**Комитет по высшей школе министерства науки, высшей школы
и технической политики Российской Федерации**

СЕВЕРО-ЗАПАДНЫЙ ЗАОЧНЫЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

В.В. Спиридонов

Проектирование структур АЛУ

Утверждено редакционно-издательским советом института
в качестве учебного пособия

(электронная версия)

Санкт-Петербург 1992

УДК 681.325

Спиридонов В.В. Проектирование структур АЛУ: Учебное пособие. – СПб.: СЗПИ, 1992. – 72 с.

В учебном пособии рассматриваются вопросы организации, функционирования и структурного проектирования арифметико-логических устройств (АЛУ). Дается классификация АЛУ, описываются средства их представления, излагаются основы методологи из проектирования на базе функционально-структурного подхода, приводятся типовые схемы АЛУ для выполнения различных операций.

Пособие предназначено для студентов специальности 2201 – электронные вычислительные машины, системы, комплексы и сети.

Р е ц е н з е н т ы: кафедра вычислительной техники ЛЭТИ им. В.И. Ульянова (Ленина) (зав. кафедрой *Д. В. Пузанков***,** д-р техн. наук проф.). *М. С. Куприянов*, д-р техн. наук, проф. ЛИАП

Содержание

ВВЕДЕНИЕ 4

Глава 1. АРИФМЕТИЧЕСКО-ЛОГИЧЕСКИЕ УСТРОЙСТВА ЭВМ 6

1.1. Назначение, состав и структура АЛУ 6

1.2. Классификация АЛУ 9

1.3. Средства представления АЛУ 14

Глава 2. ОБЩИЙ ПОРЯДОК ПРОЕКТИРОВАНИЯ АЛУ НА ОСНОВЕ ФУНКЦИОНАЛЬНОСТРУКТУРНОГО ПОДХОДА 17

2.1. Обобщенные функции систем переработки информации 17

2.2. Общая методология и этапы проектирования АЛУ 21

2.2.1. Декомпозиции функций АЛУ 26

2.2.2. Способы преодоления противоречий при формировании концепции системы 28

2.2.3. Формирование таблиц соответствия функций, функциональных и конструктивных моделей 29

2.3. Формирование функциональной структуры АЛУ 30

2.4. Модификация алгоритмов и структур АЛУ 42

2.4.1. Модификация алгоритмов выполнения операций 42

2.4.2. Модификация структуры АЛУ 43

2.5. Оценка структур АЛУ 49

Глава 3. СТРУКТУРЫ АЛУ ДЛЯ ВЫПОЛНЕНИЯ РАЗЛИЧНЫХ ОПЕРАЦИЙ 53

3.1. АЛУ для сложения и вычитания двоичных чисел с фиксированной запятой 53

3.2. АЛУ для сложения и вычитания чисел с плавающей запятой 58

3.3. АЛУ для сложения двоично-десятичных чисел 64

3.4. АЛУ для умножения чисел с фиксированной запятой 73

3.5. АЛУ для деления чисел с фиксированной запятой 79

3.6. Особенности обработки смещенных порядков (характеристик) чисел с плавающей запятой 82

ЛИТЕРАТУРА 87

ВВЕДЕНИЕ

Исследования общих принципов проектирования систем различных классов в последние годы становятся все более интенсивными. Интерес к разработкам в этой области стимулируется развитием методов и средств информатики и вычислительной техники, сделавшим возможным создание различных систем автоматизированного проектирования, экспертных систем, систем автоматизации инженерного труда. Непрерывное расширение функциональных возможностей этих систем позволяет решать с их помощью задачи проектирования все более высокого уровня. Однако для этого, в первую очередь, требуются немалые усилия по выявлению и упорядочению профессиональных знаний, методов, приемов и средств, используемых для проектирования.

Стремление к такому упорядочению нашло отражение в создании рядом авторов обобщенных моделей и раскрытии наиболее общих особенностей технических[1] и иных систем и процессов их построения, основанных на изучении накопленного опыта разработки и истории развития техники. Важнейшие положения, сформулированные в этих работах, утверждают:

наличие общих закономерностей в развитии систем различных классов и эффективность их применения при разработке систем;

многоуровневую и неоднозначную связь между функциями и структурой систем при определяющей роли функций;

целесообразность представления систем различных классов и на различных уровнях в виде совокупности (узлов, обеспечивающих реализацию) основных системных процессов: преобразования, хранения, передачи и управления, осуществляемых над веществом, энергией и информацией.

В настоящее время процессы проектирования систем переработки информации (СПИ), ЭВМ и отдельных подсистем детально исследуются с целью создания более или менее формализованной методологии проектирования. Причем наибольшие результаты в этой области достигнуты для завершающих этапов разработки СПИ, связанных с непосредственной технической реализацией систем. Этого нельзя сказать в отношении начальных этапов проектирования, например, этапа выбора структуры системы. Наряду с термином "технические системы" используются и некоторые другие термины, например, "антропогенные системы" (системы, созданные человеком).

В данном пособии рассматриваются вопросы организации и проектирования структур АЛУ с учетом названных выше положений. Причем в пер

[1] Наряду с термином “технические системы” используются и некоторые другие термины, например, “антропогенные системы” (системы, созданные человеком).

вой главе излагаются общие вопросы организации АЛУ, их классификации, языков описания.

Во второй главе приводятся общие основы методологии проектирования структур АЛУ, базирующиеся на концепции функционально-структурного подхода к проектированию систем. Центральным звеном этой концепции является рассмотрение процесса разработки, как перехода от функций к структуре системы (устройства).

В третьей главе пособия рассматриваются структурные схемы АЛУ, ориентированные на выполнение различных операций. Включение в пособие этих вопросов обусловлено, в основном, практическими потребностями организации учебного процесса по дисциплине "Организация ЭВМ и систем", связанными с выполнением курсового проекта.

Глава 1. АРИФМЕТИЧЕСКО-ЛОГИЧЕСКИЕ УСТРОЙСТВА ЭВМ

В этой главе рассматриваются общие вопросы, связанные с назначением, организацией и основными характеристиками АЛУ, а также приводится их классификация. Кроме того, даются сведения о средствах описания АЛУ и процессов их функционирования.

1.1. Назначение, состав и структура АЛУ

Все основные операции по преобразованию данных в ЭВМ производятся в операционных блоках, которые в большинстве случаев называются арифметическо-логическим устройством. Набор операций, выполняемых АЛУ универсальных ЭВМ, должен быть функционально полным, т.е., обеспечивать реализацию любого вычислительного алгоритма. И хотя функциональную полноту можно обеспечить очень узким набором операций (см., например, [6]), число различных операций, выполняемых в АЛУ, обычно составляет от нескольких десятков до нескольких сотен [17]. Это обеспечивает сокращение длины программ и повышение быстродействия ЭВМ в целом.

Как правило, в любом АЛУ предусмотрена возможность выполнения четырех основных арифметических операций, нескольких логических операций, а также сдвигов. Набор операций АЛУ является одной из основных его характеристик.

Так как АЛУ является законченным в функциональном отношении устройством, то на него распространяются общие закономерности технических систем (см. введение и гл. 2). Поэтому в составе АЛУ в общем случае можно выделить четыре группы узлов, соответствующих основным системным процессам: хранения, передачи, преобразования, управления.

К узлам хранения в АЛУ относятся:

- регистры, обеспечивающие хранение операндов, промежуточных и окончательных результатов;

- триггеры, позволяющие хранить различные признаки результатов или какие-либо вспомогательные биты.

В некоторых случаях регистры АЛУ образуют блок регистровой памяти, а триггеры (называемые также флажками) объединяются в регистр состояния.

К узлам передачи, имеющимся в АЛУ, относятся:

- шины, соединяющие отдельные блоки АЛУ;

- блоки вентилей (схем И) и мультиплексоры, обеспечивающие выполнение передачи по выбранному направлению и в нужный момент времени.

К группе узлов преобразования могут относиться:

- сумматоры, выполняющие в ряде случаев несколько различных микроопераций;

- схемы выполнения логических операций, иногда совмещаемые с сумматорами;

- схемы коррекции, например, для операций десятичной арифметики;

- схемы сдвига (сдвигатели);

- преобразователи кодов, служащие для получения обратных или дополнительных кодов;

- счетчики, используемые для вспомогательных преобразований и для подсчета числа циклов в циклических операциях.

К узлам управления можно отнести:

- блок управления АЛУ (если таковой имеется отдельно);

- дешифраторы управляющих сигналов (кодов);

- схемы формирования логических условий (признаков), используемых для организации ветвлений в микропрограммах выполнения операций.

Конечно, следует учитывать, что любое разделение, классификация имеют элемент условности и можно найти достаточно примеров, когда один узел можно отнести к различным группам, В частности, в рассматриваемом случае мультиплексоры и блоки вентилей можно отнести как к узлам передачи, так и к узлам управления, так как они разрешают или запрещают передачу.

Арифметическо-логическое устройство включает узлы перечисленных групп, соединенные тем или иным способом в некоторую структуру. Структурные схемы АЛУ могут быть различными, что определяется различием принципов их построения. Основные особенности организации АЛУ рассмотрены в следующем параграфе, посвященном классификации этих устройств.

Типовая структурная схема АЛУ показана на рис. 1, где *Р1, Р2, Р3, Р4* – регистры, *МП1, МП2* – мультиплексоры, *См* – комбинационный сумматор, *Сдв* – сдвигатель, *СхФормПр* – схема формирования признаков (флажков), *РП* – регистр признаков. В этой схеме основным узлом преобразования информации является сумматор, выполняющий операции суммирования и логические операции. В некоторых АЛУ логические операции выполняются в специальных узлах. Кроме того, преобразования осуществляются и в сдвигателе. Регистры *Р1...Р3* служат для хранения операндов и промежуточных результатов, регистра *Р4* – выходной, используется для промежуточного хранения результатов, снимаемых с выхода сумматора (сдвигателя). Мультиплексоры *МП1* и М*П2* обеспечивают коммутацию на входы сумматора содер

жимого регистров *Р1, ... Р3,* а в некоторых случаях – и инвертирование, т. е. получение обратных кодов их содержимого (если в сумматоре не выполняется операция вычитания непосредственно). СхФормПр обеспечивает формирование значений логических условий, отражающих получение на выходе сумматора нулевого числа, отрицательного числа, переполнения результата, переноса из старшего разряда, четности результата и др., а регистр признаков *РП* (или набор триггеров) сохраняет значения этих условий.

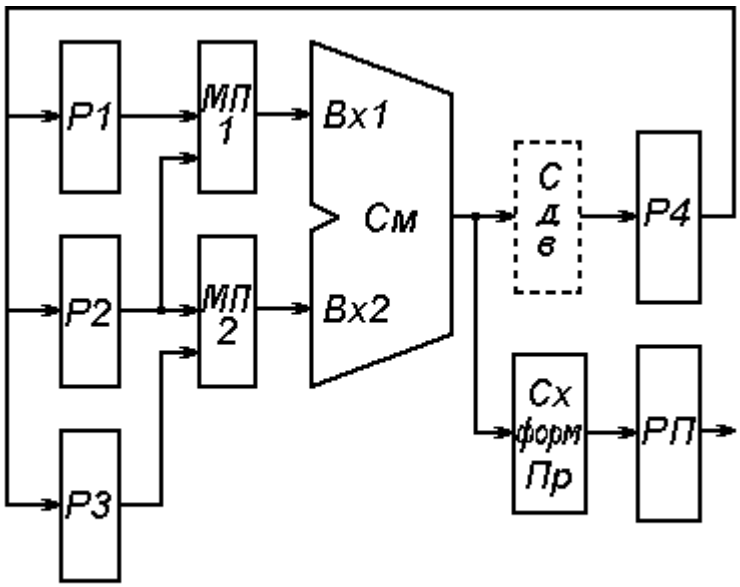

Рис.1. Типовая структурная схема АЛУ

Рассмотренная структурная схема имеет обобщенный характер. На ней не показаны узлы управления, вспомогательные узлы. Использование в схеме сумматора комбинационного типа обусловливает определенные особенности связей между узлами АЛУ. Приводимые в некоторых литературных источниках схемы с сумматором накапливающего типа имеют несколько иной регистровый состав и вид связей. Но поскольку накапливающий сумматор обычно реализуется на базе комбинационного сумматора и регистра, то структура такого АЛУ будет мало отличаться от рассмотренной. Некоторые примеры такой организации будут представлены в третьей главе.

Помимо набора операций и структурной организации, АЛУ характеризуются еще рядом показателей. К ним относятся разрядность обрабатываемых чисел (кодов), времена выполнения различных операций или, иногда, усредненное быстродействие, наличие дополнительных функциональных возможностей типа контроля правильности выполнения операций, устойчивости к отказам, а также конструктивные характеристики, такие как габариты**,** энергопотребление, надежность и пр.

1.2. Классификация АЛУ

В процессорах современных ЭВМ используются различные по своей организации АЛУ. Эти различия обусловлены функциональным назначением АЛУ, способами реализации операций, требованиями по быстродействию и др. Основные характерные особенности того или иного АЛУ можно отнести *к* одной из трех групп: особенности обрабатываемой информации, организации выполнения операций и структурной организации. Рассмотрим эти группы несколько подробнее.

Обрабатываемая в АЛУ информация представляет собой либо численные, либо логические величины (и те и другие могут иметь различную организацию). Как известно, численные величины в ЭВМ представляются поразному. Это проявляется, в основном, в используемых формах представления данных, системах счисления, разрядности, применяемых кодах. По этим признакам АЛУ можно разделить следующим образом.

По форме представления чисел: АЛУ с фиксированной запятой; АЛУ с плавающей запятой; АЛУ с фиксированной и плавающей запятыми (универсальные)**.** Причем в рамках каждого представления имеются некоторые различия, влекущие за собой особенности процедур выполнения операций либо структуры АЛУ [16]. Так, числа с фиксированной запятой могут быть представлены в виде целых или в виде дробных чисел, меньших единицы. Это сказывается на особенностях выполнения операций умножения и деления. Числа с плавающей запятой могут иметь мантиссу и порядок (целое со знаком) или мантиссу и характеристику (смещенный порядок), что влияет на процедуры обработки порядков.

Кроме того, следует также упомянуть наличие и других форм представления, используемых в калькуляторах: так называемая автоматическая запятая, при которой положение запятой в результате операции определяется количеством разрядов дробной части чисел, участвующих в операции; естественная запятая – то же для целых частей [см., например, 5].

По используемой системе счисления: АЛУ, работающие в позиционной системе счисления; АЛУ, работающие в непозиционной системе счисления.

Известно несколько позиционных систем счисления, используемых в ЭВМ. В первую очередь это двоичная и двоично-десятичная системы счисления. Причем последняя также имеет разновидности, применявшиеся в различных ЭВМ и отличающиеся весами двоичных разрядов, смещением цифры нуль и др. Кроме этих систем, широко распространены восьмеричная и шестнадцатеричная системы счисления, дающие, по сравнению с двоичной, большую наглядность в изображении чисел и расширяющие диапазон их представления при одинаковом количестве двоичных разрядов (по отношению к двоичной системе) в записи числа для формата с плавающей запятой.

Известны также случаи использования троичной системы счисления, некоторые работы с R-значными системами, разработки по использованию систем счисления, веса двоичных разрядов в которых соответствуют числам Фибоначчи.

Из непозиционных систем счисления в арифметике используется система остаточных классов (СОК), числа в которой представляются в виде остатков от деления исходного числа на набор взаимно простых чисел, называемых основаниями системы. Такое представление обеспечивает возможность независимой обработки разрядов (остатков) чисел, что, в частности, представляет интерес для цифровой оптической обработки информации [2].

По разрядности обрабатываемых чисел: АЛУ, выполняющие операции над числами (кодами) фиксированной разрядности; АЛУ, обрабатывающие операнды переменной длины.

В обоих случаях само АЛУ имеет фиксированную разрядность блоков, но во второй группе предусмотрены специальные средства, обеспечивающие обработку операндов по частям, и соответствующие микропрограммы выполнения операций. Имеются также АЛУ, в которых операции выполняются над несколькими различными видами операндов фиксированной разрядности, обычно это форматы полуслова, слова и слова двойной длины.

По кодам, используемым для представления отрицательных чисел: АЛУ с использованием обратных кодов; АЛУ с использованием дополнительных кодов.

Принципиальных особенностей структур АЛУ это различие не обусловливает. Известны и устройства, в которых одни операции выполняются с использованием обратных кодов, а другие – дополнительных.

Особенности структурной организации АЛУ определяются составом операционных блоков устройства и характером связей между ними. В этой группе признаков АЛУ можно подразделить следующим образом.

По количеству операционных блоков: одноблочные АЛУ (иначе, универсальные или многофункциональные) и многоблочные АЛУ.

В первых из них имеется операционный блок, в котором может выполняться любая из операций АЛУ. Такая организация характерна для ЭВМ невысокой производительности. Многоблочные АЛУ имеют в своем составе несколько операционных блоков, каждый из которых ориентирован на выполнение какой - либо одной операции, например умножения, или нескольких операций, например сложения и логики. Причем предусматривается одновременная работа различных блоков, что, совместно со специализацией блоков, обеспечивает более высокую производительность ЗВМ с такими АЛУ .По характеру связей: устройства с магистральными и с непосредственными связями.

Для первых из них характерно наличие внутренней шины данных, по которой осуществляются все передачи информации между любыми узлами АЛУ. В случае непосредственных связей в структуре предусматривается набор индивидуальных шин, связывающих пары узлов, между которыми должны выполняться передачи.

Структурные особенности определяются также и назначением ЭВМ, в состав которых входит АЛУ, в целом. Так, например, в специализированных системах для обработки сигналов операционные блоки могут иметь структуру, наиболее приспособленную к алгоритмам обработки [18]. Однако в данном пособии речь идет об ЭВМ общего назначения.

Особенности организации выполнения операций (процесса обработки) проявляются в принципах получения результатов и порядке обработки данных. По этим признакам возможны следующие подразделения.

По принципу получения результата: АЛУ с алгоритмической реализацией операций; табличные АЛУ; таблично-алгоритмические АЛУ.

АЛУ с алгоритмической реализацией операций – наиболее распространенный тип. В них каждая операция (кроме самых простых) представляется в виде последовательности более простых преобразований – микроопераций. Последовательность этих преобразований определяется алгоритмом выполнения операций. Реализуется такая последовательность либо за несколько тактов под управлением соответствующей микропрограммы, обеспечивающей необходимую настройку узлов АЛУ в каждом такте, либо за один такт, при наличии для всех микроопераций отдельных узлов, соединенных в требуемой последовательности.

В табличных АЛУ результат операции не вычисляется каждый раз при ее выполнении. Он выбирается из таблицы – постоянной памяти, в которой заранее записаны назначения результатов, соответствующие всем возможным значениям операндов. Такой способ наиболее эффективен для вычисления сложных функций одного аргумента при небольшой его разрядности, например, тригонометрических функций. Применим он и для реализации обычных арифметических операций.

При выполнении операций в табличных АЛУ значение аргумента (аргументов) используют в качестве адреса ячейки ПЗУ, в которой записан результат, соответствующий этому значению (значениям). Табличный способ обеспечивает высокую скорость обработки, так как независимо от сложности реализуемых преобразований все действия сводятся к считыванию готового результата из ПЗУ. Однако недостатком его является необходимость очень большого объема памяти (таблицы) при увеличении разрядности операндов.

Таблично-алгоритмические АЛУ представляют собой компромисс между первыми двумя способами. В них результат получается сочетанием этих способов. Часть разрядов операндов (обычно старшие разряды) используется

для получения приближенного значения результата табличным способом. По остальным разрядам вычисляется поправка к предварительному результату. Этот метод позволяет сократить объем таблиц при сохранении относительно высокой скорости и находит применение в мощных ЭВМ.

По порядку обработки данных: последовательные АЛУ, параллельные АЛУ и конвейерные АЛУ.

Эти АЛУ различаются между собой по степени параллелизма в выполнении операций. Так, в АЛУ последовательного типа обработка операндов осуществляется последовательно разряд за разрядом. В АЛУ параллельного типа операции производятся одновременно над всеми разрядами операндов.

Известны также промежуточные варианты организации АЛУ – параллельно-последовательные, в которых обработка операндов осуществляется одновременно по группам разрядов, тогда как группы обрабатываются между собой последовательно.

В АЛУ конвейерного типа параллелизм имеет место на уровне операций, т. е. в них возможно выполнение нескольких операций одновременно. Термин «конвейерные АЛУ» имеет различные интерпретации. В ряде случаев его применяют к многоблочным АЛУ. Например, АЛУ, имеющее в качестве отдельных блоков сумматор, устройство умножения и устройство деления может обеспечивать конвейерную обработку. С другой стороны, сами устройства умножения и деления могут быть конвейерного типа и реализовать сразу несколько операций умножения или деления, которые в один и тот же момент времени пребывают в разных стадиях своего выполнения. Эти варианты конвейеров называют конвейерами последовательного типа в отличие от векторных конвейеров [13], выполняющих операции над векторами.

Очевидно, что чем выше степень параллелизма, заложенного в структуре АЛУ, тем более высокопроизводительным оно является. Однако сложность схем, а следовательно, аппаратные затраты на реализацию таких АЛУ и управление ими тоже возрастают.

Общий вид рассмотренной классификации показан на рис. 2.

Следует отметить, что возможны также подразделения АЛУ и по некоторым другим признакам.

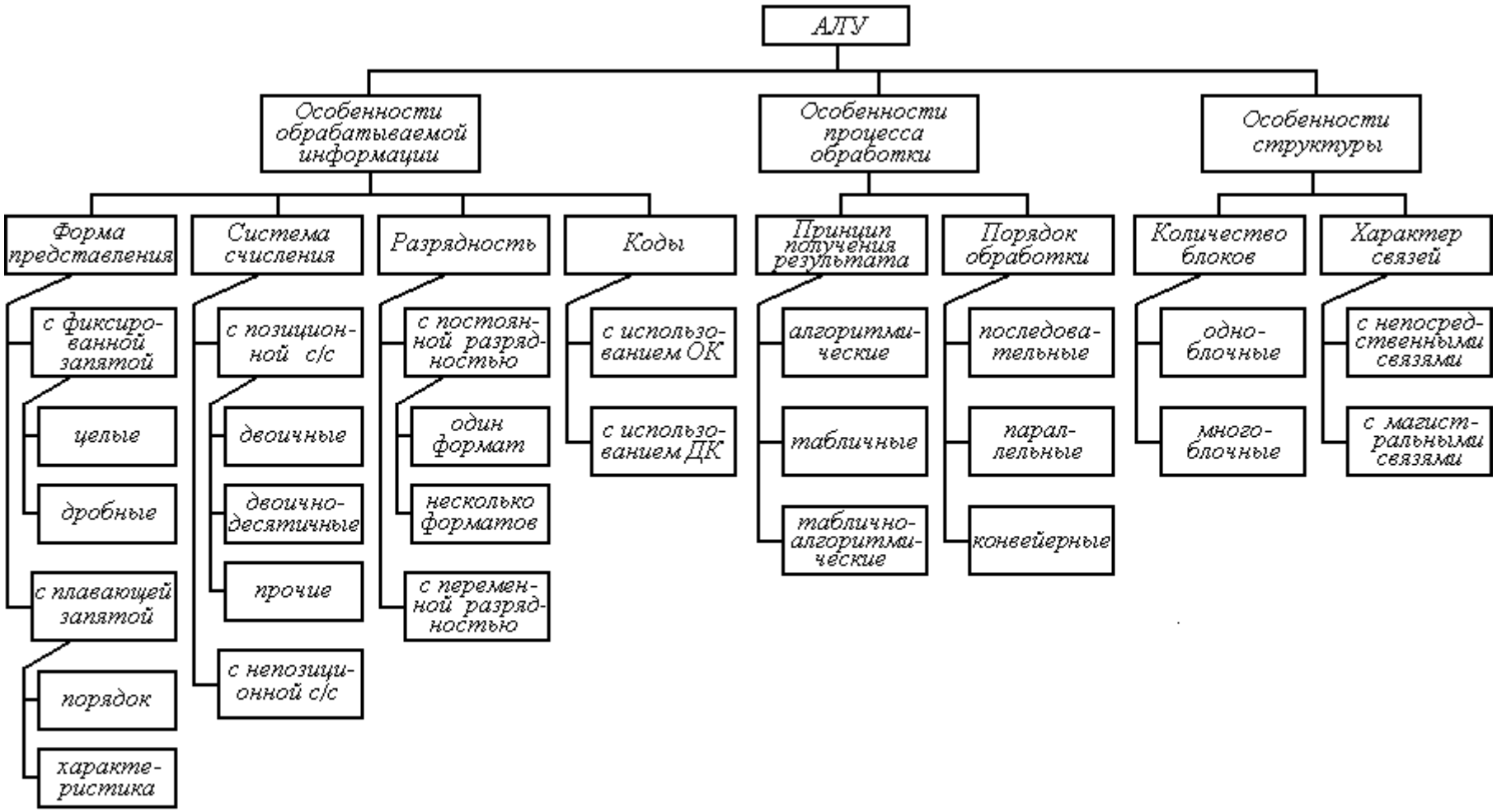

Рис. 2. Классификация АЛУ

1.3. Средства представления АЛУ

Необходимость в представлении АЛУ возникает в различных случаях: а) при описании устройств в литературе, документации, рекламе; б) при их проектировании, как ручном, так и автоматизированном; в) при моделировании функционирования с целью анализа характеристик и проверки правильности работы. При этом, поскольку ЭВМ в целом и АЛУ можно отнести к классу многоуровневых систем, их представление может осуществляться на различных уровнях детализации: начиная от уровня электронных компонентов логических элементов и кончая уровнем всего устройства в целом. Кроме того, представление АЛУ (ЭВМ) может быть ориентировано на описание структуры устройства, его функционирования и его технической реализации. Поэтому для различных целей и уровней представления АЛУ должны существовать различные средства, формальные и языковые. (Помимо этого, несколько самостоятельной, но тесно связанной с ними областью являются описания процессов проектирования цифровых устройств.)

Начало систематической разработки средств представления ЗВМ принято относить к 60-м годам XX столетия, когда стали появляться языки для описания аппаратных средств ЭВМ. Нетрудно заметить, что появление этих языков последовало за разработкой языков программирования. Но понятно, что средства представления ЗВМ существовали и на более ранних периодах развития вычислительной техники. Они заимствовались как из математических, так и из инженерных (электротехнических, радиотехнических, связных) дисциплин.

Из математических дисциплин были заимствованы аппарат булевых функций и алгоритмические представления процессов вычислений. Из инженерных дисциплин были использованы представления в виде электрических схем, блок-схем.

На первых порах, когда вычислительные машины были относительно просты, а проектирование их велось исключительно вручную, таких представлений было достаточно. Однако с усложнением самих ЭВМ, увеличением числа уровней иерархии в них, развитием методов их проектирования и теоретических разработок понадобились более пригодные для машинного использования формализованные методы описания ЭВМ. В результате были созданы различные формальные модели и языковые средства, ориентированные на различные уровни представления и целевое назначение. В большинстве случаев эти средства не были узкоспециализированными и значительная часть из них пригодна для использования на различных уровнях и для разных целей.

При представлении ЭВМ и систем в зависимости от степени детализации различают от четырех до семи и более уровней. Один из возможных вариантов, включающий шесть уровней представления структуры и соответствующие им процессы функционирования и техническую реализацию, а также

средства представления, приведен в табл. 1. Многие средства, перечисленные в таблице, не требуют специальных пояснений. Необходимо привести лишь некоторые уточнения, касающиеся языков описания. Эти языки можно разделить на три группы, соответствующие различным аспектам описания вычислительных средств: языки структурного описания, языки функционального описания и языки графического описания [10].

