Министерство образования Республики Беларусь

Учреждение образования
Белорусский государственный университет
информатики и радиоэлектроники

В. П. Качков, И. Я. Доморадов, Р. Е. Сердюков

АССОЦИАТИВНАЯ ПАМЯТЬ И АССОЦИАТИВНЫЕ ПРОЦЕССОРЫ В ИНТЕЛЛЕКТУАЛЬНЫХ КОМПЬЮТЕРАХ

Рекомендовано УМО вузов Республики Беларусь по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для студентов учреждений, обеспечивающих получение высшего образования по специальности «Искусственный интеллект»

Под научной редакцией профессора В. В. Голенкова

Минск БГУИР 2009

УДК 004.382(076)
ББК 32.973я7
К30

Рецензенты :

кафедра информатики и вычислительной техники
Высшего государственного колледжа связи
(заведующий кафедрой, кандидат технических наук Е. В. Новиков);

заведующий отделом Объединенного института проблем
информатики НАН Беларуси, кандидат технических наук
Н. Н. Парамонов

Качков, В. П.

К30 Ассоциативная память и ассоциативные процессоры в интеллектуальных компьютерах : учеб.-метод. пособие / В. П. Качков, И. Я. Доморадов, Р. Е. Сердюков ; под науч. ред. В. В. Голенкова. – Минск : БГУИР, 2009. – 188 с.

ISBN 978-985-488-176-8

Рассматриваются программные и аппаратные способы организации ассоциативной обработки информации в вычислительных системах на теоретическом и практическом материале.

Для студентов специальности «Искусственный интеллект», изучающих ассоциативную память и ассоциативные процессоры интеллектуальных компьютеров в рамках дисциплины «Аппаратное обеспечение интеллектуальных систем». Будет полезно и для студентов других специальностей в области информационных технологий.

УДК 004.382(076)
ББК 32.973я7

ISBN 978-985-488-176-8

СПИСОК СОКРАЩЕНИЙ

АЗУ – ассоциативное запоминающее устройство

АЛУ – арифметико-логическое устройство

АП – ассоциативный процессор

АУ – арифметическое устройство

АУП – ассоциативный управляющий переключатель

БД – база данных

БИС – большая интегральная схема

БП – буферная память

ВС – вычислительная система

ВТ – вычислительная техника

ДПСЧ – датчик псевдослучайных чисел

ДСЧ – датчик случайных чисел

ЗУ – запоминающее устройство

ИИ – искусственный интеллект

ИС – импульс синхронизации

КП – конечный пользователь

КлСл – ключевое слово

ЛЗ – линия задержки

ЛП – локальная память

ЛПМ – лентопротяжный механизм

МД – магнитный диск

МЛ – магнитная лента

МП – матричный процессор

НМД – накопитель на магнитном диске

ОБ – операционный блок

ОП – основная память

ОУ – операционный узел

ОЭ – обрабатывающий элемент

ПАС – память с адресацией по содержанию

ПБД – процессор баз данных

ПРЛ – память с распределенной логикой

ПОБ – память операционного блока

ПТ – плавающая точка

ПрАдр – пространство адресов

ПфУ – периферийное устройство

ПЭВМ – персональная электронная вычислительная машина

РЛС – радиолокационная станция

СБИС – сверхбольшая интегральная схема

ТС – техническое средство

ТХ – таблица хеширования

ФП – функциональная память

ЦП – центральный процессор

ЭБ – элементная база

ЭД – элемент данных

ЭП – элементарный процессор

ЯВУ – язык высокого уровня

В В Е Д Е Н И Е

Данное учебное издание рекомендуется для студентов специальности «Искусственный интеллект», изучающих ассоциативную память и ассоциативные процессоры в интеллектуальных компьютерах по дисциплине «Аппаратное обеспечение интеллектуальных систем», и может быть полезно для студентов других специальностей в области информационных технологий.

Издание включает в себя теоретическую часть и лабораторный практикум.

В результате изучения дисциплины студенты должны освоить теоретические основы построения, логические и программные способы реализации памяти с адресацией по содержанию (называемой *ассоциативной памятью*), знать тенденции развития ассоциативных средств хранения и обработки информации, типы ассоциативных процессоров, а также применение принципов ассоциативного поиска в интеллектуальных компьютерах.

Выполняя лабораторные работы по данной дисциплине, студенты должны научиться строить и анализировать программные модели ассоциативной памяти и ассоциативных процессоров.

Изучение изложенного в издании материала базируется на сведениях из дисциплин: «Высшая математика», «Теория вероятностей и математическая статистика», «Основы алгоритмизации и программирования», «Конструирование программ и языки программирования», «Физика», «Электротехника», «Электронные приборы», «Организация и функционирование традиционных и интеллектуальных компьютеров».

Все рассматриваемые в издании вопросы связаны с ассоциативным способом доступа к информации в запоминающей среде (памяти), базирующемся на механизме ассоциации. В данном случае ассоциация трактуется как взаимосвязь между информацией (образом) на входе запоминающей среды и информацией (образом), хранящейся в запоминающей среде [5].

Ассоциативный способ позволяет преодолеть многие ограничения, присущие адресному доступу к памяти (в котором при всех обращениях по одному адресу выдается один и тот же ограниченный объем информации вне зависимости от содержания), за счет задания некоторого критерия отбора и проведения необходимых преобразований только над теми данными, которые удовлетворяют этому критерию. Критерием может быть совпадение с любым элементом данных, достаточным для выделения искомых данных из всех имеющихся. Поиск

данных может происходить по фрагменту, имеющему большую или меньшую корреляцию с заданным элементом данных.

В зависимости от полноты реализации модели ассоциативной обработки различаются несколько подходов. Если реализуется только ассоциативная выборка данных с последующим поочередным использованием найденных данных, то говорят об ассоциативной памяти или памяти, адресуемой по содержанию. При достаточно полной реализации всех свойств ассоциативной обработки используется термин «ассоциативный процессор».

В [5] отмечаются основные отличия ассоциативного способа доступа к информации от адресного. Ассоциативный способ обеспечивает:

– практически одновременный доступ ко всей хранящейся в памяти информации;

– относительную независимость времени поиска информации от емкости памяти;

– внесение элементов обработки информации непосредственно в процесс самого доступа;

– обработку информации непосредственно в среде ее хранения.

Рассмотрению ассоциативных средств хранения и обработки информации и посвящено данное учебное издание.

Издание соответствует учебной программе по дисциплине «Аппаратное обеспечение интеллектуальных систем» Государственного образовательного стандарта специальности «Искусственный интеллект» и разработано по материалам источников, указанных в разделе «Литература».

Авторы выражают благодарность Н. Н. Парамонову, Г. К. Афанасьеву, М. М. Татуру, Ф. И. Брудно за полезные советы, замечания и содействие при подготовке пособия, а также Н. В. Качковой за большую работу по оформлению издания.

1. АССОЦИАТИВНАЯ ПАМЯТЬ. ОПРЕДЕЛЕНИЯ И КОНЦЕПЦИИ

1.1. Обстоятельства, способствующие развитию ассоциативных средств хранения и обработки информации

Современные системы обработки данных требуют все более высокого быстродействия, однако на пути решения этой проблемы стоит серьезное противоречие, характерное для существующих систем и заключающееся в разрыве быстродействия памяти и процессора: быстродействие процессора на порядок превышает быстродействие памяти из-за более сложной организации последней, в результате чего память не успевает обеспечивать процессор необходимой ему информацией в нужном темпе. За счет иерархической организации памяти, опережающей выборки информации из памяти, использования кэш-памяти в определенной мере удавалось ускорить подачу информации, однако это требовало огромных затрат вычислительных ресурсов (до 50 % от общих ресурсов). Наиболее эффективным способом разрешения этого противоречия считается совмещение функций хранения и обработки информации с использованием ассоциативного метода доступа [2, 5].

Основными вычислительными системами, в которых применяются ассоциативные средства хранения и обработки информации, являются интеллектуальные системы для решения ряда задач: распознавания и анализа образов, сцен и ситуаций; обработки изображений; распознавания и синтеза речи; высокопроизводительных параллельных вычислений; обработки нечеткой информации; принятия решений в условиях неопределенности; в базах данных и знаний; в системах машинного перевода и логического вывода. Эти задачи требуют при их решении на универсальных ЭВМ значительных вычислительных ресурсов и параллелизма обработки, который характерен для ассоциативных запоминающих устройств (АЗУ).

Требования к ассоциативной памяти по скорости, объему и структуре постоянно возрастают. Если ранее потребности большинства архитектур систем обработки данных удовлетворялись частично ассоциативными запоминающими устройствами, то сейчас появляется все большая необходимость в полностью ассоциативных запоминающих устройствах большой емкости и высокого быстродействия (ассоциативных процессорах) [4].

Частично ассоциативны запоминающие устройства проектируются обычно в виде СБИС на основе стандартных запоминающих элементов. Запись и хранение данных в них осуществляется путем жесткого программирования маскированием при изготовлении или путем задания соответствия между ячей-

ками признаков и областями памяти, что является недостатком, так как не позволяет проводить какие-либо изменения данных в АЗУ во время вычислений. Разные типы таких устройств различаются запоминающей средой и схемотехникой логического сравнения и управления. Такие АЗУ позволяют создавать модули ассоциативной памяти емкостью до 1 Гбайта.

В полностью ассоциативных запоминающих устройствах операции сравнения реализуются непосредственно в самих ячейках памяти, что позволяет достичь исключительно высокого быстродействия при ассоциативной выборке данных, хотя при этом увеличиваются затраты на хранение одного бита информации. На такие АЗУ возлагаются основные надежды при создании интеллектуальных систем. Их разработка ориентируется на использование новых системотехнических и технологических принципов проектирования (субмикронные и нанотехнологии, 2D- и 3D-размещение элементов на кремниевой пластине, самотестирование с возможностью реконфигурации и др.).

В настоящее время из-за отсутствия стандарта разрабатываются заказные варианты АЗУ, специализированные для конкретных приложений, что не гарантирует их высокорентабельное серийное производство. Устойчивый рынок может обеспечить применение АЗУ в качестве кэш-памяти, которая сокращает время реакции системы, уменьшает разрыв в быстродействии центрального процессора и основной памяти и является одним из основных способов повышения производительности современных ЭВМ.

Перспективным для реализации интеллектуальных интерфейсов, логического вывода, обработки изображений, распознавания образов считается создание АЗУ, построенных по принципам нейронной сети, которые рассматриваются как распределенная сеть с внутренними обратными связями между ассоциативными ячейками [5].

1.2. Определение и модель ассоциативной памяти

Ассоциативная память может быть определена [5] как система для записи, хранения, поиска, обработки и считывания информации, в которой данные (знания) об объекте могут быть инициализированы (дополнены) по заданному фрагменту этих данных (знаний), используемому в качестве поискового.

В соответствии с определением ассоциативная память решает следующие задачи:

– соотнесение поисковой информации с хранимой и дополнение ее до точного описания объекта, т.е. всей информации, которая доступна ассоциативной памяти;

– коррекция поисковой информации относительно всего объема информации, хранимой в ассоциативной памяти, выделение недостоверной информации и на основе оставшейся – решение первой задачи.

Простейшая модель ассоциативной памяти показана на рис.1.1 [5].

Модель состоит из ассоциативной логико-запоминающей среды, которая связана с двумя каналами ввода и одним каналом вывода информации.

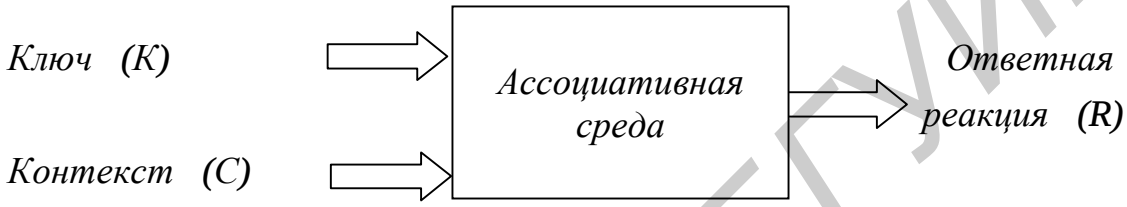


Рис. 1.1. Модель ассоциативной памяти

При записи на вход *K* подается входная информация, на вход *C* – признаковая информация, представляющая *контекст*, в котором входная информация записывается в память (контекст – сопутствующие обстоятельства).

При ассоциативной выборке по ключу (или его фрагменту) на входе *K* на выходе памяти формируется ответная реакция *R*, связанная с ключом *K*. Это значит, что для выбора записанной в память информации может быть использован любой ее фрагмент в качестве поискового. Подлежащую выборке информацию можно конкретизировать точнее путем задания различного контекста *C*.

Возможны и другие, более сложные модели ассоциативной памяти: например, с обратной связью выхода на третий вход памяти, с несколькими входами и выходами, с совмещенной записью и выборкой и др.

Логико-запоминающая среда является носителем информации. Развитие вычислительной техники сопровождалось разработкой запоминающих сред и запоминающих элементов (ЗЭ), наиболее удовлетворяющих требованиям, предъявляемым к памяти ЭВМ. Это и магнитные ЗЭ (ферриты, ленты, барабаны, диски, цифровые домены), и ламповые, и полупроводниковые, и приборы с зарядовой связью, и многие специальные (оптические, сверхпроводящие, хими-

ческие, биологические, молекулярные, голографические среды, нейронные сети) и др.

Наиболее распространены сейчас схемы памяти на полупроводниковых элементах.

Функционально ассоциативные памяти различаются способом записи, опроса, поиска и выборки результатов опроса.

Запись может быть или по адресу ячейки, или в ячейки памяти, в которых находятся слова с ассоциативными признаками, совпадающими с признаками опроса.

Способы опроса зависят от схемной реализации и могут осуществляться:

– последовательно по словам и параллельно по разрядам;
– параллельно по словам и последовательно по разрядам;
– параллельно по словам и параллельно по разрядам;
– параллельно по группе разрядов и последовательно по группам;
– последовательно по группе разрядов и параллельно по группам.

Ассоциативный поиск может выполняться по полному или частичному совпадению признака опроса и ассоциативного признака хранимой информации. Сюда входит ассоциативный поиск слов с максимальным или минимальным значением ассоциативного признака, поиск слов, численное значение ассоциативного признака которых больше или меньше некоторой заданной величины или находится в заданных (или вне) пределах, поиск слов, двоичное представление которых содержит максимальное число разрядов, совпадающих с разрядами поискового аргумента и др.

Наиболее распространенной выборкой результатов ассоциативного поиска из памяти является упорядоченная выборка. Она производится в порядке возрастания (убывания) выделенных ассоциативных признаков и состоит из циклически повторяющихся операций поиска слов с максимальным (минимальным) численным значением ассоциативного признака. Методы упорядоченной выборки разделяются на алгоритмические и аппаратные. Алгоритмические методы состоят в том, что выборка слов производится за серию опросов, причем каждый признак следующего опроса вырабатывается в зависимости от результатов предыдущего опроса. Аппаратные методы отличаются тем, что при выработке серии опросов не учитываются результаты предыдущих опросов.

Методы и алгоритмы поиска и упорядоченной выборки информации в ассоциативной памяти, а также сравнительные данные об их эффективности (по необходимому числу обращений к памяти для выборки *m n*-разрядных слов) подробно рассмотрены в [5].

Для обозначения ассоциативной памяти применялось большое число терминов, например [3]: память, адресуемая по содержанию (ПАС), ассоциативное запоминающее устройство (АЗУ), память с адресацией по данным, память-каталог, файл с быстрой выборкой, файл с параллельным поиском, память с параллельным поиском и др.

Пример наиболее простого варианта ассоциативной памяти, называемого *памятью-каталогом* [3], приведен на рис.1.2.

