% от всего объема устройства. Это показывает, что при определенных условиях описанные методы деления могут успешно сочетаться с известными методами выполнения других операций, не требуя значительных дополнительных затрат оборудования.

Следует отметить, что в настоящее время, по-видимому, не существует других машин, в арифметических устройствах которых синхронные методы деления без восстановления остатка обеспечивали бы получение за один цикл сразу четырех или хотя бы трех двоичных разрядов частного.

Получение более чем четырех разрядов частного за цикл могут обеспечить устройства, выполняющие деление при помощи итераций (циклов), во время каждой из которых производятся операции умножения (см. § 5.3). В таких устройствах количество полученных разрядов частного может, например, удваиваться после каждого цикла. Однако применение этих методов оправдано лишь в арифметических устройствах, выполняющих умножение с весьма большой скоростью. Напротив, обоб-

щенный метод синхронного деления без восстановления остатка позволяет значительно повысить скорость деления в арифметических устройствах, использующих сравнительно недорогие (в смысле аппаратных затрат) методы ускорения умножения и содержащие поэтому сравнительно небольшое количество оборудования.

6. ПОСТРОЕНИЕ ОСНОВНЫХ УЗЛОВ ВЫЧИСЛИТЕЛЬНЫХ СРЕДСТВ ВЫСОКОЙ ПРОИЗВОДИТЕЛЬНОСТИ НА ОСНОВЕ БОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ

На нынешнем этапе развития вычислительной техники, когда полупроводниковая промышленность переходит к массовому выпуску больших интегральных схем (БИС), особое значение приобретает проблема анализа и модернизации известных и создания новых методов выполнения арифметических и логических операций. Цель этих работ — использование широких возможностей современной электронной техники. Разработка принципов выполнения арифметических и логических операций при помощи БИС становится одной из наиболее важных задач в теории и практике построения арифметических устройств. Следует также отметить, что БИС, созданные для построения таких узлов, как коммутаторы, регистры и т. п., могут успешно применяться и в других устройствах ЦВМ.

Большой интегральной схемой обычно называют схему, в корпусе которой содержится большое количество (сто и более) логических элементов, соединенных в сложную функциональную структуру. Применение БИС позволяет преодолеть ограничения, накладываемые на линейные размеры и надежность ЦВМ.

Необходимость уменьшения линейных размеров ЦВМ обусловлена соизмеримостью достигнутого наносекундного быстродействия логических элементов с потерями времени на распространение электрических сигналов (в современных платах с печатным монтажом задержка составляет ~ 7 нс/м).

Повышение надежности связано, в основном, с применением в БИС более качественных компонентов, уменьшением количества технологических операций и числа сварных соединений.

Развитию техники БИС предшествовал период разработки и использования интегральных схем с малым и средним уровнем интеграции. С 1970 г. стандартной продукцией зарубежных фирм стали счетчики, сумматоры, дешифраторы, сдвиговые регистры и другие узлы на интегральных схемах с малым и средним уровнем интеграции.

Арифметические устройства зарубежных ЦВМ третьего поколения, выпущенных или намеченных к выпуску на рынок в 1971—1972 гг., были практически полностью выполнены на интегральных схемах с тем или иным уровнем функциональной сложности. При этом происходило (и происходит сейчас) постоянное снижение стоимости и повышение функциональной плотности интегральных схем.

При разработке логики интегральных схем и БИС одной из задач следует считать унификацию и уменьшение количества типов схем, повышение регулярности схемных построений, приводящие к снижению стоимости этих схем. В частности, повторяемость или регулярность функциональных узлов предлагается рассматривать как один из важных критериев оптимальности построения вычислительных машин [1, 2].

Примером арифметического устройства, построенного в начале 70-х годов на БИС с малой номенклатурой, может служить 32-разрядное арифметическое устройство ЦВМ RAC-251, разработанное одной из зарубежных фирм (Raytheon) и содержащее БИС всего лишь восьми типов [3, с. 123]. В те же годы были опубликованы описания ЦВМ, целиком или почти целиком построенных на БИС, например описание универсальной ЦВМ с коммерческим уклоном фирмы Four — Phase Systems. Эта ЦВМ была построена на БИС типа *MOS*, содержащих не менее 1000 вентилей на кристалл [4]. Стали известны реализованные или находящиеся в стадии разработки проекты с еще более высоким уровнем интеграции. Например, фирма ICL (Англия) собирается к концу 70-х годов закончить создание гигантского вычислительного комплекса, в котором каждый из 30 тыс. процессоров целиком размещается на одной БИС [5].

В последние годы общими тенденциями развития БИС является непрерывное повышение функциональной сложности схем, их надежности и быстродействия. Одновременно снижается стоимость вентильной схемы.

Примером БИС с высоким уровнем интеграции может служить схема, в корпусе которой содержится целиком однотактный умножитель, перемножающий два 16-разрядных сомножителя [6].

Значительный прогресс в технологии интегральных схем достигнут благодаря усовершенствованию методов изоляции, фотолитографии и диффузии и широкому внедрению автоматизации на этапах раскладки схем, изготовления оригиналов фотошаблонов и проверки схем. Наиболее перспективными направлениями считаются развитие методов оксидной изоляции, тройной диффузии (*3D*), ионного легирования и электронно-лучевой и рентгенолучевой литографии [7].

Для построения арифметических устройств современных ЦВМ используются различные типы схем *TTLS*, *ECL*, I^2L, МОП-схемы и др.

Биполярные логические схемы обладают бóльшим быстродействием, чем МОП-схемы, но и большей стоимостью. В биполярных схемах увеличение функциональной плотности и рост быстродействия влекут за собой нежелательное повышение мощности рассеяния. В некоторых случаях при высоком уровне интеграции, свойственном БИС, плотность мощности рассеяния может потребовать специального охлаждения. Это обстоятельство объясняет относительно медленный прогресс в производстве биполярных логических быстродействующих БИС и их использовании в вычислительных системах высокой производительности. В настоящее время элементную базу центральных процессоров большинства зарубежных вычислительных систем высокой производительности составляют биполярные схемы типа *ECL* и *TTL* с диодами Шотки.

По-видимому, *ECL*-схемы и их модификации будут применяться в более быстродействующих системах, а *TTL*-схемы с диодами Шотки — в более медленных.

На *ECL*-схемах были реализованы, например, процессоры машин 470*V*/6 фирмы Amdahl. Уровень интеграции в этих БИС составляет 100 вентилей на кристалл, а время задержки на вентиль — 600 пс. Мощность рассеяния кристалла составляет от 1 до 3,5 Вт и проблема

охлаждения решается путем использования специальных теплоотводов и особого характера движения воздушной струи [7].

Ряд фирм, например Дженерал Электрик, считает наиболее перспективным для БИС и супер-БИС (еще более сложных схем) *TTL*-схемы с диодами Шотки, обеспечивающими задержки порядка 1,5 нс на вентиль [7].

Технические и экономические преимущества БИС в настоящее время не вызывают сомнения. Переход на БИС приводит не только к повышению надежности, скорости и снижению стоимости, но и к изменению архитектуры ЦВМ, так как внедрение БИС резко изменяет отношение стоимостей самих схем и межсхемных связей. При разработке вычислительных средств на БИС основной тенденцией является не минимизация количества элементов, а оптимизация числа соединений между БИС даже за счет введения некоторой аппаратурной избыточности. Кроме минимизации числа межмодульных связей, перед разработчиками ЦВМ на БИС стоит задача уменьшения количества типов БИС; это требование является, в частности, одним из основных в военных системах [2]. По мнению многих ведущих разработчиков ЦВМ, реализация преимуществ, свойственных БИС, требует существенного изменения методов проектирования ЦВМ вообще.

Ниже в этой главе рассматриваются некоторые пути выполнения основных арифметических операций на БИС. Некоторые из изложенных ниже способов выполнения арифметических операций (например, метод построения многослойных множительных устройств на преобразователях $N \to 2$) изменяют принципы построения узлов арифметического устройства, другие фактически направлены на создание новых структур БИС, реализующих известные методы выполнения операций.

6.1. КРИТЕРИИ ОПТИМАЛЬНОСТИ БОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ

При разработке логических структур БИС могут использоваться различные критерии оптимальности БИС. В общем случае этот критерий является некоторой функцией переменных, которые в той или иной форме оценивают такие свойства и параметры БИС или устройств, синтезируемых из них, как

1) быстродействие,
2) количество применяемых типов БИС,
3) количество выводов БИС и количество связей в устройстве,
4) регулярность внутренней структуры БИС,
5) надежность

и др. Рассмотрим кратко каждый из перечисленных пунктов.

1. Скоростными параметрами, учитываемыми критерием оптимальности, могут являться как быстродействие самой БИС, так и быстродействие узла (сумматора, умножителя и т. д.), построенного на БИС.

2. Минимизация количества типов БИС, их унификация имеют большое практическое значение, так как приводят к упрощению и ускорению организации массового производства БИС, снижению их стоимости, упрощению и удешевлению обслуживания и ремонта вычислительных средств [1, 2, 8, 9]. В качестве критерия оптимальности (или его составной части) может, в частности, использоваться отношение количества типов БИС к их общему числу [2].

3. Минимизация межсхемных связей является одной из важнейших проблем, которую следует решать разработчикам БИС и ЦВМ [2, 9—12]. Фактически задача сводится к созданию новых методов построения узлов ЦВМ из соединенных более или менее регулярным образом БИС, содержащих минимальное количество внешних выводов, приходящихся на единицу объема аппаратуры.

4. Увеличение регулярной внутренней структуры повышает технологичность БИС, а также ее быстродействие (при прочих равных условиях).

5. О надежности БИС уже немного говорилось в начале главы. Здесь мы отметим, что при создании логической структуры БИС в ней могут быть предусмотрены аппаратные средства для обнаружения, а также для исправления ошибок, вызываемых неисправностями элементов БИС. Повышение надежности при этом достигается соответственно за счет сокращения времени восстановления и увеличения времени безотказной работы. Количественно повышение надежности может оцениваться теми или иными вероятностными параметрами.

В данной главе рассматриваются и предлагаются некоторые структуры БИС,

использующих наиболее мощные методы ускорения основных арифметических операций,

позволяющих строить тот или иной узел арифметического устройства (сумматор, умножитель и т. д.) на БИС с минимальной номенклатурой (1—3 типа),

обладающих сравнительно небольшим отношением f/v количества f внешних выводов БИС к общему объему v находящейся в ней аппаратуры, что позволяет минимизировать количество межсхемных связей.

Отдавалось также предпочтение БИС с регулярной внутренней структурой.

6.2. ПОСТРОЕНИЕ БЫСТРОДЕЙСТВУЮЩИХ СУММАТОРОВ НА БОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМАХ

Из всего многообразия структур ускоренных сумматоров обычно выбирают для реализации в виде интегральных схем такие построения, которые можно разделить на узлы (интегральные схемы) одного-двух типов. В гл. 3 были рассмотрены основные виды ускоренных сумматоров. Из этого рассмотрения с очевидностью следует, что сумматоры с пирамидой переносов (§ 3.4) и условные сумматоры (§ 3.5) не удовлетворяют указанному требованию и, следовательно, плохо подходят для реализации при помощи БИС. Что касается описанных в § 3.1—3.3 сверхпараллельных сумматоров (рис. 3.1.2, 3.1.6,*а*) и сумматоров, которые мы формально считали разновидностями сверхпараллельного сумматора (рис. 3.1.6,*б*—*е*), то они легко разбиваются на узлы 1—2 типов и, следовательно, могут быть выполнены в виде БИС.

В качестве первого примера на рис. 6.2.1 приведена структура интегральной схемы *SN*5483A (фирмы Texas Instruments). Из рисунка видно, что этот 4-разрядный сумматор с полным ускорением всех переносов соответствует рис. 3.1.6,*г*. Соединяя последовательно такие 4-разрядные сумматоры, можно строить $4p$-разрядные сумматоры для любого целого p. Очевидно, что БИС этого типа по мере повышения уровня интеграции смогут содержать не одну, а несколько групп разрядов, и сами группы, возможно, станут длиннее.

Другим примером может служить интегральная схема *CD*4008 фирмы *RCA*, [13], содержащая 4-разрядный сумматор с последовательным переносом внутри группы

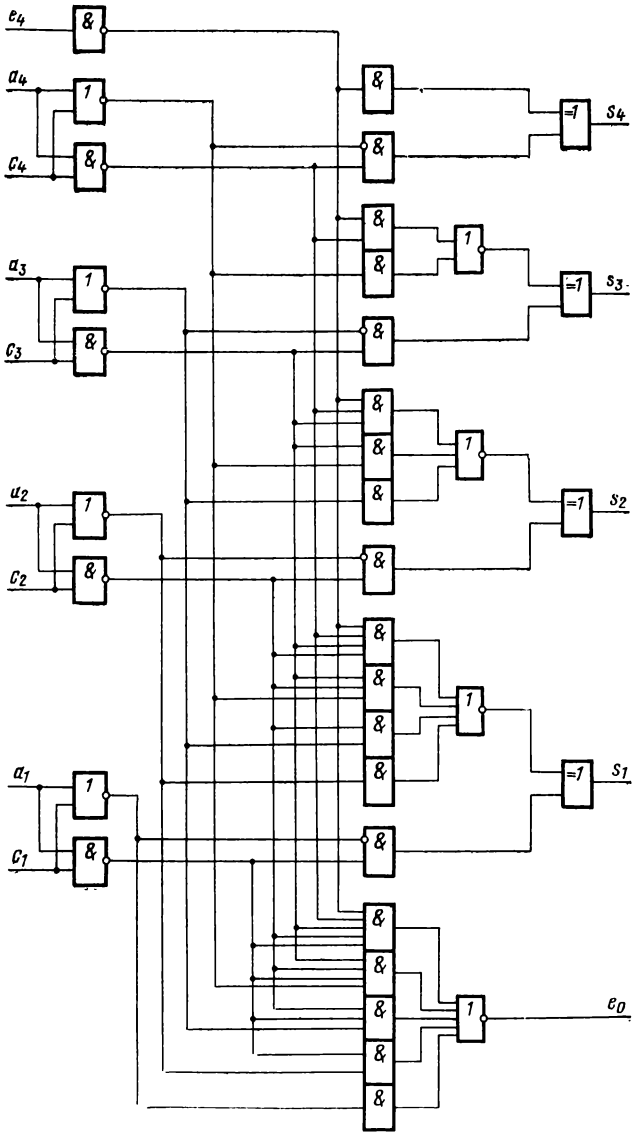


Рис. 6.2.1

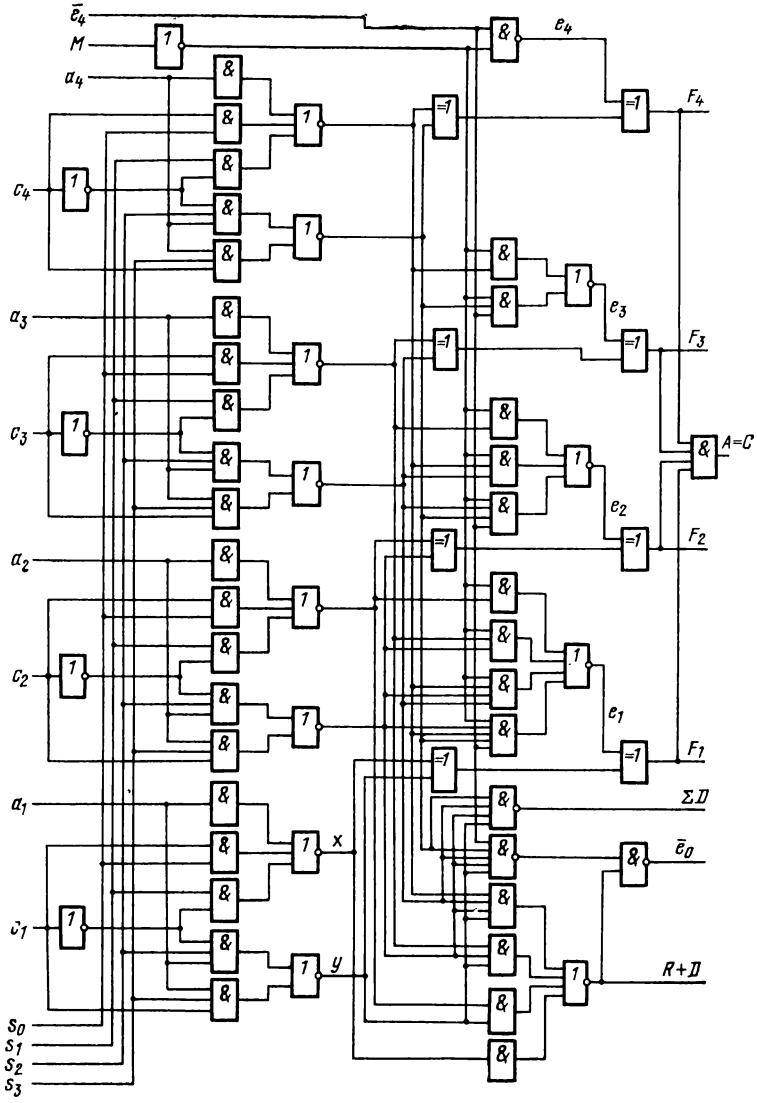

$\bar{e}_4$
M
a_4
c_4
a_3
c_3
a_2
c_2
a_1
c_1
s_0
s_1
s_2
s_3
x
y
e_4
e_3
e_2
e_1
F_4
F_3
F_2
F_1
A=C
$\Sigma\Pi$
$\bar{e}_0$
R+Д

Рис. 6.2.2

и ускоренным переносом из группы. Сумматоры, составленные из таких групп, рассматривались в § 3.1 (рис. 3.1.6,*д*).

Построение сверхпараллельных сумматоров из интегральных схем покажем на примере использования схем *SN*54181 (рис. 6.2.2) и *SN*54182 (рис. 6.2.3) фирмы Texas Instruments.

Микросхема, изображенная на рис. 6.2.2, представляет собой универсальное арифметико-логическое устройство, способное выполнять разнообразные операции сложения-вычитания и 16 поразрядных логических операций над двумя 4-разрядными двоичными числами $A=a_1a_2a_3a_4$ и $C=c_1c_2c_3c_4$. Код выполняемой операции задается сигналами s_0, s_1, s_2, s_3. Еще одним управляющим сигналом является сигнал M. Результат операции формируется в виде 4-разрядного числа $F_1F_2F_3F_4$. Перенос, поступающий на вход младшего разряда, и перенос, выходящий из старшего разряда, представлены в инверсной форме ($\bar{e}_4$, $\bar{e}_0$) и используются при операциях сложения-вычитания. Подготовительные функции ΣD и $R+D$ предназначены для организации ускорения переносов в многоразрядном сверхпараллельном сумматоре. Сигнал $A=C$ может использоваться как признак равенства чисел A и C при операции $A-C$ и операции $(A\wedge C)\vee(\bar{A}\wedge\bar{C})$. Этому сигналу можно найти и другие применения: его можно, например, расценивать как признак неравенства чисел A и C при операции $(A\wedge\bar{C})\vee(\bar{A}\wedge C)$.

В табл. 6.2.1 приведены коды операций интегральной схемы *SN*54181. Видно, что при $M=1$ схема выполняет любую из 16 поразрядных логических операций, в том числе и такие операции, как const 0 и const 1. При этих двух операциях на выходах F_i просто вырабатываются нули (операция const 0) или единицы (операция const 1).

Если $M=0$, то в схеме выполняется одна из 16 операций типа сложение-вычитание. Можно сказать, что при этом в 4-разрядном сумматоре складывается перенос e_4 и два 4-разрядных двоичных числа, каждое из которых является результатом некоторой логической поразрядной операции над числами $a_1a_2a_3a_4$ и $c_1c_2c_3c_4$. Например, операция $A-C$ выполняется так:

$$\begin{array}{r} a_1\,a_2\,a_3\,a_4 \\ +\;\bar{c}_1\,\bar{c}_2\,\bar{c}_3\,\bar{c}_4 \\ \qquad\quad 1 \\ \hline \end{array}$$

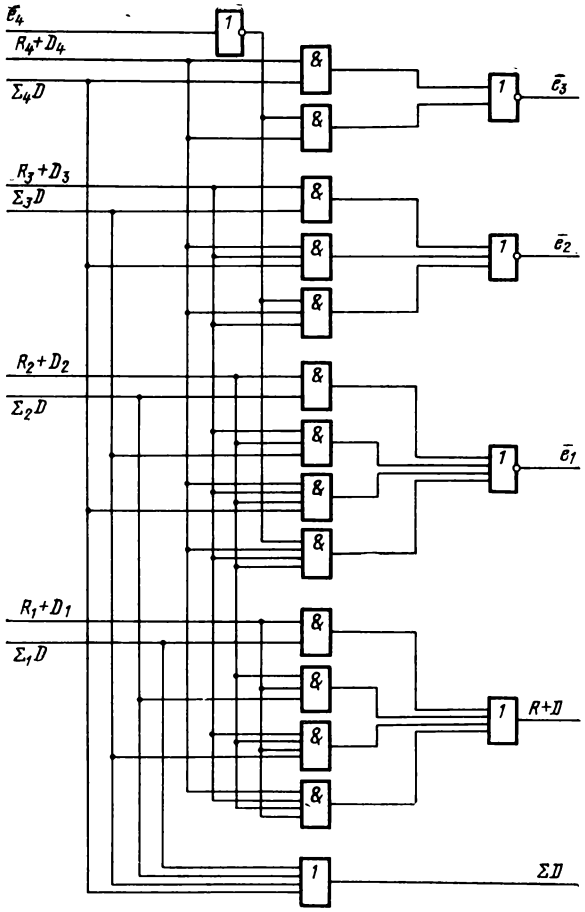


Рис. 6.2.3

(e_4=1). Операция A—1 выполняется так:

$$\begin{array}{r} \ a_1\ a_2\ a_3\ a_4 \\ +\ 1\ \ 1\ \ 1\ \ 1 \\ \hline 0 \end{array}$$

(e_4=0. Число 1111 формально можно тоже считать результатом логической операции.) Операция $(A \vee C)+$

Таблица 6.2.1

Код операции s_0 s_1 s_2 s_3	$M=1$	$M=0$	
		$e_4=0$ $(\overline{e}_4=1)$	$e_4=1$ $(\overline{e}_4=0)$
0 0 0 0	$\overline{A}$	A	$A+1$
0 0 0 1	$\overline{A}\vee C$	$A+(A\wedge C)$	$A+(A\wedge C)+1$
0 0 1 0	$\overline{A\wedge C}$	$A+(A\wedge\overline{C})$	$A+(A\wedge\overline{C})+1$
0 0 1 1	const 1	$A+A$	$A+A+1$
0 1 0 0	$\overline{A}\wedge C$	$A\vee\overline{C}$	$(A\vee\overline{C})+1$
0 1 0 1	C	$(A\vee\overline{C})+(A\wedge C)$	$(A\vee\overline{C})+(A\wedge C)+1$
0 1 1 0	$(A\wedge\overline{C})\vee(\overline{A}\wedge C)$	$A-C-1$	$A-C$
0 1 1 1	$A\vee C$	$(A\vee\overline{C})+A$	$(A\vee\overline{C})+A+1$
1 0 0 0	$\overline{A\vee C}$	$A\vee C$	$(A\vee C)+1$
1 0 0 1	$(A\wedge C)\vee(\overline{A}\wedge\overline{C})$	$A+C$	$A+C+1$
1 0 1 0	$\overline{C}$	$(A\vee C)+(A\wedge\overline{C})$	$(A\vee C)+(A\wedge\overline{C})+1$
1 0 1 1	$A\vee\overline{C}$	$(A\vee C)+A$	$(A\vee C)+A+1$
1 1 0 0	const 0	-1 (const 1)	0
1 1 0 1	$A\wedge C$	$(A\wedge C)-1$	$A\wedge C$
1 1 1 0	$A\wedge\overline{C}$	$(A\wedge\overline{C})-1$	$A\wedge\overline{C}$
1 1 1 1	A	$A-1$	A

$+(A\wedge\overline{C})$ выполняется так:

$$+\begin{array}{cccc} a_1\vee c_1 & a_2\vee c_2 & a_3\vee c_3 & a_4\vee c_4 \\ a_1\wedge\overline{c}_1 & a_2\wedge\overline{c}_2 & a_3\wedge\overline{c}_3 & a_4\wedge\overline{c}_4 \end{array}$$

Таким образом, табл. 6.2.1 заполнена в предположении, что вычитание в схеме выполняется в дополнительных кодах (отрицательный результат на выходах F_i должен получаться в дополнительном коде). Но мы увидим, что в действительности можно выполнять вычитание и в обратных кодах, т. е. с использованием цепи кольцевого переноса.

Интегральные схемы *SN*54181 можно соединять последовательно при помощи сигналов $\bar{e}_0$, $\bar{e}_4$. При соеди-

нений m микросхем образуется $4m$-разрядный ускоренный сумматор, соответствующий рис. 3.1.6,*г*.

Для организации сверхпараллельного сумматора, кроме схем *SN*54181, используются схемы *SN*54182 (рис. 6.2.3). В качестве примера рассмотрим построение 32-разрядного сумматора с кольцевым переносом (рис. 6.2.4). Для упрощения рисунка на нем не показаны входы слагаемых и выходы суммы. Из рис. 6.2.2—6.2.4

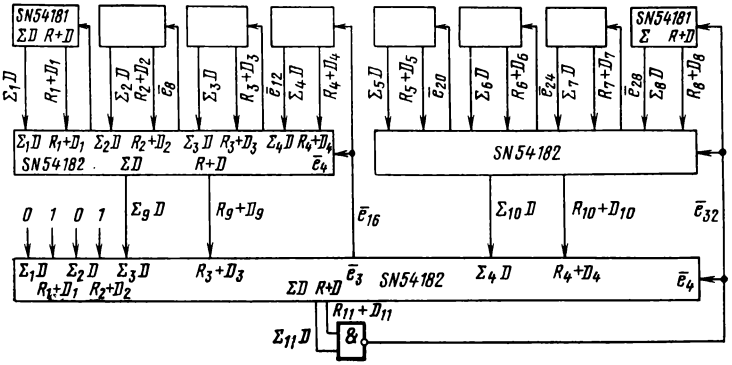


Рис. 6.2.4

видно, что при операции A—C в сумматоре фактически складываются числа

$$a_1 a_2 a_3 \ldots a_{32},$$
$$\bar{c}_1 \bar{c}_2 \bar{c}_3 \ldots \bar{c}_{32}.$$

При этом в каждом разряде сумматора вырабатываются (например, в точках y и x см. рис. 6.2.2) две подготовительные функции

$$\overline{D}_i = \overline{a_i \bar{c}_i}, \quad \overline{R_i + D_i} = \overline{a_i + \bar{c}_i}, \qquad (6.2.1)$$

где $R_i = a_i c_i + \bar{a}_i \bar{c}_i$.