Таблица 1 Уровни и средства представления вычислительных устройств

Уровень представления структуры	Процессы функционирования	Техническая реализация	Средства представления: структуры	функционирования	реализации
1.Электронные схемы	Токи в эл.цепях	Радиокомпоненты	Электрическая схема (принципиальная)	Диф.уравнения для токов и напряжений	Фотошаблоны масок интегральных схем (языка графического типа)
2. Логические схемы	Логические преобразования, переключение состояний элементов памяти	Интегральные схемы	Схема из логических элементов (электрическая принципиальная)	Булевы функции, конечные автоматы	Топология интегральных схем (языки графического типа)
3. Узлы и блоки	выполнение микроопераций	Интегральные схемы, конструктивы (плата, ТЭЗ), их фрагменты	Схемы из логических элементов и узлов (электрическая функциональная)	Языки регистровых передач (микроопераций)	Планы кристалла (языки графического типа)
4. Устройства	Выполнение операций (микропрограмм)	Интегральные схемы , платы, ТЭЗы, панели, стойки	Схемы из узлов и блоков (электрические структурные)	Языки регистровых передач, языки описания микропрограмм	Укрупненный план кристалла, чертежи конструктива
5. ЭВМ	Выполнение команд (программ)	Конструктивы всех уровней	Структурные схемы	Языки команд, языки программирования	Чертежи конструктива
6. Системы	Взаимодействие устройств, вычислительный процесс	Конструктивы всех уровней	Структурные схемы Для всех уровней : Языки структурного описания	Языки моделирования систем, сетевые модели	Чертежи конструктивов

Языки первой группы описывают, главным образом, соединения между элементами (узлами) структуры. Достаточно часто сами элементы структуры рассматриваются при этом как «черные ящики», имеющие входы и выходы. Причем в развитых языках этой группы обычно заложена возможность многоуровневого иерархического описания структуры, когда на более высоких уровнях элементами структуры являются подструктуры, описание которых построено раньше. Примерами языков этой группы могут служить языки СТРУКТУРА [1, 4], SDL, HSL [10] и другие.

Языки функционального описания описывают распространение сигнала, функцию элемента или блока, алгоритм функционирования, смену со

стояния. Если структурные описания отображают статические связи на уровне структуры, то функциональные описания представляют динамику работы устройств, последовательность развертывающихся в них процессов. В отличие от языков структурного описания, легко интегрирующих иерархическую структуру, описание функционирования на разных уровнях влечет за собой значительные различия в используемых моделях, а следовательно, и в языках представления.

На наиболее высоком уровне (системном или архитектурном) используется вероятностное описание взаимодействия различных устройств, например, модели теории массового обслуживания, сети Петри. Такому уровню соответствуют языки описания имитационных моделей типа GPSS, СИМСКРИТ, СИМУЛА [14] и другие.

Следующим является уровень программ и машинных команд, которому соответствуют языки программирования и языки машинных команд.

Еще более низким уровнем функционального описания является уровень микрооперации и микропрограмм, на котором функционирование описывается в терминах переходов регистров и триггеров из одного состояния в другое. Для этого используются языки представления микропрограмм типа языков АЛГОРИТМ [7], МИКРОПРОГРАММА [1] или языки регистровых передач CDL, DDL, HSL–FX и другие [10].

Иногда для представления АЛУ и других дискретных устройств используют языки, позволяющие описывать и структуру и функционирование одновременно, например язык ДИСТАЛ (ДИскретные СТруктуры и АЛгоритмы) [8]. Можно составлять такие описания и на алгоритмических языках типа APL и даже типа Паскаль и Си, однако такие описания могут быть не наглядными и громоздкими.

Наконец, для описания технической реализации вычислительных устройств используются графические языки. С помощью этих языков могут описываться маски и топология интегральных схем, изображаться фотошаблоны для изготовления печатных плат, задаваться конфигурации радиокомпонентов и микросхем, изображаться очертания печатных плат, размещение на них элементов, строиться символические изображения элементов и так далее. Следует учитывать, что собственно графических языков немного [8]: это, например, языки CIF и SMF. Более частым вариантом является построение и обработка файлов графических изображений с помощью специальных программных пакетов или графических программных средств, предусмотренных в той или иной системе автоматизированного проектирования.

Много примеров использования языков различных типов приведено в [8].

Глава 2. ОБЩИЙ ПОРЯДОК ПРОЕКТИРОВАНИЯ АЛУ НА ОСНОВЕ ФУНКЦИОНАЛЬНОСТРУКТУРНОГО ПОДХОДА

В данной главе рассматриваются общие вопросы проектирования структур АЛУ с точки зрения концепции функционально-структурного подхода и базирующегося на ней аппарата эволюционного синтеза систем [4]. В начале главы приводятся основные положения этой концепции. На ее основе разбираются этапы проектирования АЛУ, в соответствии с которыми формируется методика построения обобщенных структурных схем АЛУ. Эта методика иллюстрируется примером построения АЛУ для сложения чисел с плавающей запятой. В заключительной части главы рассматриваются преобразования структур АЛУ, позволяющие варьировать аппаратурные затраты на их реализацию, а такие возможные оценки эффективности получаемых структур.

2.1. Обобщенные функции систем переработки информации

Всю человеческую деятельность (не обсуждая пока вопрос об искусстве и подобных ее видах) можно рассматривать как процесс постановки человеком некоторых целей и выполнения соответствующих преобразований материального и идеального, ведущих к поставленным целям. Как отмечалось во введении, в настоящее время в различных исследованиях, посвященных вопросам проектирования сложных технических систем, в качестве объектов этих преобразований принято рассматривать вещество, энергию или информацию.

Во введении были названы также и четыре типа основных системных процессов: преобразования, храпения, передачи и управления. Понятно, что для систем переработки информации (СПИ) эти процессы в качестве своего объекта имеют, главным образом, информацию. Однако следует помнить, что всякое разделение является не абсолютным, и в любом преобразовании участвуют, в той или иной мере, все три названные категории. Ближайшим тому примером является то, что информация должна представляться каким-либо вещественным или энергетическим носителем. (Аналогично и преобразования вещества связаны с затратами или высвобождением энергии, а сама энергия, по-видимому, не существует в чистом, не зависимом от материи виде.)

Поэтому при проектировании СПИ вопросы организации подачи электропитания, отвода теплоты, экранирования от электромагнитного излучения (процессы, объектом которых является энергия), а также механических перемещений носителей информации и конструктивного оформления (объектом которых является вещество), непосредственно возникающие на этапе технического проектирования, не могут упускаться из виду полностью и на более ранних этапах создания систем. Тем не менее, в настоящем пособии все вни

мание сосредоточено преимущественно на информационной стороне протекающих в СПИ процессов.

Каждый из четырех названных типов процессов позволяет выделить различные аспекты функционирования СПИ и сформулировать соответствующую группу требований при разработке системы, совокупность которых и будет направлять процедуры формирования структуры систем. Так, процессы хранения (процессы М-типа), по сути своей, связаны с обеспечением «существования» в системе информационных объектов: констант, переменных, массивов, файлов и других, и определяют требования к составу, функциональным и техническим характеристикам средств запоминания информации всех уровней, начиная от триггеров состояния и кончая внешней памятью.

Процессы преобразования (процессы Р-типа) связаны с видоизменением существующих информационных объектов или построением новых и определяют требования к составу, функциональным возможностям и техническим характеристикам операционных элементов, блоков и устройств СПИ.

Процессы передачи (процессы Т-типа) связаны с обеспечением единства всей системы, состоящей из операционных и запоминающих узлов, позволяя перемещать информационные объекты к узлам их обработки и хранения, а также в управляющую часть (для анализа). Характер этих процессов определяет состав, функции и параметры линий связи всех рангов от простого соединения входов и выходов элементов до интерфейсов сетей ЭВМ.

Процессы управления (процессы С-типа) связаны с организацией всех остальных процессов и, по-существу, представляют собой некоторую обобщенную форму – отражение той совокупности процессов, которой они управляют. Именно поэтому процессы С типа реализуются на основе процессов М-, Р- и Т-типов более простого вида, чем те процессы, управление которыми они осуществляют. С- процессы определяют требования *к* функциональным возможностям, организации и алгоритмам управляющей части СПИ.

Использованный выше для определения человеческой деятельности термин «преобразование» является достаточно общим и может применяться для обозначения различных действий: арифметических операций, решения систем уравнений, превращения тепловой энергии в энергию механического движения, строительства дома и т. д. Поэтому для преобразования информации целесообразно несколько конкретизировать трактовку этого понятия. Такой ограниченной, но достаточно широкой (иногда считают даже слишком широкой) интерпретацией преобразований информационных объектов являются отображения множеств, в том числе множеств. с заданными на них отношениями. Рассматривая преобразования как отображения, будем обозначать их в общем случае символом F, подчеркивая их близость в такой трактовке к понятию функции.

Реализация преобразований в системе переработки информации осуществляется на основе взаимодействия узлов системы, которые можно рассматривать как элементы ее структуры. В первой главе отмечалось, что уровень такого рассмотрения может быть различен в зависимости от его целей и задач: от р – п переходов на кристалле БИС до целых систем ЭВМ. Однако при решении задач проектирования ЭВМ, связанных с разработкой структуры отдельных ее устройств, в частности арифметико-логического устройства, анализ структуры осуществляется, как правило, на уровне регистровых передач и обычно не опускается ниже уровня логических элементов.

При необходимости построить структуру, реализующую некоторое преобразование F, следует представить ее в виде связной совокупности известных структур (элементов). Для этого само преобразование F должно быть разбито (декомпозировано) на связную совокупность таких преобразований, выполнение которых возможно осуществить на известных структурах (элементах). Учитывая названный выше уровень рассмотрения структур устройств ЭВМ, к таким преобразованиям могут быть отнесены преобразования, перечисленные в табл. 2. В этой таблице каждому функциональному преобразованию поставлен в соответствие структурный элемент СПИ, реализующий это преобразование, а также указан его тип: к – комбинационный, н – накапливающий.

К функциям Т, М и С типов на рассматриваемом уровне регистровых передач можно отнести действия, представленные в табл. 3. В графе примечаний этой таблицы отражен тот факт, что разделение процессов и функций на Р, С, М и Т типы относительно, не абсолютно, как и любое разделение. Каждый тип процесса в той или иной мере сочетается с другими или даже может быть заменен ими. Например, преобразования невозможны без подачи информации на входы узла преобразования и съема результата с его выхода, условие для управления условным переходом часто приходится вычислять и т. д. Примером замены типов преобразований может служить табличное АЛУ, в котором Р процессы заменены на М-тип. Другим примером является дублирование вычислений значения одной и той же величины в разных блоках вместо вычисления ее в одном блоке с последующей передачей в другой. Такие взаимосвязи, в частности, обусловливают, наряду с иными причинами, и широкое разнообразие структур СПИ и неоднозначность проектных решений

Таблица 2 Основные функции Р-типа уровня регистровых передач

Преобразование	Структурный элемент	Тип элемента
Суммирование двух кодов/битов	сумматор	к/н
Добавление единицы (счет) Вычитание единицы	Счетчик Счетчик	к/н к/н
Сдвиг кода/бита	Сдвигающий регистр Сдвигатель	Н К

Сравнение кодов/битов (равенство, больше, меньше)	Схема сравнения Сумматор	К к/н
Сравнение с константой	Схема анализа содержимого	К
Логические функции: И, ИЛИ, исключающие ИЛИ, НЕ	Схема логических операций (сумматор)	к/н
Произвольная логическая функция	Комбинационная схема, ПЛМ	К
Шифрация (преобразование кода)	Шифратор	К
Дешифрация	Дешифратор	К

Таблица 3 Основные функции Т-, М- и С- типа уровня регистровых передач

Функция	Функциональный элемент, блок	Примечание
Т-тип		
Подача кода /бита из источника (узла , шины) на вход узла	Жесткая связь (выхода узла с источником) Связь (входа узла с источником) через блок вентилей, мультиплексор	Т+С
Подача фиксированного кода/бита на вход узла	Жесткая связь информационного входа узла с 0/1 Шина управления информационным/синхро входом узла	Т=С
Выдача кода/бита с выхода узла (в приемник : узел, шину)	Жесткая связь выхода узла с приемником Связь выхода узла с приемником через блок вентилей, мультиплексор	Т+С
М-тип		
Хранение кода/бита	Триггер, регистр, блок регистров, ЗУ	
Фиксация (запоминание) кода/бита	Триггер, регистр, блок регистров, ЗУ	М+С
Хранение фиксированного значения кода/бита	Управляющая шина	
С-тип		
Разрешение выполнения передачи, преобразования (синхронизация)	Управляющая шина (вход)	
Задание выполняемой функции	Управляющая шина Управляющий (кодовый вход)	С+Р (Дш)
Безусловный переход к следующему действию	Управляющий автомат (генератор синхроимпульсов)	
Условный переход к следующему действию	Управляющий автомат	С+Р

2.2. Общая методология и этапы проектирования АЛУ

В настоящее время существует большое разнообразие методов решения задач проектирования систем различного назначения. Среди них имеются как методы, предназначенные для решения частных задач, так и комплексные методики построения тех или иных систем.

Одной из наиболее полно охватывающих все аспекты проектирования методологий является эволюционный синтез систем, базирующийся на предложенном его автором функционально-структурном подходе к анализу и синтезу систем. Этот подход представляет собой совокупность концепций, объективных закономерностей развития, положений и выводов, определяющих стратегию анализа и синтеза [4].

Функционально-структурный подход рассматривает процесс проектирования как процесс перехода от функций разрабатываемой системы к ее структуре. Поэтому при проектировании СПИ, предназначенной для решения некоторых задач, т. е. для выполнения некоторых функциональных преобразований, такие можно говорить о переходе от функций системы переработки информации к ее структуре.

Характерными особенностями функционально-структурного подхода (ФСП) являются:

учет взаимосвязи функций и структуры объектов при определяющей роли функций по отношению к структуре;

целостность и общность подхода к анализу и синтезу многоуровневых систем;

учет вещественно-энергетических и информационных связей между элементами системы;

совместное использование общих законов материального мира и закономерностей развития систем определенного класса;

рассмотрение систем в развитии.

Исходной посылкой ФСП является единство функций и структуры системы при определяющей роли функции. Подход базируется на концепции интеграции и дифференциации функции в процессе развития – концепции многофункциональности и специализации элементов, подсистем и систем. Целостность подхода к анализу и синтезу систем обеспечивается формированием специальной модели – «дерева функций», представляющего собой декомпозицию целевой функции системы. В процессе многоуровневой декомпозиции формируются основные и дополнительные функции, реализуемые отдельными подсистемами и элементами, находящимися на различных уровнях системы. Важной отличительной особенностью подхода является совместный учет при анализе и синтезе вещественных, энергетических и информационных потоков внутри системы в процессе обмена с внешней средой.

Анализ развития систем с точки зрения функционально-структурного подхода приводит к следующим выводам:

1. Структура системы определяется совокупностью реализуемых функций.

2. Между реализуемыми функциями и структурой системы не существует взаимно однозначного соответствия. Один и тот же набор функций мотает быть реализован различными совокупностями многофункциональных и специализированных элементов, объединенных в систему.

3. При изменении условий существования системы происходит изменение спектра реализуемых системой функций, приводящее к соответствующим изменениям в структуре системы. Таким образом, осуществляется адаптация организации системы к изменяющимся условиям существования.

4. В процессе развития систем формируются различные классы и виды систем, функционально-структурная организация которых наиболее адекватна условиям существования.

Методология проектирования систем, базирующаяся на ФСП, получила название революционного синтеза систем (ЭСС). ЭСС является многофазным процессом, сводящимся к последовательному формированию и преобразованиям ряда моделей системы, соответствующих различным уровням ее организации [4]. Характерной особенностью ЗСС является широкое использование на этапах анализа систем – прототипов, формирования концепции системы и синтеза ее структуры следующих моделей:

дерева функций системы;
дерева противоречий системы;
многоуровневой структуры системы на основе функциональных модулей;
структуры системы на основе конструктивных модулей.

Основные фазы ЭСС включают по несколько этапов – процедур и сводятся к следующему:

1. Анализ систем прототипов (выявление функций систем, построение обобщенного дерева функций, определение базовых структур и анализ принципов реализации).
2. Исследование дерева противоречий системы (анализ «узких мест» прототипов, ограничивающих факторов, выявление основного противоречия, построение дерева противоречий и его анализ).
3. Формирование концепции системы (выявление способов преодоления противоречий системы, поиск альтернативы се реализации, разработка технического задания, определение совокупности показателей качества и критерия оценки системы) .
4. Формирование дерева функций системы (определение множества основных и дополнительных функции, определение числа уровней деком

позиции, декомпозиция функций системы, выделение типовых операторов, отображение функций на множество операторов).

5. Формирование функциональной структуры системы (анализ методов аппаратной и программной реализации, разработка алгоритмов функционирования системы, анализ связей между операторами различных уровней, построение диаграмм активности операторов, определение загрузки ресурсов, формирование функциональных модулей и их связей, выделение типовых функциональных подсистем).
6. Формирование технической структуры системы (выбор технических средств для реализации системы, формирование таблиц соответствия функциональных и конструктивных модулей, выявление необходимости разработки оригинальных средств, преобразование элементов функциональной структуры, покрытие функциональных подсистем конструктивными модулями, формирование конструктивных модулей высокого уровня).
7. Оценка показателей качества и выбор окончательного варианта системы по сформированному критерию.

Таким образом, формирование структуры системы осуществляются на основе принципов декомпозиции и композиции функций и структур различных подсистем, а преобразование моделей системы производится в такой последовательности:

1) формирование дерева функций системы;
2) декомпозиция функций системы до уровня принятого базового набора операторов (преобразования 1 и 2 выполняются итеративно);
3) формирование функциональной структуры системы;
4) формирование технической структуры системы.

Известны и несколько иные варианты выделения и именования этапов процесса проектирования. Так, например , приведенные выше этапы 1...3 соответствуют этапу системного проектирования [12], этапы 4 и 5 – алгоритмическому и логическому этапам проектирования [15] и т. д.

При рассмотрении определенного класса систем или подсистем, как это имеет место в настоящем пособии, посвященном арифметическо-логическим устройствам, оказывается возможным несколько ограничить и уточнить содержание ряда названных этапов. Кроме того, многие из входящих в них процедур могут быть проведены заранее, безотносительно к конкретной системе, и в дальнейшем использоваться при проектировании различных экземпляров систем данного класса. Это, например, справедливо по отношению к этапу анализа систем-прототипов, на котором выявляются функции системы и строится ее обобщенное дерево функций. Поскольку дерево функций системы определенного класса в обобщенном варианте не должно включать функций, представленных в виде конкретных действий над заданными объектами, то такое обобщенное дерево функций АЛУ может быть построено заранее и использоваться при разработке АЛУ для различных систем переработки информации. Фрагмент одного из возможных вариантов обобщенного дерева функций АЛУ представлен на рис. 3, где в качестве функций различных уровней выделены следующие:

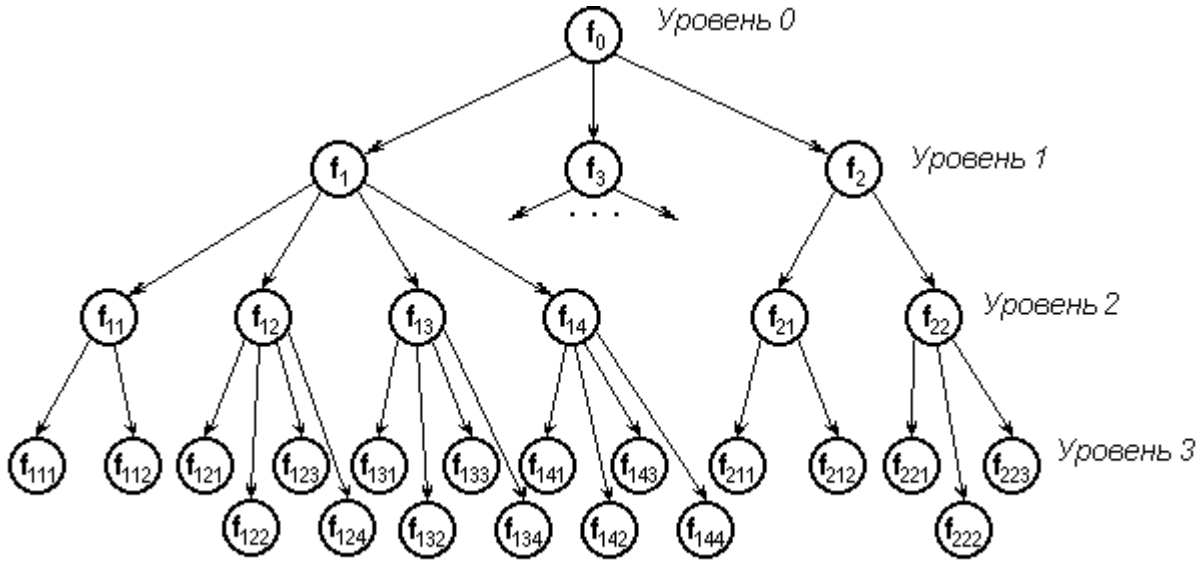

Рис. 3. Вариант обобщенного дерева функций АЛУ

Уровень 0

f_0 – выполнение заданной совокупности преобразований;

Уровень 1

f_1 – формирование результатов преобразований (основная функция);

f_2 – обеспечение взаимодействия с другими устройствами (дополнительная функция);

f_3 – выполнение контроля преобразований (дополнительная функция);

Уровень 2

f_{11} – обмен операндами и результатами внутри АЛУ;

f_{12} – хранение операндов и результатов;
f_{13} – преобразование операндов:
f_{14} – управление формированием результатов;
f_{21} – обмен данными с другими устройствами:
f_{22} – обмен управляющей информацией с другими устройствами;

Уровень 3

f_{111} – подача (промежуточных) операндов на блоки преобразований;
f_{112} – прием (промежуточных) результатов с блоков преобразований;

f_{121} – хранение операндов;
f_{122} – хранение промежуточных результатов;
f_{123} – хранение результатов;
f_{124} – хранение признаков результатов;

f_{131} – преобразование кодов;
f_{132} – сдвиг чисел;
f_{133} – арифметические преобразования;
f_{134} – логические преобразования;

f_{141} – формирование управляющих сигналов для настройки блоков преобразования;
f_{142} – формирование управляющих сигналов для передачи данных между блоками;
f_{143} – формирование признаков результатов;
f_{144} – формирование признаков занятости | готовности АЛУ;

f_{211} – прием операндов;
f_{212} – выдача результатов;

f_{221} – прием кода операции;
f_{222} – выдача признака занятости]готовности АЛУ;
f_{223} – выдача признаков результата.

(На рис. 3 не показана декомпозиция по уровням функции f_3– контроль результатов преобразований, подобная декомпозиция основной функции f_1).

Необходимо учитывать, что разбиение на функции различных уровней, как и формулировка содержания самих функций могут быть различны. Некоторые функции, например, формирование признаков результата, могут относиться к различным функциям более высокого уровня. Поэтому возможно построение различных вариантов обобщенных деревьев функций АЛУ, однако содержательный смысл входящих в них функций, в общем, будет одинаковым.

Аналогично для этапа исследования дерева противоречий может быть заранее выполнен анализ «узких мест» прототипов, построение и анализ самого дерева противоречий. Причем оно будет включать противоречия между основными показателями, которыми характеризуются функции различных уровней дерева функций системы.

При формировании концепции системы используют общие принципы разрешения противоречий, которые будут рассмотрены далее.

На шестом этапе формирование таблиц соответствия конструктивных и функциональных модулей также может осуществляться, в основном, заранее, особенно в случае автоматизированного проектирования. Однако эти таблицы обязательно будут пополняться впоследствии.

Кроме того, особенности относительно узкого класса преобразований, возлагаемых на АЛУ (арифметические и логические операции), обусловливают и некоторые изменения содержания отдельных этапов. В частности, на этапе формирования концепции системы принимаются решения о выборе табличной или алгоритмической реализации преобразований в АЛУ, о необходимости повышения надежности за счет включения в реализуемые функции функций контроля, о максимально возможном распараллеливании выполняемых действий или, наоборот, о максимальном совмещении функций в различных блоках и т. д. Последнее в значительной мере определяет порядок проведения следующих этапов разработки.

Необходимо также указать, что этапы построения дерева функций и формирования функциональной и технической структуры можно выполнять итеративно, уточняя уровень декомпозиции функций по мере формирования структур. Причем непосредственное получение оптимальных по заданному критерию структурных решений не всегда возможно. Поэтому наиболее целесообразно формировать структуру первоначально с максимальным распределением функций, которую при необходимости затем преобразовать в структуру с желаемой степенью совмещения функций. Такой путь и будет выбран в качестве базового в дальнейшем изложении. Однако прежде следует рассмотреть особенности и возможные процедуры декомпозиции функций при построении дерева функций АЛУ.

2.2.1. Декомпозиции функций АЛУ

При формировании дерева функций системы на основе анализа систем-прототипов в соответствии с требованиями технического задания определяется множество основных и дополнительных функций системы. Множество основных функций определяет [4] класс системы. Множество дополнительных функций определяет такие качества системы, как надежность, живучесть, сервис, представляемый пользователям, и т. д. Изменение множества дополнительных функций порождает различные варианты систем данного

класса, определяемого множеством основных функций. Дерево функций системы формируется затем в результате декомпозиции полученных множеств основных и дополнительных функций.

Декомпозиция функции f_i предполагает разделение ее на некоторые составляющие f_{i1}, ...,f_{in}, совместное (одновременное или поочередное) выполнение которых дает в совокупности реализацию исходной функции f_i. Для определения способов декомпозиции функций необходимо найти компоненты, входящие в описание функций, выявить структуру, типы и окружение этих компонентов.

В общем случае функции представляют собой действия (воздействия) над некоторыми объектами. Такие объекты для удобства ссылки будут далее называться актантами. Поэтому в описание функции обязательно входят собственно действия (воздействие) и актант. Кроме этого, в описании может учитываться и объект, осуществляющий действие, – актор, и объект, через посредство которого осуществляется действие – средство или инструмент. Наконец, может учитываться и внешнее окружение для функции. В качестве такового можно рассматривать как окружение актанта (и актора со средством), т.е. среду, так и внешнее воздействие, т.е. как бы окружение самого действия, активность внешней среды. Таким образом, описание функций может иметь следующий общий вид:

Действие, Актант [, Актор] [, Средство] [, Среда] [, Активность].

Декомпозиция представленной такими составляющими функции может производиться по каждой из них (по типу, структуре или другим атрибутам). Однако наиболее часто разложение производится по действию и по актанту.

Собственно действие может быть подвергнуто декомпозиции в следующих отношениях:

по фазам: инициирование, увеличение (нарастание), поддержание, уменьшение (ослабление), терминация;

по типам: преобразование, хранение, передача, управление;

по классу действия в рамках более широкого множества классов действий;

по частям среды развертывания действия, в качестве которой могут выступать время, пространство, структура и др.

Примерами подобных способов декомпозиции могут служить:

а) передача байта по последовательному интерфейсу, передача стартовых и стоповых битов начальной и конечной фаз или выполнение преобразования, включающее подачу операндов на входы блока и прием результата с выхода блока преобразования как начальную и конечную фазы;

б) формирование суммы *n* слагаемых, реализуемое поочередным добавлением слагаемых к сохраняемой сумме посредством подачи их на вход суммирующего блока и управление этим процессом;

в) разделение выполняемых в АЛУ операций преобразования на арифметические и логические;

г) хранение кода в течение некоторого времени может быть разделено на последовательно повторяющиеся интервалы хранения, как в полупроводниковых динамических ОЗУ.