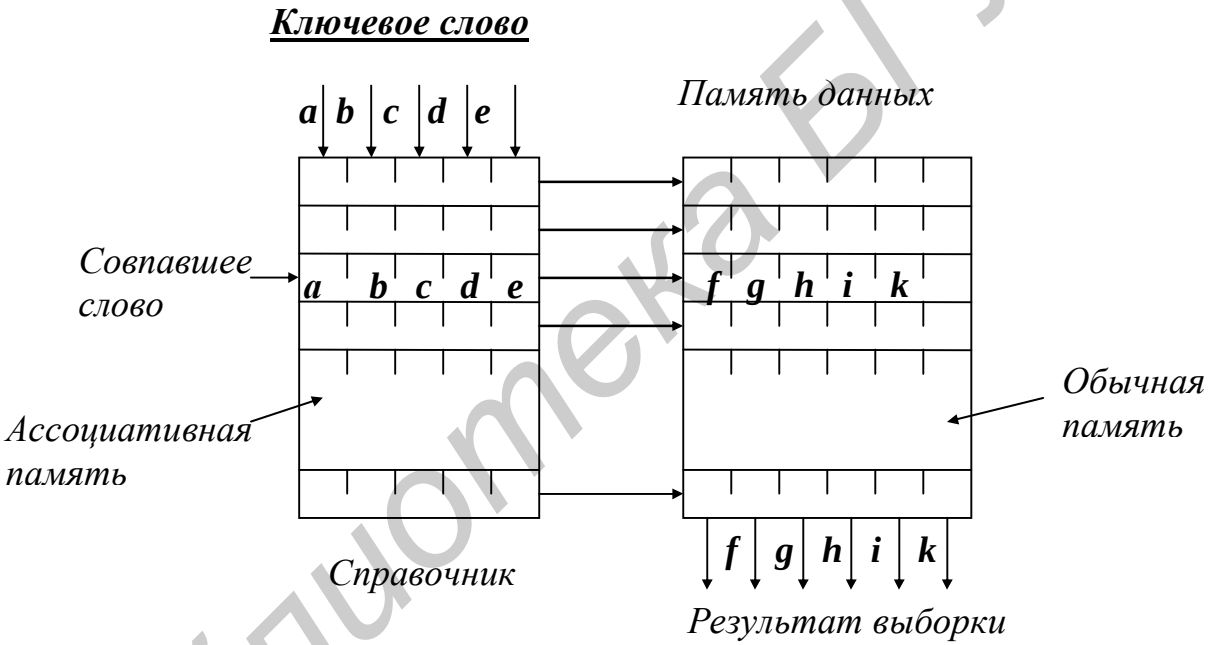


Рис. 1.2. Память-каталог

Поданная на вход справочника ключевая информация (*ключевое слово*) сравнивается одновременно со всеми словами, записанными в справочнике.

Каждая ячейка справочника представляет собой регистр для хранения одного слова. Он дополнен комбинационными схемами (КС) для сравнения содержимого регистра с ключевым словом. В каждой ячейке справочника имеется

выходная линия, которая возбуждается при совпадении ее содержимого с ключевым словом.

Память данных – это обычная память произвольного доступа с линейной выборкой. Выходные шины справочника играют роль адресных линий памяти данных.

На рис. 1.2 показаны ячейка справочника, записанное слово в которой совпало с ключевым, и выбранная в результате совпадения ячейка памяти данных, содержимое которой как результат выборки появляется на выходе памяти данных.

Показанная на рис. 1.2 память может быть дополнена схемами для маскирования отдельных частей ключевой информации и для последовательного считывания всех слов памяти данных, которым соответствуют выбранные адресные линии.

1.3. Ассоциации

1.3.1. Понятие ассоциаций

Все существующие концепции ассоциативной памяти предполагают наличие следующих компонентов [5]:

- ассоциативной среды – носителя информации;
- множества записанных в памяти информационных объектов;
- структуры взаимосвязей между информационными объектами;
- механизма информационных взаимодействий в ассоциативной среде.

Это позволяет трактовать понятие ассоциации в рамках двух подходов.

При первом подходе ассоциации между информационными объектами (образами) трактуются как некая абстрактная структура взаимозависимостей, неявно закодированная в информационных объектах и соответствующих связях между ними или в формах их представлений, при втором – ассоциации рассматриваются как коллективные или интегральные изменения в запоминающей среде, являющейся носителем информации, и отражают конкретные свойства этой среды.

Первый подход определяет ассоциации как отношения между объектами. Признаки этих отношений могут характеризовать свойства объектов, действия над ними, подчиненность и др. В этом контексте понятие ассоциации отражает наличие взаимосвязей между данными и не имеет отношения к самому механизму хранения информации.

При втором подходе ассоциативные свойства логико-запоминающей среды рассматриваются с точки зрения возможности коллективного доступа ко всей распределенной в среде информации, а также параллельной обработки и одновременного преобразования данных непосредственно в ассоциативной среде.

1.3.2. Виды ассоциаций

Ассоциируемые объекты представляются либо *прямыми*, либо *косвенными (непрямыми)* ассоциациями.

Прямые ассоциации

Прямые ассоциации являются простейшим типом ассоциаций. Они могут быть сформированы путем установления непосредственной связи (физической или логической) между хранимыми в памяти представлениями двух или большего числа объектов или событий.

В ЭВМ объекты и события хранятся в форме дискретных представлений. Эти представления также могут образовывать ассоциации, наличие которых обусловлено их логической взаимосвязью. Они могут описывать различные свойства одного и того же объекта или объединяться принудительно. К первым можно отнести документы, состоящие из ряда связанных друг с другом атрибутов. Примером второго типа могут служить элементы разного рода таблиц, словарей, справочников.

Прямые ассоциации по способу представления в виде функциональной зависимости могут инициироваться либо логически детерминированной последовательностью, либо на основе ассоциативной связи посредством прямых указателей.

Записать прямую ассоциацию, состоящую из дискретных элементов или присвоенных им значений, можно в виде *упорядоченного набора*, который в памяти ЭВМ хранится в форме *списка* (будет рассмотрено в разд. 2). Прямые ассоциации в машине могут представляться разными способами, в основе каждого из которых лежит некоторый автоматический механизм доступа к содержимому памяти.

Способы хранения представлений ассоциируемых объектов в памяти ЭВМ приведены на рис.1.3 [3].

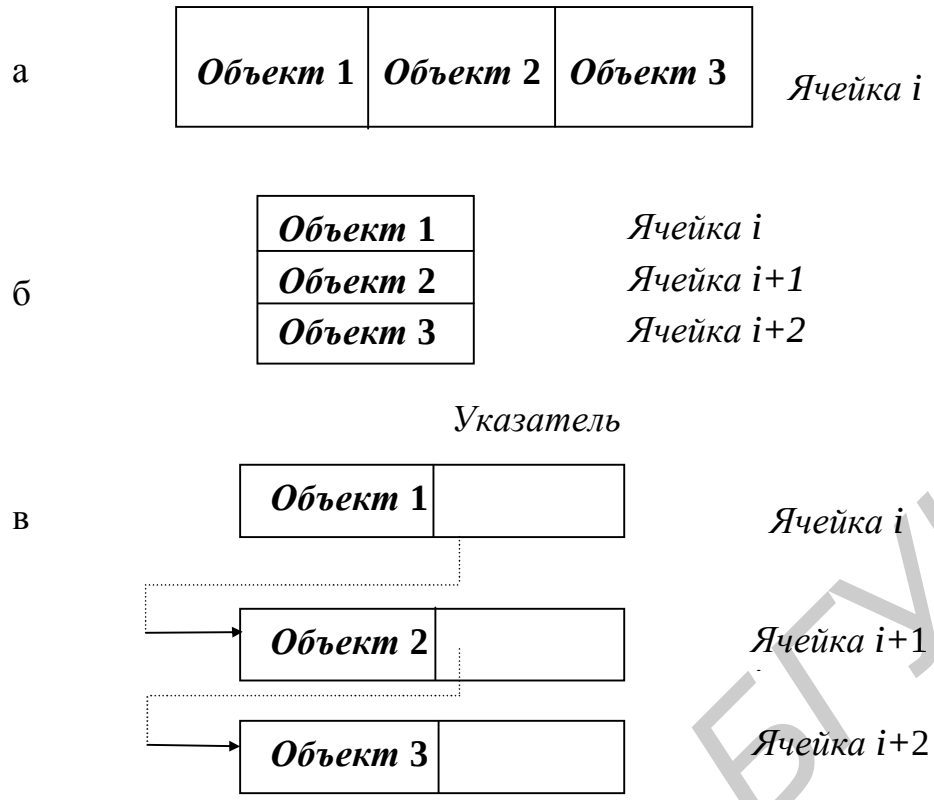


Рис. 1.3. Способы хранения представлений ассоциируемых объектов в памяти ЭВМ:
а – ассоциируемые объекты в одной ячейке; б – ассоциируемые объекты
в последовательно расположенных соседних ячейках;
в – ассоциируемые объекты, расположенные в ячейках, связанных указателями

Непрямые (или косвенные) ассоциации

Существование связей между объектами может также обнаружиться и в том случае, когда их представления объединены косвенно посредством некоторой цепочки перекрестных ссылок, наличие которой проявляется лишь на этапе выборки информации. Для этого в памяти должно быть несколько прямых ассоциаций, в состав которых входят общие или идентичные объекты. Например, пусть независимо сформированы и занесены в память упорядоченные наборы элементов (*A, B, C*) и (*C, D, E*). Если первоначально между, например, элементами *B* и *E* не было никакой связи, то после занесения в память их уже нельзя считать независимыми, так как используя, к примеру, в качестве поискового элемент *A*, можно извлечь из памяти связанные с ним элементы *B* и *C*, а по *C* – элементы *D* и *E*. Таким образом, связь между объектами *B* и *E* установлена.

Связи между объектами, формируемые на этапе выборки, называются *непрямыми* (*косвенными*) *ассоциациями*.

Существуют и другие способы представления связанных элементов данных и методы их поиска.

Отношение

Элементы $x \in E$ и $y \in F$ связаны бинарным *отношением* R, если пара (x,y) принадлежит R, где R – определенное подмножество множества всех допустимых пар элементов, т.е. множества E x F (прямого произведения E и F).

Такое отношение между x и y обозначается следующим образом:

$x\,R\,y$ или $x \xrightarrow{R} y$.

Приведенное выше абстрактное определение можно пояснить на примере простых семантических отношений. Пусть x и y – существительные, а R – некоторое сочетание слов. Рассмотрим две совокупности конкретных примеров x и y [1].

x: Статья A, статья B, книга C;

y: искусственный интеллект, распознавание образов, цифровые электронные схемы.

Приведем, например, две осмысленные пары из (x,y):

Статья A посвящена проблеме распознавания образов;

Книга C посвящена проблеме искусственного интеллекта.

Конструкция «посвящена проблеме» определяет тип отношения, т.е. выполняет функцию R (имеется, конечно, и множество других вариантов описания R). В памяти ЭВМ отношение можно представить упорядоченной тройкой вида (A, R, B), например, («статья A», «посвящена проблеме», «распознавание образов»). Приведенная запись по форме совпадает с ассоциацией элементов A, R и B.

Рассмотрим методы выборки отношений из памяти.

Поиск отношений с использованием адресации по содержанию

Этот способ среди многих других наиболее быстродействующий и дает возможность непосредственно локализовать искомый элемент в группе элементов, объединенных в список. При этом не требуется проводить сортировку информации, хранящейся в памяти. В наибольшей степени достоинства этого метода поиска проявляются при работе с большими базами данных.

Для ассоциаций косвенного типа, как следует из приведенного ранее примера, необходимо иметь возможность выборки прямой ассоциации по любому входящему в ее состав элементу.

Для выборки одного и того же отношения в рассмотренном выше примере могут применяться поисковые аргументы семи видов: A, R, B, (A, R,), (R, B), (A, B), (A, R, B).

При записи информации с использованием программных методов (хеширования) в памяти приходится хранить по семь копий всех отношений, которые заносятся в отдельные области памяти или таблицы, по одной для каждого типа поискового аргумента.

Выборка отношений существенно упрощается при хранении троек вида (*A* ,*R*, *B*) в ячейках памяти, адресуемой по содержанию. Представляющий собой любую из перечисленных выше комбинаций поисковый аргумент формируется путем маскирования отдельных информационных полей. Он должен совпасть с соответствующим фрагментом отношения. Для выборки отношения целиком достаточно выполнить всего одну операцию параллельного считывания.

Атрибуты и дескрипторы

В упорядоченном наборе позиция элемента определяет выполняемую им функцию, помещающийся в позиции атрибута элемент является конкретным значением атрибута. Значения атрибутов могут быть как числовыми, так и нечисловыми (фамилия – Иванов, лет – 50, пол – мужской и т.д.).

Набор значений атрибутов отличается от набора значений дескрипторов тем, что значения дескрипторов не связаны с их позициями и из них можно сформировать только неупорядоченный набор и образовать ассоциацию лишь между всем этим набором и обозначением соответствующего понятия (например, набором значений дескрипторов для описания содержания книги и названием этой книги).

Следует заметить, что если бы нужно было решить задачу разработки методов поиска документов по частично заданному набору его дескрипторов или атрибутов, то наиболее эффективно она решалась бы путем применения средств адресации по содержанию. Однако в некоторых задачах автоматизации проектирования и искусственного интеллекта иногда бывает трудно идентифицировать данные, подлежащие выборке, до начала поиска. В таких случаях для организации поиска можно использовать лишь косвенные ассоциации, так как для более конкретного описания искомой информации могут понадобиться данные, хранящиеся в памяти.

1.3.3. Структура ассоциаций

Рассмотрим основные понятия теории ассоциаций.

Представим *ассоциацию* как некоторое отношение вида (*A, O, V*), где *A* – атрибут; *O* – объект; *V* – значение.

Запишем ассоциацию как $O \xrightarrow{A} V$, отсюда более ясно проявляется связь между отношением, представленным в общем виде, и атрибутом. Фактически нельзя атрибут задать полностью при помощи одного аргумента *A*. Необходима пара аргументов, т.е. (*A, O*).

Например, *A=«цвет»;*

O= «шарик».

Функцию атрибута выполняет сочетание *«цвет шарика»*, а *V* – определяет его значение (например *«синий»*).

Объекту можно поставить в соответствие несколько атрибутов, использовав набор отношений вида

[(A_1, *O*, V_1), (A_2, *O*, V_2),…]

Для документа, например, это позволяет хранить в памяти одновременно несколько его разновидностей и выявлять существующие между ними связи.

Известны два подхода к построению информационных структур для их запоминания и поиска [3].

В соответствии с *первым* подходом при наличии в составе нескольких отношений идентичных элементов – из этих отношений формируются *цепочки и сети.*

В основе *второго* подхода лежит *концепция составного отношения*.

Вначале рассмотрим первый подход.

Как было сказано, наличие некоторого элемента в двух или более отношениях достаточно для определения структуры, и никаких дополнительных ссылок не требуется.

Рассмотрим следующий пример [3].

Пусть имеется некоторая конкретная информация, представленная набором отношений (табл.1.1) и соответствующим графом (рис.1.4).

В общем случае элементы, присутствующие одновременно в нескольких отношениях, в графе записываются только по одному разу, а наличие структуры следует из сочетания различных комбинаций элементов.

Сам граф в явном виде в памяти не хранится, его структура существует только в абстрактном представлении, хотя ее всегда можно восстановить по соответствующим перекрестным ссылкам.