В каждой группе первого яруса вырабатываются две подготовительные групповые функции (см. рис. 6.2.2 и 6.2.4)

$$\Sigma D = \overline{\overline{D}_{i+1}\overline{D}_{i+2}\overline{D}_{i+3}\overline{D}_{i+4}} = D_{i+1} + D_{i+2} + D_{i+3} + D_{i+4},$$

$$R+D=D_{i+1}+D_{i+2}R_{i+1}+D_{i+3}R_{i+2}R_{i+1}+$$
$$+D_{i+4}R_{i+3}R_{i+2}R_{i+1}+R_{i+4}R_{i+3}R_{i+2}R_{i+1}$$
$$(i=0,\ 4,\ 8,\ 12,\ \ldots,\ 28). \qquad (6.2.2)$$

Здесь $R=R_{i+4}R_{i+3}R_{i+2}R_{i+1}$.

(Индексы при буквах R, D соответствуют номерам разрядов чисел A и C; если группы первого яруса пронумеровать слева направо от 1 до 8, то можно сказать, что формулы (6.2.2) относятся к группе с номером $1+i/4$, где i — номер старшего разряда этой группы.)

Следует заметить, что логическое произведение сигналов ΣD и $R+D$ равно

$D_{i+1}+D_{i+2}R_{i+1}+D_{i+3}R_{i+2}R_{i+1}+D_{i+4}R_{i+3}R_{i+2}R_{i+1}=D$,

т. е. оно может служить в качестве сигнала возникновения переноса в 4-разрядной группе.

В каждой группе остальных ярусов из поступающих в нее из предыдущего яруса сигналов $\Sigma_i D$, R_i+D_i формируются при помощи схемы $SN54182$ два новых сигнала (см. рис. 6.2.3)

$$\Sigma D=\Sigma_1 D+\Sigma_2 D+\Sigma_3 D+\Sigma_4 D$$
$$R+D=\Sigma_1 D(R_1+D_1)+\Sigma_2 D(R_2+D_2)(R_1+D_1)+$$
$$+\Sigma_3 D(R_3+D_3)(R_2+D_2)(R_1+D_1)+$$
$$+(R_4+D_4)(R_3+D_3)(R_2+D_2)(R_1+D_1). \qquad (6.2.3)$$

Используя сделанные выше замечания относительно свойств произведения сигналов ΣD, $R+D$, выходящих из первого яруса, можно переписать соотношения (6.2.3) следующим образом:

$$\Sigma D=\Sigma_1 D+\Sigma_2 D+\Sigma_3 D+\Sigma_4 D,$$
$$R+D=D_1+D_2R_1+D_3R_2R_1+D_4R_3R_2R_1+R_4R_3R_2R_1.$$
$$(6.2.4)$$

Произведение этих двух сигналов также может играть роль группового сигнала D. Сигналы (6.2.4) имеют тот же логический смысл, что и сигналы (6.2.2): $\Sigma D=1$, если хотя бы в одном из четырех элементов, входящих в данную группу, вырабатывается «местный» перенос, выходящий из этого элемента (под «элементом» в случае формул (6.2.2) понимается разряд, а в случае формул (6.2.4) — группа разрядов); $R+D=1$, если в данной группе вырабатывается и выходит из нее

«местный» перенос либо через данную группу разрешается распространение переносов.

Из сигналов ΣD, $R+D$ формируются окончательные сигналы переносов. В отличие от рассмотренной в § 3.1 более известной схемы сверхпараллельного сумматора, в которой переносы организуются при помощи сигналов R и D, в данном случае роль сигнала D любой группы разрядов играет произведение $(R+D)\Sigma D$, а роль сигнала R выполняет сигнал $R+D$. Так, например, перенос e_1 в первый (старший) разряд 32-разрядного сумматора вырабатывается находящейся в первом ярусе сумматора схемой (см. рис. 6.2.4, 6.2.2), реализующей функцию, которая после несложных формальных преобразований приводится к обычной форме:

$$\overline{e_1 = \overline{R_2+D_2} + (\overline{R_3+D_3})\,\overline{D_2} + (\overline{R_4+D_4})\,\overline{D_3}D_2 + \overline{e_4}\,\overline{D_4}\,\overline{D_3}\,\overline{D_2}} =$$

$$= (R_2+D_2)(R_3+D_3+D_2)(R_4+D_4+D_3+D_2)(e_4+$$

$$+D_4+D_3+D_2) = D_2 + D_3R_2 + D_4R_3R_2 + e_4R_4R_3R_2$$

(здесь индексы при буквах R, D, e соответствуют нумерации этих величин на рис. 6.2.2 и нумерации разрядов в 32-разрядном сумматоре). Переносы в нижних ярусах формируются иначе. Например, сигнал $\bar{e}_4$ вырабатывается так (см. рис. 6.2.3, 6.2.4):

$$\overline{e_4} = \overline{\Sigma_2 D\,(R_2+D_2) + \Sigma_3 D\,(R_3+D_3)(R_2+D_2) +}$$

$$\overline{+\Sigma_4 D\,(R_4+D_4)(R_3+D_3)(R_2+D_2) +}$$

$$\overline{+e_{16}(R_4+D_4)(R_3+D_3)(R_2+D_2)} =$$

$$= \overline{D_2 + D_3R_2 + D_4R_3R_2 + e_{16}R_4R_3R_2}$$

(здесь индексы соответствуют нумерации сигналов R, D, ΣD, e на рис. 6.2.4).

Цепь кольцевого переноса замыкается через отдельный элемент И—НЕ. Это обстоятельство не следует считать недостатком схем *SN*54181, *SN*54182, так как обычно в реальных устройствах кольцевой перенос все равно приходится то замыкать (например, при вычитании), то размыкать (при сложении).

Следует обратить внимание на то, что в сумматоре, показанном на рис. 6.2.4, в нижнюю схему *SN*54182

в качестве сигналов $\Sigma_1 D$, R_1+D_1, $\Sigma_2 D$, R_2+D_2 заведены постоянные уровни 0 и 1. Этим обеспечивается правильная выработка сигналов $\Sigma_{11} D$, $R_{11}+D_{11}$. Такой прием является общим для многих стандартных схем, возможности которых в некотором отношении превышают потребности конкретного применения. Если бы, например, нам был нужен не 32-разрядный, а 31-разрядный сумматор, то в устройстве, показанном на рис. 6.2.4, следовало бы в один из разрядов завести постоянные уровни $a=0$, $c=1$ (или $a=1$, $c=0$). Выход суммы этого разряда, естественно, не использовался бы.

Интересное предложение содержится в работе [14]. В группах первого яруса сверхпараллельного сумматора, описанного в ней, вырабатываются обычные поразрядные функции D_i, R_i (в [14] они названы V_i, $\bar{Z}_i$). Далее в первом и во всех нечетных ярусах вырабатываются подготовительные групповые функции $\bar{D}$, $\bar{R}$ (в работе названы $\bar{V}$, Z^*), т. е. функции, инверсные функциям D и R, используемым в обычном сверхпараллельном сумматоре *). В четных ярусах сумматора вырабатываются подготовительные групповые функции ΣD, $R+D$ (в [14] они называются V^*, $\bar{Z}$), как в только что описанном сверхпараллельном сумматоре. Групповые сигналы переноса в нечетных ярусах вырабатываются инверсными (из сигналов ΣD, $R+D$ предыдущего яруса и сигналов переноса e, поступающих из последующих ярусов), а в четных ярусах сигналы переноса формируются прямыми (из сигналов $\bar{D}$, $\bar{R}$ предыдущего и сигналов $\bar{e}$ последующих ярусов). Основная особенность такого сумматора состоит в том, что при использовании элементов И—ИЛИ—НЕ время выработки указанных подготовительных функций (и переносов) в каждом ярусе равно 1τ (время срабатывания одного элемента И—ИЛИ—НЕ), а схемы выработки сигналов $\bar{D}$, $\bar{R}$, $\bar{e}$ соответственно тождественны схемам выработки сигналов $R+D$, ΣD, e, что дает возможность строить весь сумматор на типовых узлах двух разновидностей: для первого яруса и для последующих ярусов.

---

*) При таком сопоставлении сумматора из [14] с обычным сверхпараллельным сумматором предполагается, что в последнем поразрядные функции R_i выбираются по формуле

$$R_i=a_i+c_i, \text{ а не } R_i=a_i\bar{c}_i+\bar{a}_i c_i.$$

6.3. ПОСТРОЕНИЕ ОДНОТАКТНЫХ УМНОЖИТЕЛЕЙ НА БОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМАХ

Применение ОУ является одним из наиболее мощных методов ускорения операций умножения и деления (деления — в тех случаях, когда оно выполняется с помощью умножения). За последние годы благодаря успехам микроэлектроники и развитию техники БИС использование этих устройств значительно возросло.

Два свойства ОУ делают весьма перспективной их реализацию на БИС: сравнительно небольшое отношение количества внешних выводов к количеству аппаратуры (как для всего устройства, так и для его составных частей) и однородность структуры (последним свойством разные типы ОУ обладают в различной степени). Известны проекты и практические реализации ОУ на интегральных схемах с малой, средней [15—18] и с высокой степенью интеграции [6].

В § 6.3 рассматриваются различные схемы ОУ на БИС. В одних случаях (п. 6.3.1) речь идет об использовании БИС, представляющих собой k-разрядные двоичные параллельные сумматоры, т. е. БИС, не предназначенных, вообще говоря, для построения ОУ. В других случаях рассмотрены методы построения ОУ из специально для этого разработанных БИС—k-разрядных преобразователей N-рядного двоичного кода в двухрядный (п. 6.3.2) или из БИС, представляющих собой ОУ меньшей разрядности (п. 6.3.3).

В ОУ, построенном на БИС, можно различать микроструктуру устройства (структуру БИС) и макроструктуру (характер соединений между БИС). В некоторых случаях эти две структуры могут быть практически одинаковыми (можно, например, из БИС, представляющих собой небольшие однородные матричные ОУ, собрать большое ОУ с таким же характером структуры).

6.3.1. ПОСТРОЕНИЕ МНОГОСЛОЙНЫХ ОДНОТАКТНЫХ УМНОЖИТЕЛЕЙ ПРИ ПОМОЩИ ПАРАЛЛЕЛЬНЫХ СУММАТОРОВ

Если по тем или иным причинам в распоряжении разработчика отсутствуют БИС, специально предназначенные для построения ОУ (такие БИС будут рассмотрены в п. 6.3.2, 6.3.3), то наиболее подходящими для соз-

дания ОУ обычно оказываются БИС, содержащие малоразрядные (k-разрядные) параллельные двоичные сумматоры. Из таких интегральных схем можно строить, вообще говоря, ОУ различной конфигурации. Можно, например, реализовать любую из однородных структур (см. рис. 4.4.3). Для этого нужно схему ОУ разделить вдоль цепей переносов на участки, содержащие по k одноразрядных сумматоров. Формирование частичных произведений осуществляется при этом либо отдельными схемами (одна из возможных схем будет показана на рис. 6.4.1), либо прямо внутри БИС с сумматорами (некоторые из БИС обеспечивают такую возможность).

Однако однородные ОУ в ряде случаев оказываются недостаточно быстрыми и тогда более предпочтительны многослойные построения. В п. 4.4.4, 7°, 8°, были рассмотрены быстродействующие многослойные ОУ, построенные из одноразрядных сумматоров. Время работы такого устройства определяется количеством M его слоев; M получается из неравенства $n^{(M-1)}<n\leqslant n^{(M)}$, где n — разрядность сомножителей, а $n^{(T)}$ — определяемая соотношением $n^{(T)}=n^{(T-1)}+[n^{(T-1)}/2]$ максимальная разрядность сомножителей, которые можно перемножить на ОУ, имеющем T слоев (следует учесть, что $n^{(0)}=2$). Для краткости назовем схему такого ОУ «классической». К сожалению, классическая схема мало подходит для реализации на БИС, так как ее структура неоднородна и, по-видимому, может быть разделена на одинаковые узлы единственным способом: узел должен содержать не связанные между собой одноразрядные сумматоры. Однако при этом для каждого сумматора на БИС должны иметься 5 внешних выводов, что приводит к очень невыгодному увеличению отношения f/v количества f внешних выводов БИС к объему v находящейся в ней аппаратуры. В следующем пункте (6.3.2) будет изложен метод построения ОУ, приближающихся по быстродействию к классической схеме, из БИС нового типа. В этом пункте (6.3.1) мы тоже попытаемся строить схему, родственную классической, но в качестве составных частей ОУ будут использованы k-разрядные ускоренные двоичные сумматоры [19]. Будем полагать, что в таком сумматоре все k сигналов суммы и сигнал переноса из старшего разряда вырабатываются одновременно через 1τ после поступления всех входных сигналов, число которых, очевидно, равно $2k+1$.

Построим многослойное ОУ, в первом слое которого исходные $n=n_0$ слагаемых (рядов, частичных произведений) преобразуются в n_1 рядов, во втором слое n_1 рядов преобразуются в n_2 рядов и т. д.

Опишем организацию первого слоя. Разобьем исходные слагаемые на группы, содержащие по $2k+1$ чисел. В одной неполной группе при этом, очевидно, окажется $n-(2k+1)[n/(2k+1)]$ чисел, причем $0\leqslant n-(2k+1)\times\times[n/(2k+1)]\leqslant 2k$.

Произведем суммирование $2k+1$ слагаемых каждой полной группы при помощи k-разрядных сумматоров, расположив их в k строк таким образом, чтобы в каждой строке складывались два исходных числа, а всего на k строк будет заведено $2k$ чисел. Цифры одного оставшегося исходного числа группы заведем на входы переносов в младшие разряды этих же k-разрядных сумматоров. Для этого сумматоры каждой строки должны быть соответствующим образом сдвинуты на один разряд по отношению к сумматорам соседней строки. Выходные сигналы сумм указанных сумматоров и сигналы переносов из их старших разрядов при этом образуют $(k+1)$-рядный код. Все $\lambda_1=[n/(2k+1)]$ полных групп, следовательно, преобразовывают $(2k+1)\lambda_1$-рядный код в $(k+1)\lambda_1$-рядный.

Оставшаяся неполная группа, как мы видели, содержит $n-(2k+1)\lambda_1$ чисел. Если $n-(2k+1)\lambda_1\leqslant 2$, то эти числа обрабатывать не будем, а просто пропустим их сквозь слой. Если $n-(2k+1)\lambda_1\geqslant 3$, то произведем суммирование при помощи $[(n-(2k+1)\lambda_1-1)/2]$ дополнительных строк сумматоров. Сумматоры в строках сдвинем так же, как в полной группе. В каждой строке будут складываться 2 числа и, кроме того, на входы переносов в младшие разряды сумматоров будут заводиться цифры из еще одного числа. Следует отметить, что при $n-(2k+1)\lambda_1\geqslant 3$ часть разрядов одного числа неполной группы не преобразовывается, а проходит сквозь слой. Например, если неполная группа содержит 3 числа, то на строку сумматоров заводятся 2 числа и только одна из каждых k цифр третьего числа. Если же, кроме того, $n-(2k+1)\lambda_1$ четно, то еще одно число неполной группы целиком проходит сквозь слой без преобразования.

Аналогичным образом строится каждый i-й слой, преобразующий n_{i-1} рядов в n_i рядов. Вся структура ОУ описывается уравнениями

$$n_i=(k+1)\lambda_i+\nu_i,\ \lambda_i=\left[\frac{n_{i-1}}{2k+1}\right],$$

$$\nu_i=\begin{cases}0 \text{ при } n_i-(2k+1)\lambda_i=0,\\ \left[\frac{n_{i-1}-(2k+1)\lambda_i}{2}\right]+1 \text{ при } n_i-(2k+1)\lambda_i>0,\end{cases}$$

$$n_M=2,\quad i=1,\ 2,\ \ldots,\ M. \tag{6.3.1}$$

(Как и в п. 4.4.4, мы полагаем, что на выходе последнего M-го слоя получаются два числа, которые далее будут складываться на параллельном сумматоре.)

График функции $n_i=f(n_{i-1})$ приведен на рис. 6.3.1 (этот же график был изображен на рис. 4.4.16*е* (с. 203)). Сплошной линией показано преобразование, осуществляемое над слагаемыми одной (полной или неполной) группы; дальше график периодически повторяется (штриховая линия). Из (6.3.1) и рис. 6.3.1 следует, что при использовании этого метода построения ОУ выполняются все пять условий (условия 1—5), рассмотренных в п. 4.4.4. Поэтому каждому T соответствует $n^{(T)}$ — максимальное количество рядов (чисел), которое можно свернуть в двухрядный код на ОУ, содержащем T слоев. Известно (см. п. 4.4.4), что $\max n_{i-1}=n^{(T)}$ при $n_i=n^{(T-1)}$. Учитывая это обстоятельство, можно вывести из рис. 6.3.1 или из соотношений (6.3.1) следующую зависимость между $n^{(T)}$ и $n^{(T-1)}$:

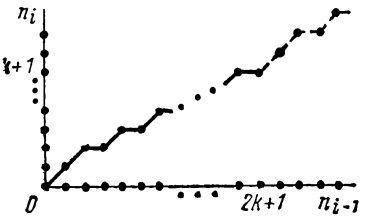


Рис. 6.3.1

$$n^{(T)}=(2k+1)\lambda+\nu,\quad \lambda=\left[\frac{n^{(T-1)}}{k+1}\right],$$

$$\nu=\begin{cases}0 \text{ при } n^{(T-1)}-(k+1)\lambda=0,\\ 2\,(n^{(T-1)}-(k+1)\lambda)-1\\ \text{при } n^{(T-1)}-(k+1)\lambda>0,\end{cases} \tag{6.3.2}$$

$$T=1,\ 2,\ \ldots$$

Подставляя сюда $n^{(0)}=2$, получим $n^{(1)}$. Подставляя далее $n^{(1)}$, $n^{(2)}$, ..., получим последовательно величины $n^{(2)}$, $n^{(3)}$, ... для любого k (табл. 6.3.1).

Таблица 6.3.1

k	$n^{(0)}$	$n^{(1)}$	$n^{(2)}$	$n^{(3)}$	$n^{(4)}$	$n^{(5)}$	$n^{(6)}$	$n^{(7)}$	$n^{(8)}$	$n^{(9)}$
2	2	3	5	8	13	21	35	58	96	160
3	2	3	5	8	14	24	42	73	127	222
4	2	3	5	9	16	28	50	90	162	291
5	2	3	5	9	16	29	53	97	177	324
6	2	3	5	9	16	29	53	98	182	338
7	2	3	5	9	16	30	56	105	196	367
8	2	3	5	9	17	32	60	113	213	382

Зная величины $n^{(0)}$, $n^{(1)}$, ..., можно определить M для любого n_0 (см. п. 4.4.4). Для этого нужно выбрать такие два числа $n^{(T-1)}$, $n^{(T)}$, что $n^{(T-1)}<n_0\leqslant n^{(T)}$. Тогда $M=T$. Например, $M=6$ при $k=5$ и $n_0=48$, так как в этом случае $n^{(5)}=29<n\leqslant 53=n^{(6)}$. Последовательность из n_i для этого примера получается из (6.3.1):

$$n_1=27,\ n_2=15,\ n_3=9,\ n_4=5,\ n_5=3,\ n_6=2.$$

Полное время умножения в устройстве, построенном на только что описанных принципах, равно

$$\tau_{\Phi}+M\tau+\tau_{\text{см}},$$

где τ_{Φ} — время формирования частичных произведений, τ — время срабатывания одного слоя, $\tau_{\text{см}}$ — время заключительного сложения двух чисел.

Количество слоев M приблизительно равно

$$M\approx\log_{\frac{2k+1}{k+1}} n,$$

а точная оценка, как сказано, дается соотношениями (6.3.2) и табл. 6.3.1.

Основная структура ОУ для $k=4$, $n_0=12$ отражена на рис. 6.3.2, имеющем тот же характер, что и аналогичные рисунки в п. 4.4.4.

Как и в других случаях (см. п. 4.4.4), существует «экономичная» модификация этого метода, при которой

$$n_1=n^{(M-1)},\ n_2=n^{(M-2)},\ \ldots,\ n_M=n^{(0)}=2.$$

«Экономичное» ОУ для $n_0=12$ иллюстрируется рис. 6.3.3. Можно заметить, что частным случаем данного метода является метод, рассмотренный в п. 4.4.4, 7°, 8° и названный нами классическим; в этом случае $k=1$.

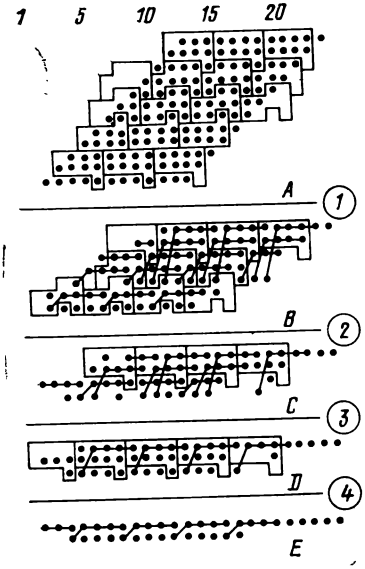


Рис. 6.3.2

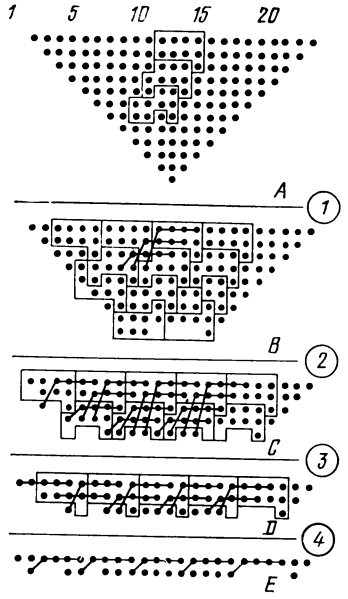


Рис. 6.3.3

Другим частным случаем ($k=\infty$) является метод, рассмотренный в п. 4.4.4, 2°, 3°.

Как уже говорилось, из ускоренных k-разрядных сумматоров можно строить ОУ и других типов. Одно из предложений содержится, например, в [26]. Однако метод, изложенный выше, обеспечивает большее быстродействие.

6.3.2. МЕТОД ПОСТРОЕНИЯ МНОГОСЛОЙНЫХ ОДНОТАКТНЫХ УМНОЖИТЕЛЕЙ ПРИ ПОМОЩИ ПРЕОБРАЗОВАТЕЛЕЙ $N\to 2$

Рассмотрим еще один метод преобразования многорядного кода, обеспечивающий примерно такую же скорость преобразования, как и классическая многослойная схема, но реализуемый одинаковыми специальными узлами, имеющими небольшое количество внешних выводов [20]. Метод состоит фактически в построении близких к классическому преобразователей n-рядного кода в двухрядный ($n=3, 4, \ldots$) из классических пре-

образователей N-рядного кода (N — фиксированное целое число, причем $N \geqslant 3$).

Пусть имеется узел (БИС), обладающий Nk основными входами, $2k$ основными выходами, $N-3$ дополнительными входами и $N-3$ дополнительными выходами (рис. 6.3.4). Здесь k — разрядность двоичных чисел, поступающих на основные входы узла, N — количество этих чисел (рядов). На рис. 6.3.4,*а* показан узел, в ко-

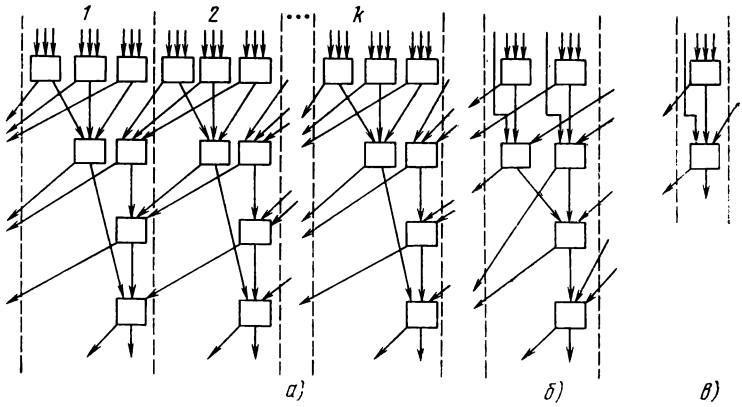


Рис. 6.3.4

тором $N=9$, на рис. 6.3.4,*б* — один разряд узла с параметром $N=8$, на рис. 6.3.4,*в* — один разряд узла для случая $N=4$. Узел содержит $(N-2)k$ одноразрядных сумматоров (на рисунке изображены в виде квадратов) и состоит из k одинаковых частей, каждая из которых представляет собой один разряд классического преобразователя N-рядного кода в двухрядный. Если несколько таких узлов соединить друг с другом цепочкой таким образом, чтобы дополнительные выходы предыдущего узла были соединены с дополнительными входами следующего так, как соединяются соседние разряды внутри узла, то получается схема, которая может служить классическим преобразователем N-рядного кода (N многоразрядных двоичных чисел) в двухрядный. Для краткости будем называть такой узел (БИС) *преобразователем* $N \to 2$, а цепочку преобразователей $N \to 2$, соединенных указанным образом для преобразования чисел большей

разрядности, чем k, назовем *рядом преобразователей.*

Преобразователь n-рядного кода в двухрядный составим из M слоев преобразователей. Выходы каждого i-го слоя являются входами следующего, $(i+1)$-го слоя. Каждый i-й слой ($i=1, 2, \ldots, M$) содержит несколько рядов преобразователей. Через n_{i-1} и n_i обозначим количество слагаемых (рядов) соответственно на входе и выходе i-го слоя.