Декомпозиция функции по объекту действия состоит обычно в разделении действия по элементам структуры этого объекта. Например, выполнение логических операций над длинными операндами может осуществляться поразрядно или побайтно. Другой возможностью является декомпозиция по разделению класса объектов действия на некоторые группы. Например, арифметические операции над числами, представленными в прямых кодах, и операции над числами, представленными в дополнительных кодах, можно рассматривать как декомпозицию функции выполнения арифметических операций.

Аналогичным образом функции могут декомпозироваться и по другим составляющим, входящим в их описание.

2.2.2. Способы преодоления противоречий при формировании концепции системы

Как отмечено выше, на этапе формирования концепции системы производится выявление способов преодоления противоречий проектируемой системы, сводящихся, в конечном счете, к ухудшению одних показателей системы при попытке улучшения других ее показателей. Следует учитывать, что поиск концептуальных решений является одной из наиболее сложных задач проектирования, рекомендации по разрешению которой могут носить лишь общий характер. При разработке системы возможны два основных пути преодоления имеющихся или возникающих противоречий:

выбор решения, обеспечивающего оптимальное по заданному критерию соотношение между сторонами рассматриваемого противоречия (баланс показателей);

частичная или полная ликвидация (снятие) рассматриваемого противоречия за счет перехода к другому способу или принципу реализации.

Первый путь связан с изменением, как правило, количественных характеристик разрабатываемой системы. Причем эти изменения могут затрагивать как параметры элементов системы, так и количественные характеристики ее структуры. Типичными примерами такого пути могут служить: а) использование более быстродействующих, менее энергоемких или более деше

вых элементов; б) различные структурные изменения, связанные с переходом от параллельной реализации функций к последовательной, или наоборот; в) замена аппаратной реализации программной (микропрограммной), и наоборот, и *т.* п.

Для второго пути решения обычно связаны с качественным изменением принципов организации структуры системы или реализации ее функции. Задачи поиска концепции системы рассматриваемым путем часто относят к изобретательским, интеллектуальным. В этом случае применяются различные приемы, систематизация и анализ которых проводится в ряде работ [3, 4, 8]. Однако построение каких-либо формальных рекомендаций по применению этих приемов довольно затруднительно. Некоторые попытки такого рода реализуются в виде экспертных систем [8].

К числу эффективных приемов, используемых на данном пути, могут быть отнесены совмещение функций (принцип многофункциональности), замена типа функций (преобразование, хранение, передача), инверсия функций, построение инвариантных систем, композиция качеств и другие. Красивым примером совмещения функций является устранение с помощью этого принципа противоречия между стоимостью точного (с малым допуском) изготовления и быстродействием интегральных схем в технологии с самосовмещением [11], в которой сам затвор транзистора используется как маска при создании областей истока и стока. Замена типа функций хорошо иллюстрируется на примере табличной реализации АЛУ, в котором, по сравнению с обычным АЛУ, преобразование операндов заменяется хранением результата. Как всегда, конечно, следует учитывать, что существуют и такие приемы, которые можно рассматривать в качестве представителей как первого, так и второго путей.

2.2.3. Формирование таблиц соответствия функций, функциональных и конструктивных моделей

При формировании функциональной и технической структуры системы на соответствующих (пятом и шестом) этапах производится покрытие функций, образующих дерево функций этой системы, функциональными и конструктивными модулями. Для проведения таких процедур необходимо располагать: а) библиотекой отображений (таблиц) функций на обеспечивающие их выполнение функциональные и конструктивные модули (с возможным различением способов их реализации); б) способом выбора совокупности модулей, реализующих требуемую совокупность функций, так как обычно имеется некоторое множество таких совокупностей модулей; в) критерием оценки предпочтительности того или иного варианта. Таблицы соответствия первоначально формируются на основании анализа деревьев функций и реализации типовых систем рассматриваемого класса. В них для каждой функции указываются:

- имя функции и ее объекта (актанта);

- перечень возможных ее параметров, относимых к различным составляющим описания функции (параметрам с помощью ссылок может быть придан вид дерева, а сам их перечень можно представить отдельными файлами);

- имя функционального блока, реализующего функцию, при необходимости дополняемое особенностями блока;

- способ реализации функций функциональным блоком, который в зависимости от уровня (сложности) функции задается либо указанием функций более низкого уровня, либо указанием структурных особенностей блока;

- имя файла конструктивных блоков, реализующих функцию, если таковые имеются. Для каждого из них должны быть указаны значения параметров, в число которых входят (означиваются) и параметры, указанные в перечне возможных параметров функций.

Перечни параметров функций, функциональных и конструктивных блоков зависят от их типов и могут быть представлены отдельными информационными структурами.

Необходимо также учитывать, что в общем случае в качестве функционального блока может рассматриваться как аппаратный, так и программный продукт без указания, каким именно типом продукта блок представлен. В роли конструктивного блока может быть представлено конкретное аппаратное, программное или аппаратно-программное средство. Возможный вид записи в библиотеке отображений (базе данных) представлен в табл. 4.

Создание библиотеки отображений является одной из наиболее трудоемких и ответственных задач построения систем проектирования АЛУ и требует детальной проработки большого количества схемных и структурных решений. При ручном проектировании сведения о способах реализации функций представляют собой личный опыт разработчика.

Перед построением библиотеки отображений необходимо тщательно определить предметную область, объем которой для небольшой системы структурного проектирования АЛУ составляет порядка нескольких сот понятий, разбиваемых на классы: объект, определенность, действие (в т.ч., отношение).

2.3. Формирование функциональной структуры АЛУ

После определения концепции реализации АЛУ и построения его дерева функций (не обязательно для всех уровней сразу) на соответствующих этапах начинается этап формирования функциональной структуры АЛУ. На этом этапе выбирается алгоритм реализации требуемых функции устройства и осуществляется переход от операторов (блоков) алгоритма к функциональ

ным блокам АЛУ. Причем, как отмечалось выше, выполнение этого этапа обычно сопровождается итеративными возвратами к этапу формирования дерева функций для выделения функций более низкого уровня и соответствующих им операторов в выбранном алгоритме с целью определения способов реализации операторов функциональными (а впоследствии и конструктивными) блоками.

В соответствии со стратегией максимального распределения функций по функциональным блокам (см. с. 29) переход от операторов алгоритма к функциональным блокам осуществляется следующими процедурами:

1. По аналогии с правилами структурного программирования, алгоритм разделяется на участки или блоки, имеющие один вход и один выход (при этом последовательные блоки, входящие в более крупный линейный участок, сперва не выделяются, т.е., разбиение первоначально производится на самые крупные участки).

2. Для каждого выделенного участка алгоритма (УчА) определяются множество *1n* входных и множество *Ot* выходных информационных объектов (ИФО). При этом в множество *Ot* выходных ИФО включаются все переменные, значения которых вычисляются операторами данного участка алгоритма, а в множество *In* входных ИФО включают все остальные переменные. Причем для участков, содержащих условные операторы (т. е. для участков алгоритма типа *if-then*, *if-then-else*), к входным ИФО относят также и переменные, определяющие значения условий операторов *if*.

3. Каждому выделенному участку алгоритма ставится в соответствие обобщенный функциональный блок (ОФБ) создаваемой функциональной структуры устройства. Количество входов и выходов таких блоков берется равным количеству информационных объектов соответствующих представляемому УчА множества *1n и Ot*.

4. Производится анализ наличия в библиотеке отображений типовых функциональных блоков, обеспечивающих реализацию функций, выполняемых операторами тех участков алгоритмов, которым соответствуют введенные обобщенные функциональные блоки.

Таблица 4. Фрагмент возможного варианта библиотеки (таблиц) отображений

	Параметры								
Функция	действия	актанта	актора	средства	Среды		Функцио-нальный блок	Способ реали-зации	Конструк-тивный блок
					окру-жения	действия			
Суммирование двоичных кодов	Время Вероятность ошибки	Разрядность					Сумматор	Параллельный Последователь-ный Табличный	Имя файла конструк-тивных реализаций сумматора
Передача двоичных кодов	Время Расстояние	Разрядность		Вид связи		Помехи	Блок вен-тилей Мультип-лексор Модем		Имя файла конструк-тивных блоков
Распознава-ние вызова (в сети ЭВМ)	Время Вероятность пропуска, ложного вызова	Структура кода вызова Частота пере-дачи кода		Тип линии передачи Волновое сопротив-ление		Амплитуда помех Величины внешних полей	Селектор Сетевой контроллер	Обнаружение посылки Прием посылки Сравнение с эталоном Реакция “да” Реакция “нет”	>> >>

Если такие типовые блоки имеются в библиотеке, то ОФБ помечаются именем типового блока с указанием значений его структурных (не физических) параметров, например количества и разрядности, входов и выходов блока. Эти значения определяются характеристиками информационных объектов блока.

Если типового блока, обеспечивающего выполнение требуемых функций, в библиотеке нет, то рассматриваемый ОФБ не может быть реализован непосредственно. Поэтому тот участок алгоритма, которому соответствует ОФБ, должен быть подвергнут дальнейшему разделению, что производится согласно пункту 1.

Выполнение рассматриваемых процедур завершается тогда, когда для всех обобщенных функциональных блоков будут найдены соответствующие им типовые функциональные блоки. Поэтому видно, что процедура перехода к функциональной структуре в целом носит итеративный характер.

Результатом выполнения перечисленных процедур является обобщенная структурная схема (ОСС), функциональные блоки которой соответствуют участкам исходного алгоритма и связаны между собой по входам и выходам в соответствии с используемыми и формируемыми информационными объектами. Такая схема имеет максимально распараллеленный для исходного алгоритма вид.

Определенные особенности имеет выполнение данных процедур для случая циклических участков алгоритма. В различных вариантах организации цикла может быть сформировано разное количество информационных объектов. Число их мотает быть как постоянным, так и переменным. В последнем случае обычно известно максимальное количество получаемых ИФО. Как правило, цикл вводится в алгоритм при ориентации его на последовательный характер преобразований и может быть всегда представлен параллельными ветвями (однотипными функциональными блоками), соответствующими всем получаемым в цикле выходным информационным объектам. Конечно, полное распараллеливание цикла приводит к получению весьма громоздких структур, но на рассматриваемом этапе это не должно сразу приводить к исключению подобных вариантов из дальнейшего анализа.

Рассмотрим выполнение процедур на примере формирования функциональной структуры АЛУ для сложения чисел с плавающей запятой.

Пусть первоначально алгоритм выполнения этой операции представлен в достаточно общем виде, показанном на рис. 4.

В соответствии с процедурой пункта 1, приведенного выше, алгоритм разбивается на два участка, первый из которых состоит из блока *1,* а второй – из всех остальных блоков и имеет тип *if-then-else*.

Далее, согласно пункту 2 определяются входные и выходные информационные объекты для выделенных участков алгоритма. Пусть операндами рассматриваемой операции являются два числа X и Y, представленные в форме с плавающей запятой, в которой каждое из чисел изображается мантиссой M и порядком P, т. е. $X=<M_x, P_x>$, $Y=<M_y, P_y>$. Тогда для первого участка алгоритма множество входных информационных объектов In_1 будет состоять из порядков P_x и P_y, а множество Ot_1 выходных ИФО этого УчА будет включать разность $P_x - P_y$ и объект, указывающий на то, какой из порядков P_x или P_y больше и поэтому является предварительным порядком результата. В роли такого информационного объекта – указателя может выступать специальный признак или просто знак разности $P_x - P_y$.

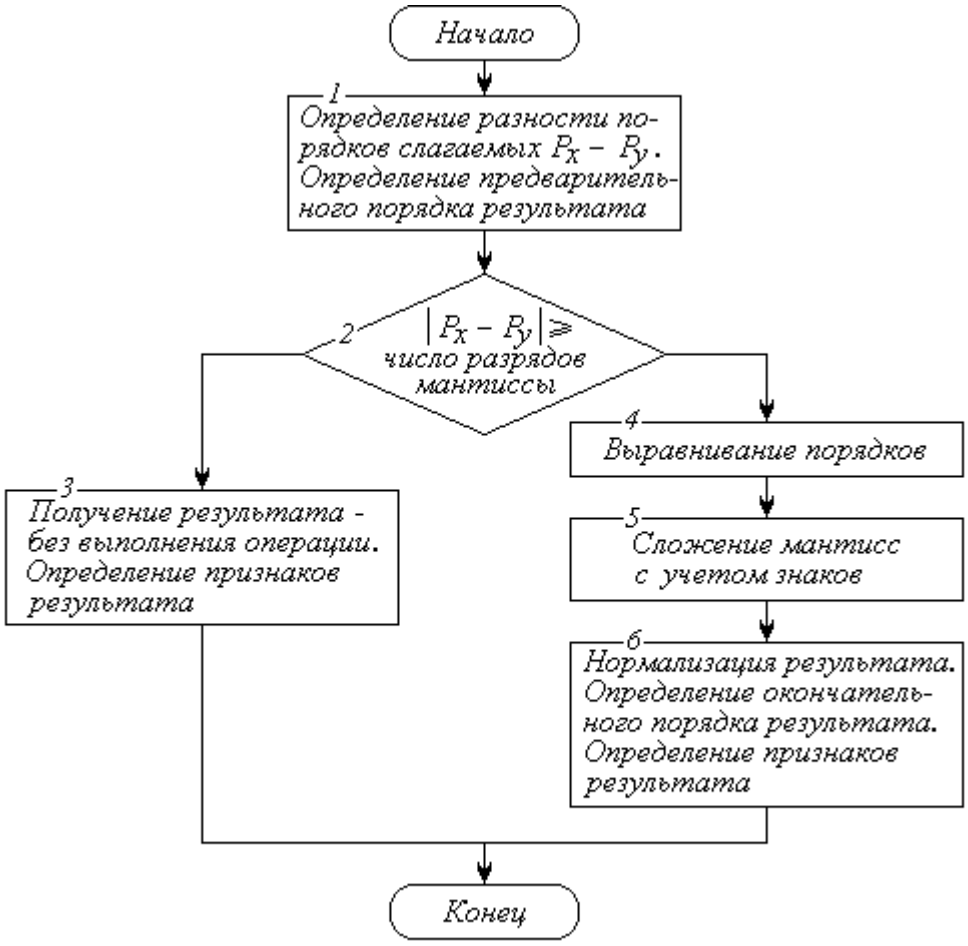

Рис. 4. Алгоритм сложения чисел с плавающей запятой

Для второго УчА множество $1n_2$ входных ИФО включает дополнительно к выходным ИФО первого блока сами мантиссы M_x и M_y и порядки P_x и P_y операндов, а множество $0t_2$ выходных информационных объектов состоит из мантиссы M_z, *и* порядка P_z, числа – результата Z (суммы) и признаков результата F_z. (К признакам обычно относятся признак переполнения *OVF*, признак потери значимости *SLS*, признак исчезновения порядка *ODA*, признак нулевого результата *N*.)

После этого, в соответствии с процедурой пункта 3, приведенного выше, строится обобщенная структурная схема (функциональная структура) АЛУ, в которой каждому УчА поставлен в соответствие обобщенный функциональный блок (ОФБ), реализующий функции этого УчА. На этой схеме для каждого такого ОФБ входы и выходы помечаются соответствующими информационными объектами.

Получающаяся в рассматриваемом случае структурная схема АЛУ представлена на рис. 5, где P_z^0 – предварительный порядок результата (равный большему из порядков слагаемых).

Рис.5. Обобщенная структурная схема (функциональная структура) устройства, реализующего алгоритм сложения чисел с плавающей запятой, при разбивке его на два участка

Выполнение следующей процедуры (пункт 4) связано с анализом наличия в библиотеке отображений типовых блоков, соответствующих обобщенным функциональным блокам построенной схемы. Поскольку состав библиотеки типовых блоков пока не уточнен, то в рассматриваемом случае можно принять за типовые блоки стандартные узлы ЭВМ, перечисленные в табл. 2.

В ходе сравнения полученных ОФБ с типовыми блоками (ТБ) можно установить, что первый ОФБ, определяющий разность двух целых чисел со знаками (порядков) и большее из этих чисел, реализуется на основе типового блока – сумматора (пока не уточняется, как учитываются знаки и выбирается большее из чисел; такое уточнение будет сделано в другой части примера). Однако второй ОФБ полученной схемы одним типовым блоком явно реализован быть не может хотя бы потому, что в нем последовательно выполняются различные функции. Конечно, в рассуждении относительно первого ОФБ также имеется определенная условность, поскольку и этот блок, кроме вычитания, осуществляет еще и выдачу большего числа, т.е. мультиплексирование.

Поэтому для второго УчА необходимо снова провести все рассмотренные процедуры, как для отдельного алгоритма. При этом начальный блок *if* данного участка следует выделить в самостоятельный блок (участок), так гак иначе правила структурирования не позволят провести дальнейшее разбиение. Такое выделение блока if вызовет появление на структурной схеме самостоятельных параллельных блоков, соответствующих обеим ветвям разделяемого участка. В свою очередь это потребует включения в схему дополнительного мультиплексора в конце данного участка для выбора результата работы одного из параллельных блоков схемы.

После разбиения второго УчА исходного алгоритма будет получено пять новых УчА, соответствующих блокам 2...6 исходного алгоритма. Сопоставление им обобщенных функциональных блоков приводит к получению обобщенной структурной схемы, представленной на рис. 6. На этом рисунке приняты следующие обозначения: G_i – указатель числа, имеющего больший порядок; M_x' и M_y' – мантиссы слагаемых после выполнения процедуры выравнивания порядков; M_z' – ненормализованная мантисса суммы; B_i – значение логического условия, устанавливаемое по результату сравнения модуля разности порядков с числом разрядов мантиссы; M_z^1, P_z^1, F_z^1 и M_z^2, P_z^2, F_z^2 – мантиссы, порядки и признаки результата, получаемые в блоках правой и левой ветвей схемы. Кроме того, на рис. 6 шестому УчА (блоку алгоритма) поставлено в соответствие сразу три блока $ОФБ_{6.1}$, $ОФБ_{6.2}$ и $ОФБ_{6.3}$, объединенных пунктирно в общий блок $ОФБ_1$ о причинах чего говорится ниже. В схеме также имеется дополнительный блок мультиплексирования $ОФБ_7$ для выдачи на выход устройства результата из соответствующей части схемы, поскольку в ней имеются две параллельных ветви, формирующих результат различным способом, что отмечалось выше.

Блок $ОФБ_1$ по сравнению с первым блоком на рис. 5 имеет дополнительный выход, указывающий, какой из порядков больше, точнее, какому слагаемому *X* или *Y* принадлежит 6больший порядок. Информация эта необходима для выбора соответствующей мантиссы в блоке $ОФБ_3$, формирующем результат без выполнения операции: полагая его равным слагаемому с большим порядком. Информационный объект на данном выходе обозначен через 6; и может представлять собой как номер слагаемого с большим порядком, так и просто логическую переменную, принимающую, например, значение 1, если больше порядок слагаемого *X*, и нуль – в противном случае. В качестве G_i удобно использовать знак разности порядков.

На выходе блока $ОФБ_2$ формируется информационный объект B_i представляющий собой логическую переменную, определяющую, какая из ветвей алгоритма должна быть реализована. Эта переменная, вырабатываемая схемой сравнения, принимает, например, единичное значение, если модуль разности $P_x - P_y$ превосходит по величине число разрядов мантиссы и результат просто равен слагаемому с большим порядком, и нулю – в противном случае.

Однако переменная B_i поступает не на вход блока ОФБ$_3$, а на вход мультиплексора ОФБ$_7$, в который также поступают результаты с выходов блоков, реализующих обе ветви алгоритма.

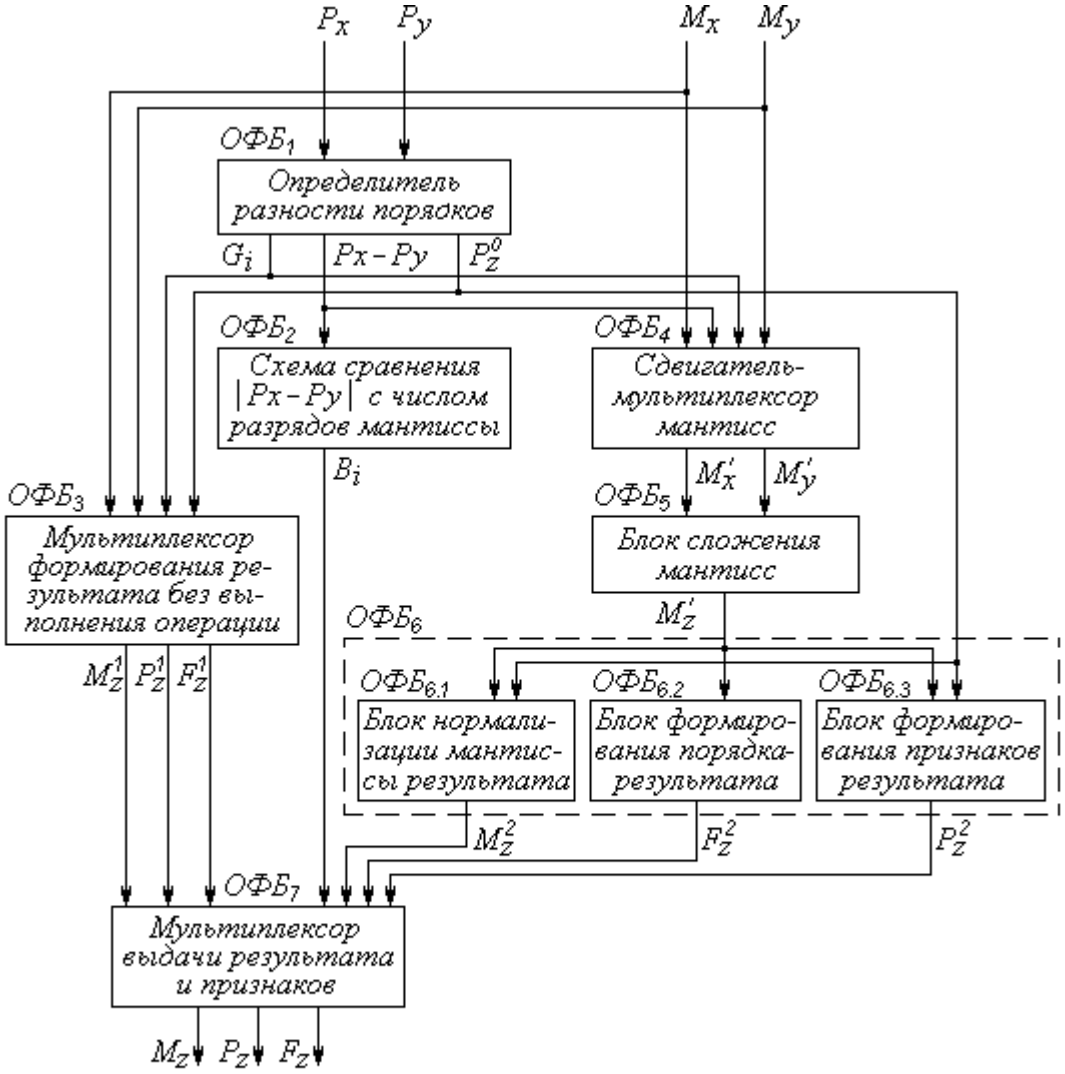

Рис. 6. Обобщенная структурная схема устройства, реализующего алгоритм сложения с плавающей запятой (рис. 4), при разбивке алгоритма на шесть участков (блоков)

В блоке ОФБ$_3$, помимо выбора мантиссы слагаемого, имеющего больший порядок, что осуществляется мультиплексором, производится также формирование признаков результата, равного этому слагаемому.

На выходах блока ОФБ$_4$ формируются информационные объекты M_x' и M_y', соответствующие мантиссам слагаемых после выравнивания порядков. При этом мантисса числа с большим порядком проходит через блок без изменений, а вторая мантисса сдвигается вправо на число разрядов, равное разности порядков. Таким образом в блоке ОФБ$_4$ производится сдвиг и мультиплексирование.

Блок $ОФБ_5$ должен выполнять сложение двух мантисс с учетом их знаков. По реализуемой функции этот блок близок к блоку $ОФБ_1$ и строится на базе сумматора.

Блок $ОФБ_6$ представлен на рис. 6 разделенным на три функциональных блока, поскольку действия (функции), перечисленные в соответствующем ему блоке 6 исходного алгоритма, достаточно сложны и обычными типовыми блоками не реализуются. Поэтому данный блок алгоритма должен быть представлен в какой-либо иной форме, более пригодной для реализации типовыми блоками.

Как отмечалось выше, задача такого представления относится к декомпозиции дерева функций системы. Однако, во-первых, в рассматриваемом случае в блоке алгоритма задана совокупность трех различных функций, для выполнения которых достаточно полученных ранее информационных объектов. Во-вторых, не был принят критерий оценки разрабатываемой системы, а используется стратегия максимального распараллеливания функций, оставляющая возможности для оптимизации структуры на последующих этапах. Поэтому блок 6 алгоритма представлен в виде трех параллельно выполняемых функций и ему сопоставлены три отдельных ОФБ: $ОФБ_{6.1}$, $ОФБ_{6.2}$ и $ОФБ_{6.3}$ (Возможна организация некоторых информационных связей между этими блоками, например, из $ОФБ_{6.1}$ в $ОФБ_{6.2}$ может поступать приращение порядка, определяемое в процессе нормализации мантиссы, на которое следует изменить предварительный порядок результата для получения окончательного его значения).

В построенную обобщенную структурную схему АЛУ входят функциональные блоки, в основе каждого из которых лежит типовой блок (сумматор, сдвигатель, мультиплексор). Но в их состав входят такие и дополнительные элементы и узлы. Поэтому такие блоки могут считаться типовыми лишь в том случае, если их проектирование уже было выполнено ранее и они введены в состав библиотеки типовых блоков. Если же принять за типовые блоки только те базовые узлы ЭВМ, которые перечислены в табл. 2, то точного соответствия между ОФБ построенной схемы АЛУ и типовыми блоками установить нельзя, так как функции этих ОФБ несколько сложнее, чем функции, реализуемые типовыми блоками. Поэтому для получения схемы, состоящей из типовых блоков, необходимо провести декомпозицию функции, реализуемых блоками исходного алгоритма до более простых функций, выполнение которых обеспечивается базовыми узлами.

Осуществление такого разбиения функций рассмотрим на примере блока $ОФБ_5$, так как проведение этого анализа для всех блоков весьма трудоемко. Кроме того, блок $ОФБ_5$, во-первых, близок по реализуемой функции к блоку $ОФБ_1$, а во-вторых, совпадает по выполняемому преобразованию с операцией сложения с фиксированной запятой.

Блок 5 исходного алгоритма задает операцию алгебраического сложения мантисс. Декомпозиция этой операции может осуществляться несколькими различными способами. Поскольку специальных критериев выбора способа декомпозиции здесь не используется, то за основу целесоо5разно взять алгоритм сложения чисел, представленных в прямом коде. Функции такого алгоритма (операции) можно распределить по трем его блокам так, как показано на рис. 6. Соответствующая этому алгоритму обобщенная структурная схема устройства, переход к которой производится по тем же правилам, представлена на рис. 7. Причем функциям блоков 5.1 и 5.3 алгоритма поставлены в соответствие отдельные (по два на каждый) функциональные блоки (рис. 8).