Таблица 1.1

Примеры отношений, хранящихся в памяти

A	O	V
Год	Статья А	1965
Год	Статья В	1978
Год	Сборник С	1979
Тема	Статья А	ИИ (Искусственный интеллект)
Тема	Статья А	Языки программирования
Тема	Статья В	ИИ
Тема	Статья В	Робототехника
Тема	Сборник С	ИИ
Часть	Сборник С	Статья А
Часть	Сборник С	Статья В

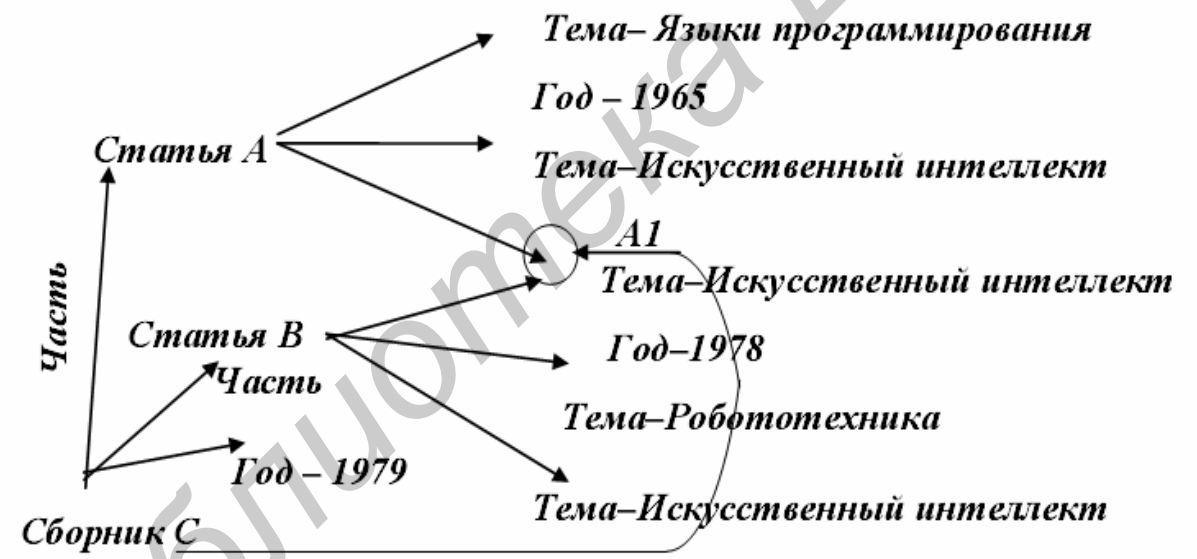


Рис. 1.4. Граф для описания структуры отношений

Существует другая логическая конструкция, называемая *составным элементом*, на основе которой также могут формироваться информационные структуры.

При вхождении составного элемента в отношение образуется *составное отношение*. Таким образом строятся *списочные структуры*. Списочная структура – это список, в котором его элемент сам является списком, а список – это любой упорядоченный элемент данных. Возможно применение составных отношений также при анализе некоторых утверждений типа, например:

«Ответом на запрос является информация, выдаваемая печатающим устройством».

Граф этого предложения показан на рис. 1.5.

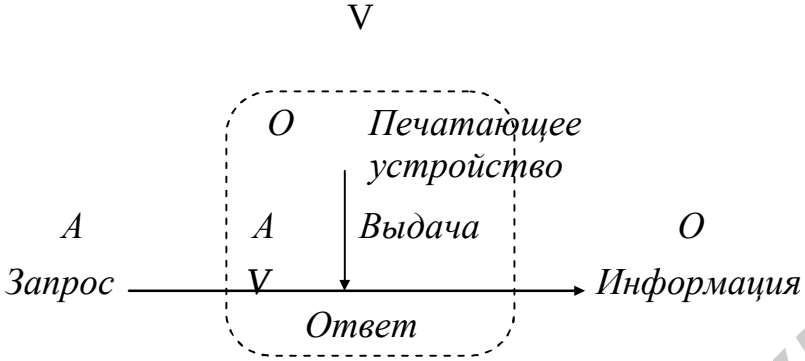


Рис. 1.5. Составное отношение

В завершение рассмотрим случай, если задачу поиска нельзя сформулировать в явном виде. Для ее решения необходимо использовать структуры, состоящие из ассоциаций.

Например [3]:

«Найти все статьи, опубликованные в 1998 г. по ИИ».

Способов решения может быть несколько. Рассмотрим такой вариант:

Полагаем, что в памяти записаны отношения только 2-х типов:

(*год, статья* A_i, B_i)

(*тема, статья* A_i, C_i).

Обозначим искомый элемент переменной *x*.

(*год, x, 1998*)

(*тема, x, ИИ*)

Поиск состоит из двух этапов:

1) поиск всех ассоциаций, отвечающих поисковому аргументу (*год, 1998*). В результате поиска сформируется набор возможных значений *x*;

2) отбираются все отношения, которые соответствуют поисковому аргументу (*тема, ИИ*). Образуется второй набор значений *x*.

Выделив элементы, которые присутствуют одновременно в обоих наборах, найдем требуемые статьи.

Поиск может выполняться и другими способами (программным или с применением аппаратуры для параллельных вычислений) [3] .

1.3.4. Классические законы ассоциаций

Обобщая наблюдения над явлениями человеческой памяти, греческий философ Аристотель (*384–322 гг. до н.э.*) *выдвинул ряд постулатов, впоследствии послуживших основой при построении классических законов ассоциаций* [3].

Между объектами мышления (идеями, восприятиями, ощущениями, чувствами) в памяти человека устанавливаются связи при наличии следующих условий (*законов*):

1) если они возникли одновременно (*«пространственный контакт»*);

2) если они возникли с небольшим разрывом во времени (*«временный контакт»*);

3) если между ними имеется сходство;

4) если они противоположны.

По современным представлениям, с учетом принципов действия средств электронной вычислительной техники, в работе памяти физической системы можно обнаружить проявление этих законов на этапах *записи* (1 и 2 законы) и *считывания* (3 и 4 законы) информации.

Действительно, чтобы в физической системе два сигнала оказались связанными (совместно закодированными), они должны появиться либо одновременно, либо с небольшим сдвигом относительно друг друга, а считанный элемент (или его часть) может, например, иметь значительную положительную корреляцию (сходство) или отрицательную корреляцию (различие) с элементом, используемым в качестве входного ключа или поискового аргумента.

Следует напомнить еще об одном факторе человеческой памяти, которому она обязана своей громадной емкостью и способностью к селекции, о *контексте*, или об обстоятельствах, при которых произошло первичное восприятие объекта. В памяти человека восприятие любого объекта из внешнего мира сопровождается возникновением эмоций и запоминается не только информация об объекте, но и о вызванных им эмоциональных ощущениях и времени. В памяти физической системы в процессе запоминания изображения объекта не происходит какой-либо обработки.

Отметим наиболее важные особенности памяти человека [3]:

1) поиск информации в памяти основывается на некоторой мере, определяющей меру сходства с ключевым образом;

2) память способна хранить образы структурированных последовательностей;

3) выборка информации из памяти представляет собой динамический процесс, подобный процессам, протекающим в различных физических системах непрерывного типа.

1.4. Отношения сходства между информационными объектами

Если рассматривать определение ассоциации как установление отношений соответствия между представлениями двух или большего числа объектов или событий, то определяющим свойством ассоциативной памяти является реализация в ней процедуры сравнения на основе выбранной *меры сходства*, трактуемой как расстояние между информационными объектами, представленными в виде упорядоченных наборов элементов некоторых множеств.

Далее рассмотрим различные информационные меры сходства и способы определения отношений сходства/различия между информационными объектами [3, 5].

Хэммингово расстояние. Данная мера применяется для сравнения любых упорядоченных наборов равной длины, принимающих дискретные значения.

Пусть имеется два набора: $x = (x_1 ,\ldots,x_N)$ *и* $y = (y_1 ,\ldots,y_N)$. Хэммингово расстояние для них можно получить путем подсчета числа несовпадающих элементов в одинаковых позициях наборов, так что для бинарных наборов можно записать

$$\rho_H(x,y) = bc\{(\bar{x}_K \wedge y_K) \vee (x_K \wedge \bar{y}_k)| \; k =1,\ldots,N\}. \qquad (1.1)$$

Функция $bc\{S\}$ здесь определяется как число элементов набора S, принимающих значение логической «1».

Функция корреляции. Для двух упорядоченных наборов или двух последовательностей действительных чисел $x = (x_1 ,\ldots,x_N)$ *и* $y = (y_1 ,\ldots,y_N)$. функция корреляции определяется следующим образом:

$$C = \sum_{k=1}^{N} x_K y_K . \qquad (1.2)$$

Например, для двух действительных векторов x и y в евклидовом пространстве величина C *равна их скалярному произведению*.

Корреляционные методы часто применяются для сравнения непрерывных сигналов, а также при распознавании образов.

Если наборы могут сдвигаться друг относительно друга, для их сравнения можно использовать меру, инвариантную к такому, например, смещению, как максимум функции корреляции на заданном интервале:

$$C_m = \max_i \sum_{k=1}^{N} x_K y_K , \; i = –\mathrm{N}, –\mathrm{N} =1,\ldots,\mathrm{N}.$$

Расстояние пространства Бэра. Точки этого пространства – это всевозможные наборы $m=(m_1,\ldots,m_k,\ldots)$ и $n=(n_1,\ldots,n_k,\ldots)$ натуральных чисел. Расстояние пространства Бэра определяется

$$\rho(m,n) = 1/h(m,n), \tag{1.3}$$

где $h(m,n)$ – наименьшее натуральное число, для которого $m \neq n$.

Направляющие косинусы. Косинусом угла между двумя векторами x и y в евклидовом пространстве называется величина

$$\cos\theta = \frac{< x, y >}{\| x \| \cdot \| y \|}, \tag{1.4}$$

где $<x,y>$ – скалярное произведение векторов x и y, а $\|x\|$ – евклидова норма вектора x.

Если векторы промасштабированы так, что их нормы равны единице, то (1.4) перейдет в (1.2) и $\cos\theta = C$. Если $\cos\theta = 0$, то векторы взаимно ортогональны; если $\cos\theta = 1$, то вектор $y = ax$, а направления совпадают .

Евклидово расстояние. Для упорядоченных наборов $x = (x_1,\ldots,x_N)$ и $y=(y_1,\ldots,y_N)$ из N действительных чисел определяется как

$$\rho_E(x,y)=\|x-y\|=\sqrt{\sum_{k=1}^{N}(X_k - Y_k)^2}\ . \tag{1.5}$$

Если векторы нормированы, так что их длины равны, то сравнение с помощью евклидова расстояния дает такие же результаты, что и методы корреляции и направляющих косинусов.

Расстояние по Манхэттену для двух упорядоченных наборов **x** и y из N действительных чисел представляет собой *покоординатное смещение:*

$$\rho(x,y)= \sum_{k=1}^{n}|x_k - y_k|\ . \tag{1.6}$$

Чебышевское расстояние – для двух упорядоченных наборов из N действительных чисел x *и* y определяется выражением

$$\rho(x,y) = \max_k |x_k - y_k|\ . \tag{1.7}$$

Меры сходства в метрике Минковского

$$\rho(x,y) = \left(\sum_{k=1}^{n}|x_k - y_k|^{\lambda} \right)^{1/\lambda}\ , \quad \lambda \geq 1. \tag{1.8}$$

При λ=1 получаем расстояние по Манхэттену, при λ=2 – евклидово расстояние, а при $\lambda \to \infty$ – чебышевское расстояние.

Мера сходства Танимото. Она может использоваться для сравнения векторов различной длины

$$\rho\ (x,y) = \frac{< x, y >}{\| x \|^2 + \| y \|^2 - < x, y >} \qquad (1.9)$$

или неупорядоченных наборов A и B нечисловых элементов (например, идентификаторов и дескрипторов в документах)

$$\rho\ (A,B) = \frac{N(A \cap B)}{(N \cup B)} = \frac{N(A \cap B)}{N(A) + N(B) - N(A \cap B)}\ . \qquad (1.10)$$

В рассмотренных мерах сходства компоненты наборов x и y считались независимыми. В [3, 5] рассматриваются также взвешенные меры сходства для статистически зависимых последовательностей, в этом случае в методах, где применялось скалярное умножение, оно должно быть заменено векторно-матричным произведением, что требует большого количества вычислений.

Применяются меры сходства на основе бинарной многозначной и непрерывнозначной логики, нечеткие меры, сравнение по неизменным признакам, вариационные меры сходства (к ним тесно примыкает *левенштейново расстояние).*

Левенштейново расстояние используется для оценки степени различия символьных строк x и y и записывается в виде

$$\rho(x,y) = \min_k\ (ap^k + bq^k + cr^k), \qquad (1.11)$$

где p,q,r – соответственно операции замены, включения и исключения символов для получения строки x из строки y в процессе редактирования, а коэффициенты a,b,c учитывают частоту этих замен (ошибок в строках) и постоянны для задач определенного класса.

Контрольные вопросы к разделу 1

1. Назовите основные обстоятельства, способствующие развитию ассоциативных средств хранения и обработки информации.

2. Что является основной особенностью памяти с адресацией по содержанию?

3. Приведите определение, виды и структуру ассоциаций.

4. Приведите законы ассоциаций (по Аристотелю).

5. Дайте определение понятия «отношение» и обозначение отношения.

6. Перечислите некоторые меры сходства.

2. ПРОГРАММНЫЙ ПОДХОД К АДРЕСАЦИИ ПО СОДЕРЖАНИЮ

2.1. Основные принципы хеширования

2.1.1. Общие понятия о хешировании

В ассоциативных (или с адресацией по содержанию) способах поиска информации доступ к любой информационной единице (записи, документу, массиву или элементу данных и т.д.) осуществляется:

– либо по специальному ключу;

– либо по некоторому фрагменту самой информационной единицы.

Поиск может быть реализован как аппаратно, так и программно. В данном разделе пособия рассмотрен ряд методов программного поиска. Эти методы рассчитаны на применение ЭВМ универсального типа. Предполагается, что информация хранится в запоминающем устройстве произвольного доступа со стандартной системой адресации.

Программные методы называют *методами хеширования*. Применяются и другие названия, в частности [3]:

– адресация в разброс;

– адресация с перемешиванием;

– ключевое преобразование;

– рандомизация;

– преобразование типа «ключ-адрес» и др.

С хешированием приходится встречаться едва ли не на каждом шагу [15]: при работе с браузером (список Web-ссылок), текстовым редактором и словарем, языками скриптов (Perl, Python, PHP и др.), компилятором (таблица символов). Упорядочение по алфавиту в адресной книге, энциклопедии и др. является не чем иным, как хешированием.

Приведем простой пример [16]. Предположим, что некоторая фирма выпускает детали и кодирует их семизначными цифрами. Для применения прямой индексации с использованием полного семизначного ключа потребовался бы массив (таблица, память) из 100 млн элементов. Ясно, что это привело бы к потере неприемлемо большого пространства, поскольку совершенно невероятно, что какая-либо фирма может иметь больше тысячи наименований изделий. Поэтому необ-

ходим некоторый метод преобразования ключа в какое-либо целое число внутри ограниченного диапазона. Тогда для хранения всего файла будет достаточно массива из 1000 элементов. Этот массив индексируется целым числом в диапазоне от 0 до 999 включительно. В качестве индекса записи об изделии в этом массиве пусть используются три последние цифры номера изделия. Отметим, что два ключа, которые близки друг к другу как числа (например 4618396 и 4618996), могут располагаться в массиве дальше друг от друга, чем два ключа, которые значительно различаются как числа (например 0000991 и 9846995).

В данном примере сведение одного большого множества к меньшему и есть х*еширование*.

Функция, которая трансформирует ключ в некоторый индекс, называется *хеш-функцией*.

Все допустимые ключевые слова образуют так называемое *пространство имен*, а все адреса памяти, в которые преобразуются ключевые слова, – *адресное пространство*.

Получаемые из ключевых слов методом хеширования адреса называются *хеш-адресами*.

Хеш-таблица, *или таблица хеширования* (*ТХ*) – это таблица с адресацией, задаваемой хеш-функцией.

Пусть далее в рассмотренном примере требуется добавить в таблицу запись с ключом 0596397. Оказывается, что данная позиция уже занята записью с ключом 4957397 (так как три последние цифры ключей совпадают). Следовательно, запись с ключом 0596397 должна быть вставлена в таблицу в другом месте.