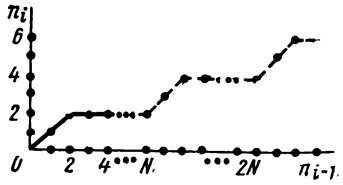


Рис. 6.3.5

Разобьем исходные $n=n_0$ слагаемых на группы, содержащие по N чисел. У нас получится $\lambda_1=[n/N]$ полных групп и одна неполная группа, содержащая $n-N\lambda_1$ чисел (очевидно, что $0\leqslant n-N\lambda_1<N$). Каждую из λ_1 групп будем преобразовывать в двухрядный код при помощи одного ряда преобразователей. Неполную группу при $n-N\lambda_1\geqslant 3$ тоже преобразуем в двухрядный код еще одним рядом преобразователей. Если $n-N\lambda_1\leqslant 2$, то пропустим числа неполной группы сквозь первый слой без преобразования. Таким образом, на выходах слоя получится n_1 чисел, причем

$$n_1=2\lambda_1+\nu_1, \quad \lambda_1=[n_0/N],$$

$$\nu_1=\begin{cases} n_0-N\lambda_1 & \text{при } n_0-N\lambda_1\leqslant 2, \\ 2 & \text{при } n_0-N\lambda_1\geqslant 3. \end{cases}$$

Аналогичным образом строится каждый i-й слой, преобразующий n_{i-1} чисел в n_i. График функции $n_i=f(n_{i-1})$ показан на рис. 6.3.5. (Этот же график был приведен на рис. 4.4.16*ж*, с. 203). Сплошной линией показано преобразование, осуществляемое над слагаемыми одной (полной или неполной) группы; дальше график периодически повторяется (штриховая линия). Видно, что в данном случае условия 1—5 (см. п. 4.4.4) выполняются. Вся структура ОУ описывается следующими соотношениями *):

---

*) Очевидно, что при $3\leqslant n_{i-1}-N\lambda_i\leqslant N-1$ аппаратура того ряда преобразователей, на который поступает $n_{i-1}-N\lambda_i$ слагаемых неполной группы, используется неполностью.

$$\left.\begin{aligned} &n_i = 2\lambda_i + \nu_i, \quad \lambda_i = [n_{i-1}/N], \\ &\nu_i = \begin{cases} n_{i-1} - N\lambda_i & \text{при } n_{i-1} - N\lambda_i \leqslant 2, \\ 2 & \text{при } n_{i-1} - N\lambda_i \geqslant 3, \end{cases} \\ &n_M = 2, \quad i = 1, 2, \ldots, M. \end{aligned}\right\} \qquad (6.3.3)$$

Любому целому T соответствует $n^{(T)}$ — максимальное количество исходных слагаемых, которое может быть свернуто в двухрядный код устройством, содержащим T слоев. Так как $n^{(T)} = \max n_{i-1}$ при $n_i = n^{(T-1)}$, то из (6.3.3) можно получить следующую зависимость между $n^{(T)}$ и $n^{(T-1)}$:

$$n^{(T)} = n^{(T-1)} + (N-2)\,[n^{(T-1)}/2]. \qquad (6.3.4)$$

В самом деле, при данном n_i максимальное n_{i-1} получается, если каждая из $[n_i/2]$ пар чисел получена путем преобразования N чисел одним рядом преобразователей $N \to 2$, а $n_i - 2[n_i/2]$ чисел прошло сквозь слой без преобразования. Полагая $n_i = n^{(T-1)}$, получаем:

$$n^{(T)} = N\,[n^{(T-1)}/2] + (n^{(T-1)} - 2\,[n^{(T-1)}/2]),$$

откуда следует (6.3.4). Подставляя в (6.3.4) $n^{(0)} = 2$, можно по очереди вычислять $n^{(1)}$, $n^{(2)}$, ... для любого N.

Для четного N можно получить функцию $n^{(T)} = f(T)$ в явном виде. В самом деле, при четных $n^{(T-1)}$ и N из (6.3.4) имеем:

$$n^{(T)} = n^{(T-1)} + (N-2)\,\frac{n^{(T-1)}}{2} = \frac{N}{2}\,n^{(T-1)},$$

т. е. $n^{(T)}$ — тоже четное. Поэтому из четности $n^{(0)}$ следует четность $n^{(1)}$, далее — четность $n^{(2)}$ и т. д. Следовательно, $n^{(T)}$ — четное при любом T:

$$n^{(0)} = 2 = 2\left(\frac{N}{2}\right)^0, \quad n^{(1)} = \frac{N}{2}\,2\left(\frac{N}{2}\right)^0 = 2\left(\frac{N}{2}\right)^1,$$

$$n^{(2)} = \frac{N}{2}\,N = 2\left(\frac{N}{2}\right)^2, \ \ldots, \ n^{(T)} = 2\left(\frac{N}{2}\right)^T. \qquad (6.3.5)$$

Таблица 6.3.2

N	$n^{(T)}$						N	$n^{(T)}$						
	$T=0$	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$		$T=0$	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$	
3	2	3	4	6	9	13	6	2	6	18	54	162	486	
4	2	4	8	16	32	64	9	2	9	37	163	730	3285	

Некоторые из величин $n^{(T)}$ приведены в табл. 6.3.2. Так как условия 1—5 выполняются, то $M=T$ при $n^{(T-1)}<n_0\leqslant n^{(T)}$. Например, при $N=4$ и $n_0=51$ количество слоев в ОУ равно $M=5$ (так как $32=n^{(4)}<n_0\leqslant n^{(5)}=64$).

Величины n_i, получаемые из (6.3.3), для этого примера равны $n_1=26$, $n_2=14$, $n_3=8$, $n_4=4$, $n_5=2$.

Можно заметить, что при $N=3$ ОУ рассматриваемого типа превращаются в классическую схему, описанную в п. 4.4.4.

В табл. 6.3.3 величины $n^{(M)}$ размещены в несколько ином порядке. В каждую колонку таблицы сведены величины $n^{(M)}$ устройств, обладающих одинаковым быстродействием, т. е. одинаковым произведением MK, где K — количество слоев одноразрядных сумматоров внутри преобразователя $N\to 2$ (считаем, что каждый слой одноразрядных сумматоров срабатывает за одно и то же время). Для таблицы выбраны преобразователи с параметрами N, равными $n^{(K)}$ для случая $N=3$, т. е. выбраны преобразователи $N\to 2$ с максимально возможным количеством слагаемых N при данном количестве K слоев внутри преобразователя $N\to 2$. Из таблицы видно, что при заданном n произведение MK, т. е. быстродействие ОУ, в общем мало зависит от N. Это обстоятельство можно объяснить следующим образом. Количество слоев сумматоров в каждом из классических преобразователей N-рядного кода, используемых для составления

Таблица 6.3.3

N, K	$n^{(M)}$														
$N=3$, $K=1$	3	4	6	9	13	19	28	42	63	94	141	211	316	474	711
$N=4$, $K=2$		4		8		16		32		64		128		256	
$N=6$, $K=3$			6			18			54			162			486
$N=9$, $K=4$				9				37				163			
$N=13$, $K=5$					13					79					508
$N=19$, $K=6$						19						172			
$N=28$ $K=7$							28							392	
MK	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

преобразователей n-рядного кода, примерно равно

$$K \approx \log_{3/2} \frac{N}{2}$$

(см. формулы (4.4.12), (4.4.13) в п. 4.4.4). Количество слоев в преобразователе n-рядного кода, как видно из (6.3.4), (6.3.5), примерно равно

$$M \approx \log_{N/2} \frac{n}{2}.$$

Поэтому общее количество слоев сумматоров

$$MK \approx \log_{3/2} \frac{N}{2} \log_{N/2} \frac{n}{2} = \log_{3/2} \frac{n}{2}$$

не зависит (точнее, мало зависит) от N и примерно равно количеству слоев в классическом ОУ, перемножающем n-разрядные сомножители. Такая оценка, однако, является довольно грубой.

В качестве примера рассмотрим построенное на четырехразрядных преобразователях 4⟶2 устройство, сворачивающее 9-рядный код (9 частичных произведений) в двухрядный ($N=4$, $k=4$, $n=9$). Структура основной модификации ОУ отражена на рис. 6.3.6. Из рисунка видно, что для формирования двухрядного кода произведения требуется 12 преобразователей и что общее количество слоев M равно трем. Так как $K=2$, то все преобразование производится за $6\tau_c$, где τ_c — время срабатывания одноразрядного сумматора. (При использовании классической схемы при $n=9$ количество слоев M равно 4 и время преобразования, следовательно, равно $4\tau_c$, однако, например, при $n=16$ обе схемы выполняют преобразование за одно и то же время — $6\tau_c$.)

Следует отметить, что метод построения ОУ на преобразователях $N\to2$ может иметь и «экономичную» модификацию, как и методы, рассмотренные в п. 4.4.4, 6.3.1, т. е. модификацию, при которой $n_i=n^{(M-i)}$. «Экономичная» структура для случая $n=9$ иллюстрируется на рис. 6.3.7.

Сопоставляя методы построения многослойных матричных ОУ на параллельных счетчиках (3,2), (7,3), (15,4), (k, m) (см. п. 4.4.4), на k-разрядных параллельных сумматорах (п. 6.3.1) и на преобразователях $N\to2$, можно придти к выводу, что для реализации ОУ на БИС, по-видимому, весьма перспективна последняя группа методов. Умножитель, построенный на преобразователях $N\to2$, мало уступает в быстродействии ОУ на счетчиках (3,2) (классической схеме), но выгодно отличается от нее небольшим отношением f/v количества

f внешних выводов БИС к объему v находящейся в ней аппаратуры. Другие перечисленные ОУ при прочих равных условиях уступают ОУ на преобразователях $N \to 2$ в быстродействии. К сожалению, в настоящее время преобразователи $N \to 2$, выполненные в виде БИС, промышленностью, по-видимому, не выпускаются.

Выполненный в виде БИС преобразователь $N \to 2$, описанный в этом параграфе, состоит, как мы видели,

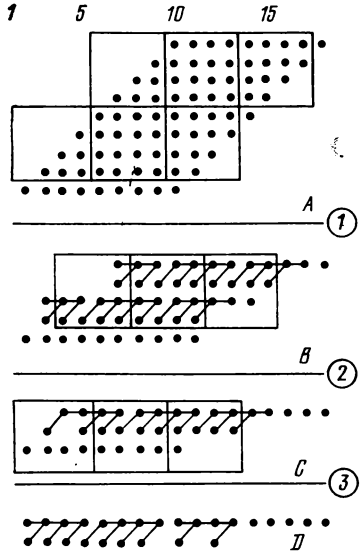


Рис. 6.3.6

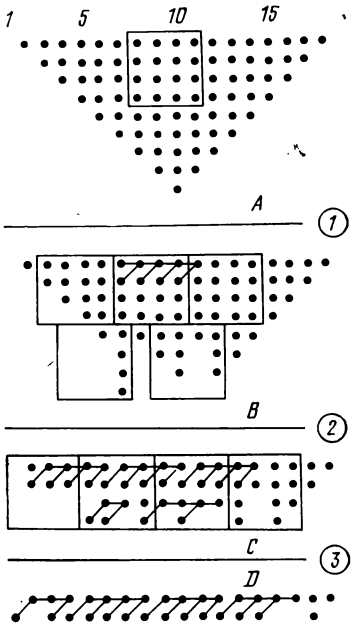


Рис. 6.3.7

из k одинаковых разрядных схем и имеет $f=kN+2N+2k-6$ внешних выводов. Можно показать, что если допустить, чтобы в БИС крайняя левая и крайняя правая разрядные схемы отличались от остальных (одинаковых) разрядных схем, то можно, вообще говоря, добиться уменьшения количества f внешних выводов БИС при сохранении того же объема v аппаратуры преобразователя $N \to 2$ и при том же для данного n количестве M слоев (и, следовательно, том же быстродействии) ОУ. Сказанное поясняется рис. 6.3.8, на котором показан

преобразователь 9→2, содержащий столько же аппаратуры, как и преобразователь 9→2, изображенный на рис. 6.3.4,*а*. Новый преобразователь имеет на 6 внешних выводов меньше. Экономия достигнута путем такого проведения «боковых» границ БИС, при котором «перерезается» минимальное количество «боковых» сигналов *).

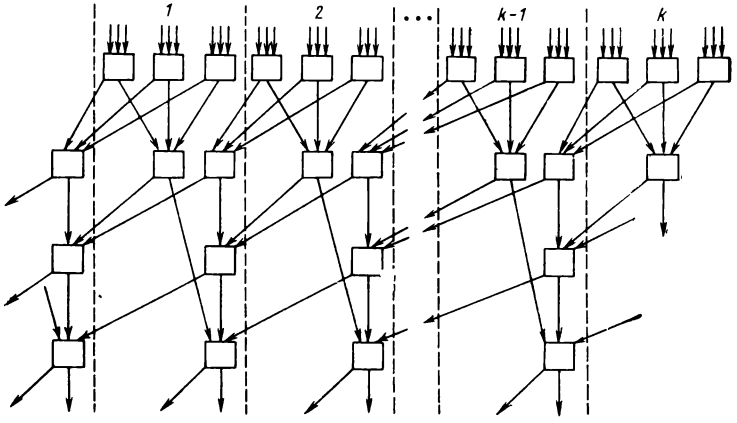


Рис. 6.3.8

Ясно, что узлы, один из которых показан на рис. 6.3.8, тоже могут соединяться боковыми сторонами в длинную цепочку, образуя такой же ряд преобразователей 9→2, как и цепочка узлов, один из которых изображен на рис. 6.3.4,*а*.

6.3.3. СОСТАВНЫЕ ОДНОТАКТНЫЕ УМНОЖИТЕЛИ НА БОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМАХ

Если в распоряжении разработчика имеются БИС, каждая из которых представляет собой ОУ того или иного типа, способный перемножать малоразрядные сомножители, то составить из таких «элементарных» ОУ большой ОУ можно, вообще говоря, многими способами. Рассмотрим несколько примеров.

Пусть для синтеза ОУ используются умножители 4×4 [21—23], каждый из которых выполнен в виде ДЗУ (долговременное запоминающее устройство, «read

*) Эта возможность указана В. Я. Горштейном.

only memory») [21, 22] или в виде матричного ОУ [23] и допустим, что требуется составить ОУ, перемножающий 8-разрядные числа.

Для такого устройства нужно использовать 4 БИС. Одна БИС формирует произведение (назовем его частичным произведением) четырех старших (или младших) разрядов множимого на четыре старших (или младших) разряда множителя. Частичные произведения можно складывать сумматорами так [21, 22], как это показано на рис. 6.3.9, или так [23], как показано на

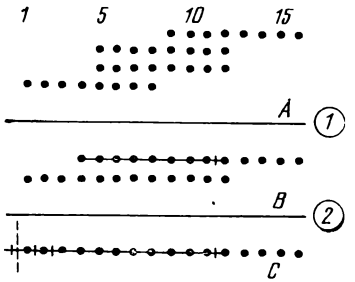


Рис. 6.3.9

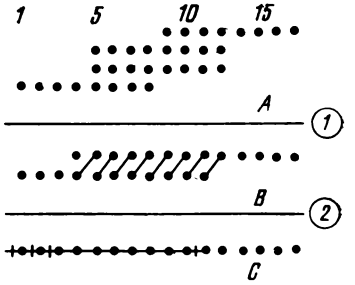


Рис. 6.3.10

рис. 6.3.10 (на рисунках использованы те же обозначения, что и в п. 4.4.4). Видно, что в первом случае (рис 6.3.9) используется 8-разрядный и 12-разрядный параллельные сумматоры, а во втором случае — 8 одноразрядных сумматоров и 11-разрядный параллельный сумматор. Из рис. 6.3.9, впрочем, видно, что если в младшем разряде 8-разрядного сумматора складывать не 2, а 3 цифры, то второй сумматор станет 11-разрядным и объемы аппаратуры суммирующих схем, показанных на рис. 6.3.9 и 6.3.10, сравняются. Достоинство второй из этих двух схем состоит в том, что ее легче ускорить (нужно ускорять только 11-разрядный сумматор).

Интересная идея построения ОУ использована в микросхемах *SN*54284, *SN*54285 фирмы Texas Instruments. Одна из этих схем формирует старшие четыре, а другая — младшие четыре разряда 8-разрядного произведения двух 4-разрядных сомножителей. Благодаря этому каждая микросхема располагается в корпусе с 16 выводами. Одна пара схем *SN*54284, *SN*54285 представляет собой, таким образом, ОУ 4×4. В каталоге [24] на стр. 496, 497 описано составное ОУ 8×8, построенное

в соответствии с рис. 6.3.10. В этом умножителе четыре пары схем *SN*54284, *SN*54285 за 40 нс вырабатывают четыре произведения 4×4, затем четыре схемы *SN*54*H*183, каждая из которых содержит два одноразрядных сумматора, за 12 нс преобразуют трехрядный код произведения в двухрядный, и, наконец, сверхпараллельный сумматор, составленный из трех схем *SN*54*S*181 и одной схемы *SN*54*S*182 (см. § 6.2), за 22 нс формирует 11 старших разрядов произведения. Время умножения, таким образом, равно 74 нс.

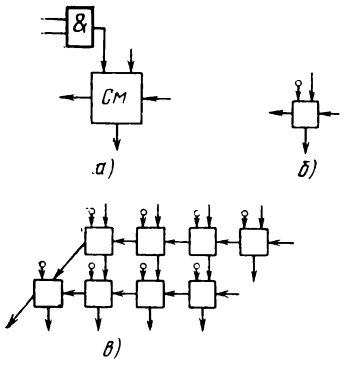


Рис. 6.3.11

Необходимость использовать дополнительные сумматоры для сложения частичных произведений, вырабатываемых элементарными ОУ (из которых составляется большой ОУ), вообще говоря, характерна для подобных построений. Однако этого можно избежать, если в схеме самой БИС имеется аппаратура для прибавления некоторого числа к формируемому частичному произведению. Пример такого составного ОУ приведен в [16]. В данном случае интегральная схема состоит из восьми одноразрядных двоичных сумматоров, каждый из которых соединен с двухвходовой схемой И (рис. 6.3.11,*а*). Такой узел в условном виде изображен на рис. 6.3.11,*б*. Восемь указанных узлов соединены в интегральной схеме так, как это показано на рис. 6.3.11,*в*. Видно, что эта схема может играть роль ОУ 4×2 (на входы каждой схемы И подается одна цифра множимого и одна цифра множителя), причем к 6-разрядному произведению может прибавляться произвольное 4-разрядное число и еще две двоичные цифры. Эта схема требует 20 контактов (18 логических и 2 для питающих напряжений) и может быть размещена в корпусе с 24 выводами.

Схема составного ОУ 8×8, содержащего 8 таких интегральных схем, приведена на рис. 6.3.12. Весь ОУ составлен из одинаковых узлов и не требует дополнительных сумматоров. Если считать, что в каждом одноразрядном сумматоре $\tau_{см}=\tau_e=\tau$, то время работы та-

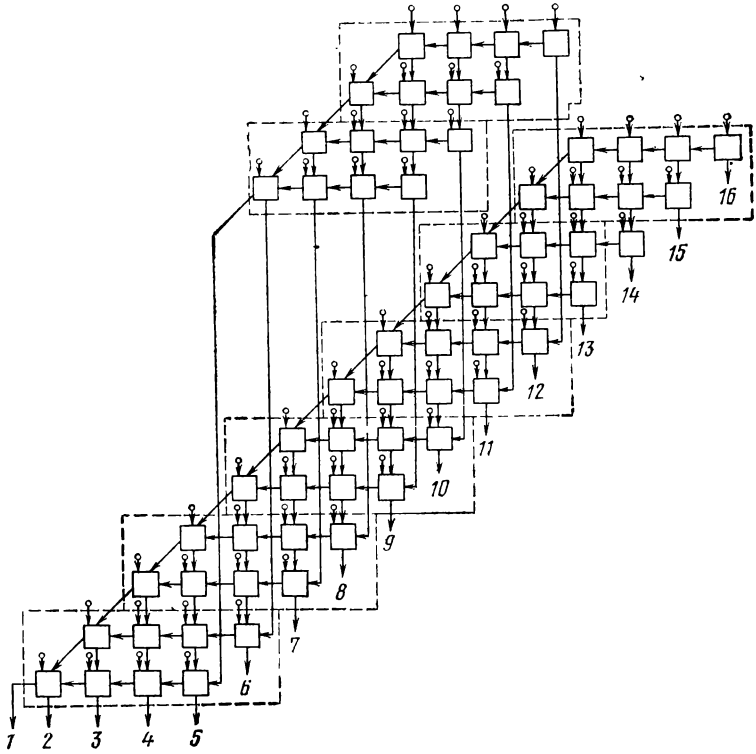


Рис. 6.3.12

кого ОУ равно $25\tau+\tau_и$, где $\tau_и$ — время срабатывания входной схемы И.

В рассмотренных примерах элементарные ОУ были малоразрядными, но и более крупные схемы могут соединяться аналогичными способами.

6.4. ПОСТРОЕНИЕ СДВИГАТЕЛЕЙ, ДЕШИФРАТОРОВ, КОММУТАТОРОВ НА БОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМАХ

Много места в арифметических устройствах современных ЦВМ обычно занимают сумматоры, умножители и регистры. Построение быстродействующих сумматоров и умножителей на БИС мы рассматривали в § 6.2, 6.3. Существует множество интегральных схем разного уровня интеграции, содержащих триггерные регистры того

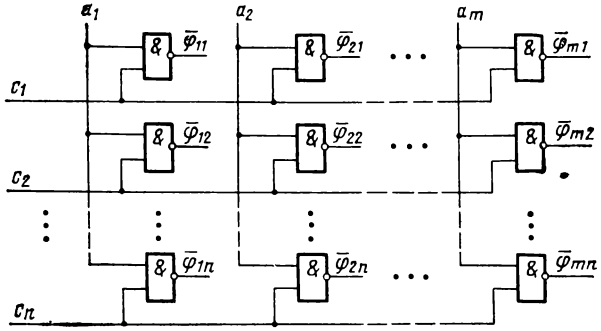


Рис. 6.4.1

или иного типа. Эти схемы широко известны и мы не будем их рассматривать.

Значительную часть арифметического устройства составляют также дешифраторы, коммутаторы, сдвигатели и счетчики. Эти узлы широко используются также и в других устройствах вычислительных машин и систем — в устройствах центрального управления и в логической части ЗУ и устройств ввода—вывода. Номенклатура интегральных схем, содержащих подобные узлы, тоже достаточно широка. Мы отметим здесь только одно обстоятельство: в ряде случаев объем v аппаратуры, находящейся в БИС, и количество f внешних выводов БИС являются функциями двух структурных параметров и задача минимизации критерия оптимальности может иметь строгое решение.

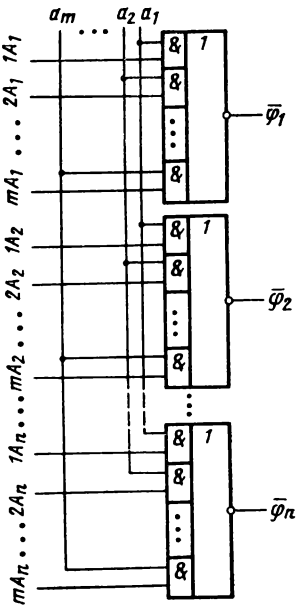


Рис. 6.4.2

На рис. 6.4.1 приведена известная схема адресного дешифратора, которая имеет ряд других применений и,

в частности, может быть использована в качестве узла формирования частичных произведений в матричном умножителе. Узел вырабатывает mn логических функций вида $\overline{\varphi_{ij}}=\overline{a_i c_j}$ ($i=1, 2, 3, \ldots, m$; $j=1, 2, \ldots, n$) и хорошо подходит для реализации в виде БИС, так как обладает сравнительно небольшим числом f внешних выводов ($m+n+mn$) при значительном объеме v аппаратуры (mn узлов формирования функций $\overline{\varphi_{ij}}$). Таким образом, отношение f/v равно

$$f(m, n)/v(m, n)=(mn+m+n)/mn. \qquad (6.4.1)$$

На рис. 6.4.2 приведена схема n-разрядного коммутатора, сквозь который может передаваться одно из m n-разрядных чисел $1A$, $2A$, ..., mA:

если $a_j=1$, $a_1=a_2=\ldots=a_{j-1}=a_{j+1}=a_{j+2}=\ldots=$
$=a_m=0$, то $\overline{\varphi_1}=\overline{jA_1}$, $\overline{\varphi_2}=\overline{jA_2}$, ..., $\overline{\varphi_n}=\overline{jA_n}$.

Кроме коммутирования потоков информации, схемы этого типа могут выполнять ряд других функций. В частности, на их основе могут создаваться разнообразные быстродействующие сдвигатели [25]. Структура коммутатора, приведенная на рис. 6.4.2, в определенном смысле обратна структуре только что рассмотренного дешифратора. Параметр f/v БИС, содержащей этот коммутатор, определяется тем же соотношением (6.4.1) (mn — количество элементов И).