Рис. 7. Алгоритм алгебраического сложения мантисс

В этой схеме входными информационными объектами (ИФО) для блоков ОФБ$_{5.11}$ и ОФБ$_{5.1.2}$ являются мантиссы M_x. и M_y (или просто числа с фиксированной запятой) слагаемых, состоящие из знака S и цифр D каждая, т.е., M_x=<S_x, D_x>, M_y=<S_y, D_y>. Выходными ИФО данных блоков являются цифры D_x' и D_y' обратных кодов мантисс, которые вместе со знаками образуют собой входные ИФО блока ОФБ$_{5.2}$, представляющего собой сумматор. Причем блоки ОФБ$_{5.1.1}$ и ОФБ$_{5.1.2}$ образуют параллельные цепи схемы в соответствии с принятой стратегией максимального распределения функций алгоритма по блокам схемы. На выходе блока ОФБ$_{5.2}$ формируются ИФО: знак S_z, и цифры D_z' обратного кода M_z' мантиссы суммы, которые преобразуются блоком ОФБ$_{5.3.1}$, полностью аналогичным блокам ОФБ$_{5.1.1}$ и ОФБ$_{5.1.1}$ в прямой код мантиссы суммы M_z= <S_z, D_z>.

Кроме того, выходными ИФО блока ОФБ$_{5.2}$ являются н сигналы переноса C_0 и C_1 из знакового и старшего цифрового разрядов сумматора, поступающие в блок ОФБ$_{5.3.2}$ для определения значения признака переполнения

OVF, являющегося выходным информационным объектом данного блока. (Следует отметить, что значение признака переполнения может определяться и иначе, например, по значениям основного S_z, и дополнительного S_z^{∂} знаковых разрядов суммы, что показано на рис. 8 пунктирными связями, либо по трем знакам: слагаемых и результата.)

Полученная функциональная структура (обобщенная структурная схема) устройства содержит типовые блоки, реализующие простейшие для структурного представления АЛУ микрооперации: суммирование, получение обратных и прямых кодов, логическую операцию «исключающее ИЛИ» (для вычисления значения признака переполнения *OVF*).

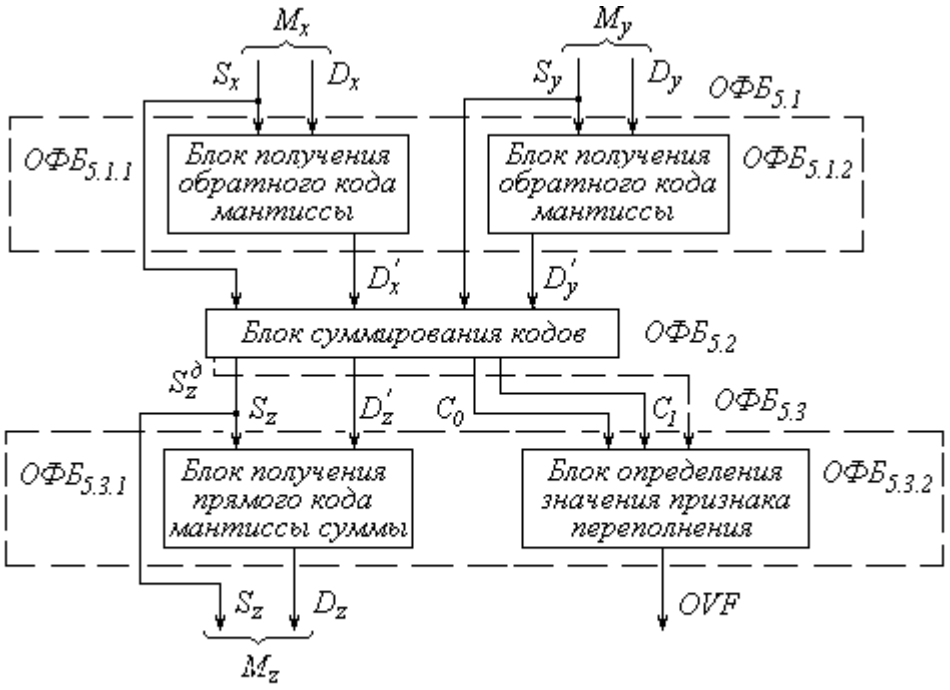

Рис. 8. Обобщенная структурная схема устройства, реализующего алгоритм алгебраического сложения мантисс

Замена обобщенных функциональных блоков полученной схемы на типовые функциональные блоки дает структурную схему, представленную на рис. 9. На этой схеме обобщенные функциональные блоки получения обратных и прямых кодов $ОФБ_{5.1.1}$ $ОФБ_{5.1.2}$ и $ОФБ_{5.3.2}$ представлен в виде блока инверторов, инвертирующего цифровые *D* разряды кодов, и мультиплексора *MUX,* управляемого знаковым *S* разрядом преобразуемого кода и коммутирующего на вход сумматора *См* (или на выход устройства) прямые или инверсные значения цифр. Это объясняется тем, что блок получения обратного кода сопоставим с блоком типа *ifthen* алгоритма и предусматривает два способа формирования выходного информационного объекта (ИФО). Следовательно, соответствующий ему структурный блок (как отмечалось выше) должен содержать мультиплексор, выбирающий один из двух формируемых ИФО в качестве выходного. Впрочем, возможна и иная реализация этого блока непосредственно из соотношения для цифр обратного кода

$$Di' = S\overline{Di} \vee \overline{S}Di$$

D_i' – i-я цифра ОК, D_i – i-я цифра исходного кода, а S – знак кода

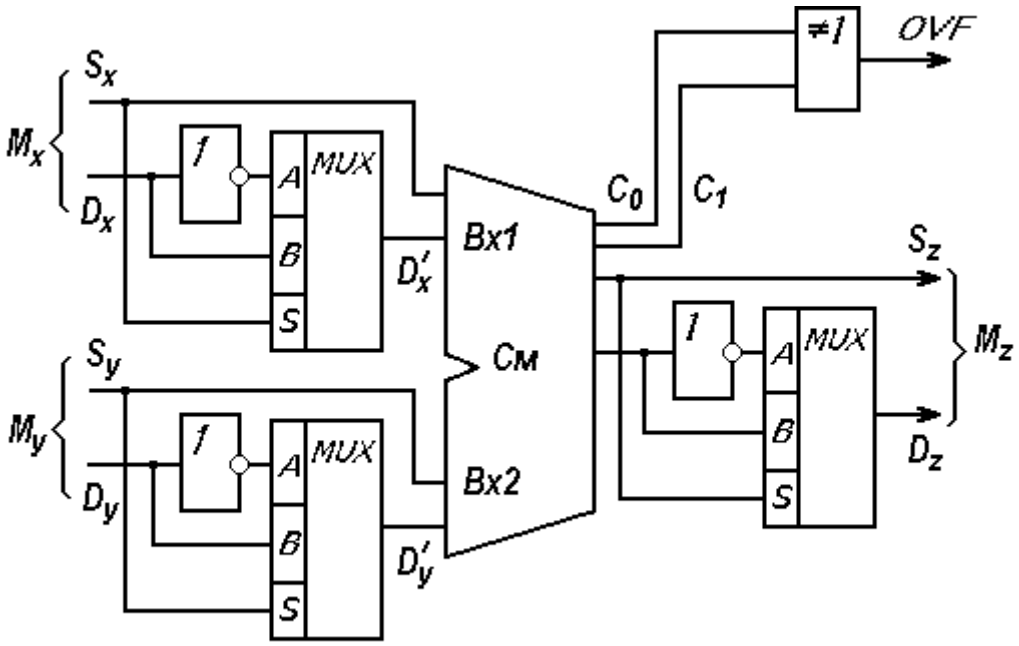

Рис. 9. Структурная схема устройства, реализующего сложение мантисс

Аналогичным образом могут быть получены структурные схемы для всех остальных участков рассматриваемого исходного алгоритма сложения чисел с плавающей запятой.

Результатом выполнения этапа формирования функциональной структуры будет являться структурная схема устройства, состоящая из типовых (комбинационных) функциональных блоков, выполняющая всю операцию, описываемую исходным алгоритмом, без управляющих сигналов и буферизации (запоминания) промежуточных результатов. Иными словами, можно считать, что построенная схема выполняет заданную операцию за один (достаточно для этого продолжительный) такт. Такая схема соответствует предельному распределению выполняемых функций по функциональным блокам, когда каждый из блоков выполняет только одну функцию.

Таким образом, получение данной схемы можно было бы рассматривать как завершение этапа формирования функциональной структуры АЛУ. Однако процедура разработки ее предполагала построение максимально распараллеленной структуры, которая в дальнейшем может подвергаться преобразованиям с целью уменьшения аппаратных затрат. Определенные преобразования могут потребоваться и на этапе формирования конструктивной структуры (см. перечень этапов проектирования на с.24). Если эти преобразования не затрагивают алгоритмов, взятых за основу при выполнении этапа формирования функциональной структуры, то модификации ее не потребуют повторения данного этапа. Если же изменения производятся в самом алго

ритме, то это может привести к необходимости сформировать новую функциональную структуру устройства.

Рассмотрим основные такие преобразования.

2.4. Модификация алгоритмов и структур АЛУ

Между преобразованиями алгоритмов и функциональных структур имеются определенные аналогии, так как и первые и вторые описывают некоторую последовательность функций. Но есть между этими преобразованиями и различия, поэтому целесообразно рассмотреть их раздельно.

2.4.1. Модификация алгоритмов выполнения операций

Основные преобразования алгоритмов, эквивалентные в функциональном отношении, которые могут использоваться при проектировании АЛУ, сводятся к следующим группам: изменение порядка следования блоков алгоритма, замена функционального базиса алгоритма и замена части или всего алгоритма.

К изменению порядка следования блоков алгоритма относятся:

а) перестановка местами независимых блоков (участков) алгоритма. Причем независимость блоков понимается в двух аспектах: в информационном смысле, как отсутствие в них общих переменных, значения которых могут изменяться блоками, и в логическом смысле, как отсутствие условий (внутри или вне блоков), одни и те же значения которых по-разному влияют на вызов рассматриваемых блоков на выполнение. При перестановке местами двух независимых блоков алгоритма информационная и логическая независимость должна также распространяться на все блоки, попадающие в область перестановки (т.е., между переставляемыми блоками);

б) распараллеливание (переход к одновременному выполнению) независимых участков или блоков алгоритма;

в) переход к последовательному выполнению одновременно выполняемых независимых участков или блоков алгоритма.

К преобразованиям, связанным с заменой функционального базиса алгоритма, относятся:

а) объединение (композиция) функциональных блоков алгоритма, приводящая *к* появлению новой функции (например, последовательный счет – сложение, последовательность сложений – умножение, вычисление результата – использование таблицы результатов);

б) разделение (декомпозиция) функциональных блоков алгоритма (например, умножение – серия сложений);

в) использование других типов операций (например, вычитание – сложение с обратным колом, сдвиг влево – суммирование числа с самим собой).

Примерами замены алгоритма или части его, относящихся к третьей группе, могут служить переходы к использованию различных методов или формул для решения одной и той же задачи. Так, подсчет площади S треугольника можно производить по известной формуле $S = b*h/2$ или вместо этого по формуле $S = 1/2* a* b* sin\ \gamma$, сумму чисел отрезка натурального ряда от n_1 до n_2 можно получить последовательностью сложений или по формуле $(n_2 * (n_2 + 1) - n_1*(n_1 - 1))/2$. Но такие замены осуществляются за счет использования некоторой скрытой полностью или частично в алгоритме взаимосвязью между объектами и операциями преобразования. И в этом смысле замены могут относиться не столько к самому алгоритму, сколько к более высокому уровню знаний о преобразуемых алгоритмом объектах, решаемых задачах. Так, в приведенных примерах это знания о соотношениях элементов треугольника и натурального ряда чисел.

Необходимо указать такие, что преобразования, отнесенные к разным группам могут быть близки друг другу, например, таковыми являются “б” из первой группы и “а” из второй, преобразование “в” из второй группы к преобразованиям третьей группы и т. д.

2.4.2. Модификация структуры АЛУ

После выполнения этапа формирования функциональной структуры АЛУ в соответствии с принятой стратегией получается максимальное распараллеливание функций. Преобразования структуры устройства в общем случае могут преследовать различные цели, связанные с улучшением тех или иных их характеристик. Наиболее часто в качестве таковых рассматриваются быстродействие и аппаратные затраты.

Повышение быстродействия

В структуре с максимальным распараллеливанием функций повышение быстродействия возможно только за счет использования более быстродействующих элементов или уменьшения длины цепей преобразования (глубины схемы) устройства, что дает сокращение длительности такта и повышение рабочей частоты.

Первый из способов не связан с модификацией структуры. А уменьшение глубины схемы непосредственно предполагает структурные изменения и может быть выполнено в структурах с максимальным распараллеливанием функций двумя путями:

сокращением числа блоков и их упрощением, главным образом, за счет исключения сложных операций из числа операций, реализуемых устройством;

максимально возможным сокращением количества коммутирующих узлов (блоков вентилей, мультиплексоров) в цепях устройства.

Исключение сложных операций в принципе снижает функциональные возможности устройства, однако этот путь имеет и свои преимущества, что подтверждается жизнеспособностью ЭВМ с RISC архитектурой (ЭВМ с сокращенным набором команд). Структура этих ЭВМ ориентирована на выполнение коротких и простых операций, а длинные – реализуются программным путем.

Сокращение количества коммутирующих узлов в структуре с максимальным распараллеливанием функций возможно в тех случаях, когда в алгоритмах реализуемых операций имеются разветвления. В этом случае каждой из ветвей может быть поставлен в соответствие свой блок преобразований, а коммутация будет производиться только на выходе устройства посредством выбора нужной ветви.

Наконец, сокращение глубины схем возможно за счет использования блоков с меньшей глубиной цепей. Например, в качестве сумматора может использоваться сумматор с параллельными переносами, имеющий наиболее короткие, по числу элементов, схемы формирования сигналов переноса.

Сокращение аппаратных затрат

Сокращение аппаратных затрат обычно понимается как уменьшение количества блоков, входящих в устройство, или их упрощение. Если не допускать дробления блоков, имеющихся в преобразуемой структуре, то осуществление таких модификаций возможно лишь за счет следующих процедур:

1. Совмещение выполнения одинаковых преобразований, соответствующих различным блокам алгоритма, в одном и том же типовом функциональном блоке (например, для рассмотренного выше устройства сложения мантисс операции преобразования слагаемых в обратный код и суммы в прямой код можно возложить на один и тот же блок).

2. Перенос выполнения некоторого более простого преобразования P_i в блок, предназначенный для выполнения более сложного преобразования P_j, реализуемого в устройстве в том случае, если преобразование P_i можно рассматривать как частный случай преобразования P_j (например, добавление единицы можно производить не в счетчике, а в сумматоре, если таковой уже имеется в устройстве).

3. Уменьшение количества входов многовходовых типовых блоков, реализующих некоторое преобразование P_i^n преобразование P_i от n аргументов), за счет использованы~ блоков, реализующих то же преобразование P_i^k

от меньшего числа аргументов, $k < n$. (Например, замена трехвходового сумматора двухвходовым, переход от параллельной обработки к параллельно-последовательной и последовательной поразрядной обработке.)

Рассмотрение этих способов позволяет отметить следующее.

Совмещение выполнения нескольких одинаковых преобразований в одном типовом блоке, как и сокращение количества входов блока, в большей части случаев требует перехода к многотактной реализации всего преобразования, сопровождающейся буферизацией промежуточных результатов и, соответственно, мультиплексированием входов и(или выходов блоков. Поэтому использование совмещенного выполнения преобразований оправдано с точки зрения снижения сложности устройства (точнее, упрощения его блочного состава, так как появление новых связей также может требовать решения дополнительных задач проектирования, особенно в технологии БИС), если вводимые буфер и мультиплексоры проще исключаемого блока (блоков) . С этой точки зрения совмещение целесообразно для блоков, начиная только с некоторого порога их сложности. Так, совмещение блоков вентилей, мультиплексоров, инверторов и т.д. может быть нецелесообразным, если буферизация и мультиплексирование (шинная организация) не предусмотрены в устройстве по другим причинам.

Совмещение одинаковых преобразований в одном типовом блоке имеет свои особенности в зависимости от взаимного расположения блоков, реализующих эти преобразования, в устройстве. Так, в некоторых случаях можно не использовать буферизацию.

К основным вариантам взаимного расположения блоков структурной схемы устройства, выполняющих одинаковые преобразования, можно отнести:

1) расположение блоков в параллельных цепях схемы;
2) расположение блоков в последовательной цепи (целях) схемы;
3) смешанный вариант: расположение блоков и в параллельных и в последовательных цепях.

Причем при расположении в параллельных цепях возможны случаи следующего использования результатов:

из N результатов преобразований, выполняемых параллельно над N различными входными ИФО, используется только один, выбираемый по значению некоторого условия, что будет обозначаться через $N \rightarrow 1$;

используются все N результатов одинаковых преобразований над N различными входными ИФО, что будет обозначено как $N \rightarrow N$.

Соответствующие этим случаям модификации структуры устройства, уменьшающие количество блоков преобразования, показаны на рис. 10,а и б,

где *M* – буфер для хранения результата (регистр, триггер), *DMX* – демультиплексор.

Для цепи $N \rightarrow 1$ следует также различать ситуации, в которых выбор зависит от значений входных ИФО или от результатов их преобразования. Модификация структуры, представленная на рис. 10, *а,* соответствует первому случаю. Во втором случае вариант принципиально не отличается от рис. 10, *б*. Из рис. 10, *б* также видно, что количество буферов, вводимое при такой модификации, должно быть не менее $N - 1$, а кроме входного мультиплексора, требуется еще и демультиплексор, обеспечивающий занесение очередного результата в соответствующий буфер.

Возможен и промежуточный вариант, когда используются *n* результатов из *N* преобразований. Этот случай может быть представлен как сочетание первых двух.

При расположении одинаковых блоков в последовательной цепи их совмещение также требует использования буфера, мультиплексора и демультиплексора, как и в случае, показанном на рис. 10, *б*. Однако буфер может быть поставлен в любой части цепи, расположенной между совмещаемыми блоками. Соответствующая этому случаю модификация структуры представлена на рис. 10, *в*, причем возможные различные расположения буфера показаны на рисунке пунктирным изображением: буфер может занимать любое одно из указанных мест.

Второй способ модификации структур устройств состоит в переносе выполнения простых преобразований в имеющиеся в устройстве блоки, ориентированные на выполнение преобразований, частным случаем которых являются переносимые простые преобразования. По получаемым вариантам модификации структуры этот способ полностью аналогичен рассмотренному выше совмещению одинаковых преобразований.

Третий из указанных способов упрощения структуры: уменьшение количества входов типовых блоков за счет реализации преобразований от большого числа аргументов одноименными преобразованиями от меньшего числа аргументов (для преобразований, допускающих такое представление, например, для суммирования), соответствует замене одного многовходового блока цепочкой блоков или деревом блоков с меньшим числом входов.

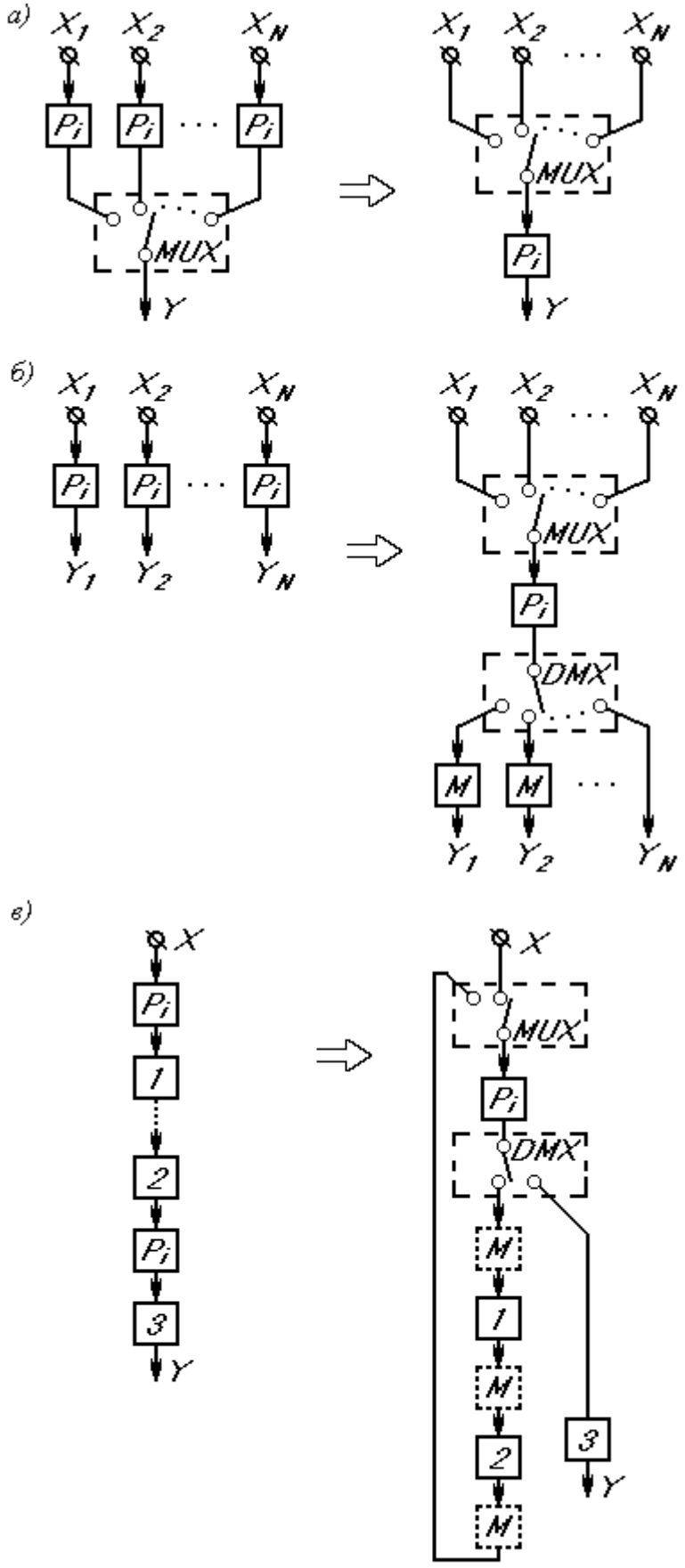

Рис. 10. Преобразования различных типов структур
за счет совмещения выполнения одинаковых функций
в одном блоке

Переход к цепочке блоков показан на рис. 11, *а,* где P^N –*N* -местное преобразование, P^2 – одноименное двухместное преобразование. Такой переход соответствует переходу от преобразования $Y= P^N(X_1, X_2, ..., X_N)$ к рекурсивной подстановке вида $Y_i = P^2(X_i, Y_i-1)$, где $i= 2 \div N$, $Y_i = X_i$, $Y_N= Y$.

К полученной последовательной цепочке одинаковых блоков P^2 может быть применен первый из рассмотренных способов преобразования структуры устройства: совмещении одинаковых операций, что дает схему, показанную в правой части рис. 11, *а.*

Замена *N*-входового преобразователя на дерево блоков сохраняет поярусную параллельность структуры, которая оказывается более быстродействующей, по сравнению с предыдущим вариантом. Такая замена показана на рис. 11, *б*.

Количество используемых при этом блоков такое же, как и в последовательной цепочке. Но существенно упростить полученную структуру не удается, так как на первом ярусе дерева приходится буферизовать *N*/n информационных объектов (где *n* – количество входов блоков преобразования). Результат такого совмещения показан в правой части рис. 11,6, где допущены определенные условности: так как часть буферов используется многократно (в нескольких ярусах), то мультиплексоры и демультиплексор должны были бы иметь больше положений: по несколько для многократно используемых буферов.

При всех рассмотренных преобразованиях структуры устройства предполагалось наличие на входах блоков преобразования входных ИФО в течение всего времени преобразования. Такое допущение вызвано ограничением задачи проектирования разработкой собственно АЛУ (или любого отдельного блока, устройства и т. д.), решением этой задачи отдельно от построения других устройств процессора. Это, по сути, эквивалентно выделению, по правилам формирования функциональной структуры, участка общего алгоритма работы процессора, соответствующего выполнению операций в АЛУ.

Приведенные варианты преобразования структур устройств позволяют осуществлять переход от структуры с максимальным распараллеливанием функций к структуре с минимальным составом блоков. При этом возникают проблемы выбора и оценки различных структур.

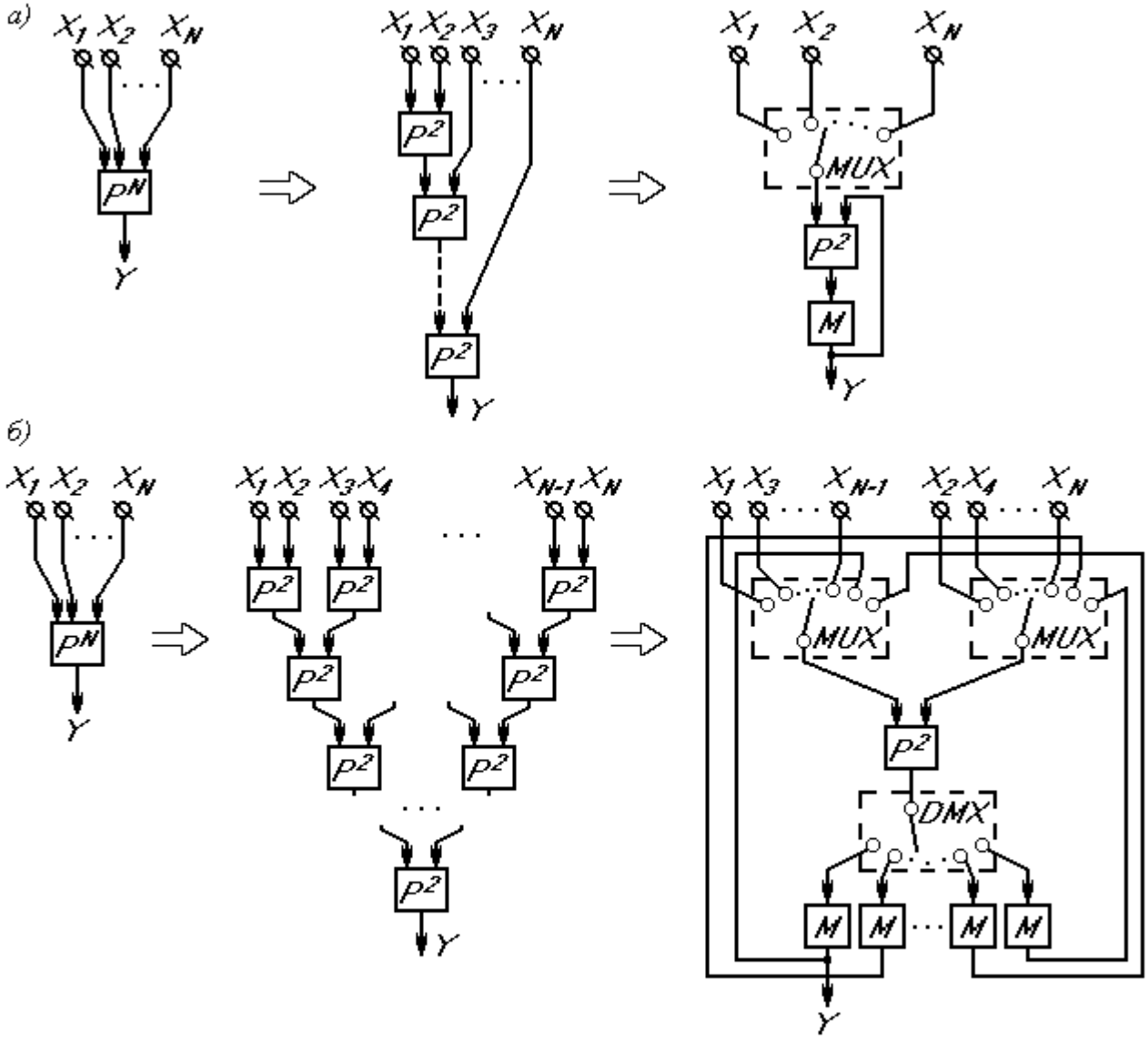

Рис. 11. Преобразования структуры, дающие уменьшение количества входов типовых блоков за счет реализации их функций цепочкой (*а*) и деревом (*б*) типовых блоков с меньшим количеством входов

2.5. Оценка структур АЛУ

К основным техническим характеристикам устройств обработки информации, выполненных на современной технологической базе: больших и сверхбольших интегральных схемах, относят занимаемую на кристалле площадь, быстродействие, рассеиваемую мощность. Кроме того, часто учитываются и такие особенности схем, как тестопригодность, иногда, топология связей, характеризующая сложность проектирования и изготовления схемы.