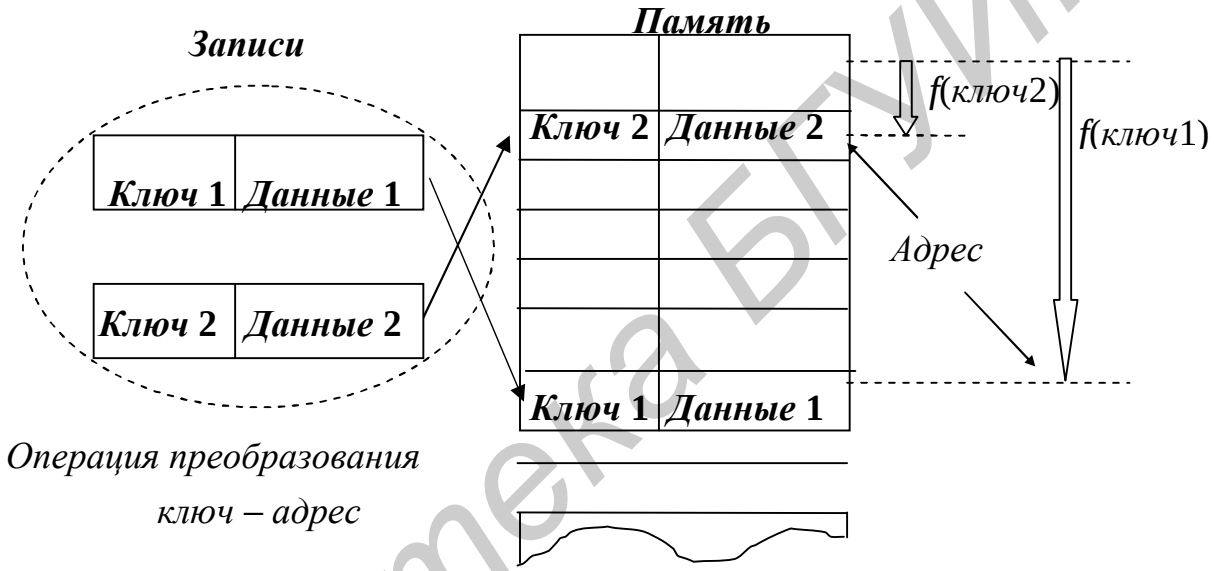
Ситуация, когда два или более ключа ассоциируются с одной и той же позицией, называется *коллизией* (конфликтом) при хешировании.

Хорошей хеш-функцией считается такая функция, которая минимизирует коллизии и распределяет записи равномерно по всей таблице. Совершенная хеш-функция – это функция, которая не порождает коллизий.

Проблема коллизии может быть решена достаточно просто и без особых затрат времени: для претендующего на уже занятую ячейку элемента (данных) отводится новое место, расположение которого легко установить, зная адрес точки коллизии, путем последовательного просмотра ячеек с адресами, сле-

дующими за вычисленным хеш-адресом до тех пор, пока не будет найдена незанятая ячейка. Но в любом случае при наличии одной или нескольких резервных ячеек с одинаковым хеш-адресом необходимо иметь возможность идентифицировать попавшие туда элементы данных по ключевым словам, так как неизвестно, как эти элементы распределены по ячейкам. Для этого обычно вместе с данными записывается соответствующее ключевое слово (или результат его однозначного преобразования), и на этапе выборки оно сравнивается с ключом, используемым в качестве аргумента поиска (рис. 2.1).

Более подробно методы разрешения коллизий будут рассмотрены ниже.



. Рис. 2.1. Распределение записей при хешировании (формирование хеш-таблицы)

В случае установления расположения элементов данных исходя из нескольких различных ключей вычисляется требуемое количество хеш-адресов и по каждому из них записывается независимая (отдельная) копия.

Для избежания чрезмерного расхода памяти на запись нескольких копий одного элемента данных (ЭД) в хеш-таблице он может заменяться более коротким идентификатором, например, адресной ссылкой в область памяти, где хранится сам элемент. Такие ссылки называются *указателями*. Это значительно упрощает логические манипуляции с данными, не прибегая к их фактической выборке.

Итак, ключ преобразуется в адрес – в этом суть метода хеширования.

Из известных методов программного поиска хеширование является самым быстродействующим. Это особенно важно при работе с наборами данных большого размера. Высокое быстродействие обусловлено тем, что элементы

данных запоминаются, а затем выбираются из ячеек памяти, адреса которых являются *простыми арифметическими функциями содержимого соответствующих ключевых слов.* Вычисление подобных функций в современных ЭВМ занимает весьма малое время по сравнению с продолжительностью доступа к памяти.

Методы хеширования получили наиболее широкое распространение в системном программировании. В настоящее время практически все символьные таблицы ассемблеров и компиляторов строятся по принципу хеширования, поскольку эти таблицы очень часто и в произвольном порядке опрашиваются в процессе компиляции и выполнения программы.

Эффективно применение методов хеширования в *управлении базами данных* (*УБД*) *в* алгоритмах выполнения операций поиска по многим ключам, в системах с многоуровневой памятью, при поиске информации в базах данных по заданным идентификаторам или *дескрипторам* (поиск документов, публикаций и т.д.), а также при создании языков высокого уровня и при построении лингвистических структур [1, 4, 5].

2.1.2. Функции хеширования

При хешировании адрес памяти, по которому заносится элемент данных, олределяется содержимым ключевого слова (КлСл), присвоенного этому элементу. В качестве КлСл могут использоваться обычные имена или произвольные числовые коды. Длина КлСл выбирается произвольно, хотя в вычислениях участвуют обычно только несколько символов. В результате набор допустимых слов (т.е. пространство имен) может оказаться весьма обширным. Например, из шести букв алфавита можно составить 256 млн слов. Однако, как уже говорилось ранее, пространство имен (ПрИм) реально оказывается крайне малым, поэтому встает задача подобрать такую функцию преобразования пространства имен в пространство адресов (ПрАдр), у которой в области значений (совпадающей с адресным пространством) адреса распределялись бы более равномерно и с большей плотностью.

Формирование хеш-функции обычно проводится в два этапа:

1) выбор способа перевода ключевых слов в числовую форму;
2) выбор алгоритма преобразования числовых значений в хеш-адреса.

Рассмотрим эти этапы.

Перевод ключевых слов в числовую форму

Информация, вводимая в ЭВМ, хранящаяся на носителях, передаваемая между устройствами, закодирована определенным образом по соответствующим стандартам (например, ASCII, КОИ-8, ДКОИ и др.). Для кодирования чисел, букв, других символов установлены стандартные форматы. Чтобы выполнять арифметические операции, числа обычно преобразуются либо в двоичную, либо в двоично-десятичную форму. Таким образом, так как все машинные переменные так или иначе уже представлены в числовой форме, то их дополнительных преобразований обычно проводить уже не надо, лишь в некоторых алгоритмах, предназначенных для вычисления адресов, требуется, чтобы все исходные числовые величины располагались в пределах допустимого диапазона целых чисел в ЭВМ.

Ключевое слово обычно представляет собой строку цифр, букв или алфавитно-цифровых символов. В алфавит могут входить разные символы. Наиболее удобными считаются алфавиты, включающие простое число символов, которое при проведении арифметических преобразований выступает в качестве основания системы счисления [3].

Пусть имеется алфавит, в котором каждому символу i по некоторому правилу поставлен в соответствие номер d_i $\hat{I}$ {0, 1, 2,…, W-1}.

Тогда строка из этих символов, записанная в форме $k = d_{N-1} \ldots d_1 \cdot d_0$, представляет собой некоторое число в системе с основанием W. Его значение определяется по формуле

$$v = \sum_{i=0}^{N-1} d_i \cdot W^i . \qquad (2.1)$$

Например, следующим образом присвоим номера буквам английского алфавита:

$$d_i = \{A=0, B=1, \ldots, Z=25\}.$$

Тогда слову *“ABC”* соответствует числовое значение

$0 \cdot 26^2 + 1 \cdot 26^1 + 2 \cdot 26^0 = 28.$

Данный метод перевода ключевых слов в числовую форму дает хорошие результаты для английских имен и слов, а также для других, например русских.

Преобразование числовых значений в хеш-адреса

Применяются различные методы хеширования, например, следующие [3]:

– *метод деления* ;

– *метод умножения*;

– *метод извлечения битов;*

– *метод квадрата*;

– *метод сегментации* (строка делится на сегменты, равные по длине хеш-адресу, и далее – различные сочетания и операции с сегментами);

– *переход к новому основанию* (системы счисления);

– *алгебраическое деление* (группирование ключевых слов, применение полиномов);

– *оптимальные функции хеширования* (попытки обобщить методы алгебраического кодирования);

– *специальные методы хеширования* (для решения задач классификации, сокращенные хеш-таблицы).

Некоторые из методов хеширования рассмотрим более подробно.

Метод деления

Данный метод является наиболее распространенным.

Пусть хеш-функция должна заполнить хеш-адресами всю область памяти размером *H,* располагая их последовательно. Хеш-функция, с помощью которой вычисляются хеш-адреса, в этом методе имеет вид

$$h(v) = v \bmod H, \qquad (2.2)$$

т. е. в качестве значения хеш-функции берется остаток от деления численного значения ключа на *H*.

Метод обеспечивает достаточно равномерное распределение хеш-адресов в пределах любой заданной области памяти, однако при его реализации требуется соблюдать ряд условий [3]:

– длина таблицы хеширования *H* должна выражаться нечетным (по мере возможности простым) числом;

– длина таблицы не должна выражаться степенью основания, по которому производится перевод ключей в цифровую форму.

Метод умножения (мультипликативный)

При этом методе размер таблицы хеширования задается без каких-либо ограничений, характерных для метода деления.

Для мультипликативного хеширования используется следующая формула:

$$h(v) = H\,cv \bmod 1\,, \qquad (2.3)$$

где v – неотрицательное целое число, полученное в результате преобразования ключевого слова, лежащее внутри допустимого диапазона представления чисел в данной ЭВМ, а c – некоторая константа из интервала [0,1].

Нормализованный адрес $cv \bmod 1$ представляет собой дробную часть произведения cv.

В данном методе допускается любая длина таблицы, в том числе и равная степени 2.

Качество хешировния по данному методу зависит во многом от выбора c.

Хорошие результаты отмечаются [17] для $c = Ø - 1 = 0{,}618034$, где Ø – так называемое «золотое отношение (сечение)».

Метод извлечения битов

Это один из старейших способов формирования хеш-функции. Двоичное число, соответствующее хеш-адресу, образуется просто путем *сцепления* нужного количества битов, извлекаемых из определенных позиций внутри указанной битовой строки ключевого слова. Рекомендуется биты хеш-адреса извлекать из позиций ключевых слов, в которых 1 и 0 появлялись бы с приблизительно равной частотой (по результатам статистического анализа).

Метод квадрата

В этом методе числовое значение ключевого слова v умножается на самого себя (возводится в квадрат) и в качестве хеш-адреса берутся биты из средней части результата. Ключевые слова могут иметь разную длину, что повышает степень рандомизации (распределения случайным образом).

Метод сегментации

Если числовое значение ключевого слова имеет большую длину, оно делится на сегменты, равные по длине хеш-адресу, а содержимое сегментов объединяется каким-либо способом в единый хеш-адрес. Из всех способов предпочтительным считается способ «Исключающее ИЛИ», когда биты всех сегментов с одинаковыми весами суммируются по модулю 2.

Описанные выше методы хеширования являются статическими, в которых сначала каким-то образом выбирается размер H хеш-таблицы, а далее под ее размер подбираются константы для хеш-функции. К сожалению, это не подходит для задач, в которых размер базы данных меняется часто и значительно

[17]. По мере роста базы данных в этом случае можно было бы пользоваться изначальной хеш-функцией, теряя производительность из-за роста коллизий; можно было выбрать хеш-функцию «с запасом», что повлекло бы неоправданные потери дискового пространства; можно было периодически менять функцию, пересчитывать все адреса, что отнимает очень много ресурсов и выводит из строя базу на некоторое время.

Существуют методы, позволяющие динамически менять размер хеш-структуры [17]. Это – д*инамическое хеширование*.

Хеш-функция генерирует так называемый псевдоключ («pseudokey»), который используется лишь частично для доступа к элементу. Другими словами, генерируется достаточно длинная битовая последовательность, которая должна быть достаточна для адресации всех потенциально возможных элементов. В то время как при статическом хешировании потребовалась бы очень большая таблица (которая обычно хранится в оперативной памяти для ускорения доступа), здесь размер занятой памяти прямо пропорционален количеству элементов в базе данных. Каждая запись в таблице хранится не отдельно, а в каком-то блоке. Эти блоки совпадают с физическими блоками на устройстве хранения данных. Если в блоке нет больше места, чтобы вместить запись, то блок делится на два, а на его место ставится указатель на два новых блока. Задача состоит в том, чтобы построить бинарное дерево, на концах ветвей которого были бы указатели на блоки, а навигация осуществлялась бы на основе псевдоключа. Узлы дерева могут быть двух видов: узлы, которые показывают на другие узлы, или узлы, которые показывают на блоки.

Вначале имеется только указатель на динамически выделенный пустой блок. При добавлении элемента вычисляется псевдоключ, и его биты поочередно используются для определения местоположения блока. Например, элементы с псевдоключами 00… будут помещены в блок А, а 01…– в блок В. Когда А будет переполнен, он будет разбит таким образом, что элементы 000…и 001… будут размещены в разных блоках.

Близко к динамическому *расширяемое хеширование*. Этот метод также предусматривает изменение размеров блоков по мере роста базы данных, но это компенсируется оптимальным использованием места. Так как за один раз разбивается не более одного блока, накладные расходы достаточно малы [17]. Вместо

бинарного дерева расширяемое хеширование предусматривает список, элементы которого ссылаются на блоки. Сами же элементы адресуются по некоторому количеству *i* битов псевдоключа. При поиске берется *i* битов псевдоключа и через список (directory) находится адрес искомого блока. Добавление элементов производится сложнее. Сначала выполняется процедура, аналогичная поиску. Если блок неполон, добавляется запись в него и в базу данных. Если блок заполнен, он разбивается на два, записи перераспределяются по описанному выше алгоритму. В этом случае возможно увеличение числа битов, необходимых для адресации. Размер списка удваивается и каждому вновь созданному элементу присваивается указатель, который содержит его родитель. Таким образом, возможна ситуация, когда несколько элементов показывают на один и тот же блок. Следует заметить, что за одну операцию вставки пересчитываются значения не более чем одного блока. Удаление производится по такому же алгоритму, только наоборот. Блоки соответственно могут быть склеены, а список – уменьшен в 2 раза.

Итак, основным достоинством расширяемого хеширования является высокая эффективность, которая не падает при увеличении размера базы данных. Кроме этого, разумно расходуется место на устройстве хранения данных, так как блоки выделяются только под реально существующие данные, а список указателей на блоки имеет размеры, минимально необходимые для адресации данного количества блоков. За эти преимущества разработчик расплачивается усложнением программного кода.

Функции, сохраняющие порядок ключей

Существует класс хеш-функций, сохраняющих порядок ключей [17], т.е. выполняется

$$K1 < K2 \rightarrow h(K1) < h(K2).$$

Эти функции полезны для сортировки, которая не потребует никакой дополнительной работы. Другими словами, не требуется множества сравнений, так как для того, чтобы отсортировать объекты по возрастанию, достаточно просто линейно просканировать хеш-таблицу. В принципе, всегда можно создать такую функцию, при условии, что хеш-таблица больше, чем пространство ключей. Однако задача поиска правильной хеш-функции нетривиальна. Разумеется, она очень сильно зависит от конкретной задачи. Кроме того, подобное ограничение на хеш-

функцию может пагубно сказаться на ее производительности. Поэтому часто прибегают к индексированию вместо поиска подобной хеш-функции, если только заранее не известно, что операция последовательной выборки будет частой.

2.2. Обработка коллизий

2.2.1. Категории методов обработки коллизий

Формирование хеш-функции – это часть работы, которую выполняет программист, реализующий поиск на основе хеширования. Кроме этого, необходимо реализовать механизм разрешения коллизий.