6.5. ОПТИМИЗАЦИЯ СТРУКТУРЫ ДВУМЕРНЫХ МАТРИЧНЫХ ИНТЕГРАЛЬНЫХ СХЕМ

В современных быстродействующих ЦВМ времена срабатывания логических элементов соизмеримы с потерями времени на распространение электрических сигналов. Поэтому одной из важнейших проблем, связанных с проектированием вычислительных систем, стала минимизация межсхемных связей. Если некоторая машина или устройство проектируется из интегральных схем определенного размера, то, очевидно, количество и длины межсхемных связей при прочих равных обстоятельствах тем меньше, чем меньше количество использованных интегральных схем и чем меньше количество внешних выводов на этих схемах. Если считать, что объем аппаратуры проектируемого устройства — заданная по-

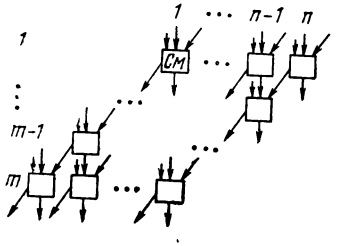


Рис. 6.5.1

стоянная величина, то для достижения этой цели нужно стремиться во-первых, к увеличению количества аппаратуры v в каждой интегральной схеме (при этом будет уменьшаться число этих схем, общее количество связей и их длины) и, во-вторых, к уменьшению числа f внешних выводов интегральной схемы, что приводит к уменьшению количества связей. В силу этих соображений отношение f/v может считаться одним из важнейших критериев оптимальности структуры БИС [2, 12, 20, 25].

При проектировании структуры БИС конструктивные требования обычно выражаются неравенствами

$$v \leqslant V, \quad f \leqslant F, \tag{6.5.1}$$

где V, F — максимально допустимые значения величин v, f.

Рассмотрим задачу минимизации критерия f/v при ограничениях (6.5.1).

В п. 6.3.2 была рассмотрена структура БИС (k-разрядный преобразователь $N \to 2$), для которого критерий оптимальности f/v выглядит следующим образом:

$$p = f/v = (kN + 2N + 2k - 6)/(kN - 2k), \tag{6.5.2}$$

где $v = kN - 2k$ — количество одноразрядных сумматоров в БИС.

В § 6.4 были описаны структуры двух других БИС (дешифратор и коммутатор), для которых критерий f/v тоже является функцией двух параметров:

$$p = f/v = (mn + m + n)/mn.$$

На рис. 6.5.1 приведена схема БИС, представляющей собой однородную структуру, содержащую одноразрядные сумматоры См, и пригодной для синтеза различных однородных ОУ. Для этой БИС

$$p = f/v = (mn + 4n + 2m - 2)/mn,$$

где $v = mn$ — количество одноразрядных сумматоров.

Подобные примеры можно было бы продолжить. Существует достаточно широкий класс БИС, имеющих разные принципы действия и различную внутреннюю организацию, но обладающих более или менее однотипными аналитическими выражениями критерия p, являющегося функцией двух параметров структуры БИС. Оптимизация таких структур, т. е. отыскание целочисленных значений двух указанных параметров, обеспечивающих минимум функции p, производится более или менее одинаковыми способами.

Рассмотрим в качестве примера методику оптимизации параметров N, k преобразователя $N \longrightarrow 2$ [20]. Зависимость $p(N, k)$ определяется соотношением (6.5.2) и иллюстрируется рис. 6.5.2.

Так как $\partial p/\partial N < 0$, $\partial p/\partial k < 0$ при $N > 3$, $k \geqslant 1$, то величина p убывает с ростом N и k. Отметим, что

$$\lim_{\substack{N\to\infty \\ k\to\infty}} p = 1.$$

Каждое из неравенств (6.5.1) ограничивает зону значений N, k, в которой ищется минимум p. Границы этих зон для некоторых значений V, F показаны на рис. 6.5.3. Уравнения границ

$$k = v/(N-2) \quad (v = V = \text{const}),$$

$$k = (f-2N+6)/(N+2) \quad (f = F = \text{const})$$

получаются из соотношений

$$f = kN+2N+2k-6, \tag{6.5.3}$$

$$v = kN-2k. \tag{6.5.4}$$

Легко показать, что значения p вдоль любой из этих границ возрастают в обе стороны от точки пересечения этой границы с прямой $N = 2k+2$ (штриховая линия на рис. 6.5.3).

Действительно, если $f = \text{const}$, то, подставив $k = (f-2N+6)/(N+2)$ в (6.5.4), получим функцию

$$v = (-2N^2+fN+10N-2f-12)/(N+2),$$

производная которой

$$dv/dN = (4f+32-2N^2-8N)/(N+2)^2$$

обращается в нуль при

$$N = N_1 = \sqrt{2f+20} - 2, \quad N = N_2 = -\sqrt{2f+20} - 2.$$

Нетрудно показать, что $N = N_1$ обеспечивает максимум v и, следовательно, минимум p, который равен

$$p_{\min} = 1+8/(N-2)-4/(N-2)^2.$$

Соотношения между N, k, v, f в точке экстремума имеют следующий вид:

$$N = 2k + 2 = 2 + \sqrt{2v} = \sqrt{2f + 20} - 2,$$

$$k = (\ -2)/2 = \sqrt{v/2} = \sqrt{f/2 + 5} - 2,$$

$$v = (N-2)^2/2 = 2k^2 = f - 4\sqrt{2f + 20} + 18,$$

$$f = (N+2)^2/2 - 10 = 2(k+2)^2 - 10 = v + 8\sqrt{v/2} - 2. \qquad (6.5.5)$$

К этим же соотношениям можно прийти, если положить $v=$ $=$const, подставить $k=v/(N-2)$ в (6.5.3) и приравнять нулю производную df/dN (минимум f в этом случае обеспечивает минимум p).

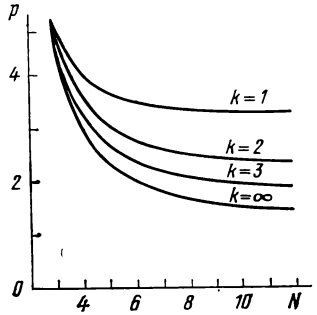


Рис. 6.5.2

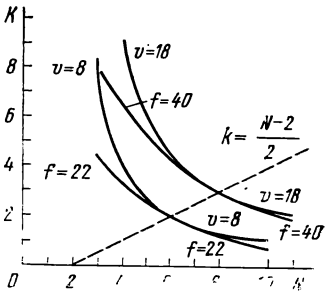


Рис. 6.5.3

Из сказанного следует, что при ограничениях (6.5.1) минимум p достигается при

$$N = \min(N_V, N_F),\ k = (N-2)/2, \qquad (6.5.6)$$

где $N_V = 2 + 2\sqrt{V}$, $N_F = \sqrt{2F + 20} - 2$. Вычисленные таким образом параметры N и k могут оказаться не целыми числами. В этих случаях следует выбирать N и k, близкие к получающимся по (6.5.6). Введем обозначение:

$$Q = \min(N_V, N_F).$$

Можно показать, что оптимальные целочисленные значения параметров N, k могут быть выбраны при помощи следующей процедуры.

Если $Q=2l$ ($l=1, 2, \ldots$), то минимум p обеспечивается при $N=2l$, $k=l-1$.

Если $2l<Q<2l+2$, то для обеспечения минимума p следует выбрать такую из четырех пар чисел

$$\left.\begin{matrix} N=2l, \\ k=l-1 \end{matrix}\right\};\quad \left.\begin{matrix} N=2l+1, \\ k=l-1 \end{matrix}\right\};\quad \left.\begin{matrix} N=2l, \\ k=l \end{matrix}\right\};\quad \left.\begin{matrix} N=2l+1, \\ k=l \end{matrix}\right\},$$

Таблица 6.5.1

v	2	8	18	32	50	72	98
f	8	22	40	62	88	118	154
N	4	6	8	10	12	14	16
k	1	2	3	4	5	6	7
p	4	2,75	2,22	1,94	1,76	1,64	1,55

для которой $v \leqslant V$, $f \leqslant F$, причем для следующей по порядку пары хотя бы одно из этих двух неравенств должно не выполняться. Решение

$$\left.\begin{aligned} N &= 2l \\ k &= l \end{aligned}\right\}$$

может быть заменено эквивалентным (те же v, f, p) решением

$$\left.\begin{aligned} N &= 2l + 2 \\ k &= l - 1 \end{aligned}\right\}.$$

На рис. (6.5.4) штриховой линией показана зависимость $p_{\min}$ от N. Здесь же приведены графики $p(N)$ для нескольких фиксированных v. В табл. 6.5.1 приведено несколько первых целочисленных решений системы уравнений (6.5.5). Видно, что уже при $N=10$ величина p оказывается в два с половиной раза меньше, чем для БИС, содержащей не связанные между собой одноразрядные сумматоры.

Следует отметить, что при проектировании структуры БИС, кроме минимизации критерия f/v, нужно стремиться к удовлетворению ряда других требований и пожеланий, анализ которых не может быть произведен в общем виде. Например, во многих случаях желательно, чтобы разрядность схемы, находящейся в БИС, была равна целой степени двойки. Можно также учитывать, что некоторый узел, разделенный на независимые части (например, сумматор, разделенный на несколько

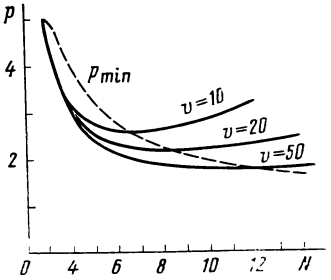


Рис. 6.5.4

более коротких сумматоров), в некотором отношении более универсален, чем неразделенная схема: эти части можно использовать порознь. Подобных соображений может оказаться множество.

7. МАТЕМАТИЧЕСКИЕ ОСНОВЫ ПОСТРОЕНИЯ ЛОГИЧЕСКИХ И АРИФМЕТИЧЕСКИХ УСТРОЙСТВ И УЗЛОВ ЦВМ

В этой главе рассматриваются форма и точность представления чисел в ЦВМ, способы (неускоренные) выполнения основных операций (сложения, вычитания, умножения, деления), системы элементов и структуры типовых узлов (регистров, сдвигателей, коммутаторов, сумматоров).

7.1. ОСНОВЫ АЛГЕБРЫ ЛОГИКИ

При создании различных узлов ЦВМ, в особенности при выборе оптимальной с некоторой точки зрения структуры узла, значительную помощь разработчику может оказать математический аппарат алгебры логики, изучающей высказывания, рассматриваемые со стороны их логических значений (истинности или ложности) и логических операций над ними. Приведем краткие сведения из этой области, достаточные, однако, для большинства практических применений.

В алгебре логики любая логическая функция

$$f(A, C, \ldots)$$

от логических переменных A, C, ..., как и любая переменная A, C, ..., может принимать только два значения (истинное и ложное), для обозначения которых можно ввести символы «И» и «Л». Вместо этих символов обычно употребляют (что мы и делаем) числа 1 и 0. Такой характер логических функций и переменных фактически совпадает с характером сигналов в большей части аппаратуры большинства вычислительных машин — это сигналы также могут принимать только два значения.

Можно показать, что количество всевозможных функций от k переменных равно 2^{2^k}. Четыре функции f_1, ..., f_4 одной переменной A приведены в табл. 7.1.1. Все функции двух переменных показаны в табл. 7.1.2.

Две первые из них (f_1, f_2) фактически не зависят от переменных A, C и называются const 0 (*константа 0*) и const 1 (*константа 1*).

Функции f_3, ..., f_6, как видно из таблицы, фактически зависят только от одной переменной, причем f_3 и f_4 просто повторяют значения переменных A и C, а функции f_5 и f_6 соответственно обратны

переменным A и C. Каждая из функций f_5, f_6 называется функцией *отрицания* (функцией НЕ, *инверсией*) и обычно обозначается, как это сделано в табл. 7.1.2, названием соответствующей переменной с черточкой сверху. Физический элемент, реализующий функцию НЕ, обычно называют *инвертором*.

Функция f_7 называется *дизъюнкцией* (функцией ИЛИ, *логической суммой*) и обозначается в виде $A \vee C$ или $A+C$.

Таблица 7.1.1

A	f_1	f_2	f_3	f_4
0	0	1	0	1
1	0	1	1	0

Функция f_8 называется *конъюнкцией* (функцией И, *логическим произведением*) и обозначается в виде $A \wedge C$ или $A\&C$, или AC.

Функция f_9 называется *стрелкой Пирса*. Она обратна (инверсна) функции f_7 и поэтому может быть записана в виде $\overline{A+C}$. Элемент, реализующий эту функцию, обычно называется элементом ИЛИ—НЕ (в зарубежной литературе принято название NOR). Эту же функцию можно рассматривать как логическое произведение функций f_5 и f_6 и, следовательно, записать в виде $\bar{A}\bar{C}$.

Функция f_{10} называется *штрихом Шеффера*. Из таблицы видно, что $f_{10}=\bar{A}+\bar{C}=\overline{AC}$. Элемент, реализующий f_{10}, называется И—НЕ (за рубежом — NAND).

Функция f_{11} называется функцией *эквивалентности* или *равнозначности*, так как она равна 1 при $A=C$ и равна 0 при $A \neq C$. Эта функция иногда обозначается в виде $A \sim C$. Функции эквивалентности обратна функции f_{12}, которая называется функцией *неэквивалентности (неравнозначности, суммой по модулю два, исключающим ИЛИ)*. Эта функция иногда изображается в виде $A \oplus C$.

Каждая из функций f_{13}, f_{14} называется *импликацией*. Иногда используются обозначения $A \longrightarrow C$, $C \longrightarrow A$.

Из табл. 7.1.2 видно, что любая из функций f_3, ..., f_{16} может быть выражена через переменные A, C и функции И, ИЛИ, НЕ. Функции f_1 и f_2 тоже можно выразить с помощью трех указанных функций: $\text{const}\,0=A\bar{A}$, $\text{const}\,1=A+\bar{A}$. Набор функций, через которые можно выразить любую другую функцию, называется *функционально полным* набором. Функции И, ИЛИ, НЕ не являются единственным функционально полным набором. С помощью элементов, реализующих все функции функционально полного набора, можно, следовательно, реализовать любые другие функции.

Функции И, ИЛИ можно распространить на случай нескольких переменных: функция И равна 1 в том и только в том случае, когда каждая из переменных равна 1; функция ИЛИ равна 1 в том и только в том случае, когда хотя бы одна из переменных равна 1.

На примере функции $f_9=\overline{A+C}=\bar{A}\bar{C}$ из табл. 7.1.2 и на примере

Переменные A C	f_1 const 0	f_2 const 1	f_3 A	f_4 C	f_5 $\bar{A}$	f_6 $\bar{C}$	f_7 $A+C$	f_8 AC
0 0	0	1	0	0	1	1	0	0
0 1	0	1	0	1	1	0	1	0
1 0	0	1	1	0	0	1	1	0
1 1	0	1	1	1	0	0	1	1

функции f_{10} мы видели, что одну и ту же функцию можно разными способами выразить через функции И, ИЛИ, НЕ и, следовательно, реализовать разными схемами. Цель преобразований логических формул обычно и состоит в том, чтобы найти такую запись формулы, которой соответствует схема, оптимальная с той или иной точки зрения.

Ниже приведены некоторые наиболее употребимые равенства, играющие важную роль при преобразовании логических формул в другие равные формулы:

$$AC=CA,\ A+C=C+A$$

(закон коммутативности),

$$(AC)B=A(CB),\ (A+C)+B=A+(C+B)$$

(закон ассоциативности),

$$A(A+B)=A,\ A+AB=A$$

(закон поглощения)

$$A(C+B)=AC+AB,\ A+(CB)=(A+C)(A+B)$$

(закон дистрибутивности),

$$A\bar{A}=0$$

(закон противоречия)

$$A+\bar{A}=1$$

(закон исключенного третьего),

$$A+A=A$$

(закон идемпотентности),

$$A+1=1,\ A1=A,\ AA=A,\ A+\bar{A}C=A+C,$$

$$\overline{AC+CB+AB}=\bar{A}\bar{C}+\bar{C}\bar{B}+\bar{A}\bar{B},$$

$$\overline{A+C+B+\ldots}=\bar{A}\bar{C}\bar{B}\ldots,\ \overline{ACB\ldots}=\bar{A}+\bar{C}+\bar{B}+\ldots$$

Некоторые из этих правил основаны на теореме *Моргана*, утверждающей, что всякая логическая функция преобразовывается в инверсную, если в формуле этой функции все операции И заменить на ИЛИ и наоборот и все переменные заменить их инверсиями.

Таблица 7.1.2

f_9 $\overline{AC}$	f_{10} $\overline{A}+\overline{C}$	f_{11} $AC+\overline{A}\,\overline{C}$	f_{12} $\overline{AC}+A\overline{C}$	f_{13} $A+\overline{C}$	f_{14} $\overline{C}+A$	f_{15} $A\overline{C}$	f_{16} $\overline{A}C$
1	1	1	0	1	1	0	0
0	1	0	1	1	0	0	1
0	1	0	1	0	1	1	0
0	0	1	0	1	1	0	0

7.2. ПРЕДСТАВЛЕНИЕ ЧИСЕЛ С ФИКСИРОВАННОЙ ЗАПЯТОЙ

В современных ЦВМ используются весьма разнообразные способы представления чисел. Мы, однако, ограничимся кратким рассмотрением самой распространенной двоичной системы счисления.

Цифра в каждом из разрядов двоичного числа может принимать только два значения — 0 и 1. В реальных ЦВМ разрядность чисел всегда конечна. Обозначив число разрядов через n, можем записать двоичное число A в виде

$$a_1a_2 \ldots a_n.$$

Здесь a_i — цифры в разрядах, i — номер разряда. Будем считать, что старший разряд написан слева, т. е. имеет номер 1, младший разряд — справа.

Запятая, отделяющая целую часть чисел с фиксированной запятой от дробной части, может стоять, вообще говоря, в любом, но постоянном для данной ЦВМ месте. Для определенности будем считать, что запятая, как это чаще всего бывает в ЦВМ, стоит слева от старшего разряда:

$$A=,a_1a_2 \ldots a_n.$$

Цена единицы каждого разряда (вес разряда) в двоичной системе в два раза больше цены единицы соседнего младшего разряда. В нашем случае вес первого разряда равен 2^{-1}, вес второго разряда равен 2^{-2} и т. д. Таким образом,

$$A=\sum_{i=1}^{n} a_i 2^{-i}.$$

До сих пор мы говорили фактически о положительных числах A. Очевидно, что величины этих чисел лежат в диапазоне

$$0 \leqslant A \leqslant 1-2^{-n}.$$

Для представления положительных и отрицательных чисел с фиксированной запятой в современных ЦВМ используются в основном три способа — в прямых, дополнительных и обратных кодах. В табл. 7.2.1 показано представление всех n-разрядных двоичных положительных и отрицательных чисел A всеми тремя способами.

Таблица 7.2.1

A	Код		
	прямой	дополнительный	обратный
$1-2^{-n}$	0,11 . . . 1111	0,11 . . . 1111	0,11 . . . 1111
$1-2\cdot2^{-n}$	0,11 . . . 1110	0,11 . . . 1110	0,11 . . . 1110
$1-3\cdot2^{-n}$	0,11 . . . 1101	0,11 . . . 1101	0,11 . . . 1101
$1-4\cdot2^{-n}$	0,11 . . . 1100	0,11 . . . 1100	0,11 . . . 1100
$1-5\cdot2^{-n}$	0,11 . . . 1011	0,11 . . . 1011	0,11 . . . 1011
. . . .	. . . . .	. . . . .	. . . . .
$4\cdot2^{-n}$	0,00 . . . 0100	0,00 . . . 0100	0,00 . . . 0100
$3\cdot2^{-n}$	0,00 . . . 0011	0,00 . . . 0011	0,00 . . . 0011
$2\cdot2^{-n}$	0,00 . . . 0010	0,00 . . . 0010	0,00 . . . 0010
2^{-n}	0,00 . . . 0001	0,00 . . . 0001	0,00 . . . 0001
$+0$	0,00 . . . 0000	0,00 . . . 0000	0,00 . . . 0000
	n разрядов	n разрядов	n разрядов
-0	1,00 . . . 0000		1,11 . . . 1111
-2^{-n}	1,00 . . . 0001	1,11 . . . 1111	1,11 . . . 1110
$-2\cdot2^{-n}$	1,00 . . . 0010	1,11 . . . 1110	1,11 . . . 1101
$-3\cdot2^{-n}$	1,01 . . . 0011	1,11 . . . 1101	1,11 . . . 1100
$-4\cdot2^{-n}$	1,00 . . . 0100	1,11 . . . 1100	1,11 . . . 1011
. . . .	. . . . .	. . . . .	. . . . .
$-(1-4\cdot2^{-n})$	1,11 . . . 1100	1,00 . . . 0100	1,00 . . . 0011
$-(1-3\cdot2^{-n})$	1,11 . . . 1101	1,00 . . . 0011	1,00 . . . 0010
$-(1-2\cdot2^{-n})$	1,11 . . . 1110	1,00 . . . 0010	1,00 . . . 0001
$-(1-2^{-n})$	1,11 . . . 1111	1,00 . . . 0001	1,00 . . . 0000
-1		1,00 . . . 0000	

Из таблицы видно, что в изображение числа A добавлен слева от запятой еще один разряд, который фактически является разрядом, изображающим знак числа: 0 в этом разряде означает знак «+», 1 означает знак «—».

В качестве примера приведем представление чисел $A=+3/4$, $A=-3/4$ всеми тремя способами ($n=5$)

$$+\frac{3}{4} \rightarrow 0{,}11000 \text{ (пр)} \quad 0{,}11000 \text{ (доп)} \quad 0{,}11000 \text{ (обр)}$$

$$-\frac{3}{4} \rightarrow 1{,}11000 \qquad 1{,}01000 \qquad 1{,}00111$$

Из таблицы и примеров видно, что при использовании первого способа (прямой код) представление числа содержит знак и абсолютную величину $|A|$, при использовании обратного кода отрицательного числа представление содержит знак и поразрядные инверсии цифр числа $|A|$, при использовании дополнительного кода отрицательного числа представление содержит знак и величину $1-|A|$, т. е. дополнение числа $|A|$ до единицы. Для положительных чисел все три способа представления совпадают.

Кроме обычного обратного и дополнительного кода, показанных в табл. 7.2.1, внутри арифметического устройства иногда использу-

ются модифицированный дополнительный и модифицированный обратный коды, отличающиеся только «раздвоением» знакового разряда. Число $-\frac{3}{4}$ из предыдущего примера в этом случае записывается так:

$$-\frac{3}{4} \to 11{,}01000 \text{ (доп)} \quad 11{,}00111 \text{ (обр)}$$

Достоинство модифицированного кода состоит в упрощении (незначительном) процедуры обнаружения выхода результата операции из допустимого диапазона, т. е. из диапазона

$$-1 \leqslant A \leqslant 1-2^{-n}$$

в случае использования дополнительных кодов и диапазона

$$-(1-2^{-n}) \leqslant A \leqslant 1-2^{-n}$$

в случае использования обратных кодов.

Для представления чисел во всех частях ЦВМ — в памяти, на входе и выходе арифметического устройства и т. д. — используется обычно либо прямой, либо дополнительный код; для представления чисел внутри арифметического устройства используются, вообще говоря, все три кода: прямой, дополнительный и обратный.

7.3. ПРЕДСТАВЛЕНИЕ ЧИСЕЛ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Недостаток способа представления чисел с фиксированной запятой состоит в том, что при неточной работе программиста абсолютная величина многих обрабатываемых чисел оказывается гораздо меньше единицы, т. е. в этих числах некоторое, иногда довольно большое количество старших разрядов равно нулю. Это приводит к уменьшению числа значащих разрядов и, следовательно, к потере точности вычислений.

Другой недостаток фиксированной запятой состоит в том, что результаты тех или иных операций могут оказаться по абсолютной величине больше единицы (это также происходит из-за неточностей в программе). В этом случае обычно вычисления прекращаются и программист корректирует свою программу. Таким образом, оба указанных недостатка связаны с проблемой масштабирования.

Представление чисел в форме с плавающей запятой обеспечивает автоматическое масштабирование чисел, благодаря чему количество значащих разрядов остается всегда постоянным. Наиболее распространен способ представления числа A в виде

$$A = 2^{\alpha}a,$$

где a — мантисса числа A, α — порядок числа A.

В машине число A фактически представляется некоторым количеством двоичных знаков. Обычно левый двоичный разряд является разрядом знака мантиссы или, что то же самое, знаком числа A. Чаще всего 1 изображает знак «—», а 0 — знак «+». Далее следуют разряды (обозначим их количество через $m+1$), изображающие порядок α, и в конце размещаются n двоичных разрядов, представляющих мантиссу a (иногда разряды порядка размещаются после разрядов мантиссы).

Порядок α является целым числом, лежащим в диапазоне

$$p_{\min} \leqslant \alpha \leqslant p_{\max},$$

где $p_{\max}$, $p_{\min}$ — максимальное и минимальное допустимые значения порядка. Наиболее распространены два способа изображения порядка α при помощи $m+1$ двоичного разряда (табл. 7.3.1). Мы будем придерживаться первого способа, при котором $p_{\max}=2^m-1$, $p_{\min}=-2^m$.

Таблица 7.3.1

α	Первый способ	Второй способ	α	Первый способ	Второй способ
2^m-1	111...111	011...111	-0	———	100...000
2^m-2	111...110	011...110	-1	011...111	100...001
2^m-3	111...101	011...101	-2	011...110	100...010
2^m-4	111...100	011...100	-3	011...101	100...011
. . . .	. . . . .	. . . . .	. . . .	. . . . .	. . . . .
3	100...011	000...011	$-(2^m-3)$	000...011	111...101
2	100...010	000...010	$-(2^m-2)$	000...010	111...110
1	100...001	000...001	$-(2^m-1)$	000...001	111...111
0	100...000 (m разрядов)	000...000 (m разрядов)	-2^m	000...000	———

Мантисса обычно представляется в прямом коде. Поэтому, говоря о мантиссе a, будем иметь в виду ее абсолютную величину (знак мантиссы, как уже сказано, является знаком всего числа A и помещается отдельно, в начале числа). Представление мантиссы фактически совпадает с описанным в § 7.2 представлением чисел с фиксированной запятой. Единственное, но весьма существенное отличие состоит в том, что старший разряд мантиссы всегда (за исключением одного особого случая, о котором речь будет ниже) равен единице. Иными словами, величина мантиссы не выходит из диапазона

$$1/2 \leqslant a \leqslant 1-2^{-n},$$

а само число $|A|$ лежит в диапазоне

$$2^{p_{\min}} \cdot 1/2 \leqslant |A| \leqslant 2^{p_{\max}}(1-2^{-n}).$$

Такое число называется *нормализованным*.