В общем случае для оценки какого-либо объекта может использоваться некоторая математическая форма: критерий, учитывающий различные характеристики оцбниваемого объекта. Методы построения и применения таких критериев рассмотрены в различных работах, в частности в [12]. Один из примеров такого критерия для АЛУ приведен в методических указаниях по

курсовому проектированию [19] и учитывает количество элементов, узлов, связей, управляющих сигналов и времена выполнения операций в АЛУ.

Однако непосредственную связь между особенностями и характеристиками собственно структурной организации устройства и характеристиками технической его реализации, такими как занимаемая на кристалле площадь, тестопригодность, проследить довольно сложно. Рассеиваемую мощность для одинаковых по составу структур можно считать примерно равной.

Поэтому основное внимание при сравнении различных вариантов структур, получаемых на этапе формирования функциональной структуры, будет уделено их временным оценкам, определяемым глубиной (количеством последовательных элементов) получаемых цепей преобразования. С этой точки зрения такт работы устройства, выполняющего преобразования за несколько тактов и имеющего выходной буферный регистр для сохранения промежуточных результатов, можно рассматривать как состоящий из следующих фаз:

- настройка многофункциональных блоков преобразования на выполнение заданных функций и настройка мультиплексоров входных цепей преобразователей;

- прохождение сигналов через преобралователи и выполнение преобразований над ними;

- прохождение сигналов через выходные коммутаторы (демультиплексоры) и стробирование их с целью «выравнивания» во времени;

- буферизация результата.

Примерное расположение этих фаз по рабочему такту устройства показано на рис. 12, где $t_т$ – длительность такта, $t_н$ – время настройки преобразователей и мультиплексоров на их входах, $\sim t_{пр}$, – время прохождения сигналов через различные блоки или цепи блоков преобразования, t_y – время задержки сигналов на выходных коммутаторах и вентилях управления и $t_б$ – время буферизации результата преобразования. Видно, что действия, непосредственно связанны с функциональными преобразованиями, составляют часть такта, Действия по коммутации, настройке и управлению, в общем случае, могут быть связаны с особенностям реализуемых алгоритмов и присутствовать даже в структур с максимальным распараллеливанием. Но по большей мере эти действия связаны с мультиплексированием и настройкой блоков преобразования, совмещающих выполнение нескольких одинаковых или различных функций. Поэтому такие действия не могут считаться эффективными с точки зрения выполнения преобразований, как не может считаться эффективной и сама буферизация. Следовательно, можно определить коэффициент $\eta_р$, полезного использования рабочего такта как $\eta_р = t_{пр}/t_т$. Тогда этот коэффициент окажется тем выше, чем сложнее реализуемое преобразование, чем меньше мультиплексоров и демультиплексоров имеется в структуре и чем выше бы

стродействие буферов. Типовое «длинное» преобразование – суммирование, при большой глубине цепи переноса даст высокий η_p, но ценой этому будет более низкое быстродействие устройства, особенно при выполнении логических операций.

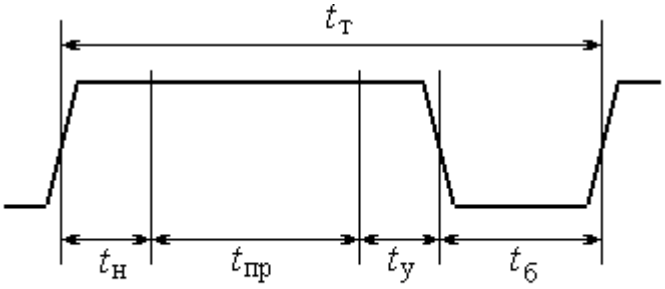

Рис. 12. Рабочий такт устройства
с выходным буфером

Можно считать, что из двух структур одинакового состава лучше та, которая имеет более высокий коэффициент η_p. Однако этот коэффициент не полностью определяет выбор структуры. Ведь если частота использования цепи преобразования наибольшей глубины невысока или цепь эта, по преимуществу, используется для выполнения более простых действий (как частных случаев такого преобразования), то может оказаться целесообразным сократить или изъять эту цепь, несмотря на снижение коэффициента η_p.

Действительно, пусть $P=(p_1, p_2, ..., p_r)$ – множество различных преобразований, которые могут быть выполнены в устройстве со структурой S , и пусть d_i – измеряемая в количестве последовательно включенных вентилей максимальная глубина цепи, реализующей преобразование p_i. (От глубины цепи легко перейти к времени преобразования через длительность τл задержки сигнала на одном элементе.) Далее, пусть каждое преобразование p_i выполняется в АЛУ с относительной частотой f_i, наконец, пусть d_{max} максимальная глубина цепи из всех цепей, реализующих различные преобразования, соответствующая некоторому преобразованию p_m, где $p \in P$.

Тогда можно определить потери времени t_L, вызванные необходимостью выбора длительности такта работы АЛУ из расчета на самое длинное преобразование. Очевидно, $t_L=\tau_л \sum_{i=1}^{i=r} f_i(d_{max} - d_i)$, где $\tau_л$ – время задержки сигнала на одном элементе. Соотнося время i-го преобразования с максимальным, можно получить коэффициент ν_p полезного использования быстродействия цепей преобразования, определяемый как $\nu_p = 1 / d_{max} \sum_{i=1}^{i=r} f_i d_i$

Для улучшения значения этого коэффициента имеются две возможности модификации структуры S АЛУ: а) исключить преобразование p_m из числа реализуемых устройством за один такт, разбив его на более простые преобразования p_{m1} и p_{m2}, выполняемые в двух последовательных тактах работы устройства (желательно, чтобы выполнялись соотношения $p_{m1} \in P$ и

$p_{m2} \in P$); б) уменьшить глубину $d_{\max}$ цепи, реализующей преобразование p_m за счет усложнения этой цепи.

В первом случае потери времени на всех преобразованиях, кроме p_m, уменьшатся до $t_L' = \tau_л \sum_{i \neq m} f_i (d'_{\max} - d_i)$, где $d'_{\max}$, – глубина цепи, следующей по величине после глубины (исключаемой цепи) d_{max}, а время выполнения преобразования p_m возрастет на $\Delta t_L' = \tau_л (2d'_{\max} - d_{\max})$. Если $f_m \Delta t_m < t_L - t_L'$, то с точки зрения времени преобразования такая модификация структуры АЛУ представляется оправданной (аналогичная ситуация характерна для RISC архитектуры, отказывающейся от сложных команд).

Во втором случае, если $d_{\max} \leq d'_{\max}$, потери t_L' составят $t_L = \tau_л \sum_{i=1}^{i=r} f_i (d'_{\max} - d_i)$, где сумма взята по всем преобразованиям, включая p_m т. е. выигрыш во времени будет несколько большим. Однако при этом возрастут и аппаратные затраты, что потребует соотнесения их с получаемым выигрышем во времени.

Влияние преобразований структуры устройства на значения коэффициентов η_p и ν_p может быть противоположным. Например, ориентация устройства на более сложные (базовые) преобразования приводит к большей глубине схем преобразователей и увеличению η_p. Однако относительные частоты использования преобразований в общем случае снижаются с повышением их сложности (это видно и на примерах смесей команд), что уменьшает ν_p. И наоборот, ориентация структуры на более простые базовые преобразования увеличивает , но значительно снижает η_p.

Рассмотренные коэффициенты позволяют оценить структуру АЛУ с точки зрения эффективности использования ее возможностей. (Можно также ввести коэффициент, аналогичный ν_p, но учитывающий эффективность использования схемы АЛУ в различных операциях: долю элементов, участвующих в различных операциях.) Однако необходимо помнить, что АЛУ, как правило, входит в состав более сложной системы, например ЭВМ, и общий критерий его оценки в этом случае желательно строить с точки зрения эффективности всей системы в целом. Обычно такие критерии связаны с соотношением стоимости и производительности.

Глава 3. СТРУКТУРЫ АЛУ ДЛЯ ВЫПОЛНЕНИЯ РАЗЛИЧНЫХ ОПЕРАЦИЙ

В этой главе рассматриваются простые варианты структур АЛУ, ориентированных на выполнение основных арифметических операций. Приводимые структуры можно получить с помощью процедур, рассмотренных в предыдущей главе. Однако привести полное описание таких действий в рамках настоящего пособия не представляется возможным. Поэтому далее дается, в основном, описание представляемых структур, а не способы их получения.

Изложенные в главе 2 методы построения предусматривали получение структур, соответствующих максимальному распараллеливанию преобразований по блокам устройства, обеспечивающему однотактную реализацию операции.

В конце п. 2.4 указывалось также, что рассматриваемая методика построения структур АЛУ предполагала наличие на входах устройства преобразуемых данных в течение всего времени выполнения операции. Для обеспечения этого требования в случае перехода к многотактному выполнению операции при сокращении числа блоков в устройстве во всех рассматриваемых далее структурах будут использоваться регистры. В этих регистрах будут размещаться операнды в течение всей операции или того времени, пока в них сохраняется необходимость. Кроме того, результат операции также будет располагаться в регистре.

3.1. АЛУ для сложения и вычитания двоичных чисел с фиксированной запятой

В п.2.3 была получена структура устройства для алгебраического сложения мантисс чисел с плавающей запятой (см. рис. 9). Было отмечено, что это устройство можно рассматривать так же, как устройство для сложения чисел с фиксированной запятой, представленных в прямом коде.

Добавим к этой структуре, в соответствии с отмеченным выше, регистры для операндов и результата и триггер для хранения признака переполнения. Кроме того, наличие трех одинаковых преобразователей кода позволяет применить совмещение выполняемых ими действий, если необходимо уменьшить аппаратные затраты. Такие совмещения могут быть выполнены различными способами. Если исключить из структуры, показанной на рис. 9, преобразователь на выходе сумматора и возложить его функции на преобразователь, связанный со вторым входом сумматора, то схема примет вид, показанный на рис. 13, *а.* Причем на этом рисунке, в отличие от рис. 9, преобразование колов обеспечивается подачей соответствующих управляющих

сигналов (A_1, ..., A_5), а не непосредственным управлением передачей прямых или инверсных значений цифровых разрядов с помощью знакового разряда регистра. Это сделано как с учебной целью, так и для большей гибкости получаемого устройства.

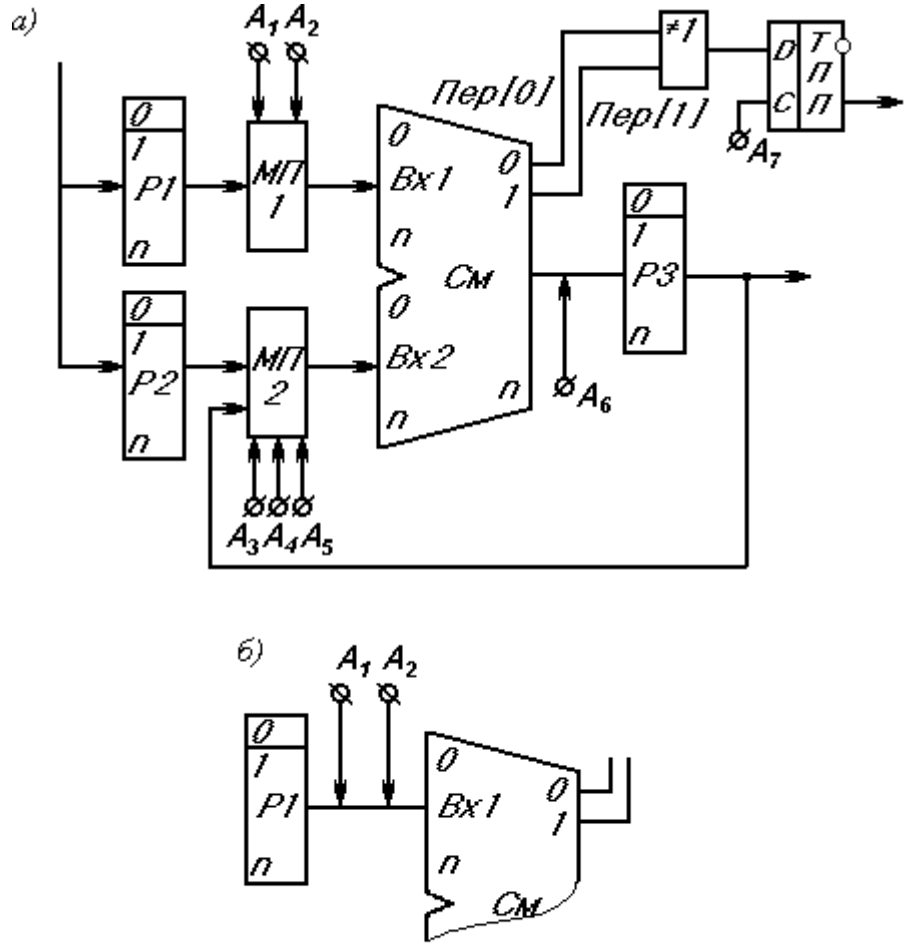

Рис. 13. Структура АЛУ для сложения двоичных чисел с фиксированной запятой (а) и вариант упрощенного изображения связи регистра *P1* со входом сумматора (б)

На рис. 13, а приняты следующие обозначения: *P1, P2* и *P3* – (*n+1*)-разрядные регистры, *n* разрядов у которых цифровые, и один разряд, 0,– знаковый; *ТПП* – триггер, хранящий значение признака переполнения; *МП1* и *МП2* – мультиплексоры, осуществляющие передачу прямых или инверсных значений цифровых разрядов в зависимости от поданного управляющего сигнала; A_1, ..., A_7 – управляющие сигналы, содержание которых следующее:

A_1: *Bx1* := (*P1[0 : n]*) – подача на вход 1 сумматора содержимого всех разрядов регистра *P1*,

A_2: *Bx1* := (*P1[0]*).($\overline{P1[1:n]}$) – подача на вход 1 сумматора содержимого знакового разряда регистра *1* и инверсии содержимого всех цифровых

разрядов регистра *P1,* т. е., обратного кода числа, записанного в регистре *P1,*

A_3: *Вх2* := (*P2[0 : n]*) – подача на вход 2 сумматора содержимого всех разрядов регистра *P2,*

A_4: *Вх2* := (*P2[0]*).($\overline{P2[1:n]}$)– аналогично A_2,

A_5: *Вх2* := (*P3[0]*).($\overline{P3[1:n]}$) – аналогично A_2,

A_6: (*P3*) := *ВыхСм* – занесение в регистр *P3* информации с выхода сумматора,

A_7: (*ТПП*)*:= Пер[0]⊕Пер[1]* – занесение в триггер признака переполнения суммы по модулю два значения сигналов переносов из нулевого и первого разрядов сумматора.

Во всех случаях в квадратных скобках указываются номера разрядов соответствующего регистра. Круглые скобки, означающие содержимое соответствующего узла, иногда для сокращения записи могут опускаться, например, можно записать A_2 как $Вх1 := P1[0].\overline{P1[1:n]}$.

Для упрощения структурной схемы при изображении мультиплексоры могут опускаться. В этом случае показываются только связи, на которых наносятся соответствующие управляющие сигналы так, как представлено на рис. 13, б.

В таком устройстве выполнение операции сложения осуществляется в зависимости от знака результата за один или два такта. В первом такте слагаемые подаются на входы сумматора, причем передача цифр осуществляется прямо или инверсно, что определяется знаками слагаемых. С выхода сумматора, в этом же такте, результат заносится в регистр Р3, а в триггер *ТПП* заносится значение признака переполнения при подаче на его С-вход разрешающего сигнала A_7. Второй такт необходим в случае получения отрицательной суммы для преобразования ее в прямой код инверсной передачей цифровых разрядов содержимого регистра Р3 через *МП2*. Нужно учитывать, что для выполнения этого действия триггеры регистра Р3 должны быть двухтактными (построенными по M–S схеме). В противном случае потребуется дополнительный такт – передачи суммы в регистр *P2* перед преобразованием ее в прямой код.

Микропрограмма выполнения этой операции представлена на рис. 14. Предполагается, что операнды (слагаемые) к началу операции находятся в регистрах *P1* и *P2,* а результат должен размещаться по окончании операции в регистре Р3. Заметим также, что управляющий сигнал A_7, фиксирующий значение признака переполнения, подастся только при одинаковых знаках слагаемых, так как в противном случае переполнение невозможно.

Выполнение операции вычитания сводится обычно к сложению

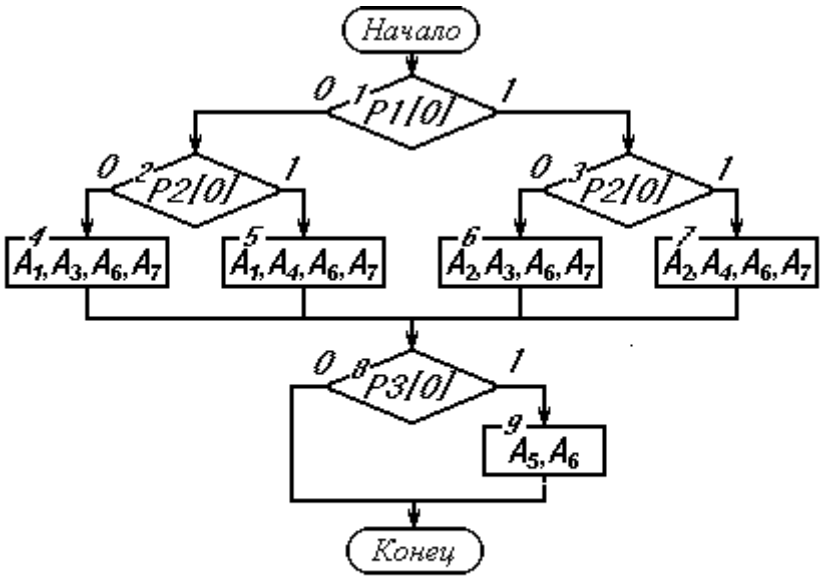

Рис. 14. Микропрограмма сложения двоичных чисел с фиксированной запятой

уменьшаемого с вычитаемым, взятым с обратным знаком. Если уменьшаемое находится к началу операции в регистре *P1,* а вычитаемое – в регистре *P2,* то для выполнения вычитания можно либо изменить знак содержимого *P2* на обратный и выполнить микропрограмму сложения , либо заменить в микропрограмме сложения микрооперацию A_3 на передачу *Bx2* := (*P2[0* : *n]*), а микрооперацию A_4 на передачу *Bx2* := (*P2[0]*).(*P2[1* : *n]*) или просто на *Bx2* := $\overline{P2[1:n]}$, так как эта микрооперация используется для отрицательного вычитаемого.

Второй способ выполнения операции вычитания короче по времени на один такт чем первый, но требует двух дополнительных микроопераций.

Рассмотренное устройство построено на основе комбинационного сумматора. Если использовать сумматор накапливающего типа, то структура АЛУ будет несколько иной.

Известно, что накапливающий сумматор обычно реализуется на основе комбинационного сумматора, на выходе которого устанавливается регистр. Причем выход этого регистра связывается с одним из входов сумматора, второй из входов которого становится входом получающегося накапливающего сумматора. В этом случае в АЛУ сумматор (точнее его регистр) служит для хранения как одного из слагаемых, так и суммы. Поэтому количество регистров в таком устройстве меньше, чем в предыдущем случае. Один из возможных вариантов его структуры приведен на рис. 15, а, где *PC* – регистр сумматора, а обозначения остальных узлов совладают с принятыми на рис. 13. Управляющие сигналы (или микрооперации) A_1 и A_2 совпадают с этими же сигналами для предыдущего случая, A_5 совпадает с ранее введенным A_7, а содержание A_3 и A_4 следующее:

A_3: $(PC[1 : n]) := (\overline{PC[1:n]})$ – инвертирование цифровых разрядов содержимого регистра *PC*,

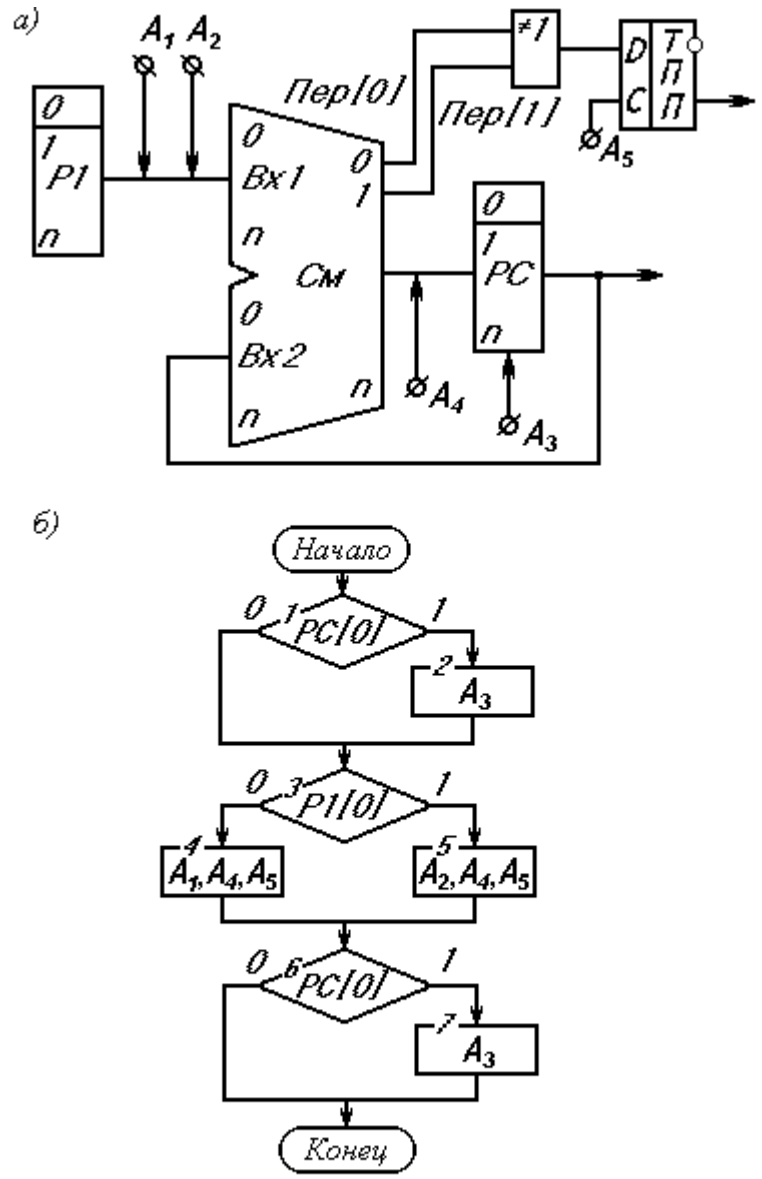

Рис. 15. Структура АЛУ на базе накапливающего сумматора для сложения чисел с фиксированной запятой (*а*) и микропрограмма выполнения сложения (*б*)

A_4: $(PC) := ВыхСм$ – аналогично A_6 в предыдущем устройстве.

Следует отметить, что на вход *Вх2* сумматора содержимое *PC* подается жестко, без управляющего сигнала.

Различные варианты структуры этого АЛУ, в основном, могут отличаться способом получения обратного кода числа, записанного в регистре *PC*. В рассматриваемой структуре этого преобразование выполняется непосредственно в регистре *PC*, для чего триггеры этого регистра должны иметь вход *T,* подача единицы на который инвертирует содержимое триггера.

Можно выполнять это преобразование аналогично тому, как это было сделано в предыдущей схеме, подавая инверсию содержимого цифровых разрядов *РС* на *Вх1* или *Вх2* сумматора и принимая инвертированные цифры в *РС* с выхода сумматора.

Выполнение операции сложения в данном устройстве потребует от одного до трех тактов. Если в исходном состоянии первое слагаемое записано в *РС*, а второе в *Р1,* то микропрограмма может выглядеть так, как показано на рис. 15, б. Первый оператор микропрограммы (вершина с номером 2) обеспечивает получение обратного кода в случае отрицательного первого слагаемого. Операторы с номерами *4* и 5 служат для добавления в прямом или обратном коде второго слагаемого к первому, а оператор с номером 7 используется в случае получения отрицательной суммы для преобразования ее в прямой код.

Если использовать для получения обратного кода числа, записанного в регистре *РС*, передачу инверсии его цифр через вход *Вх2* сумматора, как в предыдущей схеме, то вход *Вх2* сумматора будет связан с выходом *РС* управляемыми цепями, а не жестко, и микропрограмма операции будет аналогична приведенной на рис. 14.

Для выполнения операции вычитания в данном устройстве используются те же способы, что и в вышерассмотренном.

3.2. АЛУ для сложения и вычитания чисел с плавающей запятой

0бобщенная структура АЛУ для сложения чисел с плавающей запятой (ПЗ) рассмотрена в п. 2.3. Там же указан путь получения структурной схемы этого устройства с максимальным распараллеливанием на основе типовых функциональных блоков.

При соответствующей декомпозиции и совмещении выполнения преобразований, представляющих отдельные этапы операции сложения чисел с ПЗ, можно сформировать структуру АЛУ, представленную на рис. 16. В этом АЛУ числа с ПЗ имеют порядок в виде целого числа со знаком, содержащего p цифровых разрядов, и мантиссу в виде дробного числа со знаком, содержащего m цифровых разрядов. Под знак в обоих случаях отведено по одному разряду (0-й – знак порядка, $(p + 1)$-й – знак мантиссы). На этом рисунке использованы обозначения: *ТП* – триггер переполнения порядка, *ТМ* – триггер переполнения мантиссы, *ТЗ* – триггер потери значимости (получения нулевой мантиссы результата при ненулевом порядке его), *ТИ* – триггер исчезновения порядка (получения порядка меньше минимально представимого в разрядной сетке порядка, т.е. “отрицательное переполнение” порядка), остальные обозначения аналогичны ранее рассмотренным.