Известные методы обработки коллизий можно разделить на две основные категории:

1) методы, в которых как ячейки для вычисленных хеш-адресов, так и резервные ячейки для хранения элементов с одинаковыми хеш-адресами содержатся в одной области памяти, занимаемой таблицей хеширования, называемые методами *внутренней адресации*;

2) методы, в которых для резервных ячеек отводится специальная *область переполнения*.

Процедуру, при помощи которой в методах внутренней адресации отыскиваются резервные (незанятые) ячейки в таблице хеширования, называют *пробингом – методом проб и ошибок*. Разработано много различных высокоэффективных алгоритмов пробинга, о некоторых из них кратко будет изложено ниже.

Как при записи, так и при считывании (выборке) последовательность операций доступа к ячейке с вычисленным адресом или к резервной ячейке выполняется всегда одним и тем же способом, кроме случаев удаления записи из таблицы хеширования.

Для получения информации о месте нахождения искомого элемента в резервные, а также в вычисленную ячейки записываются копии соответствующих ключевых слов (при хешировании по методу деления в некоторых случаях может использоваться более короткий идентификатор).

При поиске эти копии сравниваются с аргументом (поиска) до их совпадения.

Для отметки ячеек, в которые записывается информация, а также для ускорения поисковых операций могут приниматься различные меры, например:

– один из разрядов ячейки используется для отметки занятости: разряду присваивается значение 1– если в ячейке хранится информация, 0 – если ячейка свободна или хранившийся в ней элемент помечен как вычеркнутый из таблицы;

– для обозначения конца цепочки резервных ячеек вводится еще один разряд в каждой ячейке или ячейка в конце цепочки, в которой должен содержаться определенный *идентификатор* (например запрещенное ключевое слово). В последнем случае ячейка играет роль маркера.

2.2.2. Основные методы пробинга при внутренней адресации

Имеется несколько типов пробинга:

1) линейный *пробинг* – к хеш-адресу последовательно прибавляется или вычитается из него по единице до тех пор, пока не обнаружится пустая ячейка;

2) квадратичный *пробинг* – последовательность резервных адресов определяется с использованием некоторой квадратичной функции;

3) *случайный* пробинг, в котором смещение вычисляется с помощью датчика псевдослучайных чисел (ДПСЧ).

Пусть хеш-адрес $h\ (k)$ и адреса резервных ячеек, соответствующих ключу k, образуют последовательность

$$\{ f_0(k), f_1(k), f_2(k),\ldots\},\ f_0(k) = h(k),$$

где $f_i(k) = [h(k) + g_i]\ mod\ H$,

g_i – смещение

($i = 0, 1, 2,\ldots,\ g_0 = 0$),

H – число ячеек в таблице хеширования.

Если по какой-либо причине совпали адреса двух ячеек, отвечающих двум различным ключевым словам k_1 и k_2, т.е.

$$f_i(k_1) = f_j(k_2) \qquad (2.4)$$

$$i \neq j \cdot mod\ H \text{ и } k_1 \neq k_2,$$

а

$$f_{i+L}\ (k_1) = f_{j+L}\ (k_2) \text{ для } L = 1, 2, 3,\ldots, \qquad (2.5)$$

то говорят, что имеет место *первичное группирование ячеек*.

Группирование характерно для пробинга *линейного типа* (g_i=1), поскольку условие (2.5) должно выполняться для любых значений $f_i(k_1)$ и $f_j(k_2)$, принадлежащих двум независимым последовательностям резервных адресов.

Опыты показывают [17], что алгоритм линейного типа хорошо работает в начале заполнения таблицы, однако по мере заполнения процесс замедляется, а

длинные серии проб становятся все более частыми. И все же он достаточно прост и эффективен: при заполнении таблицы на 90 % для поиска элемента в среднем требуется около пяти с половиной проб.

Первичное группирование можно устранить применением нелинейных методов, в частности, *квадратичным пробингом*. Этот метод не требует сложных вычислений, дает хорошее распределение по таблице, но занимает больше времени для просчета [17].

Процедура квадратичного пробинга может быть описана в общем виде следующим выражением:

$$g_i = a \cdot i + b \cdot i^2 . \qquad (2.6)$$

Для ЕС ЭВМ, например, было принято: $a = -787$; $b = 1$.

Но это выражение (2.6) накладывает определенные ограничения на длину хеш-таблицы и выбор значений a и b.

Случайный пробинг

В процедуре случайного пробинга используется заранее сгенерированный список псевдослучайных чисел $\{ d_i \}$.

Адрес первой просматриваемой ячейки определяется по формуле

$$f_1 = (h + d_1) \bmod H; \qquad (2.7)$$

адреса остальных резервных ячеек – по формуле

$f_i = (f_{i-1} + d_i) \bmod H$, $i = 1, 2, \ldots$

Если все числа $d_i < H$ и не имеют с H общих множителей (кроме 1), то гарантируется отсутствие повторных (двойных просмотров). Например, если H является степенью числа 2, то d_i может быть любым нечетным числом, меньшим H, а если H – простое число, то d_i может быть любым положительным числом, меньшим H.

Вторичное группирование. Двойное хеширование

В методе внутренней адресации всякий раз, когда двум и более ключевым словам присваиваются одинаковые хеш-адреса, формируется одна и та же последовательность резервных ячеек. Это явление называют *вторичным группированием*. Если средняя длина цепочек резервных ячеек достаточно велика, вторичное группирование может снизить эффективное быстродействие процедур квадратичного или случайного пробинга.

Для устранения этого явления используется адресация с двойным хешированием. Этот алгоритм выбора цепочки очень похож на алгоритм для линейной адресации, но он проверяет таблицу несколько иначе, используя две хеш-функции:

$h_1(к)$ – хеш-адрес, соответствующий численному значению $к$ ключевого слова;

$h_2(к)$ – «хеш-смещение», соответствующее значению $к$, формируемое процедурой пробинга.

Последняя функция должна порождать значения в интервале от 1 до H –1, взаимно простые с H.

Этот вариант дает значительно лучшее распределение и независимые друг от друга цепочки. Однако он несколько медленнее из-за введения дополнительной функции.

Дональд Кнут [17] предлагает несколько различных вариантов выбора дополнительной функции. Если H – простое число и $h_1(к) = к \bmod H$, можно положить $h_2(к) = 1 + (к \bmod (H - 1))$; в общем случае, если H нечетно, что всегда выполняется для простых чисел, предлагается лучше положить $h_2(к) = 1 + (к \bmod (H - 2))$.

Здесь обе функции достаточно независимы. Гари Кнотт (Gary Knott) в 1968 г. предложил при простом H использовать следующую функцию [17]:

$h_2(к) = 1$, если $h_1(к) = 0$,

$h_2(к) = H - h_1(к)$ – в противном случае (т.е. если $h_1(к) > 0$).

Этот метод выполняется быстрее повторного деления, но приводит к увеличению числа проб из-за повышения вероятности того, что два или несколько ключей пойдут по одному и тому же пути. Если число занятых ячеек обозначить N, то среднее количество проб в этом алгоритме будет составлять $(M+1) / (M-N+1)$.

При использовании алгоритма вычисления хеш-смещения, обеспечивающего равномерное появление любых допустимых пар [$h_1(к)$, $h_2(к)$], метод называют *независимым двойным хешированием* (а также *равномерным хешированием*).

Рассмотрим методы устранения группирований.

1. *Квадратичный пробинг с переменным коэффициентом*

Используется для устранения вторичного группирования (сам квадратичный пробинг позволяет избавиться от первичного группирования). Для этого

используется квадратичная функция, параметры которой зависят от численного значения ключевого слова. В этом случае алгоритм имеет вид

$$f_i(k) = [\ h\ (k) + g_i\ (k)\ ]\ mod\ H, \qquad (2.8)$$

где $g_i\ (k) = a \cdot i + b\ (k) \cdot i^2$, $b\ (k)$ – переменный коэффициент.

Для определения $b\ (k)$ используются различные методы (деления и др.).

2. *Линейный пробинг с переменным смещением*

С помощью этого метода удается избавиться также от первичного группирования.

Для этого пробинга (применительно к хешированию по методу деления) предложен такой алгоритм:

$$fi\ (k) = [\ f_{i-1}\ (k) + Q\ ]\ mod\ H, \qquad (2.9)$$

где Q – частное от деления $v\ (k)$ на H, *а* H – простое число.

Имеются и другие алгоритмы с теми или иными ограничениями на H.

3. *Сокращение числа проб в ходе выборки за счет перестройки ранее сформированных цепочек* (*на этапе записи*)

Идея этого метода заключается в перестройке таблицы хеширования при записи в нее нового ключевого слова. Хранившийся ранее в проверяемой ячейке элемент перемещается в другую ячейку, освобождая место для новой информации, а ячейка, в которую перезаписывается старый элемент, находится автоматически как очередная резервная ячейка в последовательности, к которой относился указанный элемент. Алгоритмов тоже несколько.

4. *Метод пересекающихся цепочек*

Результаты вычислений, выполненных в ходе пробинга (при первичном заполнении таблицы или в процессе ее модификации), можно запомнить и для ускорения поиска использовать их на этапе выборки. В задачах управления данными в первую очередь стремятся минимизировать продолжительность выборки ЭД, а на запись в таблицу обычно отводят больше времени.

Для запоминания в каждой ячейке памяти выделяется специальное поле, в которое, начиная с ячейки, расположенной по вычисленному хеш-адресу, заносится адрес следующей резервной ячейки (который был вычислен в момент записи соответствующего элемента данных), в результате чего на этапе поиска для получения адреса очередной ячейки никаких арифметических операций производить не требуется. Резервные ячейки находятся сразу, без лишних проверок. Последовательность адресов резервных ячеек образует цепочку или свя-

занный список. Адреса, помещаемые в ячейки, называются *указателями*. Такую структуру определяют как *цепочку с непосредственными связями*.

Метод цепочек отличается простотой добавления к связанному списку новых элементов (рис. 2.2).

После нахождения новой резервной ячейки в очередной коллизии ее адрес заносится на место старого указателя в точке коллизии, а старый – запоминается в поле указателя новой резервной ячейки; остальная часть адресной цепочки не меняется, и ни один из ранее записанных ЭД не переносится в другую ячейку. Следует заметить, что после очередной коллизии элементы данных располагаются уже не в том порядке, в котором шел просмотр потенциальных резервных ячеек в процессе пробинга. Элементы, попавшие в таблицу последними, располагаются ближе остальных к вычисленному адресу, их выборка производится наиболее быстро, что важно, так как согласно статистике новые ЭД запрашиваются чаще старых.

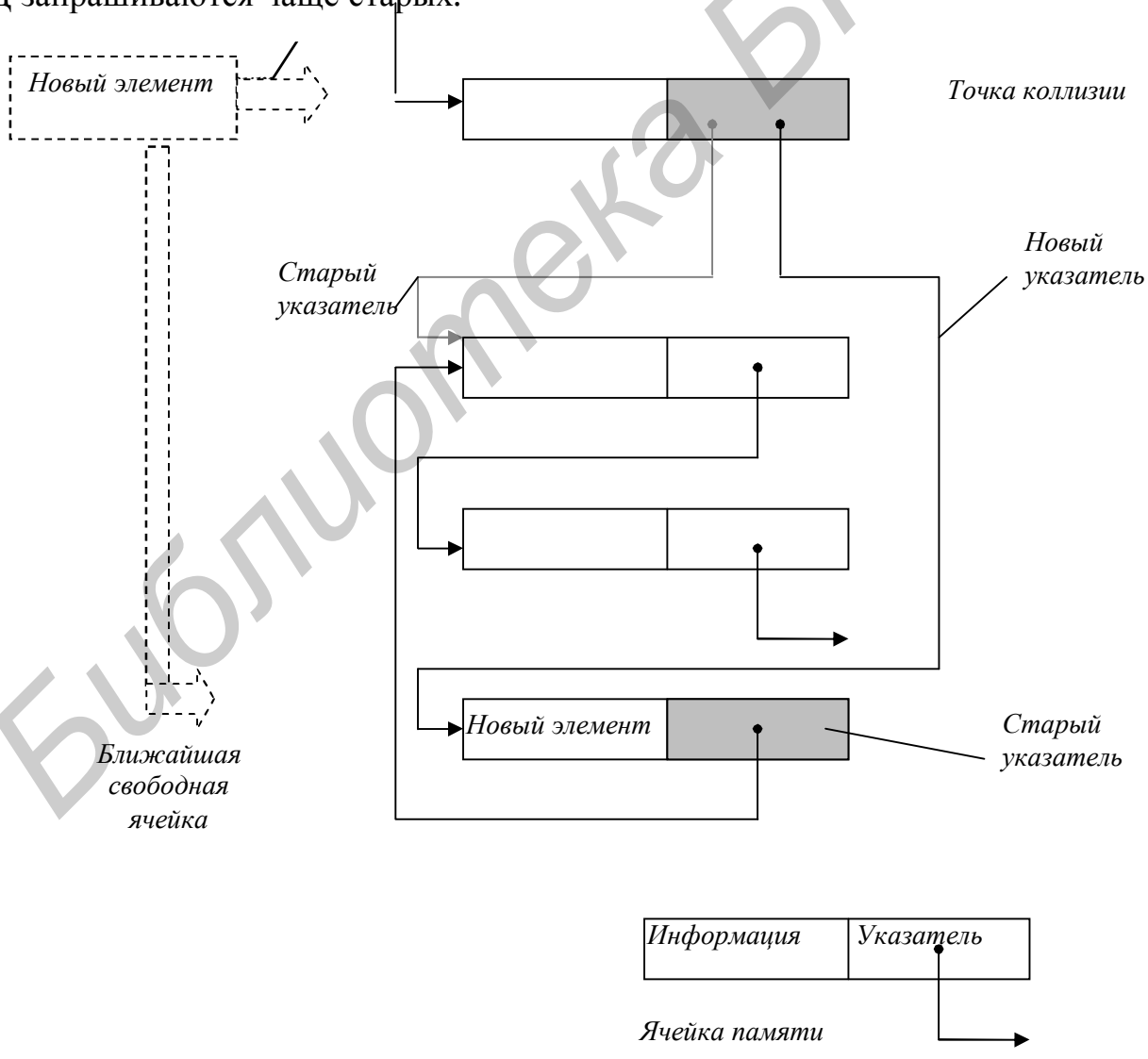


Рис. 2.2. Метод пересекающихся цепочек

В этом методе цепочек не исключается возможность того, что несколько цепочек, начинающихся с различных вычисленных адресов, пересекутся, в связи с чем метод называют еще *методом пересекающихся цепочек.*

5. *Удаление элементов хеш-таблицы*

С течением времени возникает необходимость в удалении неиспользованных элементов из таблицы хеширования для записи новой информации и сокращения задержек, связанных с обработкой элементов данных, тоже не участвующих в решении задачи.

Такие ячейки нельзя просто пометить как пустые, иначе они при пробинге будут рассматриваться как концы цепочек резервных ячеек. Например, если удалить ключ, который находится в цепочке, по которой идет поиск, использующий открытую адресацию, то все последующие элементы будут потеряны, так как алгоритм производит поиск до первого незанятого элемента.

Поэтому приходится всю последовательность записей, расположенных вслед за удаляемой, сдвигать на одну ячейку в направлении вычисленного адреса. Наиболее удобен в данном отношении *линейный* пробинг. *Случайный* пробинг, наоборот, оказывается крайне громоздким, поскольку алгоритмы получения псевдослучайных чисел принципиально необратимы.

Ячейку с подлежащим удалению ЭД можно пометить специальным маркером (например выделенным отдельным разрядом). Для записи новой информации такая ячейка считается свободной. При встрече такого маркера в процессе выполнения процедуры пробинга поиск продолжается дальше (ячейка считается занятой).

В случае совпадения вычисленного адреса с адресом ячейки, помеченной как свободная, она может служить началом новой цепочки, при этом резервные ячейки выстраиваются вслед за старой цепочкой, которая обходит эту ячейку, и сокращается время операции поиска.

Этот метод особенно удобен при применении линейного пробинга.