Исключением является число $A=0$, которое обычно представляется в виде $A=+2^{p_{\min}} \cdot 0$. Очевидно, что при использовании первого способа представления порядков (см. табл. 7.3.1) такое число фактически представляется $1+(m+1)+n$ нулями.

7.4. ТОЧНОСТЬ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ

Погрешности представления чисел не следует смешивать с погрешностями операций или, как их еще называют, аппаратными погрешностями. Аппаратные погрешности вызываются тем, что в ряде случаев арифметическое устройство вычисляет не точный, а приближенный результат операции (например, умножения или деления) над двумя числами A и C, поступившими в устройство.

Абсолютной погрешностью представления числа назовем разность $|A^*-A|$ между точным значением A^* некоторого числа и тем значением A, которое может быть введено в конкретную ЦВМ или конкретное арифметическое устройство. Будем считать, что число A^*, будучи представлено в двоичной системе счисления, может содержать любое количество значащих двоичных разрядов. Разрядность числа A определяется разрядностью машины.

Обычно различают несколько разновидностей абсолютных и относительных погрешностей представления (табл. 7.4.1). Среди них несколько типов максимальных и средних погрешностей.

Таблица 7.4.1

	Δ	Макс	Макс макс	Ср	Ср ср
Абс	$\|A^*-A\|$	$\|A^*-A\|_{макс}$	—	$\|A^*-A\|_{ср}$	—
Отн	$\dfrac{\|A^*-A\|}{\|A^*\|}$	$\dfrac{\|A^*-A\|_{макс}}{\|A^*\|}$	$\dfrac{\|A^*-A\|_{макс}}{\|A^*\|_{мин}}$	$\dfrac{\|A^*-A\|_{ср}}{\|A^*\|}$	$\dfrac{\|A^*-A\|_{ср}}{\|A^*\|_{ср}}$

Наиболее универсальным показателем точностных свойств представления чаще всего считается погрешность

$$\Delta_{отн\ ср\ ср}=\frac{|A^*-A|_{ср}}{|A^*|_{ср}}, \qquad (7.4.1)$$

при расчете которой производится, во-первых, усреднение по всевозможным разностям $|A^*-A|$, во-вторых, усреднение по самим величинам $|A^*|$.

Для n-разрядного двоичного числа с фиксированной запятой следует считать, что

$$\Delta_{отн\ ср\ ср}=2^{-n-1}. \qquad (7.4.2)$$

Эта формула выводится следующим образом. Так как мы полагаем, что число $|A|$ содержит справа от запятой n разрядов, а число $|A^*|$ — сколько угодно разрядов (предполагается, что $|A^*|<1$, $|A|<1$), то будем считать, что число A получается из A^* путем отбрасывания разрядов, более младших чем n-й, и одновременного округления. Иными словами, чтобы получить A из A^*, нужно отбросить в A^* разряды с номерами $n+1$, $n+2$, ... и после этого прибавить к оставшемуся числу величину 2^{-n}, если в старшем из отброшенных разрядов стояла единица. При таком способе уменьшаются как средние, так и максимальные погрешности. Очевидно, что при этом

$$0\leqslant|A^*-A|\leqslant 2^{-n-1}.$$

Полагая, что все разряды числа A^* являются независимыми случайными величинами, принимающими с одинаковой вероятностью значения 0 или 1, приходим к тому, что средняя величина этой разности, стоящая в числителе дроби (7.4.1), равна

$$|A^*-A|_{ср}=2^{-n-2}. \qquad (7.4.3)$$

Из допущения о независимости цифр в разрядах числа A^* следует, что число $|A^*|$ в диапазоне

$$0\leqslant|A^*|<1$$

принимает все значения с одинаковой вероятностью (точнее, имеет в этом диапазоне равномерный закон распределения). Поэтому стоящую в знаменателе дроби (7.4.1) среднюю величину $|A^*|_{ср}$ числа $|A^*|$ следует считать равной 1/2. Подставляя это значение и (7.4.3) в (7.4.1), получаем (7.4.2).

Для чисел с плавающей запятой анализ погрешностей представления приводит к не столь очевидным результатам.

Рассмотрим более общий случай представления чисел с плавающей запятой, чем мы это делали в § 7.3. Пусть число $|A|$ представляется в следующей форме:

$$|A|=P^{\alpha}a, \qquad (7.4.4)$$

где α — порядок; a — мантисса; $P=2^k$ (k целое положительное).

Будем считать, что α — целое число и представляется, как обычно, $(m+1)$-разрядным двоичным числом, т. е.

$$-2^m\leqslant\alpha\leqslant2^m-1.$$

Будем также считать, что мантисса a представляется n-разрядным двоичным числом с запятой, фиксированной слева от старшего разряда. При этом величина мантиссы (не считая случая $a=0$) должна лежать в диапазоне

$$1/P\leqslant a\leqslant1-2^{-n}. \qquad (7.4.5)$$

В частном случае, когда $P=2$, получается уже знакомая нам двоичная система. В общем случае число $|A|$, записанное в форме (7.4.4), фактически представлено в P-ичной системе счисления, причем каждая P-ичная цифра мантиссы представляется k-разрядным двоичным числом (поэтому желательно, чтобы n было кратно k).

Определим значения всех погрешностей, указанных в табл. 7.4.1.

Очевидно, что

$$P^{-2^m-1}=P^{\alpha_{\min}}\frac{1}{P}\leqslant|A|\leqslant P^{\alpha_{\max}}(1-2^{-n})=P^{2^m-1}(1-2^{-n}),$$

$$0\leqslant\Delta_{абс}=|A^*-A|\leqslant P^{\alpha}2^{-n-1},$$

$$\Delta_{абс\ макс}=|A^*-A|_{макс}=P^{\alpha}2^{-n-1}$$

(имеется в виду погрешность, максимальная при данном α),

$$\Delta_{абс\ ср}=|A^*-A|_{ср}=P^{\alpha}2^{-n-2}$$

(среднее значение погрешности при данном α),

$$0 \leqslant \Delta_{\text{отн}} = \frac{|A^* - A|}{|A^*|} \leqslant \frac{P^{\alpha} 2^{-n-1}}{P^{\alpha} a_{\min}} = P 2^{-n-1},$$

$$\Delta_{\text{отн макс}} = \frac{|A^* - A|_{\max}}{|A^*|} = \frac{P^{\alpha} 2^{-n-1}}{P^{\alpha} a} = \frac{2^{-n-1}}{a}$$

(погрешность, максимальная при данном a)

$$\Delta_{\text{отн макс макс}} = \frac{|A^* - A|_{\max}}{|A^*|_{\min}} = \frac{P^{\alpha} 2^{-n-1}}{P^{\alpha} a_{\min}} = P 2^{-n-1},$$

$$\Delta_{\text{отн ср}} = \frac{|A^* - A|_{\text{ср}}}{|A^*|} = \frac{P^{\alpha} 2^{-n-2}}{P^{\alpha} a} = \frac{2^{-n-2}}{a}.$$

Закон распределения мантиссы a в диапазоне (7.4.5) не является равномерным и среднее значение мантиссы $a_{\text{ср}}$ не лежит в середине этого диапазона. По данным работ [1, 2],

$$\Delta_{\text{отн ср ср}} = \frac{|A^* - A|_{\text{ср}}}{|A^*|_{\text{ср}}} = \frac{P^{\alpha} 2^{-n-2}}{P^{\alpha} a_{\text{ср}}} = \frac{2^{-n-2}}{a_{\text{ср}}} = \frac{(P-1)\, 2^{-n}}{4 \ln P}.$$

Именно эта формула делает целесообразным использование некоторых форм представления, невыгодных на первый взгляд. Сравним, например, два способа представления чисел (табл. 7.4.2). В двух сравниваемых случаях ЦВМ имеет одну и ту же разрядность, так как сумма $m+n$ в обоих случаях одна и та же. Близки

Таблица 7.4.2

P	$m+1$	n	$A_{\text{макс}} = P^{2^m-1}$	$\Delta_{\text{отн макс макс}} = P \cdot 2^{-n-1}$	$\Delta_{\text{отн ср ср}} = \frac{(P-1)\, 2^{-n}}{4 \ln P}$
2	8	38	2^{127}	$0{,}5 \cdot 2^{-37}$	$0{,}18 \cdot 2^{-37}$
16	6	40	2^{124}	2^{-37}	$0{,}17 \cdot 2^{-37}$

между собой и диапазоны представимых чисел (так как величины 2^{127} и 2^{124} отличаются друг от друга не слишком сильно). Максимальная относительная погрешность в первой из этих двух систем, как это видно из таблицы, в два раза меньше, чем во второй системе. (Понять это нетрудно: во второй системе мантисса содержит на два двоичных разряда больше, чем в первой, но из 40 разрядов мантиссы при $1/16 \leqslant a < 2/16$ три старших разряда оказываются нулями и тогда остается фактически 40—3=37 значащих разрядов, что на один разряд меньше, чем в первой системе). Поэтому, на первый взгляд, вторая система хуже. Но по усредненной относительной

погрешности вторая система не уступает первой. Указанное обстоятельство было, по-видимому, одним из решающих при выборе способа представления чисел в сериях машин ЕС ЭВМ, ИБМ 360, ИБМ 370, в которых, как известно, $P=16$ *).

7.5. ЭЛЕМЕНТЫ И УЗЛЫ ЦВМ

Не только в этой главе, но и во всей книге основное внимание уделяется логическим и математическим идеям и принципам, лежащим в основе вычислительных устройств, машин и систем. Техническими деталями мы занимаемся лишь в мере, необходимой для понимания путей реализации указанных идей. Поэтому основные типы элементов и узлов рассмотрим кратко, выделяя их назначение и логические возможности.

Рис. 7.5.1

В настоящее время наибольшее распространение получили потенциальные системы элементов. Кроме того, применяются импульсные, импульсно-потенциальные и другие системы.

В потенциальной системе элементов информация представляется электрическими потенциалами, которые могут принимать только два фиксированных значения (говоря так, мы пренебрегаем процессом перехода с одного уровня на другой; во время этого процесса потенциал сравнительно быстро изменяется от одного фиксированного значения до другого). Будем считать, что более высокий из двух уровней изображает (представляет) цифру 1, более низкий — цифру 0 (существуют ЦВМ, в которых принята обратная договоренность). Основная особенность потенциальных систем состоит в том, что электрический сигнал может принимать любое из этих двух значений в течение интервала времени произвольной длительности (рис. 7.5.1,*а*).

В импульсных системах элементов электрические сигналы имеют форму стандартных импульсов и поэтому в системе с положительными (отрицательными) импульсами потенциал может быть высоким (низким) только в течение определенного небольшого интервала времени, определяемого стандартной формой импульса (рис. 7.5.1,*б*).

В импульсно-потенциальной системе выходные сигналы одних элементов имеют импульсный характер, а выходные сигналы элементов других типов — потенциальный характер.

В данном параграфе описываются только потенциальные элементы и построенные из них узлы. Следует заметить, что во всех

*) Одним из основных достоинств 16-теричной системы является также относительная простота реализации процедур сдвига чисел (например, при нормализации, денормализации и др.), так как сдвиги при этом производятся только на количество разрядов (двоичных), кратное четырем.

случаях рассмотрения конкретных узлов и устройств в данной книге предполагалось использование потенциальных элементов (это не означает, что принципы построения того или иного устройства не могут быть реализованы на других элементах).

Одним из основных параметров потенциальной системы элементов является набор логических функций, реализуемых этими элементами. Наиболее распространенными являются функции И, ИЛИ, И—ИЛИ, И—НЕ, ИЛИ—НЕ, И—ИЛИ—НЕ, ИЛИ—И—НЕ. На рис. 7.5.2 показаны условные изображения элементов, реализующих указанные функции.

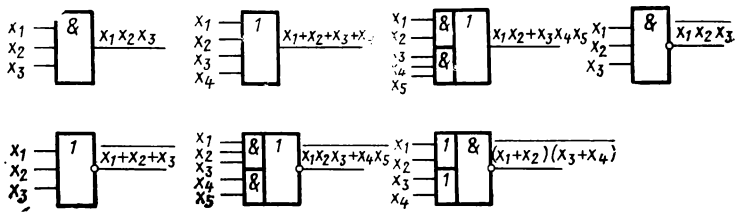


Рис. 7.5.2

Среди других параметров, которые важны при синтезе различных функциональных схем, следует отметить число входов схемы И, число входов схемы ИЛИ, коэффициент разветвления по выходу (нагрузочную способность), объем оборудования каждого логического элемента, динамические параметры элементов. В определенных случаях большое значение приобретают конструктивные параметры, устойчивость к механическим, климатическим воздействиям, потребляемая мощность и др.

Микросхема, размещенная в одном корпусе, обычно содержит некоторое количество элементов, реализующих указанные выше элементарные логические функции. Характер соединений между элементами внутри корпуса определяет функции, выполняемые всей микросхемой. В одном корпусе может, например, размещаться 4-разрядный ускоренный сумматор (см. § 6.2) или устройство, перемножающее 4-разрядные сомножители (п. 6.3.3) и др.

Основными типами функциональных узлов, построенных на элементах, реализующих указанные выше элементарные логические функции, можно считать триггеры, сдвигающие регистры, коммутаторы, сумматоры.

Триггер — это логическая схема с положительной обратной связью, имеющая два устойчивых состояния, которые принято называть единичным и нулевым и обозначать 1 и 0. Таким образом, триггер является запоминающим устройством с объемом памяти в один бит. Существует множество построений триггерной схемы (см., например, [3]). Один из простейших триггеров и его условное обозначение показаны на рис. 7.5.3. Если $D=1$, то поступлении на вход C положительного импульса (*строба приема*) триггер устанавливается в единичное состояние ($Q=1$, $\bar{Q}=0$); если $D=0$, то этот же импульс устанавливает триггер в нулевое состояние ($Q=0$, $\bar{Q}=1$). Принятое состояние сохраняется до тех пор, пока $C=0$, т. е. поведение информационного сигнала D при $C=0$ на состояние триггера не влияет

Группа триггеров с общим стробом приема называется *триггерным регистром*. Такой узел способен хранить двоичное число, разрядность которого равна количеству триггеров в регистре.

Триггерный регистр, к которому добавлена некоторая аппаратура, позволяющая сдвигать вдоль регистра хранящуюся в нем информацию, будем называть *сдвигающим регистром*. Одно из возможных построений сдвигающего регистра показано на рис. 7.5.4. Узел состоит из двух триггерных регистров (Рг1 и Рг2), один из которых (например, Рг1) можно считать основным, а другой — вспомогательным. Из рисунка видно, что если передать информацию из Рг1 в Рг2 при помощи строба *С2*, а затем передать ее обратно в Рг1 при помощи С1, то информация в Рг1 сдвинется на один разряд. В зависимости от наличия или отсутствия связи между выходом нижнего триггера регистра

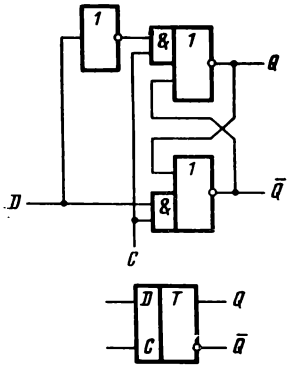


Рис. 7.5.3

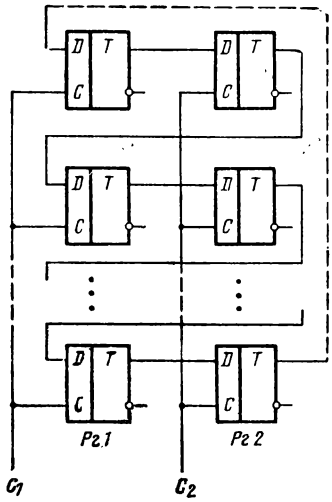


Рис. 7.5.4

Рг2 и входом верхнего триггера Рг1 (показана штриховой линией на рис. 7.5.4) сдвигающий регистр называется *кольцевым* или *разомкнутым*. В кольцевой схеме информация сдвигается по кольцу.

Из рис. 7.5.4 видно, что Рг2 служит для кратковременного хранения информации. Поэтому этот регистр может быть заменен линиями задержки или еще какими-либо элементами, задерживающими сигналы. В некоторых случаях для этой цели могут быть использованы паразитные задержки основного регистра. Можно утверждать, что принцип передачи информации на короткое время в некоторую схему используется в той или иной форме в любом сдвигающем регистре.

Коммутатор — это узел, служащий для коммутации информационных сигналов, принимаемых одним приемником с нескольких возможных направлений. Направление коммутации определяется управляющими сигналами $У_i$. При передаче информации параллельным кодом, т. е. при необходимости коммутировать сразу n разрядов,

коммутатор строится из n одинаковых одноразрядных схем. На рис. 7.5.5 показан один разряд такого коммутатора, построенный на элементе И—ИЛИ—НЕ. Принцип работы коммутатора прост. Если один из сигналов $У_i$ равен единице, а остальные — нулю, то информационный сигнал, соответствующий высокому управляющему сигналу, «проходит» сквозь коммутатор. Из рисунка видно, что в данном случае информационные входные сигналы инверсны выходному сигналу: если входные сигналы поданы в прямом (обратном) коде, то выходной сигнал получается обратный (прямой).

Одна из немногих тонкостей использования коммутаторов связана как раз с элементами И—ИЛИ—НЕ. В некоторых устройствах возможен режим, при котором $У_1=У_2=\ldots=У_k=0$. Если требуется, чтобы в этом режиме число на выходах коммутатора равнялось нулю, и если выходные сигналы коммутатора представлены в прямом коде (входные — в обратном), то, как видно из рисунка, в этом режиме на выходах появятся высокие уровни, которые будут расцениваться как единицы. В таком коммутаторе нужно в каждом разряде на одну из входных схем И (одновходовую) завести дополнительный управляющий сигнал $У=\overline{У_1}\,\overline{У_2}\ldots\overline{У_k}$, который и обеспечит нули на выходах в указанном режиме.

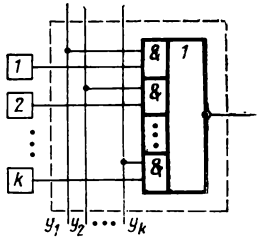


Рис. 7.5.5

Различные типы параллельного *сумматора* — устройства для сложения двух n-разрядных двоичных чисел — подробно рассматриваются в третьей главе. Здесь мы только заметим, что сумматоры делятся на комбинационные и накапливающие. *Комбинационный сумматор* не содержит запоминающих элементов и поэтому сумма слагаемых формируется в нем и сохраняется на его выходах только в течение того времени, пока слагаемые находятся на входах сумматора. Внутри *накапливающего сумматора* имеется триггерный регистр, в котором перед началом сложения содержится одно из двух слагаемых, а после окончания сложения — полученная сумма. Второе слагаемое подается на отдельные входы сумматора. Производя сложение числа, находящегося в регистре сумматора, с несколькими числами, поочередно поступающими на указанные входы накапливающего сумматора, можно «накопить» в нем сумму всех этих чисел.

Подробнее с различными функциональными узлами на потенциальных элементах можно познакомиться, например, по книге [3].

7.6. ОСНОВНЫЕ МЕТОДЫ НЕУСКОРЕННОГО СЛОЖЕНИЯ-ВЫЧИТАНИЯ

Существует множество деталей процесса сложения-вычитания двух двоичных чисел с фиксированной запятой. Поскольку некоторую комбинацию этих деталей можно назвать способом или методом сложения-вычитания, то количество методов становится неопределенным. Тем не менее представляется целесообразной классификация методов сложения-вычитания, приведенная на рис. 7.6.1. На рисунке цифрами выделены пять методов, которые предлагается счи-

тать основными. Сама классификация выполнена по трем признакам. Во-первых, разделение произведено по форме представления операндов и результата операции — в прямом или в дополнительном коде. Вторым классификационным признаком выбран вопрос о том, какой или какие операнды могут быть преобразованы перед поступлением в сумматор (имеется в виду преобразование формы представления числа). Будем считать, что устройство должно уметь выполнять три операции: $A+C=B$, $A-C=B$, $C-A=B$. По второму классификационному признаку методы делятся на две группы. В одних случаях (методы 4, 5) преобразовываться может только один операнд (назовем его операндом C), в других случаях (методы 2, 3)

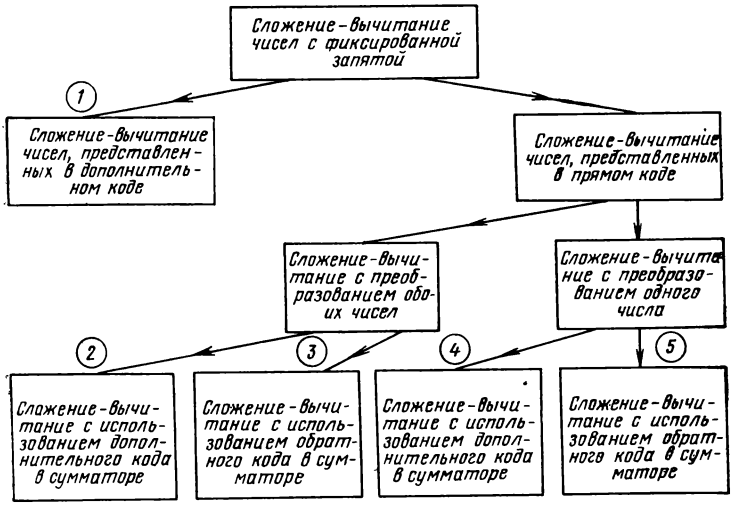


Рис. 7.6.1

преобразовываться могут оба операнда. Третьим классификационным признаком является тип кода, применяемого в сумматоре при вычитании, — дополнительный или обратный.

Рассматриваемые методы могут использоваться в трех модификациях. Назовем модификацией «а» вариант устройства, в котором в сумматоре имеются два знаковых разряда и используется модифицированный код (дополнительный или обратный), модификацией «б» — вариант с одним знаковым разрядом в сумматоре и модификацией «в» — вариант, в котором сумматор вообще не содержит знакового разряда (при этом о знаке числа, получающегося в сумматоре, судят по сигналу переноса из старшего разряда сумматора).

1°. Рассмотрим метод 1. Он используется обычно в ЦВМ, в которых числа на входах и выходе арифметического устройства представлены в дополнительном коде. В сумматоре при этом методе используется дополнительный код.

Операции $A+C$, $A-C$, $C-A$ выполняются по методу 1 в два этапа: вначале в нужных случаях преобразуется один из операндов, затем выполняется сложение на сумматоре.

Преобразование операнда A производится при операции $C-A$, преобразование C — при операции $A-C$. Само преобразование состоит во взятии дополнения.

Сложение при использовании этого и всех других методов выполняется на сумматоре, содержащем, как уже отмечалось, $n+2$ (модификация «а»), $n+1$ (модификация «б») или n (модификация «в») разрядов. Полученная в сумматоре после сложения по методу 1 сумма представляет собой готовый результат операции, что является характерным признаком метода 1.

Правила обнаружения переполнения, т. е. правила выработки сигнала ω в трех модификациях метода, различны.

Приведем примеры выполнения сложения-вычитания по методу 1а. В этом случае переполнение обнаруживается по несовпадению цифр s_1, s_2 на выходах двух знаковых разрядов сумматора, т. е.

$$\omega = \bar{s}_1 s_2 + s_1 \bar{s}_2. \qquad (7.6.1)$$

Во всех примерах, которые будут рассмотрены в этом параграфе, сперва будут написаны числа A и C в виде правильных десятичных дробей, рядом (в скобках) будут показаны соответствующие двоичные числа A и C в том виде, в котором они поступают в арифметическое устройство. Далее будут написаны слагаемые, поступающие на входы сумматора, и полученная сумма. Сбоку в скобках будет приводиться в виде неравенства соотношение между операндами, выполнение которого в данном примере определяет знак результата операции и наличие или отсутствие переполнения.

В примерах, относящихся к методу 1а, дополнение берется путем инвертирования всего числа и прибавления единицы в младший разряд (такой способ в ряде случаев проще для реализации). Крайний левый (знаковый) разряд числа, поступающего в сумматор, перед сложением раздваивается (для получения модифицированного дополнительного кода).

Примеры. Метод 1а. Операция $A-C$, $n=4$.

1) $A = +\frac{5}{16}$ (0,0101) 00,0101
$C = +\frac{3}{16}$ (0,0011) + 11,1100
1
$B = +\frac{1}{8}$ $(A>C)$ 00,0010

2) $A = +\frac{3}{16}$ (0,0011) 00,0011
$C = +\frac{5}{16}$ (0,0101) + 11,1010
1
$B = -\frac{1}{8}$ $(A<C)$ 11,1110

3) $A = +\frac{7}{16}$ (0,0111) 00,0111
$C = -\frac{3}{8}$ (1,1010) + 00,0101
1
$B = +\frac{13}{16}$ $(A-C<1)$ 00,1101

4) $A = +\frac{10}{16}$ (0,1010) 00,1010
$C = -\frac{7}{16}$ (1,1001) + 00,0110
1
$\omega = 1$ $(A-C>1)$ 01,0001
s_1 s_2

5) $A = -\frac{11}{16}$ (1,0101) 11,0101
$C = +\frac{5}{16}$ (0,0101) 11,1010
1
$B = -1$ $(A - C > -1)$ 11,0000

6) $A = -\frac{11}{16}$ (1,0101) 11,0101
$C = +\frac{3}{8}$ (0,0110) 11,1001
1
$\omega = 1$ $(A - C < -1)$ 10,1111

7) $A = -\frac{15}{16}$ (1,0001) 11,0001
$C = -\frac{9}{16}$ (1,0111) 00,1000
1
$B = -\frac{3}{8}$ $(|A| > |C|)$ 11,1010

8) $A = -\frac{9}{16}$ (1,0111) 11,0111
$C = -\frac{9}{16}$ (1,0111) 00,1000
1
$B = 0$ $(|A| \leqslant |C|)$ 00,0000

Пользуясь приведенными примерами, читатель сумеет разобрать выполнение операций $A+C$, $A-C$, $C-A$ по методу 1а при любом n. Методы 1б, 1в мы рассматривать не будем. Модификации «б» и «в» будут описаны применительно к другим методам. Отличия между модификациями «а», «б», «в» не велики. В основном они касаются способа обнаружения переполнения.