Микрооперации, реализуемые устройством, следующие:

A_1: $Bx1 := P1[0 : p]$ – подача на $Bx1$ порядка из регистра $P1$,

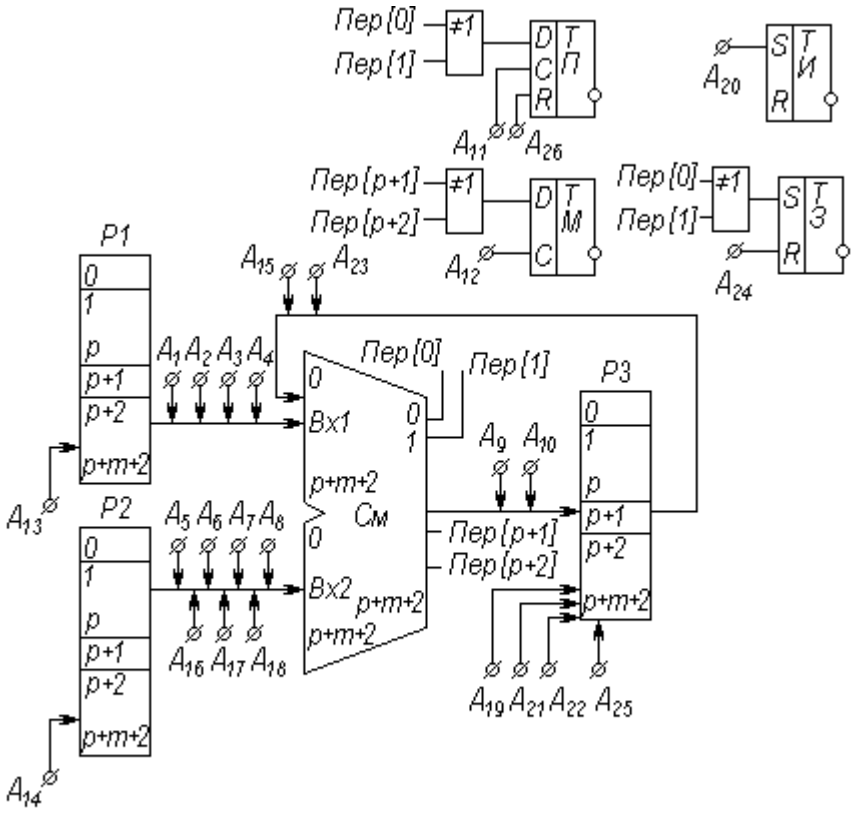

Рис. 16. Структура АЛУ для сложения чисел с плавающей запятой

A_2: $Bx1 := P1[0].\overline{R1[1 : p]}$ – подача на $Bx1$ порядка из $P1$ обратным кодом,

A_3: $Bx1 := P1[(p+1) : (p+m+2)]$ – подача на $Bx1$ мантиссы из $P1$,

A_4: $Bx1 := P1[p+1].\overline{R1[(p+2) : (p+m+2)]}$ – подача на $Bx1$ мантиссы из $P1$ обратным кодом;

A_5: $Bx2 := \overline{R2[0 : p]}$ – подача на $Bx2$ инверсии порядка из $P2$,

A_6: $Bx2 := P2[1: p]$ – подача на $Bx2$ цифровых разрядов порядка $P2$,

A_7: $Bx2 := P2[(p+1) : (p+m+2)]$ – аналогично A_3,

A_8: $Bx2 := P2[p+1].\overline{R2[(p+2) : (p+m+2)]}$ – аналогично A_4

A_9: $P3[0 : p] := ВыхСм[0 : p]$ – запись в $P3$ порядка с выхода сумматора,

A_{10}: $P3[(p+1) : (p+m+2)] := ВыхСм[(p+1) : (p+m+2)]$ – запись в регистр $P3$ мантиссы с выхода сумматора,

A_{11}: $ТП := Пер[0] \oplus Пер[1]$ – фиксация переполнения порядка,

A_{12}: $ТМ := Пер[p+1] \oplus Пер[p+2]$ – фиксация переполнения мантиссы,

A_{13}: $R1(P1[(p+2) : (p+m+2)])$ – сдвиг вправо на один разряд содержимого цифровых разрядов регистра $P1$,

A_{14}: $R1(P2[(p+2) : (p+m+2)])$ – то же для регистра *P2,*

A_{15}: $Bx1 := P3[0 : p]$ – аналогично A_1,

A_{16}: $Bx2 :=$ 111...10 – подача «–1» порядка на вход *Bx2,*

A_{17}: $Bx2 :=$ 000...01 – подача «+1» порядка на *Bx2,*

A_{18}: $Bx2 := P2[0 : p]$ – аналогично A_1,

A_{19}: $P3[(p+2) : (p+m+2)] := \overline{R3[(p+2) : (p+m+2)]}$ – инвертирование цифровых разрядов мантиссы в регистре *P3*,

A_{20}: $T3 := 1$ – установка в «1» триггера потери значимости,

A_{21}: $R1(P3[(p+2) : (p+m+2)])$ – аналогично A_{13},

A_{22}: $P3[p+1] := \overline{R3[p+1]}$ – инвертирование знакового разряда мантиссы в регистре *P3*,

A_{23}: $Bx1 := P3[0].\overline{R3[1 : p]}$ – аналогично A_2,

A_{24}: $TИ := Пер[0] \oplus Пер[1]$ – фиксация исчезновения порядка,

A_{25}: $L1(P3[(p+2) : (p+m+2)])$ – сдвиг влево на один разряд содержимого цифровых разрядов мантиссы регистра *P3*,

A_{26}: $TП := 0$ – сброс триггера переполнения порядка.

Если принять, что в исходном состоянии слагаемые находятся в регистрах *P1* и *P2,* то микропрограмма выполнения операции сложения чисел с П3 может быть представлена в виде граф-схемы, приведенной на рис. 17. В этой микропрограмме:

условия и операторы с *1* по *7* обеспечивают получение разности порядков слагаемых, нужной для последующего выравнивания порядков;

условия *8* и *9* проверяют возможность получения результата без вычислений при разности порядков, превышающей по модулю число разрядов мантиссы *m;*

операторы и условия *10, 13, 15* и *16* обеспечивают выравнивание порядков сдвигом соответствующей мантиссы;

операторы и условия *11, 12, 19* и *20* формируют результат без вычислений, полагая его равным слагаемому с большим порядком;

условие и операторы *14, 17* и *18* присваивают результату (регистру *P3*) предварительно больший из порядков слагаемых;

условия и операторы с *21* по *27* формируют мантиссу суммы (возможно в обратном коде в случае отрицательной мантиссы);

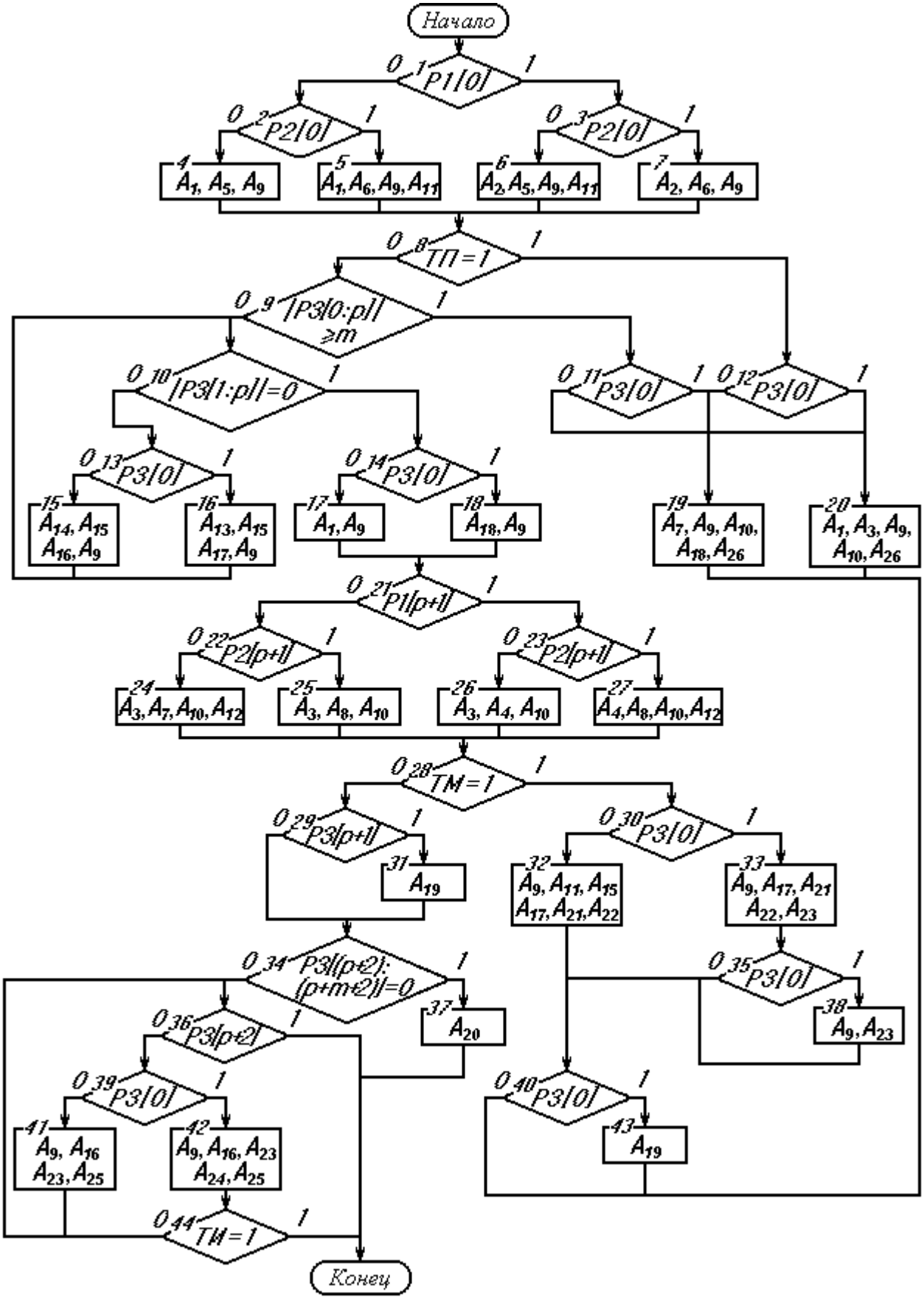

Рис. 17. Микропрограмма выполнения операции сложения чисел с плавающей запятой в АЛУ, представленном на рис. 16

условие *28* проверяет нарушение нормализации мантиссы результата влево (переполнение мантиссы, а условия и операторы *30, 32, 38, 35* и *38* обеспечивают нормализацию результата сдвигом вправо);

условие и оператор *29* и *31* получают прямой код мантиссы перед нормализацией сдвигом влево, а условие и оператор *40* и *43* получают прямой код мантиссы после нормализации сдвигом вправо;

условие *34* проверяет, а оператор *37* фиксирует потерю значимости, устанавливая в «1» триггер *ТЗ*;

условие *36* проверяет нарушение нормализации мантиссы результата вправо (наличие нулей в старших цифрах), а условия и операторы *39, 41, 42* и *44* производят нормализацию сдвигом влево мантиссы суммы и уменьшением на единицу порядка результата при каждом сдвиге, проверяя возможность исчезновения порядка при этом условием *44*.

Логика микропрограммы, в основном проста, некоторые пояснения можно дать к формированию результата (блоки *11, 12, 19* и *20*) без вычислений и к последней части микропрограммы, начиная с блока с номером *28*.

Так, в случае, если разность порядков превосходит по модулю число разрядов мантиссы *m,* то сумма равна слагаемому с большим порядком. Но значение разности порядков, отвечающее этому соотношению, может находиться как в пределах разрядной сетки порядка, так и вызывать переполнение. В первом случае нулевое значение знакового разряда разности говорит о ее положительной величине и, следовательно, о том, что порядок слагаемого, записанного в регистре *Р1,* больше порядка слагаемого в регистре *Р2* (и наоборот при единичном знаке разности). В случае переполнения при вычитании порядков нулевое значение знакового разряда разности, напротив, говорит о том, что больше порядок второго слагаемого, поскольку нулевой знак при переполнении имеет место для отрицательного результата. Это и учтено блоками *11, 12, 19* и *20* микропрограммы. Причем при установке значения результата в РВ сбрасывается триггер переполнения порядка.

Конечная часть микропрограммы связана с нормализацией результата и формированием его признаков. При этом блок *28* микропрограммы выделяет случай нарушения нормализации влево при переполнении в мантиссе суммы. Этот случай обрабатывается блоками *30, 32, 33, 35, 38, 40, 43*, которые обеспечивают сдвиг мантиссы суммы вправо на один разряд с установкой правильного значения знака мантиссы (учитывая изменение знака при переполнении), получение ее прямого кода, наращивание предварительного порядка результата на единицу и проверку его при этом на переполнение с установкой триггера *ТП* в единицу в соответствующей ситуации.

В левой ветви конечного участка микропрограммы условие *34* выявляет случай получения нулевой мантиссы результата, что соответствует потере значимости и вызывает установку в единицу триггера *ТЗ* (оператор *37*). Если мантисса ненулевая, то проверяется ее старший цифровой разряд (условие

36). Если при этом его значение единичное, то мантисса нормализована и операция заканчивается. В противном случае начинается цикл нормализации сдвигом влево, при котором каждый сдвиг мантиссы влево на один разряд сопровождается уменьшением на единицу порядка результата и проверкой на выход порядка за минимальный предел (максимальный по модулю отрицательный порядок, представимый *p* разрядами), т.е. на исчезновение порядка, что производится блоками *36*, *39*, *41, 42* и *44* микропрограммы.

С учетом перечисленного назначения блоков микропрограммы можно установить функциональное соответствие между ними и обобщенными функциональными блоками структурной схемы АЛУ, представленной в главе 2 на рис.6. Так, блок $ОФБ_1$ (см. рис. 6) соответствует блокам *1, ..., 7, 14, 17* и *18* микропрограммы, блок $ОФБ_2$ – блокам *8* и *9*, блок $ОФБ_3$ – блокам *11, 12, 19* и *20*, блок $ОФБ_4$ – блокам *10, 13, 15* и *16*, блок $ОФБ_5$ – блокам *21, ..., 27,* блок $ОФБ_6$ – блокам *28*, ..., *44*. При этом функции обобщенных функциональных блоков реализуются в микропрограмме за несколько тактов в виде последовательных, разветвляющихся и циклических участков, а не чисто схемным способом за один такт, как в структуре с максимальным распараллеливанием функций.

Следует отметить, что разнообразие вариантов структуры, отличающихся различными способами реализации отдельных функций, для данной операции, значительное. Однако эти различия больше касаются связей между узлами устройства и состава микроопераций, чем собственно состава устройства.

Из возможных модификаций настоящей структуры следует отметить использование представления порядков без знаков, в виде целых чисел, называемых характеристиками. Последние можно рассматривать как смещенные порядки, минимальному (т. е. максимальному по модулю отрицательному) из которых соответствует нулевая характеристика, а максимальному – характеристика из всех единиц. Применение такого представления избавляет от необходимости анализа знаков и преобразования кодов порядков при действиях над ними, что несколько упрощает микропрограмму операции и состав микроопераций устройства.

Если для построения устройства необходимо использовать накапливающий сумматор, то изменения в приведенной структуре будут аналогичными тем, которые имели место в устройстве для сложения и вычитания чисел с фиксированной запятой.

Аналогично, для выполнения операции вычитания в рассматриваемом устройстве при обработке мантисс следует использовать способы, изложенные при описании организации вычитания чисел с фиксированной запятой в предыдущем параграфе.

3.3. АЛУ для сложения двоично-десятичных чисел

Десятичные числа могут представляться в ЭВМ различными способами, как с точки зрения изображения самих цифр числа, так и в плане размещения чисел в памяти ЭВМ.

Наиболее распространенным вариантом двоичного кодирования десятичных чисел является преставление каждой десятичной цифры X_i десятичного числа X четверкой двоичных разрядов, называемой тетрадой. Веса двоичных разрядов тетрады соответствуют весам обычного двоичного числа (8-4-2-1), а коды десятичных цифр при этом соответствуют их обычному двоичному представлению: 0 – 0000, 1 – 0001, 2 – 0010, . . . , 9 – 1001.

Такая форма представления, называемая двоично-десятичной, удобна как для обработки чисел в ЭВМ так и для их восприятия человеком. Однако, в каждой тетраде может быть представлено 16 кодовых комбинаций (фактически соответствующих шестнадцатеричным цифрам), шесть из которых не являются десятичными цифрами: 1010, 1011, . . ., 1111. Поэтому сложение и вычитание двоично-десятичных чисел несколько отличается от сложения и вычитания обычных двоичных чисел.

Одним из наиболее распространенных вариантов выполнения этих операций является добавление кода “6” при сложении (вычитании) двоично-десятичных чисел, позволяющее легко идентифицировать те случаи, когда результат в тетраде попадает в диапазон недействительных для десятичных цифр комбинаций 1010 – 1111.

Правила сложения, основанные на это приеме, определяются следующим образом.

Пусть X и Y – два двоично-десятичных числа, а X_i и Y_i – двоично-десятичные цифры i-го разряда этих чисел соответственно.

Сложение этих чисел производится в два этапа.

На первом этапе цифры S_i i-го разряда суммы ($i = 1 \div n$, где n – количество десятичных разрядов в изображении чисел) определяются как

$$S_i = X_i + 6 + Y_i + p_{i+1},$$

где p_{i+1} – перенос из более младшей десятичной тетрады.

При этом в зависимости от соотношения чисел X и Y для каждой из тетрад возможны два следующих случая: $X_i + Y_i + p_{i+1} > 9$ и $X_i + Y_i + p_{i+1} \le 9$.

Тогда с учетом добавления шестерки сумма S_i в этих случаях окажется $S_i > 15$ и $S_i \le 15$ соответственно.

В первом случае $X_i + Y_i + p_{i+1} > 9$ будет сформирован сигнал переноса из данной тетрады в более старшую, уносящий с собой 16 единиц. Тогда в этом случае i-я тетрада суммы S_i будет равна

$$S_i = X_i + Y_i + p_{i+1} + 6 - 16 = X_i + Y_i + p_{i+1} - 10,$$

то есть в такой тетраде будет получено правильное (с учетом переноса десятки в более старший разряд) значение двоично-десятичной суммы.

Во втором случае $X_i + Y_i + p_{i+1} \le 9$ перенос в старшую тетраду не возникает и десятичная сумма в i-м разряде будет определена с избытком 6:

$$S_i = X_i + Y_i + p_{i+1} + 6.$$

Поэтому ее придется корректировать, изымая эту лишнюю шестерку.

Можно заметить, что эта процедура есть вычитание той самой шестерки, которую сначала при сложении цифр слагаемых. Действительно, это так и есть. Однако такой способ позволяет легко отличить рассматриваемые случаи $X_i + Y_i + p_{i+1} > 9$ и $X_i + Y_i + p_{i+1} \le 9$ один от другого.

Конечно, их можно различить, не добавляя шестерки, с помощью специальной комбинационной схемы, анализирующей значение суммы в каждом десятичном разряде. Такой прием тоже используется. Однако, добавление “6” не требует дополнительной схемы анализа суммы, хотя и предполагает дополнительные затраты времени на это действие. Поэтому выбор варианта зависит от разработчика.

На втором этапе сложения производится коррекция суммы, полученной на первом этапе, сводящаяся к вычитанию 6 из тех тетрад, в которых не было сформировано переноса в более старшие десятичные разряды.

Такая коррекция производится добавлением обратного или дополнительного кода 6 в соответствующие тетрады. При этом для получения правильного результата в случае использования обратного кода к тетрадам, из которых был перенос, следует добавить обратный код нуля (1111). При использовании дополнительного кода этого делать не надо, но вместо этого следует блокировать передачу переносов между тетрадами.

Ниже приводится пример, иллюстрирующий рассмотренное правило.

Пример 1.

$X = 542$	0101.0100.0010	
$+\ Y = 374$	$+$ 0110.0110.0110	Добавление “6” ко всем тетрадам X
$S = 916$	1011.1010.1000	
	$+$ 0011.0001.1100	Добавление Y и фиксация переносов из тетрад
	1111.0001.1100	
	$+$ Нет пер Пер Нет пер	Коррекция обратным кодом (+ обр. код 6 к тетрадам, где нет переноса)
	1001.1111.1001	
	$+$ 1001.0001.0101	
	1	циклический перенос
$S = 916$	1001.0001.0110	

- - - - - - - - - - - - - - - - - - -

	1111.0001.1100	
	$+$ Нет пер Пер Нет пер	Коррекция дополнит. кодом (+ доп. код 6 к тетрадам, где нет переноса, с блокировкой переносов между тетрадами)
	1010.0000.1010	
$S = 916$	1001.0001.0110	

Пример 1а (без добавления 6)

$+\begin{array}{l}X=542\\ \underline{Y=374}\end{array}$	$+\begin{array}{l}0101.0100.0010\\ \underline{0011.0001.1100}\end{array}$	Сложение X и Y
	$+\begin{array}{l}1000.1011.0110\\ \underset{\le 9}{}\ \underset{>9}{}\ \underset{\le 9}{}\\ \underline{0000.0110.0000}\end{array}$	Коррекция (+6 к тетрадам, превосходящим 9)
$S=916$	1001.0001.0110	

В примере 1 на первом шаге первого этапа к слагаемому X добавляется код, содержащий шестерку во всех тетрадах. На втором шаге к полученной сумме добавляется второе слагаемое Y тоже как обычное двоичное число. При этом фиксируются значения переносов между тетрадами.

(Нужно отметить, что добавление 6 можно производить с помощью специальной схемы преобразователя кода без использования сумматора, что позволяет сократить время сложения.)

Далее выполняется второй этап операции, состоящий в коррекции тетрад, их которых не было переносов. В рассматриваемом примере это первая и третья тетрады. Как отмечалось, коррекция может быть выполнена добавлением к этим тетрадам как обратного, так и дополнительного кода шестерок. Оба этих варианта и показаны в примере.

Для сравнения приведен и пример 1а, в котором на первом этапе используется сложение с 6 одного из слагаемых. Второй этап такой процедуры заключается в коррекции, состоящей в добавлении прямого кода 6 к тем тетрадам, где сумма $X_i + Y_i + p_{i+1}$ превышает 9.

Для вычитания двоично-десятичных чисел X и Y используется похожее правило, а операция также выполняется в два этапа: на первом этапе получается предварительная разность, которая далее корректируется.

Разность двоично-десятичных чисел X и Y на первом этапе получается сложением кода уменьшаемого Y с обратным кодом вычитаемого Y и обратным кодом заимствования b, причем все эти коды на данном этапе обрабатываются как двоичные. То есть, i-я тетрада разности определяется как

$$D_i = X_i + (15 - Y_i) + (1 - b_{i+1}),$$

где X_i – i-я десятичная цифра уменьшаемого, Y_i – i-я десятичная цифра вычитаемого, а b_{i+1} – заимствование из более младшего десятичного разряда. (причем обратный код заимствования соответствует переносу из более младшей тетрады).

Также как и при сложении на этом этапе могут иметь место 2 случая: i-й разряд уменьшаемого X_i не меньше i-го разряда вычитаемого Y_i с учетом возможного заимствования: $X_i \ge Y_i + b_{i+1}$ и наоборот: $X_i < Y_i + b_{i+1}$.

В первом случае *i*-я тетрада разности $D_i \geq 16$. При этом возникает перенос в более старшую тетраду, уносящий с собой 16 единиц, и в *i*-й тетраде разность оказывается равной

$$D_i = X_i - Y_i - b_{i+1} + 16 - 16 = X_i - Y_i - b_{i+1},$$

т.е. определяется верно.

Перенос в более старшую тетраду и говорит об этом случае, а также об отсутствии заимствования из данной старшей тетрады и соответствует компоненте $1 - b_i$, участвующей в формировании разности в ней.

Во втором случае разность $D_i < 16$ и переноса в старшую тетраду не возникает. Сама разность при этом равна

$$D_i = X_i - Y_i - b_{i+1} + 16 ,$$

что на 6 превышает правильное значение разности, определяемое в данном случае как

$$D_i = X_i + 10 - Y_i - b_{i+1},$$

где 10 – десятка (единица более старшего десятичного разряда), заимствованная из более старшей тетрады. О заимствовании говорит именно отсутствие переноса в более старшую тетраду, обращающее в ней в 0 компоненту разности $(1 - b_{i+1})$.

Поэтому, так же как и при сложении, второй этап операции вычитания заключается в коррекции разности, полученной на первом этапе. Причем коррекция эта в точности такая же, как и при сложении: к тетрадам, из которых не было переноса в более старшую тетраду, добавляется обратный или дополнительный код шестерки.

Пример выполнения операции вычитания двух положительных двоично-десятичных чисел приведен ниже (пример 2). Также как и в примере сложения, в нем приведены варианты коррекции в обратном и дополнительном кодах.

Изложенные правила позволяют складывать и вычитать положительные двоично-десятичные числа. Для учета знаков операндов и анализа возможных случаев переполнения при сложении необходимо рассмотреть еще некоторые особенности обработки чисел со знаками.

Часто в десятичных числах под представление знака отводится целая тетрада, что вызвано стремлением исключить существование неполной старшей цифровой тетрады, которая получилась бы в случае представления знака одним разрядом в числе, занимающем целое количество байтов. Одним из наиболее распространенных вариантов является представление знака “+” четырьмя нулями.

Учет знаков операндов, как правило, производится следующим образом. Если знаки слагаемых одинаковы, то они складываются без учета знаков, и результату присваивается тот же знак, который имеют слагаемые.

Операцией сложения заменяется и операция вычитания слагаемых с разными знаками. В этом случае результату присваивается знак первого слагаемого.

Пример 2.

$X = 916$	$+\begin{array}{r}1001.0001.0110\\ 1100.1000.1011\\ \hline 0101.1010.0001\\ \text{Пер}\quad\text{Нет пер}\quad\text{Пер}\quad 1\\ \hline 0101.1010.0010\\ 1111.1001.1111\\ \hline 0101.0100.0010\\ 1\\ \hline 0101.0100.0010\end{array}$	Добавление обратного кода Y к коду X и фиксация переносов из тетрад циклический перенос Коррекция обратным кодом (+ обр. код 6 к тетрадам, где нет переноса) циклический перенос
$Y = 374$		
$D = 542$		
$D = 542$	$+\begin{array}{r}0101.1010.0010\\ 0000.1010.0000\\ \hline 0101.0100.0010\end{array}$	Коррекция дополнит. кодом (+ доп. код 6 к тетрадам, где нет переноса, с блокировкой переносов между тетрадами)

- - - - - - - - - - - - - - - - - - -

Учет знаков операндов, как правило, производится следующим образом. Если знаки слагаемых одинаковы, то они складываются без учета знаков, и результату присваивается тот же знак, который имеют слагаемые. Операцией сложения заменяется и операция вычитания слагаемых с разными знаками. В этом случае результату присваивается знак первого слагаемого.

В обоих этих вариантах следует учитывать возможность переполнения, о наличии которого будет говорить перенос из старшей цифровой тетрады.

Если же знаки слагаемых разные, то выполняется операция вычитания (Например, второго слагаемого из первого). Знак результата в этом случае полагается равным знаку уменьшаемого.

Причем, если на первом этапе вычитания будет обнаружено отсутствие переноса из старшей тетрады, то это говорит о том, что вычитаемое по модулю больше уменьшаемого: $|X| < |Y|$. В таких случаях знак результата будет обратным по отношению к знаку уменьшаемого, а полученный результат будет представлен в обратном (двоично-десятичном) коде. Для получения правильного значения разности в этой ситуации можно поступить одним из следующих способов:

а) поменять операнды местами и повторить операцию (причем это можно сделать, не доводя первое вычитание до конца, а сразу после обнаружения отсутствия переноса из старшей тетрады);

б) вычесть полученный результат из нуля (такой путь показан в примере 2а, причем первый шаг такого вычитания можно и не делать, так как добавление обратного кода к нулю этот код не изменит, и переносов между тетрадами не будет).

В любом случае первоначально установленный знак разности (по знаку первого уменьшаемого) следует поменять на обратный.

Таким образом, общий порядок сложения двоично-десятичных чисел оказывается следующим.

1. Знак результата полагается равным знаку первого операнда (слагаемого или уменьшаемого).