Такой метод хорошо работает при редких удалениях, поскольку однажды занятая позиция больше никогда не сможет стать свободной, значит, после длинной последовательности вставок и удалений все свободные позиции исчезнут, а при неудачном поиске будет требоваться H проверок (где H – размер хеш-таблицы). Это будет достаточно долгий процесс.

Данный алгоритм позволяет перемещать некоторые элементы таблицы, что может оказаться нежелательным (например, если имеются ссылки извне на элементы хеш-таблицы).

Другой подход к проблеме удаления основывается на хранении количества ссылок с каждым ключом, которое говорит о том, как много других ключей сталкивается с ним. Тогда при обнулении счетчика можно преобразовывать такие ячейки в пустые.

Для экономии памяти вместо флажкового разряда можно использовать:

– либо специальное ключевое слово;

– либо данные, которые по какой-то причине запрещено употреблять.

Содержащие подобный код ячейки играют роль *маркера*. При необходимости в эту ячейку может быть записана новая информация и пробинг будет выполняться без учета этого маркера.

Значительно проще, в отличие от обычных хеш-таблиц, осуществляется процедура удаления ЭД из пересекающихся цепочек. В ячейку, из которой удаляется запись, копируется содержимое ячейки памяти (ключ плюс данные), стоящей следующей в цепочке и помечаемой в дальнейшем как свободная. Содержимое остальных ячеек не меняется.

В поле указателя ячейки, на которой заканчивается адресная цепочка, заносится специальное условное число: константа (например нуль), вычисленный адрес, соответствующий данной цепочке, или адрес ячейки.

Некоторые методы удаления, позволяющие избежать перемещения в таблице и работающие с любой хеш-технологией, указаны в [17].

2.2.3. Методы обработки коллизий при использовании отдельной области переполнения

При работе с многоуровневой памятью обмен данными в ней ведется через буферы, информация передается поблочно, поэтому удобно, если подлежащие пересылке данные располагаются в одной локальной области памяти. В этом случае метод внутренней адресации дает определенные преимущества.

В ассемблерах и компиляторах для формирования таблиц и символов более подходящим является метод обработки коллизий с использованием отдельной области памяти, именуемой *областью переполнения*. Из этой области по

мере необходимости берутся резервные ячейки для размещения конфликтующих элементов.

И так как цепочки, соответствующие различным вычисленным адресам, формируются независимо друг от друга, чаще всего этот метод называют *методом непересекающихся цепочек* (рис. 2.3). На этом рисунке показан случай двух коллизий в первой ячейке и одной – в третьей.

Адреса ячеек области переполнения не зависят от вычисленных адресов, резервные ячейки берутся из области переполнения последовательно в порядке возрастания адресов, поэтому операции пробинга для их поиска не требуются.

Это позволяет ускорить обработку информации (при записи и при чтении) и сократить среднее количество обращений к памяти, что особенно проявляется при большом коэффициенте заполнения таблицы хеширования.

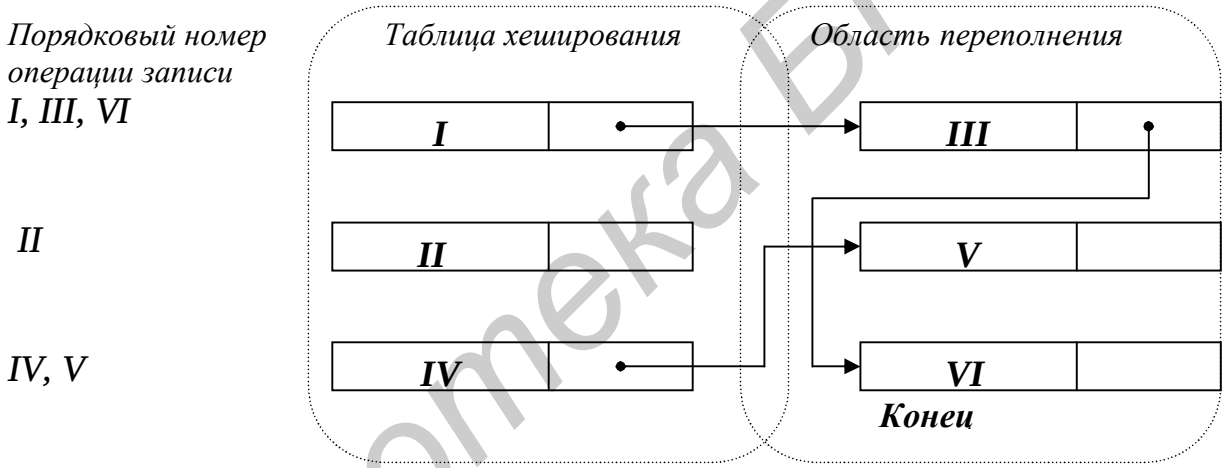


Рис. 2.3. Метод непересекающихся цепочек

Каждый новый элемент, попадающий в эту область, обычно связывается указателем непосредственно с вычисленным адресом. Поэтому при поиске недавно записанной информации не требуется просматривать всю цепочку адресов и, кроме того, упрощается процедура записи новых элементов.

Преимущества этого метода проявляются и при работе с многоуровневой памятью, когда область переполнения разделяется на части, каждая из которых закрепляется за определенной группой вычисленных адресов. Тогда при выполнении операции обмена в буферную зону основной памяти заносится не все содержимое области переполнения, а часть, которая относится к данному хеш-адресу.

С использованием отдельной области переполнения легче устраняются проблемы загруженности таблиц хеширования. На практике считается, что коэффициент заполнения таблиц хеширования не должен быть больше 0,8–0,9. Поэтому необходимо иметь автоматическую процедуру, которая помогала бы вовремя приостанавливать рост коэффициента заполнения.

Можно было бы решить эту задачу добавлением новых ячеек в таблицу хеширования путем расширения области значений исходной хеш-функции, но так как ранее записанные элементы должны оставаться на своих местах, распределение занятых ячеек в целом по таблице становится весьма неравномерным.

Для устранения этого недостатка вместо старой таблицы хеширования (*ТХ*) предлагается вводить новую хеш-таблицу большего размера и осуществлять в ней *рехеширование* (повторное размещение) всех элементов, содержащихся в старой *ТХ*, после того как загруженность старой таблицы хеширования достигнет некоторого порогового уровня, например 80 %. В новую таблицу перезапись информации производится последовательно, элемент за элементом. Для реализации этого метода в ячейках таблицы хеширования должны храниться не укороченные идентификаторы, а полные ключевые слова, так как они необходимы для вычисления значений новой хеш-функции.

2.2.4. Методы ускорения процедур поиска

Заметим, что искомый элемент может отсутствовать в таблице хеширования, поэтому значительная часть попыток выборки окажется безуспешной. Применяется несколько методов сокращения числа операций поиска.

Флажок коллизии

Чтобы быстро определить, имеется ли в таблице требуемый исковый элемент, можно зарезервировать в каждой ячейке таблицы специальный разряд, называемый *флажком коллизии*. Поисковый аргумент или заменяющий его идентификатор может не совпадать с записанным по вычисленному адресу ключевым словом (или соответствующим идентификатором) по двум причинам:

1) либо элемент с данным ключевым словом содержится в таблице, но вследствие коллизии он записан в резервную ячейку;

2) либо его просто нет в таблице.

Пусть во флажковом разряде первоначально записан “0”. После первой коллизии в данной ячейке флажок переводится в “1”. Тогда, если по вычисленному адресу ключевое слово не совпадает с аргументом поиска, а *флажок* установлен в “0”, то это означает, что ни в одной из резервных ячеек искомого ключа нет, и поиск надо прекратить.

В комбинации с флажком *“занято”* реализуется возможность определения продолжения поиска, если флажок коллизии в каждой ячейке таблицы, за которой следует резервная ячейка, устанавливается в “1”. После удаления элемента флажок *“занято”* устанавливается в “0”, и это означает, что ячейка свободна. Если *флажок коллизии* в данной ячейке установлен в “1”, то это означает, что цепочка резервных ячеек не закончилась и поиск надо продолжать.

Упорядоченные таблицы хеширования

Если элементы, образующие цепочку, упорядочить по какому-либо принципу (например, по алфавиту или в порядке нарастания численных значений ключа), то отсутствие требуемой записи при этом легко установить путем последовательной сверки поискового аргумента с элементами списка. Получение на очередном шаге проверки результата, противоположного результату предыдущего шага проверки, означает, что в оставшейся части цепочки искомый элемент заведомо отсутствует.

Виртуальные хеш-адреса

При использовании принципа внутренней адресации в ходе пробинга среди проверяемых могут оказаться ячейки, содержащие элементы, относящиеся к посторонним вычисленным адресам.

Если в процессе проверки выяснится, что очередной элемент не принадлежит к цепочке, соответствующей требуемому хеш-адресу, то дальнейший поиск не нужен.

Для их определения предлагается использовать функцию хеширования, которая формирует *виртуальный хеш-адрес*, имеющий на несколько битов больше, чем нужно для обращения к ячейкам таблицы. Дополнительные старшие биты служат *идентификатором вычисленного адреса* и хранятся в ячейке совместно с ключевым словом. Младшие разряды виртуального хеш-адреса образуют *истинный адрес* ячейки внутри таблицы.

Вначале сравнивается содержимое дополнительных разрядов (вместо ключевых слов). При большой длине ключевых слов указанная операция занимает значительно меньше времени, чем сравнение самих ключей. В итоге уве-

личивается среднее быстродействие, но ценой затрат на хранение дополнительных битов.

2.3. Структура и форматы таблиц хеширования

2.3.1. Непосредственная и косвенная адресация

Индексные ТХ

Таблицы хеширования, как правило, строятся исходя из максимальной длины ЭД, размеры которых могут изменяться, в результате чего часть памяти будет использоваться непроизводительно. Кроме того, с увеличением длины записей возрастает трудоемкость (время) поиска. Если разместить ЭД в специально отведенной области памяти, а в основной *ТХ* хранить только **указатели**, содержащие адреса элементов в этой области и соответствующие идентификаторы, то при такой структуре в процессе поиска проверке подвергаются только ключевые слова (или их идентификаторы) и обращение к отдельной области памяти производится только после того, как будет найден требуемый элемент. Таким образом, допуская некоторые увеличения количества обращений к памяти (что неизбежно при использовании многоуровневой памяти), можно повысить эффективность поиска. Преимущество предлагаемой схемы состоит также в том, что при необходимости основную *ТХ* (которую в дальнейшем будем называть *индексной ТХ*) можно без особых затруднений поместить в другую область.

Индексные *ТХ* достаточно распространены, особенно в СУБД. Однако если они используются в качестве таблиц символов с короткими именами , то косвенная адресация описанного типа приводит лишь к увеличению количества вспомогательных операций.

Комбинированная адресация в ТХ (ступенчатое хеширование)

Рассмотренный выше принцип построения *ТХ* можно рассматривать как простейшую форму косвенной адресации. Ее можно сделать более эффективной, если в одной и той же таблице наряду с косвенной адресацией использовать и прямую (непосредственную) адресацию. Для задания режима адресации в каждой ячейке таблицы отводится специальный разряд, именуемый *флажком связи.*

Напомним, как аналогичный принцип реализуется в современных ЭВМ.

На рис. 2.4 показана структура типовой машинной команды.

Можно было бы прямо в поле кода операции (*КОП*) указать, является ли содержимое адресного поля константой или адресом. Это может быть задано специальным признаком *D/I* (непосредственная или косвенная адресация).

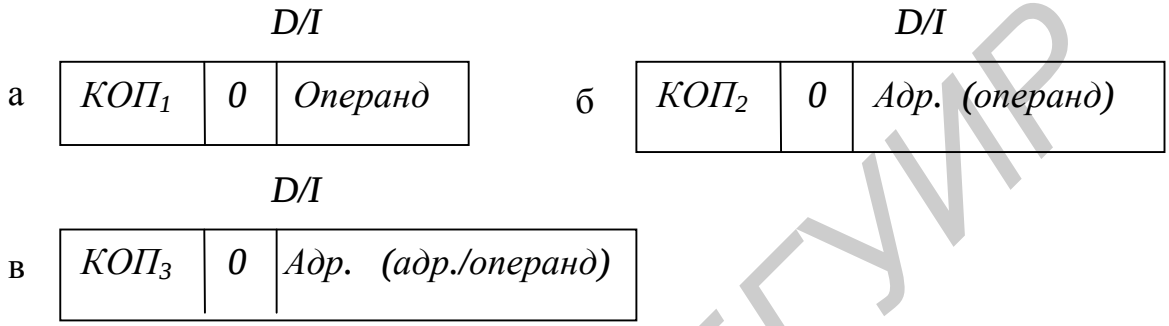


Рис. 2.4. Структура типовой машинной команды

Если *D/I* = 0, то в указанном поле записан *непосредственный*, или *абсолютный*, адрес ячейки, в которой находится искомый операнд.

Если *D/I* = 1, то истинный адрес извлекается из специальной таблицы либо из набора регистров, номер регистра или записи в таблице указывается в поле адреса (это *метод косвенной адресации*).

При построении *ТХ* может быть использован многоуровневый вариант косвенной адресации. Функцию флажка *D/I* в этом случае выполняет *флажок связи*. Значение 0 этого флажка означает, что искомые данные хранятся в той же ячейке *ТХ*, если же *флажок связи* равен 1, то соответствующее информационное поле нужно рассматривать как *хеш-связь*, т.е. адрес второй таблицы, содержащей, как правило, значительно меньшее количество слов, но большей разрядности. Для хранения записей еще большего размера можно организовать еще одну (третью) таблицу, доступ к которой производится посредством второй хеш-связи и т. д .

Для обращения к данным, расположенным во внешних запоминающих устройствах, адреса должны содержать обычно не менее 20 *разрядов*. Нерационально размещать в ТХ указатели такой длины. Введение ступенчатого хеширования в этом случае решает проблему.

Метод ступенчатого хеширования успешно применяется при решении ряда банковских задач, в языках высокого уровня, для обработки списков.

2.3.2. Форматы таблиц хеширования

При выборе размера ячейки *ТХ* следует учитывать не только объем информации, которую предполагается в ней хранить, но и возможность использования укороченного идентификатора вместо ключевого слова (при хешировании методом деления) и сокращение требуемой памяти в случае применения комбинированной адресации, а также принятую в ЭВМ систему адресации, в соответствии с которой указанные ячейки должны занимать целое число байтов или машинных слов.

Содержимое ячейки ТХ включает *обязательно*:

– ключевое слово (или его идентификатор);

– соответствующие данные (или указатель этих данных).

При внутренней адресация дополнительно резервируется место для флажка «*конец последовательности пробинга*» или дополнительная ячейка в конце последовательности со специальным кодом. В состав ячейки *ТХ* могут также входить одноразрядные маркеры (флажок “*занято*”, флажок *коллизии*, флажок *вычеркивания*, флажок *связи*) или их комбинации (рис. 2.5).

Рис. 2.5. Типовые форматы ячеек таблиц хеширования

Обозначения на рис. 2.5:

ID – идентификатор ключевого слова;

P_i – указатель области данных (индекс);

P_0 – указатель области переполнения (или следующей записи в цепочке);

T – терминальный символ;

U – флажок “занято”;

D – флажок вычеркивания;

C – флажок коллизии;

L – флажок связи.

Способы индексации слов в ТХ

Пусть запись таблицы хеширования состоит из *k* слов, следовательно, их вычисленные адреса в оперативной памяти с произвольным доступом, в которой записи обычно занимают по *k* последовательно расположенных слов, должны быть кратны *k* (т.е. полученный с помощью некоторого алгоритма исходный хеш-адрес впоследствии умножается на *k*).

Например:

(1, 1) (2, 1) (3, 1) …
(1, 2) (2, 2) (3, 2) …
(1, 3) (2, 3) (3, 3) …
(1, 4) (2, 4) (3, 4) …

k = 4 – это количество слов в записи.

Представим таблицу в виде *k* секций:

(1, 1) (1, 2) (1, 3) (1, 4)
(2, 1) (2, 2) (2, 3) (2, 4)
(3, 1) (3, 2) (3, 3) (3, 4)
(4, 1) (4, 2) (4, 3) (4, 4)
. . . .
. . . .
. . . .