2°. Перейдем к методу 2. Как видно из рис. 7.6.1, теперь мы будем иметь дело с операндами и результатами операций, представленными в прямых кодах; преобразовываться могут оба операнда, в сумматоре используется дополнительный код. Операции выполняются в три этапа: вначале в нужных случаях преобразуются один или оба операнда из прямого кода в дополнительный, затем выполняется сложение на сумматоре и, наконец, в нужном случае (при отрицательном результате операции) результат преобразуется из дополнительного кода в прямой.

Операнд A преобразуется при операциях $A+C$, $A-C$, если число A отрицательно, и при операции $C-A$, если A положительно. Аналогично C преобразуется при операциях $A+C$, $C-A$, если C отрицательно, и при операции $A-C$, если C положительно. Иными словами, A преобразуется, если в сложении на сумматоре участвует число $-|A|$, а C преобразуется, если в сложении участвует число $-|C|$; отрицательные числа $-|A|$, $-|C|$ поступают в сумматор в дополнительном коде.

Рассмотрим примеры сложения-вычитания по методу 2б. В этом случае переполнение можно обнаруживать при помощи схемы, вырабатывающей логическую функцию

$$\omega = \overline{A}_{\text{п}}\overline{C}_{\text{п}}s + A_{\text{п}}C_{\text{п}}\overline{s} + A_{\text{п}}C_{\text{п}}z, \qquad (7.6.2)$$

где s — сигнал на выходе знакового разряда сумматора, $A_{\text{п}}$ ($C_{\text{п}}$) — сигнал, равный единице в том случае, когда операнд $A(C)$ преобразовывается в дополнительный код, z — сигнал, равный единице в том случае, когда после сложения в сумматоре все n значащих разрядов на выходах сумматора оказываются равными нулю.

Примеры. Метод 2б. Операция $C-A$, $n=3$. Взятие дополнения после суммирования указывается в отдельной строке.

1) $A = +\frac{5}{8}$ (0,101) I,011
$C = +\frac{7}{8}$ (0,111) 0,111
$B = +\frac{1}{4}$ (A<C) 0,010

2) $A = +\frac{7}{8}$ (0,111) I,001
$C = +\frac{5}{8}$ (0,101) 0,101
I,110
$B = -\frac{1}{4}$ (A>C) I,010

3) $A = +\frac{1}{8}$ (0,001) I,111
$C = -\frac{1}{4}$ (I,010) I,110
I,101
$B = -\frac{3}{8}$ (C-A>-I) I,011

4) $A = +\frac{3}{4}$ (0,110) I,010
$C = -\frac{1}{4}$ (I,010) I,110
ω = I (C-A ≤ -I) I,000

5) $A = -\frac{1}{2}$ (I,100) 0,100
$C = +\frac{3}{8}$ (0,011) 0,011
$B = +\frac{7}{8}$ (C-A<I) 0,111

6) $A = -\frac{1}{2}$ (I,100) 0,100
$C = +\frac{5}{8}$ (0,101) 0,101
ω = I (C-A ≥ I) I,001

7) $A = -\frac{5}{8}$ (I,101) 0,101
$C = -\frac{5}{8}$ (I,101) I,011
$B = 0(|A| \geqslant |C|)$ 0,000

8) $A = -\frac{5}{8}$ (I,101) 0,101
$C = -\frac{3}{4}$ (I,110) I,010
I,111
$B = -\frac{1}{8}$ (|A|<|C|) I,001

Из формулы (7.6.2) и из примеров видно, что при использовании метода 2б несколько усложняется обнаружение переполнения. По сравнению с формулой (7.6.1) усложнение двух первых слагаемых в (7.6.2) объясняется переходом от модификации «а» к модификации «б»; введение третьего слагаемого ($A_{\text{п}}C_{\text{п}}z$) вызвано возможностью получения результата операции, равного —1. Такой результат является допустимым для дополнительного кода (который используется в сумматоре), но недопустим для прямого кода (в котором представляется результат операции). Можно отметить, что первое и третье слагаемое формулы (7.6.2) «сработали» соответственно в примерах 6 и 4. Второе слагаемое «срабатывает» при результате операции, меньшем чем —1.

Методы 2а, 2в мы рассматривать не будем.

3°. Третий метод отличается от второго тем, что в сумматоре используется обратный код и имеется цепь кольцевого переноса, по которой перенос из старшего разряда сумматора поступает в младший разряд. Преобразуются операнды в тех же случаях, что и при методе 2, однако не из прямого кода в дополнительный, а из прямого — в обратный. Если результат операции отрицателен, то он по-

лучается на выходе сумматора не в дополнительном коде, как при методе 2, а в обратном, и заключительное преобразование переводит это отрицательное число из обратного кода в прямой.

Рассмотрим несколько примеров для метода 3б. В этом случае переполнение может быть обнаружено по правилу

$$\omega = \bar{A}_{\text{п}}\bar{C}_{\text{п}}s + A_{\text{п}}C_{\text{п}}\bar{s}.$$

Обозначения здесь те же, что и в (7.6.2).

Примеры. Метод 3б. Операция $A-C$, $n=4$. (Операнды в примерах 1—3,5, 8 те же, что в примерах для метода 1а.)

1) $A = +\frac{5}{16}$ (0,0101) 0,0101

$C = +\frac{3}{16}$ (0,0011) I,1100

кольцевой перенос → 0,0001 → I

$B = +\frac{1}{8}$ ($A > C$) 0,0010

2) $A = +\frac{3}{16}$ (0,0011) 0,0011

$C = +\frac{5}{16}$ (0,0101) I,1010

I,1101

$B = -\frac{1}{8}$ ($A \leqslant C$) I,0010

3) $A = +\frac{7}{16}$ (0,0111) 0,0111

$C = -\frac{3}{8}$ (I,0110) 0,0110

$B = +\frac{13}{16}$ (A-C<I) 0,1101

4) $A = +\frac{9}{16}$ (0,1001) 0,1001

$C = -\frac{7}{16}$ (I,0111) 0,0111

ω = I (A - C > I) I,0000

5) $A = -\frac{11}{16}$ (I,1011) I,0100

$C = +\frac{5}{16}$ (0,0101) I,1010

ω = I(A - C < -I) 0,1110

6) $A = -\frac{11}{16}$ (I,1011) I,0100

$C = +\frac{1}{4}$ (0,0100) I,1011

0,1111 → I

$B = -\frac{15}{16}$ (A-C>-I) I,0000

I,1111

7) $A = -\frac{9}{16}$ (I,1001) I,0110

$C = -\frac{15}{16}$ (I,1111) 0,1111

0,0101 → I

$B = +\frac{3}{8}$ (|A| < |C|) 0,0110

8) $A = -\frac{9}{16}$ (I,1001) I,0110

$C = -\frac{9}{16}$ (I,1001) 0,1001

I,1111

$B = -0$ (|A| ≥ |C|) I,0000

В последнем примере был показан случай получения отрицательного нуля в результате операции.

4°. Особенностью метода (4) (как и метода 5) является то, что один из операндов (будем считать, что это операнд A) поступает на входы сумматора без преобразования, независимо от типа операции и знаков слагаемых. Второй операнд (C) при необходимости преобразуется. Методы 4, 5 могут оказаться, в частности, выгодными в таких арифметических устройствах, где в целях повышения быстродействия каждый операнд преобразуется отдельным узлом; поэтому при использовании, например, метода 2 и 3 нужны два таких узла, в то время как для метода 4 или 5 требуется только один узел преобразования C.

Сложение-вычитание по методу 4 выполняется в три этапа: вначале в нужных случаях производится преобразование операнда C из прямого кода в дополнительный, затем выполняется сложение на сумматоре и, наконец, в нужном случае производится преобразование результата из дополнительного кода в прямой.

Преобразование операнда C производится в тех случаях, когда нужно выполнить операцию $A+C$ над числами, имеющими разные знаки, или выполнить операцию $A-C$ или $C-A$ над числами с одинаковыми знаками. Будем считать, что в этих случаях в арифметическом устройстве вырабатывается управляющий сигнал «Выч». В остальных случаях, т. е. при операции $A+C$ над числами с одинаковыми знаками или операции $A-C$, $C-A$ над числами с разными знаками, вырабатывается сигнал «Слож». Очевидно, что Слож.$=\overline{\text{Выч}}$.

Рассмотрим примеры сложения-вычитания по методу 4в. При этом переполнение может быть обнаружено схемой, реализующей формулу

$$\omega = \text{Слож} \cdot e, \tag{7.6.3}$$

где e — перенос из старшего разряда сумматора.

Дополнительный код $-|C|$ будем формировать путем инвертирования числа $|C|$ и прибавления единицы в n-й разряд; разряд знака числа, полученного при этом преобразовании, будем, как это принято для модификаций «в», отбрасывать. Преобразование суммы, полученной в сумматоре, будем производить при Выч$\cdot\bar{e}=1$. Напомним, что знак числа, получающегося в сумматоре при модификации «в», определяется по сигналу e.

Примеры. Метод 4в. Операция $A+C$, $n=3$.

1) $A = +\frac{3}{8}(0{,}011)$,011
$C = +\frac{1}{4}(0{,}010)$,010
Слож $= 1$, $e = 0$,101

$B = +\frac{5}{8}$ $(A + C < 1)$

2) $A = +\frac{3}{8}(0{,}011)$,011
$C = +\frac{7}{8}(0{,}111)$ +,111
$e = 1$, $\omega = 1$,010
$(A + C \geqslant 1)$

3) $A = -\frac{1}{8}(1{,}001)$,001

$C = +\frac{3}{8}(0{,}011)$ + ,100

I

Выч = I, e = 0 ,110

$B = +\frac{1}{4}(|A| < |C|)$,010

4) $A = -\frac{3}{8}(1{,}011)$,011

$C = +\frac{1}{8}(0{,}001)$ + ,110

I

e = I, $B = -\frac{1}{4}$,010

$(|A| \geqslant |C|)$

5) $A = +\frac{1}{8}(0{,}001)$,001

$C = -\frac{3}{8}(1{,}011)$ + ,100

I

Выч = I, e = 0 ,110

$B = -\frac{1}{4}\ (|A| < |C|)$, 010

6) $A = +\frac{5}{8}(0{,}101)$,101

$C = -\frac{5}{8}(1{,}101)$ + ,010

I

e = I, B = 0 ,000

$(|A| \geqslant |C|)$

7) $A = -\frac{5}{8}(1{,}101)$,101

$C = -\frac{1}{8}(1{,}001)$ + ,001

Слож = I, e = 0 ,110

$B = -\frac{3}{4}(A + C > -1)$

8) $A = -\frac{5}{8}(1{,}101)$,101

$C = -\frac{1}{2}(1{,}100)$ + ,100

e = I, ω = I ,001

$(A + C \leqslant -1)$

Одно из отличий метода 4 (5) от методов 2, 3 состоит в том, что число, получающееся после сложения в сумматоре, преобразуется не только при отрицательном результате всей операции, но иногда и при положительном (см. пример 3).

Мы еще не описали формально правило определения знака результата операции при использовании метода 4в. Это правило для всех модификаций методов 4, 5 имеет более сложный характер, чем для методов 1—3. В частности, для метода 4в знак результата операции представляет собой следующую логическую функцию:

$$\text{Зн}\,B = \text{Слож}\,\{[(A+C)\vee(A-C)](-A)+(C-A)(+A)\}+$$

$$+\text{Выч}\,\{[(A+C)\vee(A-C)][(+A)\bar{e}+(-A)e]+$$

$$+(C-A)[(+A)e+(-A)\bar{e}]\}.$$

Здесь использованы упрощенные условные обозначения. Формула утверждает, что знак результата операции отрицателен, если Слож = =1 и при этом выполняется операция $A+C$ или $A-C$, а число A отрицательно, и т. д.

5°. Пятый метод отличается от четвертого тем, что в сумматоре используется обратный код и имеется цепь кольцевого переноса. Операнд C преобразовывается в тех же случаях, что и при методе 4, но не из прямого в дополнительный, а из прямого в обратный код. Результат преобразуется в тех же случаях (для модификации «в» — при Выч·$\bar{e}$=1); это преобразование переводит результат из обратного кода в прямой.

Рассмотрим примеры сложения-вычитания по методу 5в. Переполнение обнаруживается по правилу (7.6.3).

Примеры. Метод 5в. Операция $A+C$, $n=3$ (операнды те же, что ь примерах для метода 4в).

1) $A = +\frac{3}{8}$ (0,011) ,011
$C = +\frac{1}{4}$ (0,010) ,010
Слож = 1, e = 0 ,101
$B = +\frac{5}{8}$ ($A + C \leqslant 1$)

2) $A = +\frac{3}{8}$ (0,011) ,011
$C = +\frac{7}{8}$ (0,111) ,111
e = 1, ω =1 ,010
($A+C > 1$)

3) $A = -\frac{1}{8}$ (1,001) ,001
$C = +\frac{3}{8}$ (0,011) ,100
Выч = 1, e =0 ,101
$B = +\frac{1}{4}$ ($|A| \leqslant |C|$) ,010

4) $A = -\frac{3}{8}$ (1,011) ,011
$C = +\frac{1}{8}$(0,001) ,110
,001
e = 1, $B = -\frac{1}{4}$ → 1
,010
($|A| > |C|$)

5) $A = +\frac{1}{8}$ (0,001) ,001
$C = -\frac{3}{8}$ (1,011) ,100
,101
Выч= 1, e =0 ,010
$B = -\frac{1}{4}$ ($|A| \leqslant |C|$)

6) $A = +\frac{5}{8}$ (0,101) ,101
$C = -\frac{5}{8}$(1,101) ,010
,111
e =0, B = - 0 ,000
($|A| \leqslant |C|$)

7) $A = -\frac{5}{8}$ (1,101) ,101
$C = -\frac{1}{8}$ (1,001) ,001
Слож = 1, e = 0 ,110
$B = -\frac{3}{4}$($A + C > -1$)

8) $A = -\frac{5}{8}$(1,101) ,101
$C = -\frac{1}{2}$ (1,100) ,100
e = 1, ω = 1 ,001
($A + C \leqslant -1$)

6°. Рассмотрим кратко процедуру сложения-вычитания чисел A и C, представленных в форме с плавающей запятой, т. е. в форме

$$A = \pm 2^{\alpha} a, \quad C = \pm 2^{\gamma} c, \qquad (7.6.4)$$

где $p_{\min} \leqslant \alpha$, $\gamma \leqslant p_{\max}$ (обычно $p_{\max} = 2^m - 1$, $p_{\min} = -2^m$)

$$\frac{1}{2} \leqslant a, \quad c \leqslant 1 - 2^{-n}$$

(если не считать особых случаев $a=0$, $c=0$). Знаки $\pm$ в (7.6.4) означают, что число может иметь знак плюс или минус.

Сложение-вычитание чисел A и C производится в три этапа. На первом этапе, называемом обычно «выравнивание порядков», выбирается большее из чисел α и γ, вычисляется разность $|\alpha-\gamma|$, после чего мантисса числа, имеющего меньший порядок, сдвигается вправо относительно второй мантиссы на $|\alpha-\gamma|$ разрядов. На втором этапе производится сложение-вычитание сдвинутой и несдвинутой мантиссы. При этом могут использоваться, вообще говоря, различные методы сложения-вычитания, например уже знакомые нам методы 2—5 сложения чисел с фиксированной запятой (мы считаем, что мантиссы операндов и результата представляются в прямом коде). Абсолютная величина $b_{\text{нн}}$ полученной суммы (разности) мантисс является мантиссой ненормализованного результата операции $\pm 2^{\max(\alpha, \gamma)} b_{\text{нн}}$, где $\max(\alpha, \gamma)$ — максимальный из порядков α, γ. На третьем этапе («нормализация результата») производится сдвиг влево или вправо мантиссы $b_{\text{нн}}$ таким образом, чтобы полученная после сдвига окончательная мантисса b удовлетворяла неравенству

$$1/2 \leqslant b \leqslant 1 - 2^{-n},$$

и производится соответствующая коррекция порядка $\max(\alpha, \gamma)$, после которой получается окончательный порядок β.

Рассмотрим два примера операции $A+C$.

Пример 1. $A = 2^5 \cdot \frac{31}{32}$, $C = 2^2 \cdot \frac{1}{2}$.

Первый этап. Выбираем $\max(\alpha, \gamma)=5$ и вычисляем $|\alpha-\gamma|=5-2=3$. После сдвига мантиссы c на три разряда вправо взаимное расположение мантисс оказывается следующим:

$$\begin{array}{r} a \quad ,11111 \\ + \\ c \cdot 2^{-|\alpha-\gamma|} \quad ,0001 \\ \hline b_{\text{нн}} = 1,00001 \end{array}$$

Второй этап.

Третий этап. Так как ненормализованный результат равен

$$2^{\max(\alpha, \gamma)} b_{\text{нн}} = 2^5 \cdot 1\frac{1}{32},$$

то следует сдвинуть $b_{нп}$ на один разряд вправо, а к порядку ненормализованного результата прибавить единицу, после чего $b=b_{нн}\cdot 2^{-1}=0{,}100001$ (в двоичной форме), $\beta=\max(\alpha,\ \gamma)+1=5+1=6$. Итак, $A+C=B=2^{\beta}b=2^{6}\cdot\frac{33}{64}$. В этом примере оказалось, что $b_{нн}>1$. Так как каждая из мантисс a, c меньше единицы, то очевидно, что $b_{нн}$ всегда меньше, чем 2. Поэтому сдвиг вправо при нормализации результата может производиться только на один разряд.

Пример 2. $A=2^{-8}(1-2^{-9})$, $C=-2^{-7}\cdot\frac{1}{2}$.

Первый этап. $\max(\alpha,\ \gamma)=\gamma=-7$, $|\alpha-\gamma|=1$.

$$\begin{array}{ll} c & ,1 \\ \hline a\cdot 2^{-1} & ,0111111111 \end{array}$$

Второй этап. $b_{нн}=,0000000001$ (результат после вычитания).

Третий этап. Так как ненормализованный результат равен

$$2^{-7}\cdot\frac{1}{1024},$$

то следует сдвинуть $b_{нн}$ на 9 разрядов влево, а от порядка ненормализованного результата отнять 9, после чего $b=b_{нн}\cdot 2^{9}=0{,}1$ (в двоичной форме), $\beta=-7-9=-16$. Итак,

$$A+C=B=2^{\beta}b=2^{-16}\cdot\frac{1}{2}.$$

Из сказанного и из примеров следует, что действия, выполняемые при нормализации, можно описать следующим образом:

$$b=b_{нн}\cdot 2^{-1},\quad \beta=\max(\alpha,\ \gamma)+1 \text{ при } b_{нн}\geqslant 1,$$

$$b=b_{нн}\cdot 2^{N},\quad \beta=\max(\alpha,\ \gamma)-N \text{ при } b_{нн}<1,$$

где N — количество нулей между запятой и старшей единицей в числе $b_{нн}$. Видно, что сдвиг $b_{нн}$ и коррекция порядка $\max(\alpha,\ \gamma)$ делаются так, чтобы величина результата не изменилась, т. е.

$$2^{\beta}b=2^{\max(\alpha,\ \gamma)}b_{нн}.$$

Если $b_{нн}\geqslant 1$ и при этом $\max(\alpha,\ \gamma)=p_{max}$, то $\beta=p_{max}+1$ и, следовательно, результат операции по абсолютной величине превышает максимальное допустимое число $2^{p_{max}}(1-2^{-n})$. В этом случае в арифметическом устройстве вырабатывается сигнал переполнения разрядной сетки ($\omega=1$). Таким образом, правило обнаружения переполнения при сложении- вычитании с плавающей запятой совсем другое, чем при фиксированной запятой.

Если $b_{нн}=0$ или если $b_{нн}<1$, $\beta=\max(\alpha, \gamma)-N<p_{min}$, т. е. если результат операции меньше минимально допустимого нормализованного числа $2^{p_{min}}\cdot\frac{1}{2}$, то в качестве результата операции обычно используют число

$$+2^{p_{min}}\cdot 0.$$

(На первый взгляд кажется, что при нулевой мантиссе мог бы стоять любой порядок. Однако, если бы число, равное нулю, имело порядок, отличный от p_{min}, то при сложении такого числа с числом, имеющим меньший порядок и мантиссу, отличную от нуля, эта вторая мантисса при выравнивании порядков могла бы полностью или частично выдвинуться за пределы сдвигателя вправо, что привело бы к потере точности вычислений.)

При сложении- вычитании с плавающей запятой возникают различные аппаратные погрешности. Не рассматривая этих вопросов детально, укажем только причины возникновения погрешностей.

1) При сдвиге вправо во время выравнивания порядков мантисса числа, имеющего меньший порядок, может полностью или частично выдвинуться за пределы сдвигателя вправо. (Допустим, что этот сдвигатель, сумматор и схема нормализации содержат по $n+k$ разрядов.)

2) При сдвиге вправо во время нормализации может выдвинуться еще один разряд.

3) Во время выполнения операции отбрасываются полностью (или с тем или иным округлением) младшие k разрядов.

4) Обнуление результата операции при $\max(\alpha, \gamma)-N<p_{min}$ тоже вносит погрешность.

Максимальные значения погрешностей операции в каждом конкретном устройстве определить обычно нетрудно. Гораздо сложнее обстоит дело со средними значениями. Трудности фактически состоят в том, что до настоящего времени мало изучены статистические свойства и взаимосвязь величин $|\alpha-\gamma|$, N, a_i, c_i ($i=1, 2, \ldots, n$). (Буквами a_i, c_i здесь обозначены двоичные цифры в разрядах мантисс.) Вопросы точности сложения-вычитания с плавающей запятой рассматривались в работах [4, с. 326—331], [5] и др.

7.7. ОСНОВНЫЕ МЕТОДЫ НЕУСКОРЕННОГО УМНОЖЕНИЯ

1°. Перемножим двоичные числа 0,1101 и 0,1011:

```
      0,1101    множимое C
  ×   0,1011    множитель A
  ------------
      1101
      1101
  +   0000      } частичные произведения
    1101
  ------------
  ,10001111     произведение CA
```

В этом примере умножение начиналось с младших разрядов множителя. Каждое частичное произведение равняется либо нулю (если со-

ответствующая цифра множителя равна нулю), либо множимому, сдвинутому на соответствующее количество разрядов. Очевидно, что умножение можно начинать и со старших разрядов множителя (при этом частичные произведения меняются местами). Кроме того, ясно, что если частичные произведения складываются не все сразу, а сначала только два частичных произведения, затем к их сумме прибавляется третье и т. д., то не обязательно при формировании частичных произведений сдвигать множимое; вместо этого можно перед прибавлением множимого, умноженного на очередной разряд множителя, сдвигать на один разряд накопленную к этому моменту сумму частичных произведений. Таким образом, существуют четыре варианта умножения и им соответствуют показанные на рис. 7.7.1 четыре основные схемы неускоренного параллельного умножения [4]. Каждая из них состоит из трех основных узлов: регистра-сдвигателя множителя A, регистра (или регистра-сдвигателя) множимого C и накапливающего сумматора (или накапливающего сумматора-сдвигателя). Надписи на рисунке слева от узлов уточняют их состав и разрядность. Например, надпись $2n$-р. Рг-Сд, n-р. См (рис. 7.7.1,а) относится к $2n$-разрядному сдвигающему регистру (регистр-сдвигатель) и n-разрядному накапливающему сумматору, в котором роль внутреннего регистра выполняет половина указанного регистра-сдвигателя.

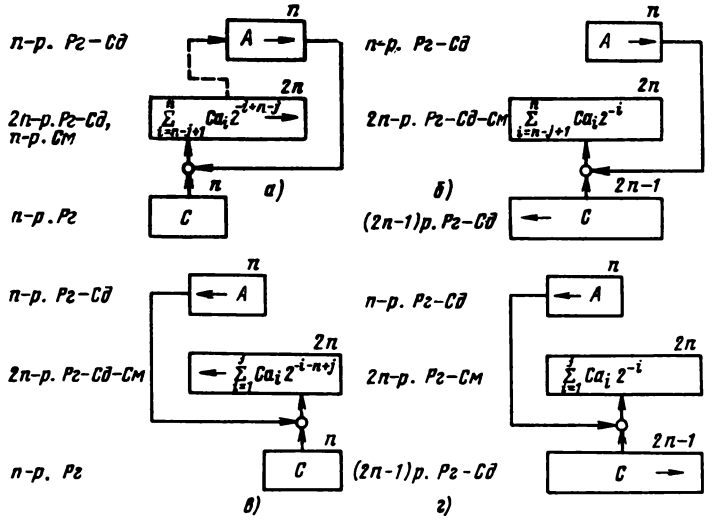


Рис. 7.7.1

В первом из четырех устройств (самом простом и самом распространенном) умножение начинается с младших разрядов множителя. Процесс умножения имеет циклический характер. Во время каждого j-го цикла ($j=1, 2, \ldots, n$) из регистра-сдвигателя выдвигается очередная цифра множителя a_{n-j+1}, которая управляет передачей через n двухвходовых схем И (обозначены кружком) в сумматор

либо числа 0 (если цифра множителя равна нулю), либо множимого (если цифра множителя равна единице), причем множимое поступает в 2 ... $(n+1)$ разряды сумматора. В конце цикла полученная сумма частичных произведений сдвигается на один разряд вправо (на последнем, n-м цикле этот сдвиг не нужен). Таким образом, во время первого цикла в сумматор передается число $Ca_1 2^{-1}$, в конце цикла оно сдвигается. В начале второго цикла прибавляется число $Ca_2 2^{-1}$, а в конце второго цикла полученная сумма $Ca_1 2^{-2} + Ca_2 2^{-1}$ сдвигается и т. д. На рис. 7.7.1 внутри контуров сумматоров написана текущая сумма частичных произведений, остающаяся в накапливающем сумматоре во время j-го цикла сразу после прибавления j-го частичного произведения (до сдвига).