2. Если операнды имеют одинаковые знаки, то выполняется их (операндов) сложение без учета знаков.

Если же знаки операндов различны, то выполняется вычитание второго операнда из первого.

3. Если при выполнении сложения возникает перенос из старшей цифровой тетрады, то это говорит о переполнении суммы.

4. Если при выполнении вычитания отсутствует перенос из старшей тетрады, то это говорит о том, что уменьшаемое по модулю меньше вычитаемого. В этом случае надо поменять первоначально установленный знак результата на обратный и либо вычесть полученный результат из нуля, либо повторить операцию вычитания, поменяв операнды местами.

Пример 2а.

		Знак разности первоначально полагается равным знаку уменьшаемого "+"
$-\frac{X = 374}{Y = 916}$	$+\frac{0011.0111.0100}{0110.1110.1001}$	**Добавление обратного кода Y к коду X и фиксация переносов из тетрад**
$D = -542$	1010.0101.1101 Нет пер Пер Нет пер	
		Уменьшаемое меньше вычитаемого (Обмен их местами и повтор операции соответствуют примеру 2. Здесь далее результат вычитается из нуля.)
	$+\frac{1001.1111.1001}{0100.0101.0110}$	**Коррекция обратным кодом (+ обр. код 6 к тетрадам, где нет переноса)**
	$+\frac{0100.0101.0110}{1}$	**циклический перенос**
(457)	0100.0101.0111	
		Разность получена в двоично-десятичном обратном коде (542 + 457 = 999)
		Замена знака разности на "–"
	$+\frac{0000.0000.0000}{1011.1010.1000}$	**Добавление обратного кода разности к коду 0 и фик – сация переносов из тетрад**
	1011.1010.1000 Нет пер Нет пер Нет пер	
	$+\frac{1001.1001.1001}{0101.0100.0001}$	**Коррекция обратным кодом (+ обр. код 6 к тетрадам, где нет переноса)**
	$+\frac{0101.0100.0001}{1}$	**циклический перенос**
$D = -542$	0101.0100.0010	

Вариант структуры АЛУ для сложения двоично-десятичных чисел приведен на рис. 18. Это устройство подобно АЛУ для сложения двоичных чисел с фиксированной запятой, но включает в свой состав дополнительный регистр фиксации межтетрадных (десятичных) переносов *РДП* и схему формирования кода десятичной коррекции *СДК*, формирующую код коррекции в соответствии с сигналами десятичных переносов.

Состав микроопераций также несколько расширился и включает в себя дополнительные микрооперации, связанные с добавлением кода “6” при сложении, запоминанием переносов между тетрадами, добавлением кода коррекции и установки знака результата и его инвертирования.

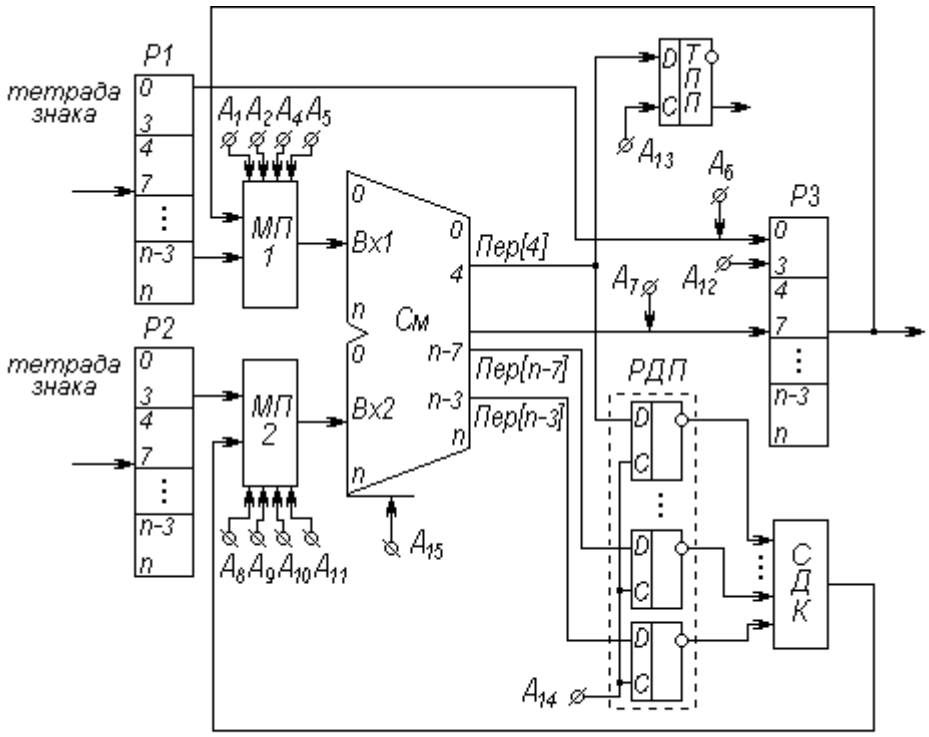

Рис. 18. Структура АЛУ для сложения двоично-десятичных чисел

Общий состав микроопераций рассматриваемого АЛУ следующий:

A_1: *Bx1* := (*P1[4 : n]*) – подача на вход *Bx1* сумматора содержимого всех тетрад регистра *P1*, кроме знаковой;

A_2: *Bx1* := ($\overline{P1[4:n]}$) – подача на вход *Bx1* сумматора инверсии содержимого всех тетрад регистра *1*, кроме знаковой;

(A_3: *Bx1* := (*P1[4 : n]*) + 0110 0110. . . 0110 – подача на вход *Bx1* сумматора увеличенного на 6 содержимого всех тетрад регистра *P1*, кроме знаковой (в рассматриваемом примере не используется));

A_4: *Bx1* := (*P3[4 : n]*) – подача на вход *Bx1* сумматора содержимого всех тетрад регистра *P3*, кроме знаковой;

A_5: *Вх1* := ($\overline{P3[4:n]}$) – подача на вход *Вх1* сумматора инверсии содержимого всех тетрад регистра *Р3*, кроме знаковой;

A_6: (*Р3[0 : 3]*) := (*Р1[0 : 3]*) – занесение знаковой тетрады регистра *Р1* в регистр *Р3*;

A_7: *Р3[4 : n])* := *ВыхСм [4 : n]* – занесение в регистр *Р3* значений тетрад всех десятичных цифр с выхода сумматора;

A_8: *Вх2* := (*Р2[4 : n]*) – подача на вход *Вх2* сумматора содержимого всех тетрад регистра *Р2*, кроме знаковой;

A_9: *Вх2* := ($\overline{P2[4:n]}$) – подача на вход *Вх2* сумматора инверсии содержимого всех тетрад регистра *Р2*, кроме знаковой;

A_{10}: *Вх2* := 0110.0110. . . . 0110 – подача на вход *Вх2* сумматора кода “6” во все цифровые тетрады;

A_{11}: *Вх2* := *Код коррекции* – подача на вход *Вх2* сумматора кода коррекции во все цифровые тетрады;

A_{12}: (*Р3[0 : 3])* := ($\overline{P3[0:3]}$) – инвертирование знаковой тетрады в регистре *Р3*;

A_{13}: (*ТПП*)*:= Пер[4]* – занесение сигнала переноса из старшей цифровой тетрады в триггер признака переполнения *ТПП*;

A_{14}: (*РДП*)*:= Пер[4]*, *Пер[8]*, . . . , *Пер[n+4]* – занесение сигналов переноса из цифровых тетрад в регистр десятичных переносов *РДП*;

A_{15}: блокировка переносов между тетрадами (при коррекции дополнительным кодом).

Для формирования кода коррекции (при коррекции обратным кодом) может быть использована схема *СДК*, показанная на рис. 19.

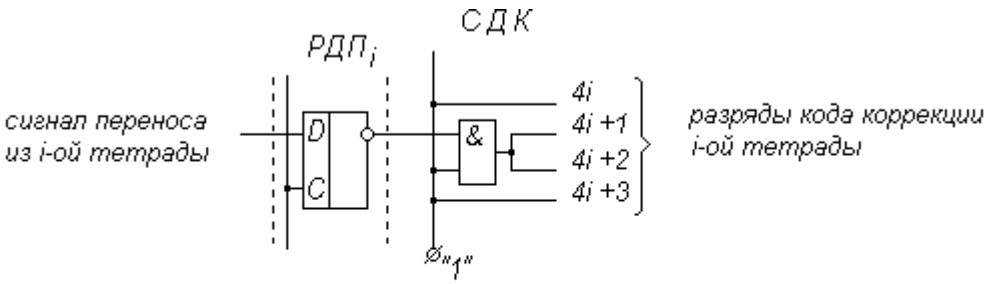

Рис.19. Возможный вариант построения схемы десятичной коррекции СДК (для одной тетрады), РДП$_i$ - *i*-ый разряд регистра десятичных переносов

Микропрограмма выполнения операции сложения двоично-десятичных чисел представлена на рис. 20.

Предполагается, что к началу операции операнды (слагаемые) находятся в регистрах *P1* и *P2*. Результат операции при нормальном ее завершении (отсутствии переполнения) размещается в регистре *P3*.

При разных знаках слагаемых в ветви вычитания в случае отсутствия переноса из старшей тетрады (условный блок *8*) первый этап операции повторяется при изменении роли операндов: операнд из второго регистра становится уменьшаемым, а операнд из регистра *P1* – вычитаемым, и подается на вход сумматора в обратном коде.

Триггеры регистра *P3*, также как и в АЛУ для сложения чисел с фиксированной запятой, показанном на рис. 13, должны быть выполнены по M-S

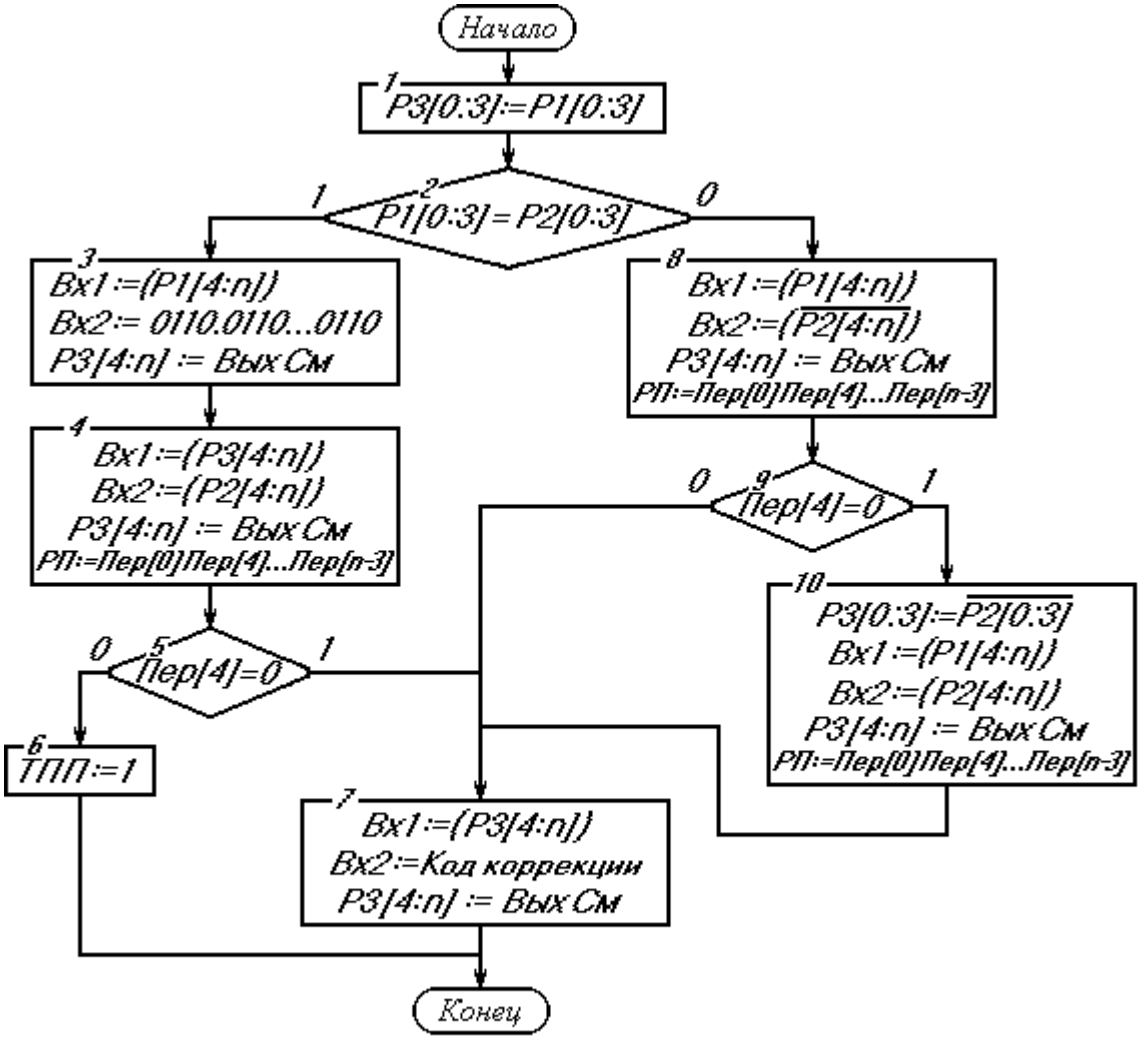

Рис. 20. Микропрограмма сложения и вычитания двоично-десятичных чисел

схеме. Это требуется потому, что в блоках *6* и *9* микропрограммы содержимое *P3* используется для подачи на вход сумматора и в этом же такте в *P3* фиксируется результат с выходов сумматора.

3.4. АЛУ для умножения чисел с фиксированной запятой

Как известно из ранее изученных дисциплин (“Дискретная математика”, “Теория автоматов”) общий ход операции умножения, хотя и подобен умножению чисел, выполняемому вручную, имеет некоторые отличия от него. Так, частичные произведения, получаемые при умножении на разряды множителя, не сохраняются до конца операции, где они суммируются на последнем этапе ее выполнения. Напротив, каждое получаемое частичное произведение сразу же подсуммируется к сумме частичных произведений, накопленной при умножении на предшествующие разряды множителя.

Из четырех возможных вариантов общей схемы умножения, различающихся порядком анализа разрядов множителя (начиная со старших или с младших разрядов) и сдвигаемым компонентой: множимым или суммой частичных произведений. (Направление сдвига при этом однозначно определяется выбранным порядком анализа разрядов множителя и сдвигаемой компонентой).

Наиболее распространен вариант умножения с анализом множителя, начиная с младших разрядов, и со сдвигом суммы частичных произведений вправо при неподвижном множимом. Схематически этот способ умножения для случая использования множимого и множителя, имеющих по *n* цифровых разрядов, показан на рис. 21.

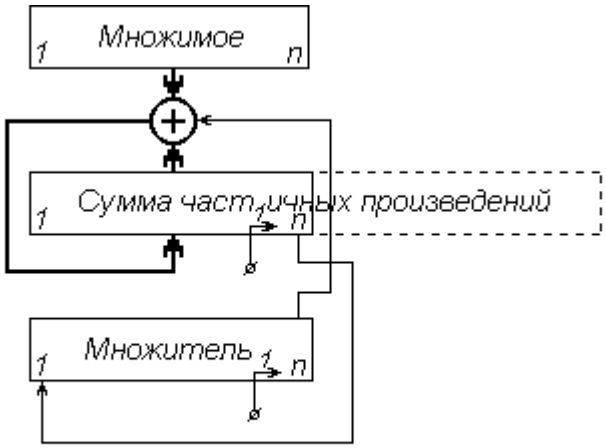

Рис. 21. Вариант структуры устройства для умножения чисел с фиксированной запятой

При такой схеме умножения выход младшего разряда регистра множителя используется для управления суммирования множителя с ранее накопленной суммой частичных произведений: единичное значение на этом выходе говорит о необходимости суммирования, нулевое – об отсутствии таковой. Регистры, в которых хранятся множитель и сумма частичных произведений, имеют цепи сдвига вправо на один разряд. Регистр суммы частичных произведений, которая в общем случае имеет двойную длину (*2n* разрядов) не обязательно должен иметь такую же разрядность. Поскольку множитель в про

цессе умножения выдвигается из регистра, то в освобождающиеся его разряды можно заносить младшие разряды суммы частичных произведений, что позволяет сделать регистр суммы такой же длины, как и регистры множимого и множителя.

Помимо собственно формирования произведения как суммы частичных произведений, необходимо также сформировать его знак и, возможно, выполнить округление.

Знак произведения определяется как сумма по модулю 2 (*mod2*) знаков сомножителей определен как до вычисления произведения, так и после этого. Сами сомножители, в случае представления их в прямом коде перемножаются без знаков (такой способ и будет рассмотрен далее).

В случае представления чисел (сомножителей и произведения) в дополнительном коде знаковый разряд множимого непосредственно участвует в умножении, а знаковый разряд множителя косвенно влияет на последний цикл умножения. (Однако такой случай здесь не рассматривается.)

Округление необходимо выполнить в том случае, если числа с фиксированной запятой рассматриваются как дробные (всегда меньшие, чем единица), а произведение должно быть представлено таким же количеством разрядов, как и сомножители. В этом варианте отбрасываются младшие n разрядов полученного $2n$-разрядного произведения, и если старший из отбрасываемых разрядов имеет единичное значение, то к младшему из остающихся разрядов произведения добавляется единица.

Напротив, если перемножаемые числа с фиксированной запятой рассматриваются как целые (больше, чем единица), то округление не требуется, но если произведение должно содержать не более n цифровых разрядов, требуется выявить возможное переполнение разрядной сетки, проверяя наличие единиц в старших n разрядах $2n$-разрядного произведения.

На рис. 22 показана структура АЛУ, реализующего рассматриваемую схему умножения для дробных двоичных чисел с фиксированной запятой, представленных в прямом коде.

Предполагается, что перед началом операции множимое и множитель размещены в регистрах *P1* и *P2* соответственно, а произведение будет расположено в регистре *P3*. Сдвиг множителя и суммы частичных произведений в рассматриваемой структуре производится в специальном блоке – сдвигателе в момент передачи через него соответствующей информации с выхода сумматора. Этот вариант рассмотрен как несколько более сложный в реализации. Если же сдвиг осуществляется непосредственно по месту хранения множителя и суммы частичных произведений, т.е. в *P2* и *P3*, то сдвигатель в схеме не нужен.

Микрооперации, выполняемые в данном АЛУ, следующие:

A_1: *СчЦ* := *n* – установка количества циклов анализа разрядов множителя в счетчике циклов;

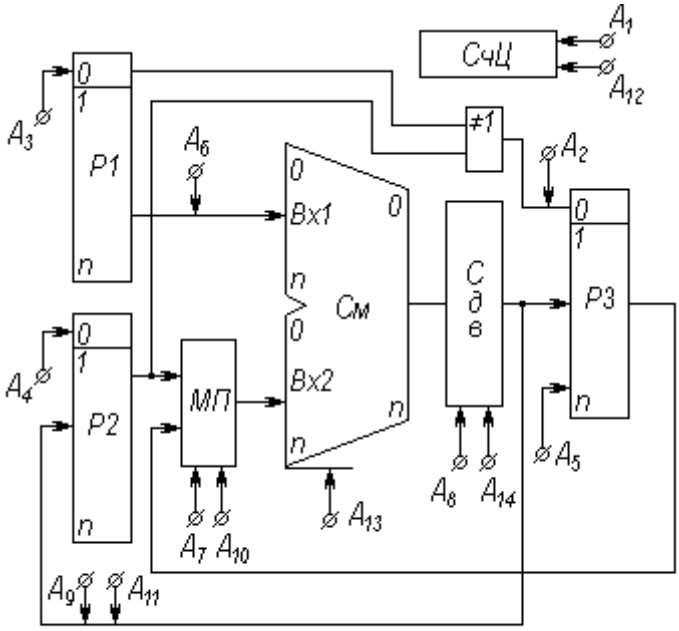

Рис. 22. Структура АЛУ для умножения чисел с фиксированной запятой

A_2: *P3[0]* := *P1[0]* ⊕ *P2[0]* – присвоение знаковому разряду регистра *P3* суммы по модулю 2 значений знаковых разрядов регистров *P1* и *P2*;

A_3: *P1[0]* := *0* – обнуление знакового разряда регистра *P1*;

A_4: *P2[0]* := *0* – обнуление знакового разряда регистра *P2*;

A_5: *P3[1 : n]* := *0* – обнуление цифровых разрядов регистра *P3*;

A_6: *Вх1* := (*P1[1 : n]*) – подача содержимого цифровых разрядов регистра *P1* (множимого) на вход *Вх1* сумматора;

A_7: *Вх2* := (*P3[1 : n]*) – подача содержимого цифровых разрядов регистра *P3* (суммы частичных произведений) на вход *Вх2* сумматора;

A_8: *P3[1 : n])* := *R1*(*ВыхСм*) – занесение в цифровые разряды регистра *P3* сдвинутой вправо на один разряд суммы частичных произведений с выхода сумматора;

A_9: *P2[0]* := (*ВыхСм[n]*) – занесение младшего разряда сдвигаемой суммы частичных произведений в 0-й (знаковый) разряд регистра *P2*;

A_{10}: *Вх2* := (*P1[0 : n]*) – подача на вход *Вх2* сумматора содержимого регистра *P2*;

A_{11}: *P2* := *R1*(*ВыхСм*) – занесение в регистр *P2* сдвинутого вправо на один разряд значения с выхода сумматора (сдвинутого множителя);

A_{12}: *СчЦ* := *СчЦ – 1* – уменьшение на единицу содержимого счетчика циклов;

A_{13}: *Вх1 := 1* – подача “1” на вход *Вх1* сумматора;

A_{14}: *Р3[1 : n]) := ВыхСм* – занесение в цифровые разряды регистра *Р3* информации с выхода сумматора.

Микропрограмма реализации в рассматриваемом АЛУ операции умножения дробных чисел с фиксированной запятой, представленных в прямом коде, приведена на рис. 23.

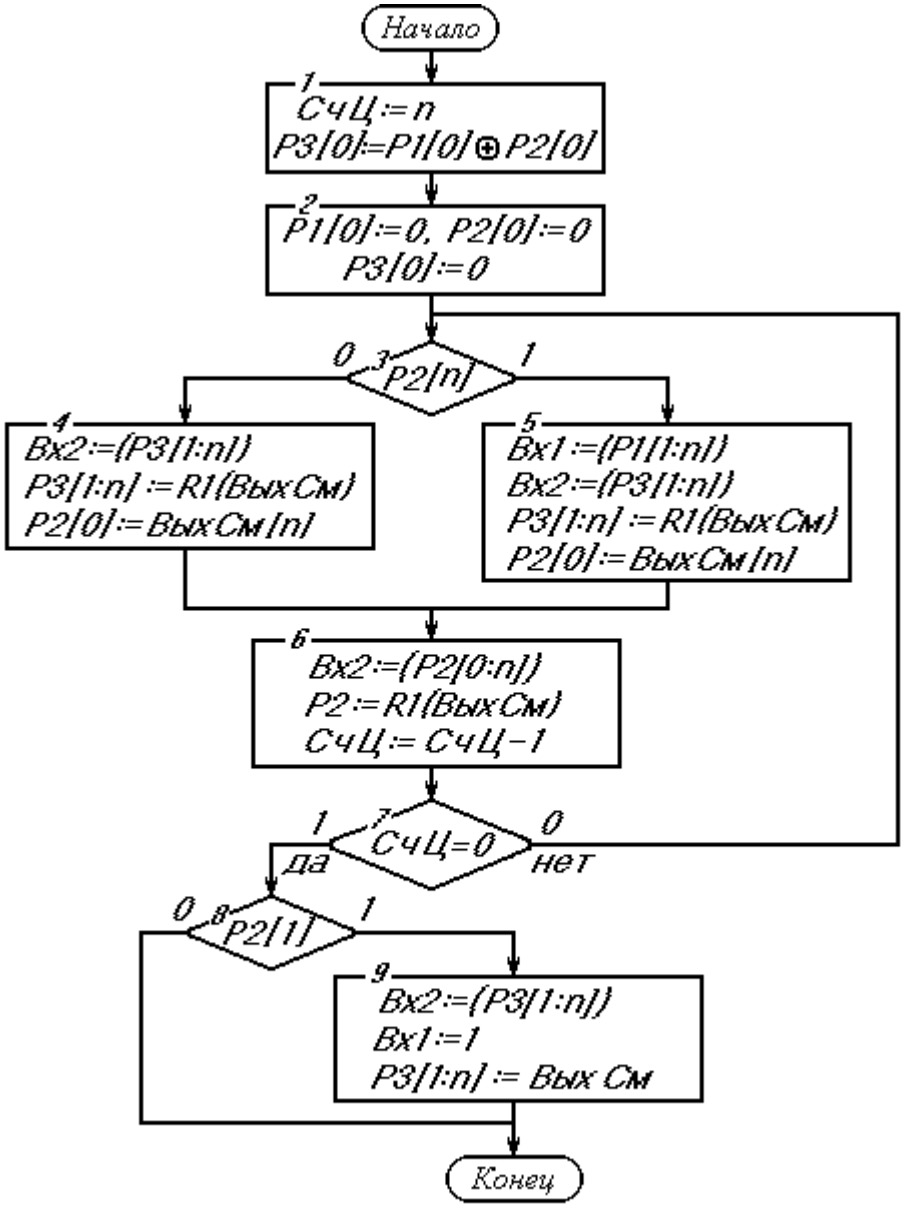

Рис. 23. Микропрограмма умножения двоичных чисел с фиксированной запятой

В блоках *1* и *2* производятся подготовительные действия: установка счетчика циклов, определение знака произведения, очистка регистра суммы частичных произведений и знаковых разрядов регистров сомножителей. Блоки *3 – 7* реализуют основной цикл анализа разрядов множителя и формирования произведения, а в блоках *8* и *9* выполняется округление.

В случае умножения целых чисел с фиксированной запятой (представленных в прямом коде) вместо округления следует осуществить проверку n старших разрядов произведения (в регистре *P3*) и, в случае наличия там значащих цифр, установить признак переполнения.

При умножении чисел, представленных в дополнительном коде, общий порядок выполнения операции умножения сохраняется. Предварительное определение знака произведения не осуществляется, а множимое при добавлении его к сумме частичных произведений передается вместе со знаком.

При сдвиге вправо суммы частичных произведений освобождающийся старший цифровой разряд заполняется значением знакового разряда, который, в свою очередь, при сдвиге сохраняет свое значение.

Если множитель положителен, то произведение сразу получается правильным. Если же множитель отрицателен, то из полученного произведения следует вычесть множимое (фактически, умножить на знаковый разряд). Последнее легко объяснимо: действительно, значение отрицательного множителя, представленного в дополнительном коде, можно записать как $2^n - Y$. Тогда сумма частичных произведений в принятой схеме умножения окажется равной $X \times (2^n - Y) = X \times 2^n - X \times Y$. Для получения правильного значения произведения $-X \times Y$ из этой суммы надо вычесть $X \times 2^n$, что и соответствует вычитанию X (добавлению дополнительного кода X с измененным знаком) из ее старших разрядов.

Ниже приведены примеры умножения двух целых чисел, представленных в дополнительном коде, для разных сочетаний знаков.

Пример 3а.