Хеш-адрес может рассматриваться в этом случае как индекс любого элемента записи, распределенной по *k* секциям, благодаря чему требуется меньше вычислений, что особенно важно при использовании хеш-таблицы в качестве таблицы символов.

2.3.3. Буферизация таблиц хеширования. Клеточная организация

Если *ТХ* имеет большой размер, то ее фрагмент, в котором находится ЭД с вычисленным адресом, помещается в *буферную* область быстродействующей основной памяти (БП), а остальная часть обычно хранится во внешнем запоминающем устройстве (ВЗУ). При обработке коллизий с использованием процедуры линейного пробинга значительная часть резервных ячеек (а иногда и все) также попадает в БП. Обмен данными между основной памятью (ОП) и ВЗУ производится *страницами* типового для каждого вида ВЗУ размера.

Если адресные смещения в процедуре пробинга вычислять циклически, по модулю, равному размеру страницы, вероятность попадания всех резервных ячеек на одну и ту же страницу можно повысить.

Использование *буферизации* повышает среднюю скорость выборки информации, однако не позволяет получать равномерное распределение занятых адресов.

Метод построения хеш-таблиц, названный *клеточной организацией*, основан на объединении в группу, называемую *клеткой*, нескольких последовательно расположенных ячеек памяти, которым присваивается общий для всех ячеек хеш-адрес. Клетка делится на фиксированное количество частей, в каждой из которых размещается по одному конфликтующему элементу. Резервные ячейки отыскиваются в клетке до тех пор, пока все части такой клетки не будут заняты. При заполнении клетки целиком берется вторая клетка, и цепочка резервных ячеек продолжает формироваться уже в ней (рис. 2.6).

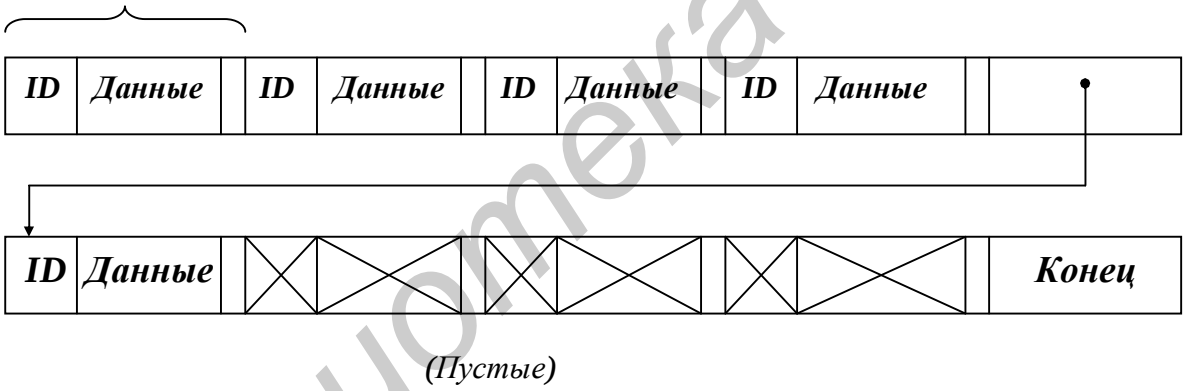


Рис. 2.6. Клеточная организация *ТХ*

Размещение в клетках записей переменной длины

Клеточная организация применяется в основном при использовании для хранения данных внешних запоминающих устройств, но в клетках могут помещаться и записи достаточно большого размера и переменной длины, непосредственно друг за другом, без разбиения на части, лишь как-то отделяясь друг от друга – например при помощи специальных кодов.

В [3] дан анализ различных вопросов, связанных со сравнительной оценкой некоторых рассмотренных ранее методов и алгоритмов хеширования, в частности:

– среднего числа обращений к памяти для различных процедур пробинга (линейного, случайного, квадратичного и др.), используемых в рамках метода внутренней адресации;

– среднего числа обращений при применении отдельной области переполнения;

– влияния упорядочивания записей в резервных ячейках;

– влияния размера клетки на количество обращений;

– влияния функции хеширования на продолжительность операции поиска и др.

2.4. Виды ассоциативного поиска

2.4.1. Многоключевой поиск

В отличие от таблиц символов, в которых обычно существует взаимно однозначное соответствие между ключевым словом и связанным с ним значением, в разного рода справочниках, списках заказчиков и других документах, содержащих информацию описательного характера, как правило, каждый запоминаемый элемент данных снабжается несколькими ключевыми словами, или *дескрипторами*, причем одно и то же ключевое слово может присутствовать в произвольном количестве записей.

Если возникает необходимость в частой выборке всех элементов, отвечающих заданному ключу, наиболее выгодными с точки зрения общих затрат считаются *последовательные методы поиска*, например, как в *задаче селективного отбора* информации в библиотеке.

При нерегулярных одиночных обращениях к БД *адресация по содержанию* имеет явные преимущества по сравнению с *последовательным поиском* (например, при оформлении банковских операций, поиске архивных документов и т.д.).

Учитывая разнообразие задач и особенности поиска информации в них, рассмотрим основные подходы к организации некоторых стандартных методов поиска информации по многим ключевым словам (многоключевой поиск) и их реализации с помощью методов хеширования.

2.4.2. Списки и списочные структуры

Список представляет собой любой упорядоченный набор элементов данных; если элемент списка сам по себе является списком, то такую конструкцию называют *списочной структурой*. Список или списочные структуры хранятся в памяти в виде отдельной записи, списку присваивается имя, называемое *заголовком*, обычно это первый элемент списка. Поиск заголовков удобно проводить *методами хеширования*. К списку могут присоединяться различными способами дополнительные элементы (самый простой – использование *последовательно расположенных ячеек*).

Цепочка резервных ячеек может служить примером списка, элементы которого разбросаны по некоторой области памяти, вычисленный адрес при этом играет роль заголовка списка.

При обращении к списку последовательность адресов либо вычисляется при помощи *пробинга*, либо определяется посредством *указателей*.

Связанная указателями любая совокупность элементов данных называется *связанным списком*.

Различные виды связанных списков показаны на рис. 2.7.

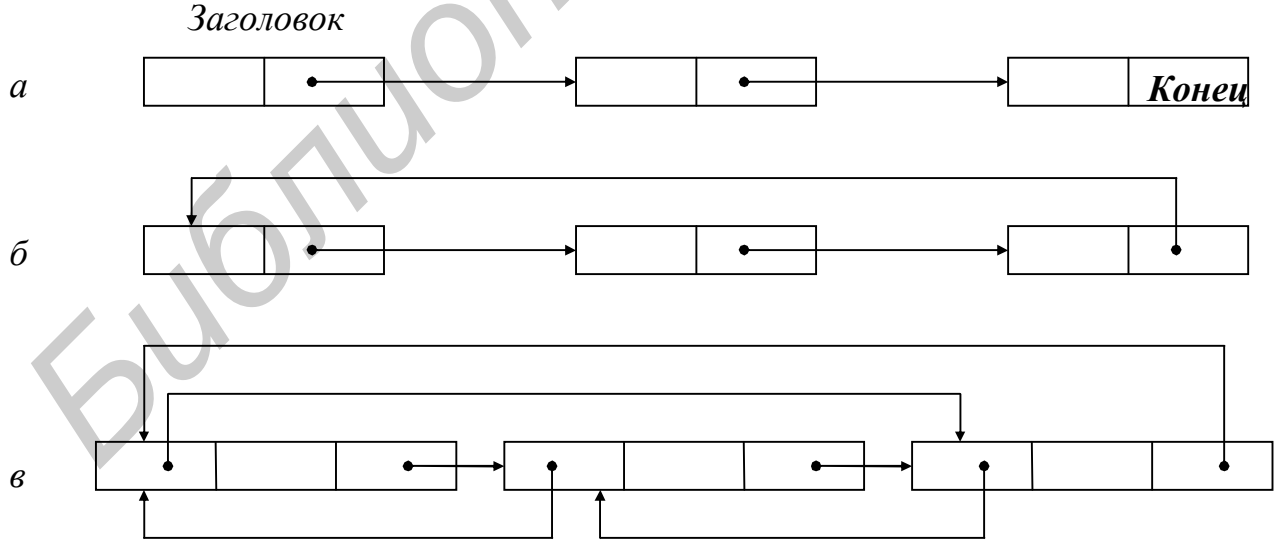


Рис. 2.7. Различные типы списков:
а – однонаправленный; *б* – циклический список;
в – двунаправленный список

В списочной структуре, в некоторых ее элементах, называемых *вершинами*, имеется не менее чем по два указателя. Если списочная структура начинает ветвиться из вершины, именуемой в этом случае *корнем*, и в ней отсутствуют циклические цепочки элементов, ее называют *деревом*.

Списочная структура с циклическими участками представляет собой *граф*.

На рис. 2.8 приведен пример *бинарного дерева*, в котором из каждой вершины (кроме последних элементов) всегда исходят по две ветви.

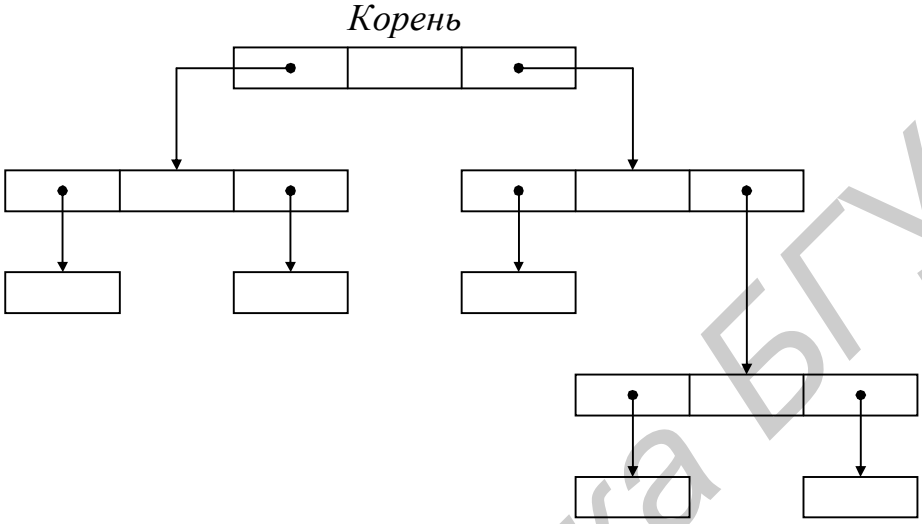


Рис. 2.8. Бинарное дерево

2.4.3. Мультисписки

Мультисписками называют структуры, в которых имеются независимые цепочки, проходящие через один и тот же набор элементов, т.е. входящих в несколько списков, каждый из которых начинается с *заголовка* и представляет собой простую неветвящуюся цепочку. Для различения цепочек в ячейках наряду с указателями помещаются также идентификаторы заголовков соответствующих частных списков (рис. 2.9).

Флажки	***Данные***
ID1	***Указатель1***
ID2	***Указатель2***
ID3	***Указатель3***

Рис. 2.9. Содержимое ячейки мультисписка

На рис. 2.10 приведена структура типового мультисписка.

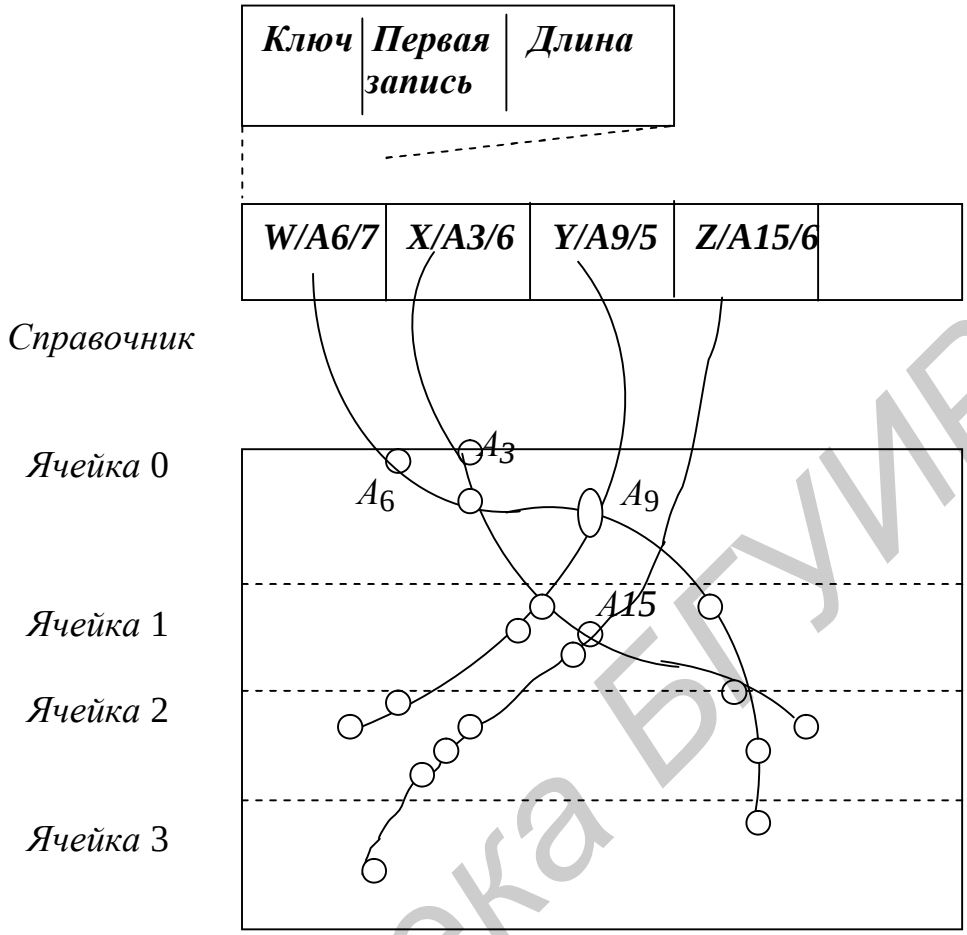


Рис. 2.10. Структура типового мультисписка

В справочнике мультисписка хранятся все ключевые слова, указатели первых элементов каждого списка и числа, задающие количество элементов в списках; его можно поместить в отдельной области памяти. Ключевые слова либо сортируются, либо путем хеширования адресов распределяются по ячейкам таблицы.

В ячейку памяти должны поместиться наряду с данными и управляющими битами m идентификаторов и m указателей (m – максимальное количество ключевых слов в записи).

При формировании мультисписковой структуры (обычно в процессе записи информации) вначале устанавливается местонахождение заголовков, содержащих ключевые слова нового элемента, затем – прослеживаются цепочки, идущие от заголовков. В последнюю запись каждой из цепочек заносится указатель, представляющий собой адрес ячейки нового элемента, который становится последним членом всех этих цепочек.

Выборка информации производится чаще всего по совокупности двух или более ключевых слов, просмотр ключей при этом целесообразно вести

вдоль более короткой цепочки. Такой просмотр возможен лишь при условии, что вместе с каждым элементом данных хранятся все его ключевые слова или идентификаторы.

Входящие в мультисписок отдельные списки могут достигать довольно больших размеров, в связи с чем при добавлении к списку новых элементов потребуются значительные временные затраты, так как придется просматривать всю ранее сформированную цепочку. При удалении элементов возникают аналогичные трудности. Их решение возможно путем *ограничения длины цепочек.*

Например, мультисписок на рис. 2.10 можно разделить на фрагменты, длина каждого из которых не превышает некоторой максимальной фиксированной величины, при этом в *справочник мультисписка* необходимо занести указатели всех фрагментов. Тогда каждый новый элемент будет добавляться к последнему частичному списку (*фрагменту*), более короткому, чем соответствующая полная цепочка мультисписка обычного типа (рис. 2.11).

Предельная длина не фиксируется в так называемых *блочных мультисписках*, однако в них каждый из фрагментов должен располагаться в пределах одного физического блока памяти , например дорожки (ячейки, клетки и др.), рис. 2.12.