Из рис. 7.7.1,*а* видно, что схемы суммирования находятся только в старшей половине узла, накапливающего частичные произведения (точнее — в $n+1$ старших разрядах узла). Младшая половина этого узла, как и узел множителя, представляет собой регистр-сдвигатель. Поэтому имеется возможность изъять из устройства младшую половину сумматора, а выдвигаемые из сумматора младшие разряды готового произведения передавать (штриховая линия) в освобождающиеся старшие разряды регистра-сдвигателя множителя.

Во втором устройстве (рис. 7.7.1,*б*) умножение тоже начинается с младших разрядов, но сдвигается не сумма частичных произведений, а множимое.

В третьем и четвертом устройствах умножение идет со старших разрядов и поэтому множитель двигается влево. Третье устройство можно упростить, изъяв из него регистр-сдвигатель множителя и поместив множитель в старшую половину регистра накапливающего сумматора. В процессе умножения множитель будет сдвигаться влево, освобождая место надвигающейся справа сумме частичных произведений. Следует еще заметить, что левая половина сумматора третьей схемы содержит не одноразрядные сумматоры, а одноразрядные полусумматоры, так как частичные произведения в старшую половину не поступают. В этом состоит преимущество третьей схемы перед второй и четвертой.

Затраты аппаратуры и другие свойства каждого из четырех вариантов множительного устройства можно оценить довольно точно прямо по виду рис. 7.7.1. Из менее очевидных особенностей следует отметить пригодность двух последних схем для выполнения деления: в регистр множимого можно поместить делитель, в накапливающий сумматор — делимое (там же будут получаться все остатки), формируемые цифры частного можно по одной вдвигать справа в регистр множителя.

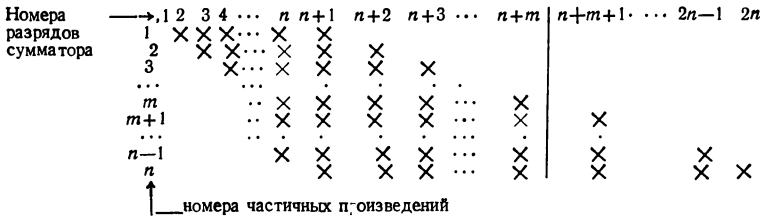
Показанные на рис. 7.7.1 устройства предназначены для получения полного, $2n$-разрядного произведения АС. Значительный интерес представляет анализ возможностей упрощения каждой из этих схем в том случае, когда устройство предназначено для формирования только n-разрядного произведения.

Очевидно, что в первом из четырех устройств в этом случае может быть изъята вся младшая половина регистра-сдвигателя суммы частичных произведений (если, конечно, эта половина не совмещена с регистром-сдвигателем множителя). При этом в старшей половине регистра-сдвигателя останутся старшие n разрядов точного произведения. (При желании можно оставить один дополнительный млад-

ший разряд и одновременно с выполнением последнего, n-го цикла произвести округление).

Очевидно также, что третья схема не поддается упрощению, так как изъятие из сумматора даже двух-трех младших разрядов приведет к аппаратной погрешности, значительно превышающей величину 2^{-n}.

Наиболее сложен анализ упрощений, возможных во 2- и 4-й схемах. Допустим, что в сумматоре имеется лишь $n+m$ старших разрядов ($0 \leqslant m \leqslant n$), а $n-m$ младших разрядов с целью экономии аппаратуры изъяты. Изобразим частичные произведения, поступающие в сумматор, в виде следующей таблицы:



Каждый крестик здесь обозначает цифру частичного произведения, т. е. либо нуль, либо соответствующую цифру множимого. Разница между 2- и 4-й схемами состоит фактически лишь в порядке, в котором частичные произведения поступают в сумматор. Допустим, что разряды частичных произведений, оказавшиеся правее младшего из оставшихся разрядов сумматора, т. е. правее вертикальной черты в таблице частичных произведений, отбрасываются. Тогда сумма чисел, стоящих справа от черты, является погрешностью полученного в сумматоре $(n+m)$-разрядного произведения. Нетрудно оценить ее максимальное значение, соответствующее случаю, когда все отброшенные цифры являются единицами. Однако максимальная погрешность умножения резко уменьшается, если производится округление путем прибавления единицы в $(n+m)$-й разряд сумматора каждый раз, когда старший из отбрасываемых разрядов очередного частичного произведения, т. е. $(n+m+1)$-й разряд равен единице. Точная оценка максимальной погрешности умножения для этого случая получена в работе [6]. Эта погрешность равна

$$\Delta_{\max} = 2^{-(n+m)} \left\{ \frac{n-m}{6} + \frac{7}{18} + \frac{1}{9} \left(-\frac{1}{2} \right)^{n-m} \right\}.$$

Следует учитывать, что указанная величина — это максимальная разность между точным произведением AC и $(n+m)$-разрядным произведением, сформированным описанным способом. Дополнительная погрешность, связанная с отбрасыванием младших m разрядов полученного $(n+m)$-разрядного произведения (при этом можно еще округлить остающийся n-разрядный результат) — это отдельный (более простой) вопрос.

2°. Рассмотрим процедуру умножения чисел A и C, представленных в форме с плавающей запятой, т. е. в форме

$$A = 2^{\alpha} a, \quad C = 2^{\gamma} c,$$

где $p_{\min} \leqslant \alpha, \gamma \leqslant p_{\max}$,

$$\frac{1}{2} \leqslant a, \; c \leqslant 1 - 2^{-n} \tag{7.7.1}$$

(пока не будем говорить об особых случаях $a=0$ или $c=0$). Умножение производится в два этапа. На первом этапе порядки складываются, а мантиссы перемножаются, т. е. вычисляется величина $AC = 2^{\alpha+\gamma}ac$. Из (7.7.1) следует, что

$$\frac{1}{4} \leqslant ac \leqslant (1-2^{-n})^2.$$

Поэтому при $\frac{1}{4} \leqslant ac < \frac{1}{2}$ выполняется второй этап (нормализация результата), во время которого ac сдвигается на один разряд влево, а от суммы порядков отнимается единица. Если обозначить результат операции через $B = 2^{\beta}b$, то можно записать, что

$$\left.\begin{array}{l} b = ac \\ \beta = \alpha + \gamma \end{array}\right\} \text{ при } ac \geqslant \frac{1}{2}, \quad \left.\begin{array}{l} b = 2ac \\ \beta = \alpha + \gamma - 1 \end{array}\right\} \text{ при } ac < \frac{1}{2}.$$

Очевидно, что b тоже находится в диапазоне (7.7.1).

Если

$$\left.\begin{array}{l} ac \geqslant \frac{1}{2} \\ \alpha + \gamma > p_{\max} \end{array}\right\} \text{ или если } \alpha+\gamma-1 > p_{\max},$$

т. е. если результат умножения превышает максимальное допустимое число $2^{p_{\max}}(1-2^{-n})$, то в арифметическом устройстве вырабатывается сигнал переполнения разрядной сетки. Иногда это правило заменяется более простым, но более грубым правилом: сигнал переполнения выдается при $\alpha+\gamma > p_{\max}$. Недостаток второго правила состоит в том, что ситуация

$$\left.\begin{array}{l} \alpha + \gamma = p_{\max} + 1 \\ ac < \frac{1}{2} \end{array}\right\}$$

считается недопустимой, в то время как при использовании первого правила вычисления могут продолжаться (в данном случае $\beta = p_{\max}$).

Если $ac=0$ (это бывает при равенстве одного или обоих сомножителей нулю), то в качестве результата операции обычно используется число

$$+2^{p_{\min}}.0;$$

то же самое, если

$$\left.\begin{array}{l} ac < \frac{1}{2} \\ \alpha + \gamma - 1 < p_{\min} \end{array}\right\} \text{ или если } \alpha + \gamma < p_{\min},$$

т. е. если результат умножения меньше минимально допустимого

нормализованного числа $2^{p_{\min}} \cdot \frac{1}{2}$. Иногда это правило заменяется более грубым:

$$\alpha+\gamma-1<p_{\min}.$$

Недостаток последнего правила состоит в том, что при

$$ac \geqslant \frac{1}{2}, \quad \alpha+\gamma=p_{\min}$$

результат умножения будет заменен нулем, в то время как при использовании предыдущего правила он сохранится (в данном случае $\beta=p_{\min}$).

7.8. ОСНОВНЫЕ МЕТОДЫ НЕУСКОРЕННОГО ДЕЛЕНИЯ

1°. Рассмотрение операции деления мы начнем с общего описания.

При делении чисел с фиксированной запятой в арифметическом устройстве формируется частное Q, которое в общем случае является приближенным частным, т. е.

$$A/C \approx Q.$$

Здесь A — делимое, C — делитель. В тех методах деления, которые будут рассмотрены в этом параграфе, Q содержит старшие n разрядов точного частного A/C.

Выработка знака частного Q на основе знаков делимого и делителя обычно не представляет особых трудностей. Поэтому для упрощения будем считать A, C, Q положительными числами.

Если $A \geqslant C$, т. е. если $A/C \geqslant 1$, то должен вырабатываться признак (сигнал) переполнения, который будем обозначать буквой ω.

Само деление представляет собой циклический процесс; на каждом i-м цикле определяется очередная двоичная цифра частного q_i. Пронумеруем разряды частного Q следующим образом:

$$Q=0, \; q_2 q_3 \ldots q_{n+1}.$$

Общее количество циклов деления равно $n+1$. Первый цикл отличается от других тем, что при его выполнении проверяется неравенство $A \geqslant C$, на втором цикле определяется q_2, на третьем — q_3 и т. д. В дальнейшем мы подробно изучим процесс определения цифр частного.

Деление чисел с плавающей запятой, т. е. вычисление частного

$$A/C = 2^{\alpha} a / 2^{\gamma} c$$

производится в два этапа. На первом определяется разность порядков и делятся мантиссы, т. е. вычисляется величина:

$$A/C = 2^{\alpha-\gamma} Q,$$

где $Q \approx a/c$. Как и при делении чисел с фиксированной запятой, Q содержит старшие разряды точного частного a/c. Так как

$$1/2 \leqslant a, \; c \leqslant 1-2^{-n} \qquad (7.8.1)$$

(мы пока не говорим об особых случаях $a=0$ или $c=0$), то

$$1/2 < a/c \leqslant 2-2^{-n+1}. \qquad (7.8.2)$$

Поэтому при $1 \leqslant Q \leqslant 2 - 2^{-n+1}$ выполняется второй этап операции (нормализация результата), во время которого Q сдвигается на один разряд вправо, а к разности порядков прибавляется единица. Если обозначить результат операции через $B = 2^{\beta} b$, то можно записать, что

$$\left.\begin{array}{l} b = Q \\ \beta = \alpha - \gamma \end{array}\right\} \text{ при } Q < 1, \quad \left.\begin{array}{l} b = Q/2 \\ \beta = \alpha - \gamma + 1 \end{array}\right\} \text{ при } Q \geqslant 1.$$

Очевидно, что мантисса b тоже находится в диапазоне (7.8.1).

Рассмотрим особые случаи. Если

$$c = 0 \text{ или } \alpha - \gamma > p_{\max}, \text{ или } Q \geqslant 1, \ \alpha - \gamma + 1 > p_{\max},$$

т. е. если результат деления превышает максимально допустимое число $2^{p_{\max}}(1 - 2^{-n})$, то вырабатывается сигнал ω переполнения разрядной сетки. Иногда вместо рассмотрения трех последних неравенств рассматривают только неравенство $\alpha - \gamma + 1 > p_{\max}$ и вырабатывают $\omega = 1$ при удовлетворении этого неравенства. Недостаток такого более простого, но более грубого правила состоит в том, что при

$$Q < 1, \ \alpha - \gamma = p_{\max}$$

вырабатывается $\omega = 1$, хотя в этом случае $\beta = p_{\max}$, т. е. вычисления можно было бы продолжать.

Если $a = 0$, то в качестве результата операции обычно используется число

$$2^{p_{\min}} . 0.$$

То же самое производится, если

$$Q < 1, \ \alpha - \gamma < p_{\min} \text{ или } \alpha - \gamma + 1 < p_{\min},$$

т. е. если результат деления меньше минимально допустимого числа $2^{p_{\min}} . \frac{1}{2}$. Иногда это правило заменяется более грубым:

$$\alpha - \gamma < p_{\min}.$$

Недостаток последнего правила состоит в том, что при

$$Q \geqslant 1, \ \alpha - \gamma + 1 = p_{\min}$$

результат деления будет заменен нулем, в то время как при использовании предыдущего правила он сохранится (в данном случае $\beta = p_{\min}$).

Мантиссы a и c делятся фактически так же, как и числа A и C с фиксированной запятой. Разница состоит лишь в том, что во время первого цикла деления делимое с делителем не сравниваются, как при делении чисел с фиксированной запятой, а определяется целая часть частного, которое теперь имеет вид

$$Q = q_1, \ q_2, \ q_3 \ldots$$

Из (7.8.2) следует, что по крайней мере одна из цифр q_1, q_2 равна единице. Учитывая, что при $q_1 = 1$ частное Q будет во время нормализации результата сдвигаться на один разряд вправо, можно записать:

$$Q=q_1,\ q_2q_3\ldots q_{n+1} \text{ при } q_1=0$$

$$Q=q_1,\ q_2q_3\ldots q_n \text{ при } q_1=1,$$

т. е. при $q_1=0$ нужно выполнить $n+1$ цикл, а при $q_2=1$ — выполнить n циклов; в обоих случаях в частном получится n значащих разрядов.

2°. Перейдем к детальному рассмотрению процесса получения последовательных цифр q_i частного Q при делении с фиксированной или с плавающей запятой. Основными двумя методами неускоренного деления следует считать *деление с восстановлением остатка* и *деление без восстановления остатка*. Рассмотрим процесс получения цифр частного по первому из этих методов.

На рис. 7.8.1 показана структурная схема одного $(i+1)$-го цикла деления с восстановлением остатка. Цикл начинается со сдвига на один разряд влево (удвоения) остатка B_i, полученного во время предыдущего, i-го цикла. После этого вычисляется разность между удвоенным остатком и делителем. Если она положительна, то в очередной разряд частного записывается цифра $q_{i+1}=1$, а эта разность является новым остатком B_{i+1}. Если разность между $2B_i$ и делителем отрицательна, то, как видно из рисунка, в частное записывается $q_{i+1}=0$, а остаток $2B_i$ восстанавливается и воспринимается как новый остаток B_{i+1}. Таким образом, во время одного цикла производится одна или две операции сложения-вычитания.

Рассмотрим пример деления чисел с фиксированной запятой.

$$\frac{A}{C}=\frac{0,100}{0,101} \qquad n=3$$

1-й цикл:

$A=2B_0$	$0,100$	
$-C$	$1,011$	
$2B_0-C$	$1,111$	$e=0,\ 2B_0-C<0,\ \omega=0$
$+C$	$0,101$	
$2B_0=B_1$	$0,100$	

2-й цикл:

$2B_1$	$1,000$	
$-C$	$1,011$	
$2B_1-C=B_2$	$0,011$	$e=1,\ 2B_1-C\geqslant 0,\ q_2=1$

3-й цикл:

$2B_2$	$0,110$	
$-C$	$1,011$	
$2B_2-C=B_3$	$0,001$	$e=1,\ 2B_2-C\geqslant 0,\ q_3=1$

4-й цикл:

$2B_3$	$0,010$	
$-C$	$1,011$	
$2B_3-C$	$1,101$	$e=0,\ 2B_3-C<0,\ q_4=0$
$+C$	$0,101$	
$2B_3=B_4$	$0,010$	$Q=0,110$

В этом примере и в других примерах деления, рассматриваемых в данном параграфе, делитель вычитается с использованием дополнительного кода. Кроме того, следует отметить, что все числа изображаются без знакового разряда (можно сказать, что используется модификация «в» одного из методов вычитания, рассмотренных в § 7.6). Разряд, расположенный слева от запятой в этом и других примерах деления, является не знаковым разрядом, а разрядом целой части и имеет вес 2^0; о знаке разности $2B_i - C$ в примерах нужно судить не по знаковому разряду (его нет), а по сигналу переноса e из старшего разряда.

Между делимым A, делителем C, приближенным частным Q и последним остатком B_{n+1} существует следующее точное соотношение:

$$A/C = Q + B_{n+1}/C2^n.$$

Так как Q содержит старшие n разрядов точного частного A/C, то в тех ЦВМ, где в качестве результата деления, кроме Q, выдается последний остаток B_{n+1}, программист может использовать этот

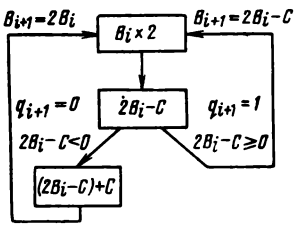


Рис. 7.8.1

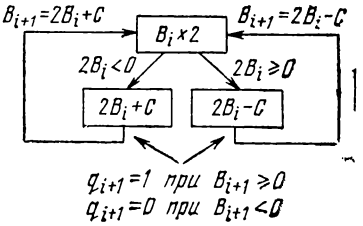


Рис. 7.8.2

остаток для повторного деления (B_{n+1} на C) и получить еще n разрядов частного. Многократно повторяя эту процедуру, можно вычислить частное с любой точностью.

Теперь рассмотрим пример деления мантисс a и c чисел с плавающей запятой.

$$\frac{a}{c} = \frac{0{,}111}{0{,}110} \qquad n = 3$$

1-й цикл

$a = 2B_0$	0,111	
$-c$	1,010	
$2B_0 - c = B_1$	0,001	$e = 1,\ 2B_0 - c \geqslant 0,\ q_1 = 1$

2-й цикл

$2B_1$	0,010	
$-c$	1,010	
$2B_1 - c$	1,100	$e = 0,\ 2B_1 - c < 0,\ q_2 = 0$
$+c$	0,110	
$2B_1 = B_2$	0,010	

$$
\text{3-й цикл}\left\{\begin{array}{rll}
- & 2B_2 & 0{,}100 \\
& -c & \underline{1{,}010} \\
& 2B_2 - c & 1{,}110 \qquad e=0,\ 2B_2 - c < 0,\ q_3 = 0 \\
& +c & \underline{0{,}110} \\
& 2B_2 = B_3 & 0{,}100 \qquad Q = 1{,}00
\end{array}\right.
$$

В этом примере выполнено только 3 цикла потому, что $q_1 = 1$. Из примера видно, что разницы между делением чисел с фиксированной запятой и делением мантисс почти нет. Соотношения между a, c, Q и последним остатком имеют следующий вид:

$$\frac{a}{c} = Q + \frac{B_{n+1}}{c2^n} \quad (\text{при } q_1 = 0),$$

$$\frac{a}{c} = Q + \frac{B_n}{c2^{n-1}} \quad (\text{при } q_1 = 1)$$

Частным случаем деления с восстановлением остатка следует считать *деление с незавершением* (nonperforming division), реализация которого возможна при использовании определенного типа накапливающего сумматора, вычисляющего разность $2B_i - C$. При этом методе в сумматоре сперва происходит процесс распространения переносов. Если в конце процесса выясняется, что $2B_i - C \geqslant 0$, то вычитание доводится до конца и в регистр накапливающего сумматора вместо $2B_i$ принимается вычисленная разность $2B_i - C$. Если же выясняется, что $2B_i - C < 0$, то вычитание до конца не доводится и в регистре остается число $2B_i$. Этим исключается необходимость восстановления остатка, т. е. экономится время.

3°. Теперь рассмотрим деление без восстановления остатка. На рис. 7.8.2 показана структурная схема $(i+1)$-го цикла деления. Остатки B_i при использовании этого метода могут быть как положительными, так и отрицательными числами. Из рисунка видно, что во время каждого цикла выполняется только одна операция сложения-вычитания.

Разделим те же числа A и C, что и в предыдущем примере деления чисел с фиксированной запятой:

$$\frac{A}{C} = \frac{0{,}100}{0{,}101} \qquad n = 3,$$

$$
\text{1-й цикл}\left\{\begin{array}{rll}
& A = 2B_0 & 0{,}100 \\
& -C & \underline{1{,}011} \\
& 2B_0 - C = B_1 & 1{,}111 \qquad e=0,\ B_1 < 0,\ \omega = 0.
\end{array}\right.
$$

$$
\text{2-й цикл}\left\{\begin{array}{rll}
& 2B_1 & 1{,}110 \\
& +C & \underline{0{,}101} \\
& B_2 & 0{,}011 \qquad e=1,\ B_2 \geqslant 0,\ q_2 = 1,
\end{array}\right.
$$

3-й цикл
$$\begin{array}{ll} 2B_2 & 0{,}110 \\ -C & 1{,}011 \\ \hline B_3 & 0{,}001 \end{array} \qquad e=1,\ B_3 \geqslant 0,\ q_3=1,$$

4-й цикл
$$\begin{array}{ll} 2B_3 & 0{,}010 \\ -C & 1{,}011 \\ \hline B_4 & 1{,}101 \end{array} \qquad e=0,\ B_4<0,\ q_4=0.$$

Как и при делении с восстановлением остатка, частное Q содержит старшие разряды точного частного A/C. Соотношения между A, C, Q, B_{n+1} имеют следующий вид:

$$A/C=Q+B_{n+1}/C2^n \quad (\text{при } B_{n+1}\geqslant 0),$$

$$A/C=Q+(B_{n+1}+C)/C2^n \quad (\text{при } B_{n+1}<0).$$

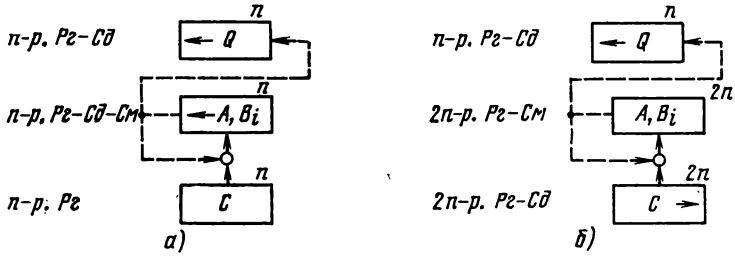


Рис. 7.8.3

Деление мантисс при делении чисел с плавающей запятой производится фактически таким же образом:

$$\frac{a}{c}=\frac{0{,}111}{0{,}110} \qquad n=3$$

1-й цикл
$$\begin{array}{ll} a=2B_0 & 0{,}111 \\ -c & 1{,}010 \\ \hline B_1 & 0{,}001 \end{array} \qquad e=1,\ B_1 \geqslant 0,\ q_1=1,$$

2-й цикл
$$\begin{array}{ll} 2B_1 & 0{,}010 \\ -c & 1{,}010 \\ \hline B_2 & 1{,}100 \end{array} \qquad e=0,\ B_2<0,\ q_2=0,$$

3-й цикл
$$\begin{array}{ll} 2B_2 & 1{,}000 \\ +c & 0{,}110 \\ \hline B_3 & 1{,}110 \end{array} \qquad e=0,\ B_3<0,\ q_3=0.$$

При таком делении

$$a/c=Q+B_{n+1}/c2^n \quad (\text{при } q_1=0,\ B_{n+1}\geqslant 0)$$

$$a/c=Q+(B_{n+1}+c)/c2^n \quad (\text{при } q_1=0,\ B_{n+1}<0),$$

$$a/c=Q+B_n/c2^{n-1} \quad (\text{при } q_1=1,\ B_n\geqslant 0),$$

$$a/c=Q+(B_n+c)/c2^{n-1} \quad (\text{при } q_1=1,\ B_n<0).$$

4°. В описанных в этом параграфе методах деления очередной остаток B_i сдвигался влево. Очевидно, что результат деления не изменится, если вместо сдвига влево остатка сдвигать вправо делитель, т. е. если вместо вычисления, например, разности $2B_i-C$ вычислять разность $B_i-C/2$. Это обстоятельство приводит к существованию двух основных схем деления — см. рис. 7.8.3. Обозначения на рисунке те же, что и на рис. 7.7.1. Штриховая линия указывает, что перенос из старшего разряда сумматора, определяющий знак числа $2B_i-C$ или $2B_i+C$, управляет выбором очередной цифры q_{i+1} и выбором числа, которое должно быть прибавлено к содержимому сумматора. Первая из двух схем деления проще, чем вторая, и поэтому чаще находит применение. Достоинство второй схемы состоит в том, что при использовании определенных типов накапливающих сумматоров можно производить сдвиг делителя до завершения сложения-вычитания в сумматоре.

СПИСОК ЛИТЕРАТУРЫ

К главе 1

1. **Глушков В. М., Капитонова Ю. В., Летичевский А. А.** Теория структур данных и синхронные параллельные вычисления. — Кибернетика, 1976, № 6, с. 2—15.

2. **Гливенко Е. В.** Об одном принципе параллельных вычислений. — Программирование, 1977, № 1, с. 3—9.

3. **Карцев М. А.** Вопросы построения многопроцессорных вычислительных систем. — Вопросы радиоэлектроники. Сер. ЭВТ, 1970, вып. 5—6, с. 3—19.

4. **Мультипроцессорные** системы и параллельные вычисления/ Под ред. Энслоу Ф. Г. Пер. с англ. — М.: Мир, 1976.—383 с.

5. **Байцер Б.** Архитектура вычислительных комплексов/ Пер. с англ. — М.: Мир, 1974.—Т. 1—498 с, Т. 2—566 с.