$\times \begin{matrix} X=-5 \\ Y=-6 \\ \hline P=+30 \end{matrix}$ $\times \begin{matrix} 1.11011 \text{ Дополнительный код множимого} \\ \underline{1.11010 \text{ Дополнительный код множителя}} \end{matrix}$

$$
\begin{array}{rl}
 & 0.00000 \\
+ & \underline{0.00000} \times \text{1-ю цифру множителя (0)} \\
 & \underline{0.00000} \\
+ & \underline{0.00000\,0} \text{ Сдвиг суммы вправо} \\
 & \underline{1.11011} \times \text{2-ю цифру множителя (1)} \\
 & \underline{1.11011\,0} \\
+ & \underline{1.11101\,10} \text{ Сдвиг суммы вправо} \\
 & \underline{0.00000} \times \text{3-ю цифру множителя (0)} \\
 & \underline{1.11101\,10} \\
+ & \underline{1.11110\,110} \text{ Сдвиг суммы вправо} \\
 & \underline{1.11011} \times \text{4-ю цифру множителя (1)} \\
 & \underline{1.11001\,110} \\
+ & \underline{1.11100\,1110} \text{ Сдвиг суммы вправо} \\
 & \underline{1.11011} \times \text{5-ю цифру множителя (1)} \\
 & \underline{1.10111\,1110} \\
+ & \underline{1.11011\,11110} \text{ Сдвиг суммы вправо} \\
 & \underline{0.00101} \text{ Вычитание множимого} \\
P = & 0.00000\,11110 = +30
\end{array}
$$

Пример 3б.

$$
\begin{array}{rl}
\times\begin{array}{l} X=-5 \\ Y=+6 \\ \hline P=-30 \end{array} & \times\begin{array}{l} 1.11011 \\ \underline{0.00110} \end{array} \text{ Дополнительный код множимого} \\
& \text{Дополнительный код множителя} \\
& \\
& +\begin{array}{l} 0.00000 \\ \underline{0.00000} \\ 0.00000 \end{array} \times 1\text{-ю цифру множителя (0)} \\
& +\begin{array}{l} 0.00000\,0 \\ \underline{1.11011} \\ 1.11011\,0 \end{array} \text{ Сдвиг суммы вправо} \\
& \quad \times 2\text{-ю цифру множителя (1)} \\
& +\begin{array}{l} 1.11101\,10 \\ \underline{1.11011} \\ 1.11000\,10 \end{array} \text{ Сдвиг суммы вправо} \\
& \quad \times 3\text{-ю цифру множителя (1)} \\
& +\begin{array}{l} 1.11100\,010 \\ \underline{0.00000} \\ 1.11100\,010 \end{array} \text{ Сдвиг суммы вправо} \\
& \quad \times 4\text{-ю цифру множителя (0)} \\
& +\begin{array}{l} 1.11110\,0010 \\ \underline{0.00000} \\ 1.11110\,0010 \end{array} \text{ Сдвиг суммы вправо} \\
& \quad \times 5\text{-ю цифру множителя (0)} \\
& \begin{array}{l} 1.11111\,00010 \\ P=\ 1.11111\,00010 \end{array} \text{ Сдвиг суммы вправо} \quad = -30
\end{array}
$$

Пример 3в.

$$
\begin{array}{rl}
\times\begin{array}{l} X=+5 \\ Y=-6 \\ \hline P=-30 \end{array} & \times\begin{array}{l} 0.00101 \\ \underline{1.11010} \end{array} \text{ Дополнительный код множимого} \\
& \text{Дополнительный код множителя} \\
& \\
& +\begin{array}{l} 0.00000 \\ \underline{0.00000} \\ 0.00000 \end{array} \times 1\text{-ю цифру множителя (0)} \\
& +\begin{array}{l} 0.00000\,0 \\ \underline{0.00101} \\ 0.00101 \end{array} \text{ Сдвиг суммы вправо} \\
& \quad \times 2\text{-ю цифру множителя (1)} \\
& +\begin{array}{l} 0.00010\,10 \\ \underline{0.00000} \\ 0.00010\,10 \end{array} \text{ Сдвиг суммы вправо} \\
& \quad \times 3\text{-ю цифру множителя (0)} \\
& +\begin{array}{l} 0.00001\,010 \\ \underline{0.00101} \\ 0.00110\,010 \end{array} \text{ Сдвиг суммы вправо} \\
& \quad \times 4\text{-ю цифру множителя (1)} \\
& +\begin{array}{l} 0.00011\,0010 \\ \underline{0.00101} \\ 0.01000\,0010 \end{array} \text{ Сдвиг суммы вправо} \\
& \quad \times 5\text{-ю цифру множителя (1)} \\
& +\begin{array}{l} 0.00100\,00010 \\ \underline{1.11011} \\ \end{array} \text{ Сдвиг суммы вправо} \\
& \quad \text{Вычитание множимого} \\
& \begin{array}{l} \\ P=\ 1.11111\,00010 \end{array} \quad = -30
\end{array}
$$

Следует обратить внимание, что в примере 3в, в отличие от первых двух, последний шаг – вычитание множимого – не требуется, так как множитель положителен.

3.5. АЛУ для деления чисел с фиксированной запятой

Известны два основных способа деления чисел в ЭВМ: деление с восстановлением остатка и деление без восстановления остатка. При делении двоичных чисел чаще используется первый способ, так как он является более быстрым.

Как и в случае умножения возможны различные варианты выполнения операции, но более распространено деление с неподвижным делителем и сдвигом частичного остатка (остатка получаемого при определении очередной цифры частного) влево. Это позволяет сократить разрядность используемых в АЛУ регистров.

Схематически такой способ выполнения деления можно представить в виде, показанном на рис. 24.

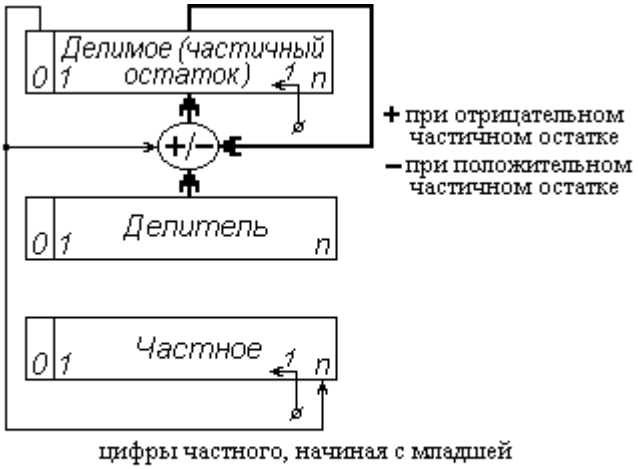

Рис. 24. Вариант структуры устройства для деления чисел с фиксированной запятой

В этой схеме цифры частного определяются по одной, начиная со старшего разряда, при каждом вычитании (или добавлении) делимого из делителя или частичного остатка от предыдущего шага.

Помимо собственно определения цифр частного, необходимо также определить знак результата и выполнить его округление.

Знак результата, как и в умножении, определяется суммированием по модулю 2 знаков делимого и делителя. Для выполнения округления определяется дополнительная цифра частного, при единичном значении которой и производится добавление единицы в младший из сохраняемых разрядов частного.

Кроме того, если делятся дробные числа с фиксированной запятой меньшие единицы, то в том случае, когда модуль делимого больше модуля делителя, модуль частного должен быть больше единицы, что не может быть представлено в разрядной сетке дробных чисел. В этом случае следует зафиксировать переполнение.

Структура АЛУ, реализующая алгоритм деления без восстановления остатка чисел с фиксированной запятой, представленных в прямом коде, приведена на рис. 25.

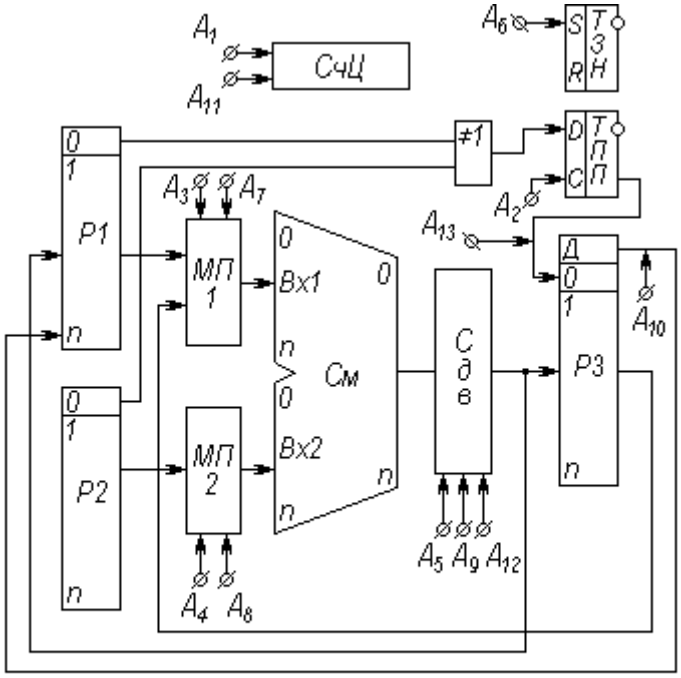

Рис. 25. Структура АЛУ для деления чисел с фиксированной запятой

В этом АЛУ реализуются следующие микрооперации:

A_1: *СчЦ* := *n* + *1* – установка количества циклов анализа разрядов множителя в счетчике циклов;

A_2: *P3[0]* := *P1[0]* ⊕ *P2[0]* – присвоение знаковому разряду регистра *P3* суммы по модулю 2 значений знаковых разрядов регистров *P1* и *P2*;

A_3: *Вх1* := (*P1[1* : *n]*) – подача содержимого цифровых разрядов регистра *P1* (множимого) на вход *Вх1* сумматора;

A_4: *Вх2* := *1*.($\overline{P2[1:n]}$)– подача на вход *Вх2* сумматора обратного кода содержимого регистра *P2* со знаком минус;

A_5: *P3* := *L1*(*ВыхСм*) – занесение в регистр *P3* сдвинутого влево на один разряд содержимого с выхода сумматора (частичного остатка, причем этот регистр увеличен на один разряд для сохранения в нем знака сдвигаемого влево остатка);

A_6: *ТПП* := *1* – установка признака переполнения в триггере переполнения;

A_7: *Вх1* := (*P3[0* : *n]*) – подача содержимого регистра *P3* (без дополнительного разряда) на вход *Вх1* сумматора;

A_8: *Вх2* := (*P2[1* : *n]*) – подача содержимого цифровых разрядов регистра *P2* (делителя) на вход *Вх2* сумматора;

A_9: *P1 := L1(ВыхСм)* – занесение в регистр *P1* сдвинутого влево на один разряд содержимого с выхода сумматора;

A_{10}: *P1[n] := #P3[Д]* – занесение в младший разряд регистра Р1 инверсного значения содержимого регистра *P3*;

A_{11}: *СчЦ := СчЦ – 1* – уменьшение на единицу содержимого счетчика циклов;

A_{12}: *P3[1 : n]) := ВыхСм* – занесение в цифровые разряды регистра *P3* информации с выхода сумматора.

A_{13}: *P3[0 : n] := ТЗН* – занесение в знаковый разряд регистра *P3* знака частного из триггера знака *ТЗН*.

Микропрограмма выполнения операции деления чисел с фиксированной запятой, меньших единицы, в этом устройстве представлена на рис. 26.

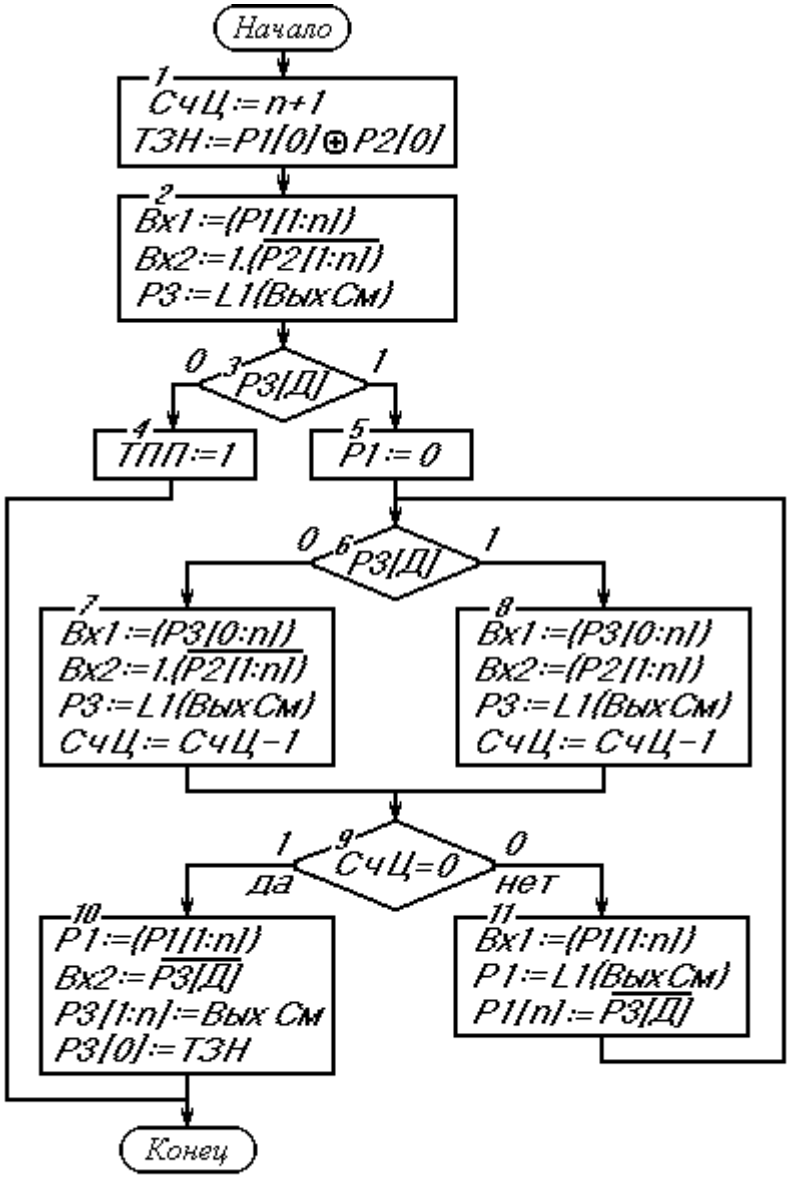

Рис. 26. Микропрограмма деления двоичных чисел с фиксированной запятой

Предполагается, что перед началом операции делимое расположено в регистре *P1*, а делитель – в регистре *P2*.

В блоке *2* микропрограммы выполняется “пробное вычитание” модуля делителя из модуля делимого с целью выявления тех случаев, когда модуль частного больше единицы. В таких ситуациях в блоке *4* регистрируется переполнение. Блоки *7*, *8* и *11* микропрограммы формируют частное в регистре *P1*, передаваемое по окончанию операции в регистр *P3*. При этом остаток, который в каждом цикле определения цифры частного формируется в регистре *P3*, теряется. Округление производится добавлением к частному значения дополнительного ($n + 1$) разряда частного же, для чего количество циклов определения его цифр увеличивается на 1, по сравнению с количеством разрядов делимого и делителя.

При делении целых чисел микропрограмма будет аналогична. Однако, в этом случае переполнение разрядной сетки произойти не может. Поэтому пробное вычитание выполняется со следующей целью.

Если модуль делимого меньше модуля делителя, то целая часть частного либо равна нулю, либо может стать равной единице после округления.

Тогда, чтобы определить, сколько разрядов будет иметь целая часть частного (а результат должен быть представлен целым числом), следует выполнять пробное вычитание, сдвигая каждый раз делитель на один разряд влево до тех пор, пока он не станет больше делителя по модулю. Количество целых цифр частного при этом предварительно принимается равным количеству сдвигов делителя, потребовавшемуся для получения отрицательного остатка при пробном вычитании.

В конце операции, при округлении, количество целых разрядов частного может либо остаться неизменным либо увеличиться на 1.

Таким образом, начало микропрограммы деления целых чисел, представленных в прямом коде, будет выглядеть так, как показано на рис. 27.

3.6. Особенности обработки смещенных порядков (характеристик) чисел с плавающей запятой

При представлении чисел с плавающей запятой могут использоваться две формы записи порядка: естественная и смещенная.

В первом случае порядок изображается целым числом со знаком, представленным в прямом коде, во втором – целым числом без знака, которое получается добавлением в исходному значению порядка некоторого смещения *D*, численно равного весу единицы старшего разряда в представлении порядка. Т.е., если разрядность порядка равна *k*, то $D = 2^{k-1}$. Представленный таким образом порядок называют иначе характеристикой.

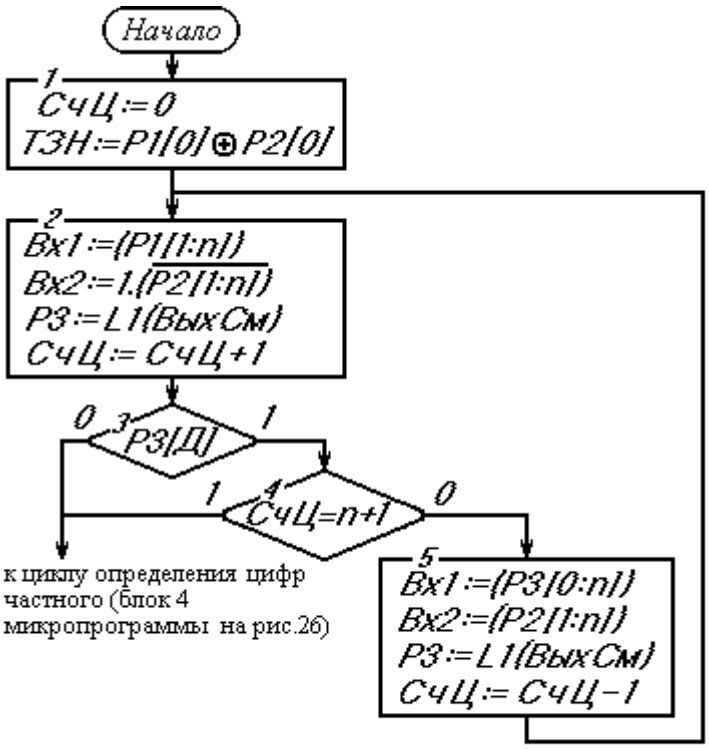

Рис. 27. Начальный участок микропрограммы деления целых двоичных чисел с фиксированной запятой

Минимальное значение характеристики, равное 0, соответствует отрицательному порядку, равному -2^{k-1}, а максимальное, равное 2^{k-1} (все единицы k-разрядного двоичного числа), соответствует положительному порядку $+2^{k-1} - 1$. Например, для характеристики, занимающей 1 байт, имеют место соотношения:

Характеристика = 00000000 Порядок = –128

Характеристика = 10000000 Порядок = 0

Характеристика = 00000000 Порядок = +127

Представление порядков в виде характеристик в определенном смысле более удобно, так как не требует непосредственного учета знаков при обработке порядков.

Например, при выполнении операции сложения чисел с плавающей запятой характеристики (как и порядки) могут изменяться только на ±1 при нормализации результата. Эти действия выполняются над характеристиками также, как и над обычными двоичными числами.

В операциях умножения и деления чисел с плавающей запятой и при выравнивании порядков в операции сложения таких чисел порядки, а следовательно и характеристики складываются или вычитаются. При этом правила

их сложения и вычитания будут отличаться от сложения и вычитания порядков, как целых чисел со знаками.

Правило сложения характеристик следующее.

Пусть p_X и p_Y – порядки чисел с плавающей запятой X и Y, а $c_X = p_X + D$ и $c_Y = p_Y + D$ (где $D = 2^{k-1}$, при разрядности порядка равной k) – их характеристики.

При сложении порядков в операции умножения чисел с плавающей запятой сумма их характеристик должна быть равна $p_X + p_Y + D$. Однако сумма S_c характеристик при непосредственном их сложении будет равна

$$S_c = c_X + c_Y = p_Y + D + p_Y + D = p_X + p_Y + 2D.$$

Тогда, учитывая ограничения разрядной сетки характеристик, надо рассмотреть два случая: $p_X + p_Y \geq 0$ и $p_X + p_Y < 0$.

В первом случае при $p_X + p_Y \geq 0$ сумма соответствующих порядкам характеристик $S_c \geq 2D$, где $2D = 2^k$ – это единица выходящая за пределы разрядной сетки характеристик. Тогда, обозначив через S_c^k значение k младших разрядов характеристики, можно записать

$$S_c^k = S_c - 2^k = S_c - 2^k - 2D = p_X + p_Y.$$

Т.е. в этом случае сумма характеристик определена с недостатком D (она должна быть равна $p_X + p_Y + D$) и для получения характеристики, соответствующей сумме порядков, к полученному значению S_c^k необходимо добавить D.

Во втором случае при $p_X + p_Y < 0$ сумма характеристик $S_c < 2D$ и следовательно она помещается в k –разрядной сетке полностью, оставаясь равной $S_c = p_X + p_Y + 2D$. Это значит, что она определена с избытком D. Тогда для получения характеристики, соответствующей порядку, равному $p_X + p_Y$, следует уменьшить полученную сумму S_c на D, т.е. добавить к ней дополнительный (обратный) код D. Но дополнительный код D равен самому D.

Поэтому для получения характеристики, соответствующей сумме порядков, необходимо к сумме характеристик добавить код D, что показано в приведенном ниже примере 4 для разрядной сетки характеристики в 8 бит (в этом случае $D = 10000000$)

При вычитании порядков характеристика c_{x-y}, соответствующая разности порядков p_{x-y}, $= p_x - p_y$, определяется аналогично. Сперва получается разность R_c характеристик вычитанием характеристики c_y из характеристики c_x, выполняемым посредством добавления к последней дополнительного кода c_y, т.е.,

$$R_c = c_X - c_Y = c_Y + (2^k - c_Y) = c_X - c_Y + 2^k = p_X + D - p_Y + D + 2^k = p_X - p_Y + 2^k.$$

Пример 4.

а)

$p_x = -7$ $\quad +c_x = 01111001$

$p_y = 9$ $\quad c_y = 10001001$

$(1)00000010$ $(+2)$ сумма порядков

$+$ Пер 10000000

10000010 характеристика порядка p_{x+y}, равного сумме порядков p_x и p_y

б)

$p_x = 7$ $\quad +c_x = 10000111$

$p_y = -9$ $\quad c_y = 01110111$

11111110 $(-2$ дополн. код$)$ сумма порядков

$+$ 10000000

Пер $(1)01111110$ характеристика порядка p_{x+y}, равного сумме порядков p_x и p_y

Здесь также возможны два случая: $p_X \geq p_Y$ и $p_X < p_Y$.

В первом случае разность $R_c \geq 2^k$ и в пределах k-разрядной сетки характеристики усекается до $R_c{}^k = R_c - 2^k$, что дает чистую разность порядков. Для получения соответствующей характеристики к этой разности необходимо, как и при сложении, добавить D.

Во втором случае: $p_X < p_Y$, – разность $R_c < 2^k$ и полностью вмещается в свою k-разрядной сетку, т.е.

$$R_c = p_X - p_Y + 2^k = p_X - p_Y + 2D.$$

Тогда для получения характеристики, соответствующей разности порядков, R_c необходимо уменьшить на D, добавив к ней дополнительный код D (равный D).

Таким образом, как и при сложении, для получения характеристики, соответствующей разности порядков, необходимо к разности порядков добавить код D, что показано ниже в примере 5.

Для выявления случаев переполнения (“положительного” и “отрицательного”), возможного как при сложении так и при вычитании характеристик, можно использовать следующее правило.

Если при сложении/вычитании характеристик есть перенос из старшего разряда результата суммирования кодов, а значение этого разряда равно “1”, то имеет место “положительное” переполнение разрядной сетки характеристики. Если же перенос из старшего разряда отсутствует, а его значение равно “0”, то имеет место “отрицательное” переполнение разрядной сетки характеристики. В остальных случаях переполнение отсутствует.

Пример 5.

а)

$p_x = -7$ $\quad + \quad c_x = 01111001$ $\qquad p_y = 9 \rightarrow c_y = 10001001$

$p_y = 9$ $\quad$ ДК $c_y = 01110111$ $\qquad$ дополн.код $c_y = 01110111$

11110000 (−16 дополн.код) разность порядков

$+_{\text{Нет пер}}\ 10000000$

01110000 характеристика порядка p_{x-y}, равного разности порядков p_x и p_y

б)

$p_x = 7$ $\quad + \quad c_x = 10000111$ $\qquad p_y = -9 \rightarrow c_y = 01110111$

$p_y = -9$ $\quad$ ДК $c_y = 10001001$ $\qquad$ дополн.код $c_y = 10001001$

$(1)00010000$ (+16) разность порядков

$+_{\text{Пер}}\ 10000000$

10010000 характеристика порядка p_{x-y}, равного разности порядков p_x и p_y

ЛИТЕРАТУРА

1. **Автоматизированное** проектирование цифровых устройств / С. С. Бадулин, Ю. М. Барнаулов, В. А, .Бердышев и др. – М.: Радио и связь, 1981. – 240 с.
2. **Акаев А. А., Майоров С. А.** Оптические методы обработки информации. – М.: Высш. школа, 1988. – 237 с.
3. **Анкудинов Г. И.** Синтез структуры сложных объектов: логико-комбинаторный подход. – Л.: ЛГУ, 1986. – 260 с.
4. **Балашов** Е. **П.** Эволюционный синтез систем. – М.: Радио и связь, 1985. – 328 с.
5. **Бухштаб А. И., Каневский Е. А., Хохлов Л. М.** Электронные клавишные вычислительные машины. – Л.: Энергия, 1974. – 160 с.
б. **Глушков В. М.** и др. Вычислительные машины с развитыми системами интерпретации. – Киев: Наукова думка, 1970. – 260 с.
7. **Глушков В. М., Капитонова Ю. В., Летичевский А. А.** Автоматизация проектирования вычислительных машин. – Киев: Наукова думка, 1975. – 230 с.
8. **Интеллектуальные** системы автоматизированного проектирования больших и сверхбольших интегральных схем: В. А. Мищенко, Л. М. Городецкий, Л. И. Гурский и др. – М.: Радио и связь, 1988. – 272 с.
9. **Каган Б. М.** Электронные вычислительные машины и системы: Учеб пособие для вузов. – М.: Энергоатомиздат, 1985. – 552 с.
10. **Киносита К., Асада К., Карацу** 0. Логическое проектирование СБИС. – М.: Мир, 1988. – 309 с.
11. **Мурога С.** Системное проектирование сверхбольших интегральных схем: 2х кн. Кн. 1. – М.: Мир, 1985. – 288 с.
12. **Николаев В. И., Брук В. М.** Системотехника: методы и приложения. – Л.: Машиностроение, 1985. – 199 с.
13. **Озкарахан Э.** Машины баз данных и управление базами данных. – М.: Мир, 1989. – 696 с.
14. **Основы** теории вычислительных систем: Учеб. пособие / Под ред. С. А. Майорова. – М.: Высш. школа, 1978. – 408 с.
15. **Проектирование** цифровых вычислительных машин: Учеб. пособие для вузов / Под ред. С. А. Майорова. – М.: Высш. школа, 1972. – 344 с.
16. **Савельев А. Я.** Прикладная теория цифровых автоматов: Учеб. для вузов по спец.: ЭВМ. – М.: Высш. школа, 1987. – 272 с.
17, **Самофалов К. Г., Корнейчук В. И., Тарасенко В. П.** Цифровые ЭВМ. Теория и проектирование.– Киев: Вища школа, 1983. – 456 с.
18. **СБИС** для распознавания образов и обработки изображений: Под ред. К. Фу.– М.: Мир, 1988. – 248 с.
19. **Теория** и проектирование ЭВМ и систем: Методические указания к выполнению курсового проекта. – Л.: СЗПИ, 1988. – 44 с.

20. **Чу Я.** Организация ЭВМ и микропрограммирование. – М.: Мир, 1975 .– 592 с.