6. **Baer J. L., Estrin G.** A priory sheduling of bi — logic graph models of computations. — INTERNET, 1969, p. 119—127.

7. **Baer J. L., Estrin G.** Bounds for maximum parallelism in a bilogic praph model of computation. — IEEE Trans. Comput., 1969 v. C-18, № 11, p. 1012—1014.

8. **Martin D., Estrin G.** Models of computations and systems-evaluation of vertex probabilities in graph models of computations. — J. ACM, 1967, v. 14, № 2, p. 281—300.

9. **Головкин Б. А.** Построение вероятностной модели и анализ параллельных вычислительных процессов. — Изв. АН СССР. Техническая кибернетика, 1973, № 3, с. 86—96.

10. **Головкин Б. А.** Об одном классе динамических многомерных моделей. — Вопросы радиоэлектроники. Сер. ЭВТ, 1973, вып. 7, с. 25—30.

11. **Евреинов Э. В., Косарев Ю. Г.** Однородные универсальные вычислительные системы высокой производительности. — Новосибирск: Наука, 1966.—308 с.

12. **Поспелов Д. А.** Введение в теорию вычислительных систем.— М.: Советское радио, 1972.—280 с.

13. **Голубев-Новожилов Ю. Г.** Многомашинные комплексы вычислительных средств. — М.: Сов. радио, 1967.—424 с.

14. **Карцев М. А.** Распараллеливание вычислительных алгоритмов итерационного типа. — Вопросы радиоэлектроники. Сер. ЭВТ, 1971, вып. 9, с. 36—39.

К главе 2

1. **Карцев М. А.** Архитектура цифровых вычислительных машин. — М.: Наука, 1978.—296 с.

2. **Мультипроцессорные** системы и параллельные вычисления/ Под ред. Энслоу Ф. Г. Пер. с англ. — М.: Мир, 1976.—383 с.

3. **Crane B. A., Cilmartin H. J., Huttenhoff J. H., Rux R. T., Shively R. R.** PEPE computer architecture. — COMPCON-72, 6th Annu. IEEE Comput. Soc. Int. Conf., San Francisco, 1972, p. 57—60.

4. **Evensen A. J., Troy J. L.** Introduction to the architecture of a 288-element PEPE. — Proc. Sagamore Computer Conf. on parallel processing, Springer-Verlag, N. Y., 1973, p. 162—169.

5. **Slotnick D. L., Borck W. C., McReynolds R. C.** The Solomon computer. — 1962 Fall Joint Computer Conf. AFIPS Proc., 1962, v. 22, Washington, p. 97—108.

6. **Barnes G. H., Brawn R. M., Kato M., Kuck J., Slotnick D. L., Stokes R. A.** The ILLIAC IV Computer. — IEEE Trans. Comput. 1968, v. C-17, № 8, p. 746—757.

7. **Bouknight W. J., Dennenberg S. A., McIntyre D. E., Sameh A. H., Slotnick D. L.** The ILLIAC-IY System. — Proc. of IEEE, 1972, v. 60, № 4, p. 369—388.

8. **Вопросы** радиоэлектроники. Сер. ЭВТ, 1980, вып. 9.

9. **Martin D. F., Estrin G.** Experiments on models of computation and systems. — IEEE Trans. Comput., 1967, v. EC-16, № 1, p. 59—69.

10. **Noor A. K., Fulton R. E.** Impact of CDC STAR-100 computer on finite element systems. — J. Struct. Div. Proc. Amer. Soc. Civ. Eng., 1975, v. 101, № 4, p. 731—750.

11. **Карцев М. А.** Вопросы построения многопроцессорных вычислительных систем. — Вопросы радиоэлектроники. Сер. ЭВТ, 1970, вып. 5, 6, с. 3—19.

12. **Abel N. E., Budnik P. P., Kuck D. J., Muraoka Y., Northoote R. S., Wilhelmson R. B.** TRANQUIL: A language for an array processing computer. Spring Joint Computer Conference, 1969, p. 57—73.

13. **Lamport L.** The Parallel exicution of DO loops. — Communications of the ACM, 1974, v. 17, № 2, p. 83—93.

14. **Lawrie D. H., Layman T., Baer D., Randal J. M.** Glypnir — a programming languagef or Illiac-IV. — Communications of ACM, 1975, v. 18, № 3, p. 157—164.

15. **Карцев М. А.** Арифметика цифровых машин. — М.: Наука, 1969.—576 с.

16. **Карцев М. А.** Вычислительная машина М-10, ДАН, 1979, т. 245, № 2, с. 309—312.

К главе 3

1. **Weinberger A., Smith J. L.** A one-microsecond adder using onemegacycle circuitry. — IRE Trans. on Electron. Comput., 1956, v. EC-5, № 2, p. 65—73.

2. **Rowe W. D.** A new approach to high-speed logic. — In: Proc. of the West Joint Computer Conf., San Francisco — N. Y., 1959, p. 277—283.

3. **Arithmetic** arrays using standart COS/MOS building blocks. — RSA Solid State Division, 1973, p. 373—378.

4. **Жук А. И., Жук В. И.** О циркуляции сигналов по цепи переноса в сумматорах обратных кодов. — Вопросы радиоэлектроники. Сер. ЭВТ, 1968, вып. 4, с. 24—33.

5. **Потемкин И. С.** Сумматоры. Министерство высшего и среднего специального образования СССР. Московский ордена Ленина Энергетический институт. — М., 1975.—66 с.

6. **Wilkes M. V.** Automatic digital computers. London: Methuen and Co., 1958.—305 p.
7. **Morgan L. P., Jarvis D. B.** Transistor logic using current switching and routing techniques and its application to a fast-carry propagation adder. — IEE Proc., 1959, v. 106, № 29, Pt. B, p. 467—468.
8. **Lehman M., Burla N.** Skip techniques for high-speed carry propagation in binary arithmetic units. — IRE Trans. on Electron. Comput., 1961, v. EC-10, № 4, p. 691—699.
9. **Heijn H. I., Selman J. C.** The Philips Computer Pascal. — IRE Trans. on Electron. Comput., 1961, v. EC-10, № 2, p. 175—183.
10. **Majersky S.** On determination of optimal distributions of carry skips in adders. — IEEE Trans. Comput., 1967, v. 16, № 1, p. 45—58.
11. **Nadler M.** A high-speed electronic arithmetic unit for automatic computing machines. — Acta Technica, 1956, v. 1, № 6, p. 464—478; 1957, v. 2, № 1, p. 101—102.
12. **Пат.** № 3230354 (США). НКИ 235—164.
13. **Sclansky J.** Conditional-sum addition logic. — IRE Trans. on Electron. Comput, 1960, v. EC-9, № 2, p. 226—231.
14. **Bedrij O. J.** Carry-select adder. — IRE Trans. on Electron. Comput., 1962, v. EC-11, № 3, p. 340—346.
15. **Беляков М. И., Брик В. А.** Определение оптимальных параметров ускоренного сумматора. — Вопросы радиоэлектроники. Сер. ЭВТ, 1970, вып. 9, с. 62—66.

К главе 4

1. **Карцев М. А.** Арифметика цифровых машин. — М.: Наука, 1969.—576 с.
2. **Хетагуров Я. А.** Схема сокращения времени умножения. — В кн.: Вычислительная техника. — М.: Атомиздат, 1960, с. 48—53.
3. **Карцев М. А.** Арифметические устройства электронных цифровых машин. — М.: Физматгиз, 1958.—158 с.
4. **Брик В. А., Лушпин Л. И.** Построение однотактных множительных устройств на больших интегральных схемах. — Вопросы радиоэлектроники. Сер. ЭВТ, 1974, вып. 11, с. 41—52.
5. **Dean K. J.** Versatile multiplier arrays. — Electronic Letters, 1968, v. 4, № 16, p. 333—334.
6. **Hoffman J. C., Lacaze B., Csillag P.** Multiplier parralèle a circuits logiques iteratifs. — Electronic Letters, 1968, v. 4, № 9, p. 178.
7. **Bandyopadhyay S., Basu S., Choudhury A. K.** An iterative array for multiplication of signed bihary numbers. — IEEE Trans. Comput., 1972, v. 21, № 8, p. 921—922.
8. **Хетагуров Я. А., Попов Ю. А., Любенцов В. М.** Матричное устройство перемножения. — В кн.: Вопросы вычислительной математики и вычислительной техники. — М.: Машгиз, 1963, с. 157—165.
9. **Брик В. А.** О быстродействии пирамид сумматоров множительных устройств ЦВМ. — Вопросы радиоэлектроники. Сер. ЭВТ, 1970, вып. 5—6, с. 79—87.
10. **Брик В. А., Борисов Ю. М., Танетов Г. И.** Пирамида сумматоров для арифметического устройства цифровой вычислительной машины. — В кн.: Элементы и устройства управляющих машин. — М., 1966, с. 159—166.

11. **Majithia J. C., Kitai R.** An iterative array for multiplication of signed binary numbers. — IEEE Trans. Comput., 1971, v. 20, № 2, p. 214—216.

12. **Deegan I. D.** Cellular multiplier for signed binary numbers. — Electronic Letters, 1971, v. 7, № 151, p. 436—437.

13. **Kingsbury N. G.** Hilgh-speed binary multiplier. — Electronic Letters, 1971, v. 7, № 10, p. 277—278.

14. **Springer J., Alfke P.** Parallel multiplier gets boots from IC iterative logic. — Electronics, 1970, v. 43, № 21, p. 89—93.

15. **Habibi A., Wintz P. A.** Fast multipliers. — IEEE Trans. Comput. 1970, v. 19, № 2, p. 153—157.

16. **Stylianos P.** A 40-ns 17-bit by 17-bit array multiplier. — IEEE Trans. Comput, 1971, v. 20, № 4, p. 442—447.

17. **Храпченко В. М.** Методы ускорения арифметических операций, основанные на преобразовании многорядного кода. — Вопросы радиоэлектроники. Сер. VII ЭВТ, 1965, вып. 8, с. 121—144.

18. **Wallace C. S.** A suggestion for a fast multiplier. — IEEE Trans. Comput., 1964, v. EC-13, № 1, p. 14—17.

19. **А. С.** 126668 (СССР). — Опубл. в Б. И., 1960, № 5.

20. **Jacobsohn D.** A suggestion for a fast multiplier. — IEEE Trans. Comput., 1964, v. EC-13, № 6, p. 754—755.

21. **Dadda L.** Some schemes for parallel multipliers. — Alta Frequenza, 1965, v. 34, № 5, p. 349—356.

22. **Карцев М. А.** Устройство для ускоренного выполнения умножения и деления. — Вопросы радиоэлектроники. Сер. XII Общетехническая, 1959, вып. 12, с. 43—61.

23. **Gex A.** Multiplier-divider cellular array. — Electronic Letters, 1971, v. 7, № 15, p. 442—444.

24. **Deegan I.** Concise cellular array for multiplication and division. — Electronic Letters, 1971, v. 7, № 23, p. 702—704.

25. **Hemel A.** Making small ROM's do math quickly, cheaply and easily. — Electronics, 1970, v. 43, № 10, p. 104—111.

26. **Johnson N.** Improved binary multiplication system. — Electronic Letters, 1973, v. 9, № 1, p. 6—7.

27. **Пат.** 3670956 (США). МКИ G06f7/52.

28. **Lamdan T., Aspinal D.** Some aspects of the design of a simultaneous multiplier. — Proc. IFIP Congr., New York City, 1965, v. 2, Washington, D. C., Spartan Books; London, Macmillan and Co., 1966, p. 440—441.

29. **Мороз И. Г.** Некоторые вопросы однотактного умножения. — В кн.: Вычислительная математика и техника. — Киев: АН УССР, 1962, с. 85—94.

30. **Hennie F. C.** Iterative arrays of logical circuits. — MIT Press, New York — London: Wiley, 1961.—242 p.

31. **Cappa M., Hamacher V. C.** An augmented iterative array for high-speed binary division. — IEEE Trans. Comput., 1973, v. 22, № 2, p. 172—175.

32. **Guild H. H.** Some cellular logic arrays for non-restoring binary division. — The Radio and Electronic Engineer, 1970, v. 39, № 6, p. 345—348.

33. **Majithia J. C.** Nonrestoring binary division using a cellular array. — Electronic Letters, 1970, v. 6, № 10, p. 303—304.

34. **Dean K. J.** Binary division using a datadependent iterative array. — Electronic Letters, 1968, v. 4, № 14, p. 283—284.

35. **Dean K. J.** Cellular logic array for extracting sguare roots. — Electronic Letters, 1968, v. 4, № 15, p. 314—315.
36. **Devries R. C., Chao M. H.** Fully iterative array for extracting square roots. — Electronic Letters, 1970, v. 6, № 8, p. 255—256.
37. **Dean K. J.** Logical circuits for use in iterative arrays. — Electronic Letters, 1968, v. 4, № 5, p. 81—82.
38. **Dean K. J.** Cellular logical array for obtaining the square of a binary number.—Electronic Letters, 1969, v. 5, № 16, p. 370—371.
39. **Anderson S. F., Earle J. G., Goldschmidt R. E., Powers D. M.** The IBM System/360, Model 91. Floating-point execution unit. — IBM J. Res. and Developm., 1967, v. 11, № 1, p. 34—53.
40. **Проектирование** сверхбыстродействующих систем. Комплекс Стретч./Под ред. В. Бухгольца. Пер. с англ. под ред. А. И. Китова. — М.: Мир, 1965.—348 с.
41. **Брик В. А., Ленгник Т. М.** Анализ процессов «затухания» в матричных множительных устройствах. — Вопросы радиоэлектроники. Сер. ЭВТ, 1971, вып. 9, с. 65—69.
42. **Храпченко В. М.** Об одном способе преобразования многорядного кода в однорядный. — Доклады АН СССР, 1963, т. 148, № 2, с. 296—299.
43. **Офман Ю.** Об алгоритмической сложности дискретных функций. — Доклады АН СССР, 1962, т. 145, № 1, с. 48—51.
44. **Борисов Ю. М.** Об аппаратных методах второго порядка ускорения умножения. — В кн. Вычислительная техника, алгоритмы и системы управления. — Труды конференции ИНЭУМ, июль 1966.— М., ИНЭУМ, 1967, с. 5—19.
45. **Брик В. А., Лушпин Л. И.** Преобразование многорядного кода в двухрядный при помощи однотипных узлов. — Вопросы радиоэлектроники. Сер. ЭВТ, 1973, вып. 7, с. 94—116.
46. **Брик В. А.** Некоторые общие свойства многослойных матричных устройств. — Вопросы радиоэлектроники. Сер. ЭВТ, 1978. вып. 6, с. 61—69.
47. **Stenzel W. J., Kubitz W. J., Garcia G. H.** A compact high-speed parallel multiplication scheme. — IEEE Trans. Comput., 1977, v. 26, № 10, p. 948—957.

К главе 5

1. **Пат.** 3293418 (США) НКИ 235—156.
2. **Акушский И. Я.** Многорегистровые схемы выполнения арифметических операций: Вопросы теории математических машин. Сборник первый/ Под ред. Ю. Я. Базилевского. — М.: Физматгиз, 1958, с. 192—218.
3. **Dean K. J.** Binary division using a datadependent iterative array. — Electronic Letters, 1968, v. 4, № 14.
4. **Cappa M., Hamacher V. C.** An augmented iterative array for high-speed binary division. — IEEE Trans. Comput., 1973, v. 22, № 2, p. 172—175.
5. **Stefanelly R.** A suggestion for a high-speed parallel binary divider. — IEEE Trans. Comput., 1972, v. C-21, № 1, p. 42—55.
6. **Wallace C. S.** A suggestion for a fast multiplier. — IEEE Trans. Comput., 1964, v. EC-13, № 1, p. 14—17.
7. **Храпченко В. М.** Методы ускорения арифметических операций, основанные на преобразовании многорядного кода. — Вопросы радиоэлектроники. Сер. VII ЭВТ, 1965, вып. 8, с. 121—144.
8. **Flynn M. J.** Very high-speed computing system. — Proc. of the IEEE, 1966, v. 54, № 12, p. 1901—1909.

9. **Ahmad M.** Iterative schemes for high-speed division. — Comput. J., 1972, v. 15, № 4, p. 333—336.

10. **Shaham Z., Riesel Z.** A note on division algorithms based on multiplication. — IEEE Trans. Comput., 1972, v. C-21, № 5, p. 513—514.

11. **Anderson S. F., Earle J. G., Goldschmidt R. E., Powers D. M.** The IBM System/360, Model 91. Floating-point execution unit. — IBM J. Res. and Developm., 1967, v. 11, № 1, p. 34—53.

12. **Брик В. А., Гаврилин В. А., Жук В. И., Златников В. М., Кислинский В. А., Ленгник Т. М., Лушпин Л. И., Петрова Г. Н.** Многопроцессорное арифметическое устройство. — Вопросы радиоэлектроники. Сер. ЭВТ, 1972, вып. 5, с. 56—67.

13. **Карцев М. А.** Арифметика цифровых машин. — М.: Наука, 1959,—576 с.

14. **Беркс А., Голдстейн Г., Дж. Нейман.** Предварительное рассмотрение логической конструкции электронного вычислительного устройства. — Кибернетический сборник № 9, М.: Мир, 1964, № 9, с. 7—67.

15. **Брик В. А., Гаврилин В. А., Жук В. И., Захаров В. Г., Лушпин Л. И., Петрова Г. Н.** Быстродействующее арифметическое устройство. — Вопросы радиоэлектроники. Сер. ЭВТ, 1970, вып. 5—6, с. 97—108.

16. **Atkins D. E.** The analysis and design of a class of quotient digit selectrors. — 5-th Annu. IEEE Int. Comput. Soc. Conf., Boston, Mass., 1971. Conf. dig., New York, 1971, p. 201—202.

17. **Брик В. А.** Об одном методе деления без восстановления остатка. — Цифровая вычислительная техника и программирование/ Под ред. А. И. Китова. — М.: Сов. радио, 1969, вып. 5, с. 72—84.

18. **Nadler M.** A high-speed electronic arithmetic unit for automatic computing machines. — Acta Technica, 1956, v. I, № 6, p. 464—478; 1957, v. 2, № 1, p. 101—102.

19. **Карцев М. А.** Об одном методе двоичного деления. — В кн.: Материалы научно-техн. конференции «Новые разработки в области вычислительной математики и вычислительной техники». — Киев: ВЦ АН УССР, 1960, с. 409—418.

20. **Robertson J. E.** A new class of digital division methods. — IRE Trans. on Electron. Comput., 1958, v. EC-7, № 3, p. 218—222.

21. **Браиловский В. Л., Глухов Ю. Н.** О методах деления без восстановления остатка. — Вопросы радиоэлектроники. Сер. XII — Общетехническая, 1961, вып. 13, с. 40—59.

22. **Пат.** 3621218 (США).

23. **Soceneantu A., Toma C. I.** Cellular logic array for redundant binary division. — Proc. IEE, 1972, v. 119, № 10, p. 1452—1456.

24. **Карцев М. А.** Устройство для ускоренного выполнения умножения и деления. — Вопросы радиоэлектроники. Сер. XII — общетехническая, 1959, вып. 12, с. 43—61.

25. **Atkins D. E.** Higher-radix division using estimates of the divisor and partial remainders. — IEEE Trans. Comput., 1968, v. C-17, № 10, p. 925—934.

К главе 6

1. **Грамолин В. В.** Повышение регулярности вычислительных устройств при использовании специальных арифметик. — Труды Московского ин-та инженеров железнодорожного транспорта, 1971, вып. 395 «Вопросы теории вычислительных систем», с. 73—83.

2. **Карасик В. М.** Некоторые задачи проектирования ЦВМ на больших интегральных схемах. — Вопросы радиоэлектроники. Сер. ЭВТ, 1971, вып. 1, с. 3—13.

3. **Зейденберг В. К., Матвеенко Н. А., Тароватова Е. В.** Обзор зарубежной вычислительной техники по состоянию на 1971 г. — М.: Институт точной механики и вычислительной техники, 1971.— 316 с.

4. **Cloyd E. M.** MOS/LSI throughout. — IEEE Int Convent. Dig., New York, N. Y., 1971. — New York, N. Y., 1971, p. 74—75.

5. **New** Scientist. — 1973, 3/V, № 844.

6. **Альтман.** Функциональный подход к проектированию БИС.— Электроника, 1975, № 17, с. 58.

7. **Тароватова Е. В., Головня Е. В., Матвеенко Н. А.** Вычислительная техника за рубежом в 1975 г./ Под ред. В. К. Зейденберга. — М.: Институт точной механики и вычислительной техники им. С. А. Лебедева АН СССР. 1976.—275 с.

8. **Микроэлектроника** и однородные структуры для построения логических и вычислительных устройств/ И. В. Прангишвили, Н. А. Абрамова, Е. В. Бабичева, В. В. Игнатущенко. — М.: Наука, 1967.—228 с.

9. **Духнич Е. И., Макаревич О. Б.** Большие интегральные схемы и математическое обеспечение цифровых интегрирующих структур.— В кн.: Вычислительные системы и среды. — Таганрог, 1972, с. 433—435.

10. **Зейденберг В. К., Матвеенко Н. А., Тароватова Е. В.** Обзор зарубежной вычислительной техники по состоянию на 1972.—М.: Институт точной механики и вычислительной техники, 1972.—355 с.

11. **Arluckle W. L., Mattson R. C.** Macromodularity: a design concept to end computer generation gaps. — Computer Design, 1970, v. 9, № 8, p. 69—79.

12. **Russo R. L.** On the tradeoff between logic performance and circuit-to-pin ration for LSI. — IEEE Trans. Comput., 1972, v. 21, № 2, p. 147—153.

13. **Arithmetic** arrays using standard COS/MOS building blocks. — RCA Solid State Division, 1973, p. 373—378.

14. **Белков М. С., Братальский Е. А., Лушпин Л. И.** Способ построения сверхпараллельного сумматора. — Вопросы радиоэлектроники. Сер. ЭВТ, 1975, вып. 7, с. 65—71.

15. **Kingsbury N. G.** High-speed binary multiplier. — Electronic Letters, 1971, v. 7, № 10, p. 277—278.

16. **Springer J., Alfke P.** Parallel multiplier gets boots from IC iterative logic. — Electronics, 1970, v, 43, № 12, p. 89—93.

17. **Habibi A., Wintz P. A.** Fast multipliers. — IEEE Trans. Comput., 1970, v. 19, № 2, p. 153—157.

18. **Stylianos P.** A 40-ns 17-bit by 17-bit array multiplier. — IEEE Trans. Comput., 1971, v. 20, № 4, p. 442—447.

19. **Брик В. А., Лушпин Л. И.** Метод преобразования многорядного кода при помощи двоичных сумматоров. — Вопросы радиоэлектроники. Сер. ЭВТ, 1977, вып. 6, с. 119—124.

20. **Брик В. А., Лушпин Л. И.** Преобразование многорядного кода в двухрядный при помощи однотипных узлов. — Вопросы радиоэлектроники. Сер. ЭВТ, 1973, вып. 7, с. 94—116.

21. **Hemel A.** Making small ROM's do math quickly, cheaply and easily. — Electronics, 1970, v. 43, № 10, p. 104—111.

22. **Johnson N.** Improved binary multiplication system. — Electronic Letters, 1973, v. 9, № 1, p. 6—7.
23. **Пат.** 3670956 (США). МКИ G06f 7/52.
24. **The TTL** Data Book Supplement to CC-401 for Disign Engineers. — Deutschland: Texas Instruments, 1973.
25. **Белков М. С., Братальский Е. А., Брик В. А., Лушпин Л. И., Пахунов В. Н.** Разработка системы БИС для построения узлов ЦВМ. — Вопросы радиоэлектроники. Сер. ЭВТ, 1975, вып. 7, с. 72—80.
26. **Пат.** 3866030 (США). МКИ G06f7/39.

К главе 7

1. **McKeeman W. M.** Representation error for real numbers in binary computer arithmetic. — IEEE Trans. Comput. 1967, v. 16, № 5, p. 682—683.
2. **Hamming R. W.** Numerical methods for scientists and engineers. — New York.: McGraw-Hill Book Co., 1962.—411 p.
3. **Потемкин И. С.** Построение функциональных узлов на потенциальных системах элементов. — М.: МЭИ, 1974.—126 с.
4. **Карцев М. А.** Арифметика цифровых машин. — М.: Наука, 1969.—576 с.
5. **Брик В. А.** Точность выполнения операций сложения — вычитания с плавающей запятой. — Вопросы радиоэлектроники. Сер. ЭВТ, 1972, вып. 5, с. 74—81.
6. **Храпченко В. М.** Об оценке погрешности двоичного умножения. — Проблемы кибернетики/ Под ред. А. А. Ляпунова. — М.: Физматгиз, 1963, вып. 10, с. 165—177.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

МИХАИЛ АЛЕКСАНДРОВИЧ КАРЦЕВ
ВЛАДИМИР АРКАДЬЕВИЧ БРИК

ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ И СИНХРОННАЯ АРИФМЕТИКА

Редактор *Н. Д. Иванушко*
Художественный редактор *Н. А. Игнатьев*
Художник *О. В. Камаев*
Технический редактор *Г. З. Кузнецова*
Корректор *О. И. Галанова*
ИБ № 377

Сдано в набор 17.10.80 Подписано в печать 26.03.81 Т-00800
Формат 84×108¹/₁₆ Бумага книжно-журнальная Гарнитура литературная
Печать высокая Усл. печ. л. 18,9 Уч.-изд. л. 19,58 Усл.-кр. отт. 18,9
Тираж 10 000 экз. Изд. № 19523 Зак. 1013 Цена 1 р. 20 к.

Издательство «Радио и связь», Москва, Главпочтамт, а/я 693

Московская типография № 10 «Союзполиграфпрома»
Государственного Комитета СССР
по делам издательств, полиграфии и книжной торговли.
Москва, М-114, Шлюзовая наб., 10