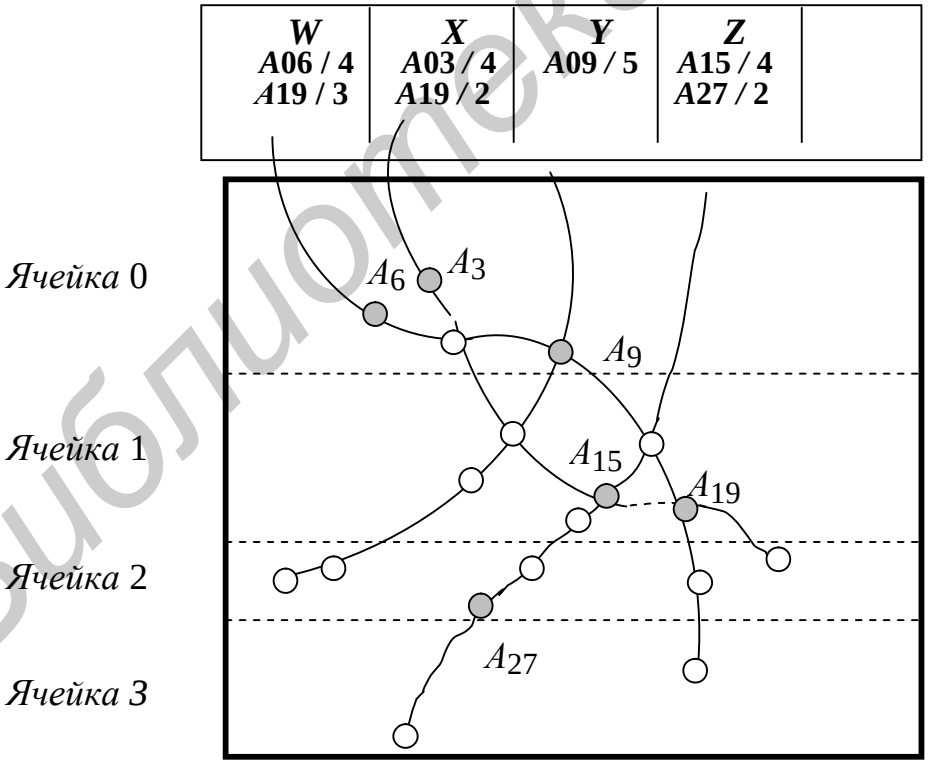


Рис. 2.11. Мультисписок с цепочками ограниченной длины

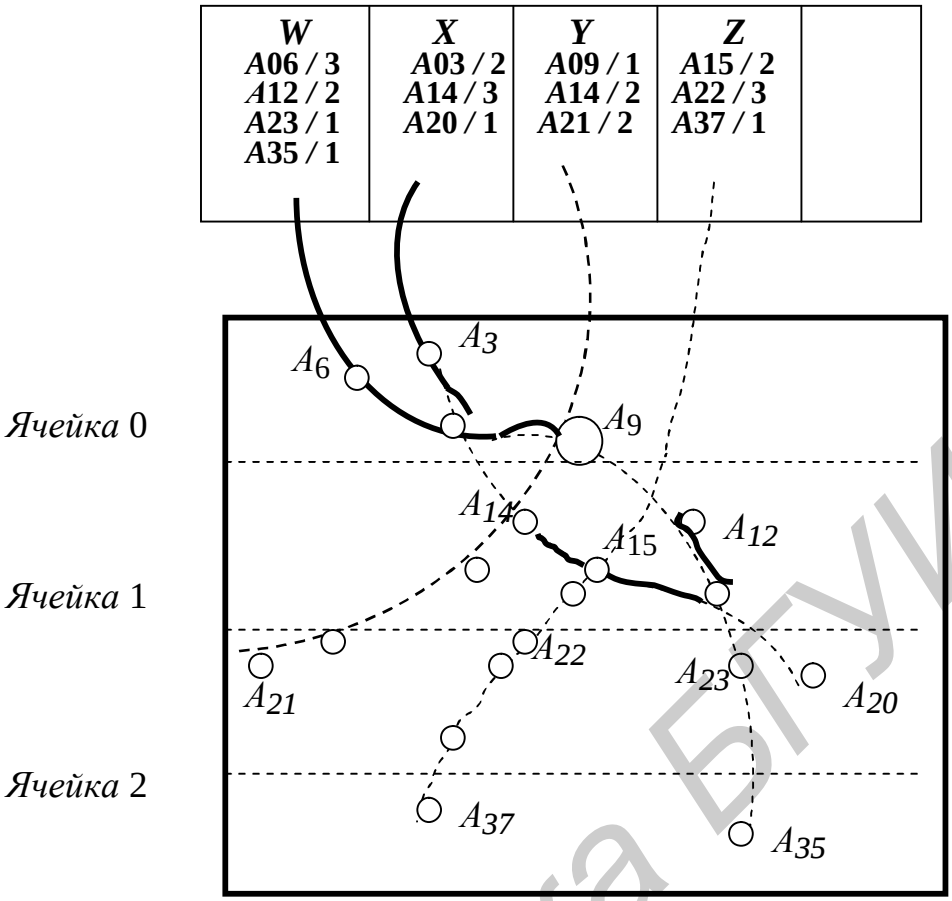


Рис. 2.12. Блочный мультисписок

При буферизации стандартного блока в этом случае в ОП пересылаются все данные, содержащиеся в некотором частичном списке, причем по справочнику можно определить, какие блоки можно исключать из рассмотрения. Например, ячейку 3 можно не проверять, если необходимо выполнить комбинационный поиск по аргументу *X и Y,* так как в ней нет записей с ключом *Y* (только с ключами *W* и *Z*). Можно также сократить продолжительность поиска, выбирая для просмотра ячейки, относящиеся к наиболее коротким цепочкам (ячейки 0 и 1 в цепочке *Y* или ячейка 2 в цепочке *X*).

При сочетании блочного мультисписка и клеточной организации образуется *блочная последовательная структура,* изображенная на рис. 2.13, для которой в справочник заносятся только индексы блоков, имеющих записи, содержащие определенные ключевые слова. Опрос соответствующих ячеек в этом случае производится последовательно.

Частным случаем блочной последовательной структуры, каждый блок которой состоит из одной ячейки, является *инвертированный список* (рис. 2.14). В отличие от всех предыдущих способов многоключевого поиска, в которых ключевые слова располагаются в самой записи, в инвертированном списке ад-

реса всех записей, связанных с данным ключевым словом, хранятся в справочнике. Справочник списка можно построить аналогично *индексной таблице хеширования*. Ключевые слова при этом играют роль вычисленных адресов ***v***, а адреса записей соответствуют указателям элементов, расположенных во внешней памяти.

Для поиска ключевых слов, если они в справочнике списка рассортированы в определенном порядке, можно применить любой стандартный метод, а указатели разместить в следующих друг за другом ячейках памяти.

В инвертированном списке вся информация, участвующая в процессе поиска, занимает сравнительно немного места, благодаря чему справочник можно хранить в ОП (для файлов не очень большого размера), что позволяет сократить до минимума в ЦП все подготовительные операции, предшествующие выборке из внешней памяти. Это является основным достоинством инвертированного списка.

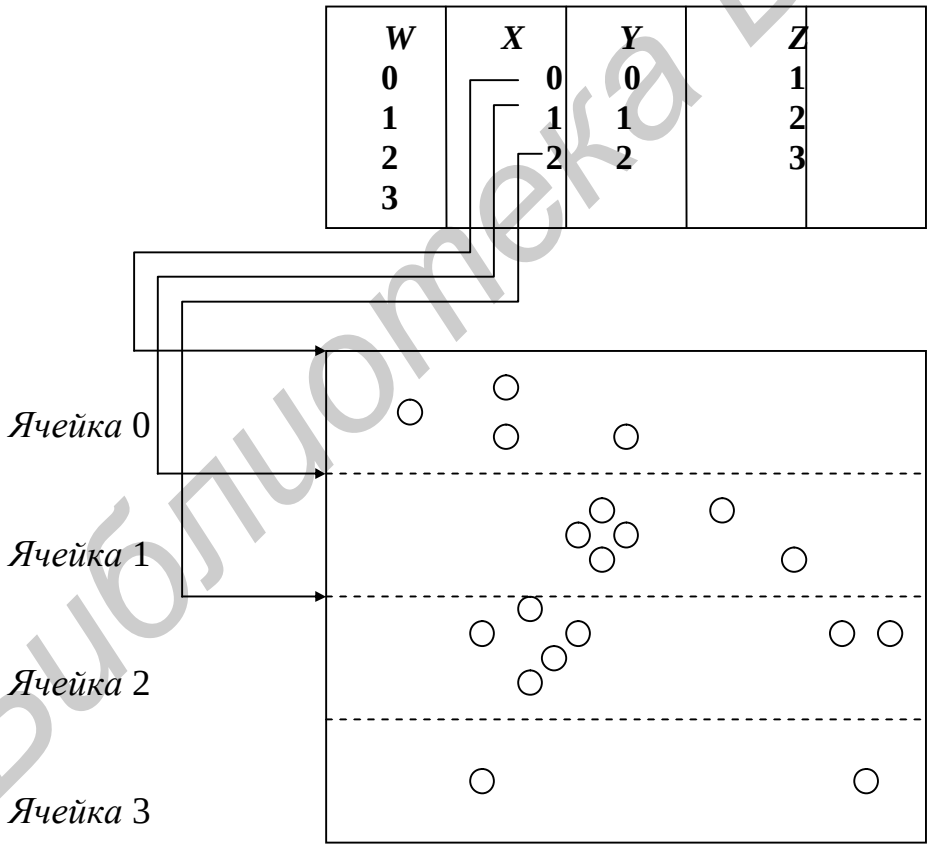


Рис. 2.13. Блочная последовательная структура

W	*X*	*Y*	*Z*	
A_{06} A_{19} A_{07} A_{23} A_{09} A_{35} A_{12}	A_{03} A_{15} A_{07} A_{19} A_{14} A_{20}	A_{09} A_{21} A_{14} A_{18} A_{16}	A_{15} A_{25} A_{17} A_{27} A_{22} A_{37}	

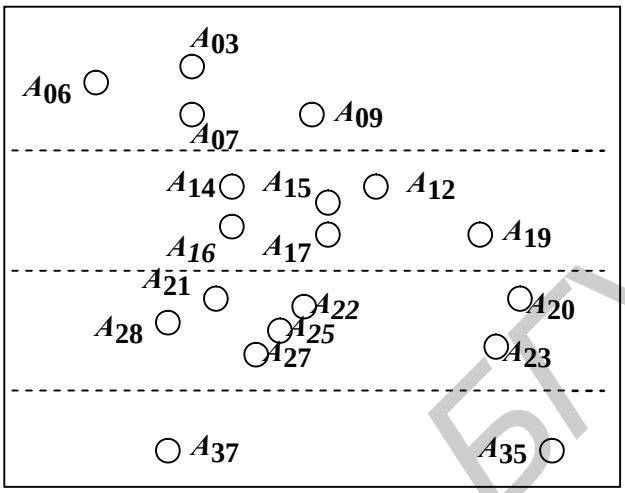


Рис. 2.14. Инвертированный список

2.4.4. Использование составных ключевых слов в процедурах хеширования

В отличие от методики многоключевого поиска, основанной на выделении совпадающих элементов в списках, связанных с различными ключевыми словами, имеется и другой подход, в котором все ключи, входящие в некоторую запись, рассматриваются как *единое составное ключевое слово*. При полном наборе ключевых слов поиск можно осуществить так же, как и с одним ключевым словом. Задача усложняется, если часть ключей *не задана*.

Далее рассматриваются некоторые варианты использования составных ключевых слов при хешировании.

Хеширование по всем сочетаниям ключевых слов

Этот метод требует самых больших затрат памяти, хотя и обеспечивает при этом высокое быстродействие. Так, если в запись входит n ключевых слов, то существует $\sum_{k=1}^{n}(\frac{n}{k})$ различных комбинаций, для каждой из которых в хеш-таблице должна храниться копия записи.

Приведем стандартную процедуру формирования хеш-адреса по составному ключу.

Пусть по ключевым словам $k_1, k_2, \ldots, k_k$ при помощи одного и того же алгоритма хеширования получены соответствующие битовые строки $B(k_1)$, $B(k_2)$, …, $B(k_k)$, по длине совпадающие с хеш-адресом. Хеш-адрес в этом случае можно определить как битовую строку B, которая вычисляется по следующей формуле:

$$B = B(k_1) \oplus B(k_2) \oplus \ldots \oplus B(k_k),$$

где символ $\oplus$ – операция “Исключающее ИЛИ”, т.е. операция “+” одноименных разрядов по модулю 2.

Если разделить все ключевые слова на группы по функциональному признаку, объем хеш-таблицы можно несколько сократить.

Предположим, что атрибуты A_1, A_2, A_3 (например, “*национальность*”, “*возрастная группа*” и “*местожительство*” граждан) имеют соответственно значения x_1, x_2, x_3. Тогда уже по виду заданного x_i можно установить значение i (1, 2 или 3), что позволяет однозначно идентифицировать каждый из компонентов составного ключевого слова и каждому типу комбинации ключей поставить в соответствие **отдельную таблицу хеширования**.

В данном примере отдельные таблицы должны быть сформированы для составных слов $A_1, A_2, A_3 \Rightarrow (A_1, A_2), (A_1, A_3), (A_2, A_3), (A_1, A_2, A_3)$.

Цепочки резервных ячеек в таких частичных хеш-таблицах существенно короче, чем в общей таблице, поэтому их обработка значительно упрощается.

Сократить хеш-таблицу можно также хешированием лишь по наиболее вероятным или по наиболее длинным комбинациям ключевых слов. По остальным комбинациям производится отдельный поиск, но если их число слишком велико, такие способы неприемлемы.

Один из вариантов частичного преодоления этой трудности заключается в построении с*цепленного хеш-адреса*.

С учетом того, что хеш-адрес представляет собой битовую строку, его можно формировать так, чтобы каждый из ключей, входящих в составное ключевое слово, определял свою группу битов. Полученные биты сцепляются, образуя *полный хеш-адрес*. Если, к примеру, каждому ключевому слову в адресе соответствует 2 бита, то функция хеширования от этого ключа может принимать лишь 4 различных значения: (00, 01, 10, 11). Следовательно, если в со-

ставном аргументе не заданы значения k ключевых слов, то поиск производится только для 4^k сочетаний битов в соответствующих группах. Это дает возможность реализовать достаточно эффективную процедуру многоключевого поиска, хотя при этом сцепленные адреса распределяются по таблице менее равномерно, чем при обычном хешировании, и скорость поиска несколько снижается.

2.4.5. Применение методов хеширования для поиска по соответствию

В технических системах обработки данных по ряду объективных причин информация представляется как последовательность символов двоичного алфавита. Такое представление информации с ростом сложности решаемых задач все более проявляется как чужеродное человеческому мышлению.

В памяти технических систем в процессе запоминания изображения (отражение одного объекта на другом), его пересылки из одного ЗУ в другие не происходит обобщений, выделения каких-либо отличительных признаков, стирания второстепенных деталей [5].

В памяти человека восприятие объекта из внешнего мира сопровождается возникновением эмоций, запоминается не только информация об объекте, но и вызванные им эмоциональные ощущения. Хранящийся в памяти образ (отображение объекта в сознании субъекта) содержит информацию о раздражителе из внешней среды, о субъективных эмоциях организма, вызванных появлением раздражителя, о времени воздействия данного раздражителя. В процессе формирования и запоминания образа происходит такая его обработка, которая соотносит данный образ с определенным классом уже хранящихся образов, т.е. осуществляется ассоциирование запоминаемого образа с уже запомненными и формирование понятий о классе образов. Процесс запоминания связан с образованием следа в мозговых структурах: поток нервных импульсов, несущих информацию об объекте, проходит через нейронные сети, возбуждая на своем пути нервные клетки, из которых формируется нейронный след. В организованных случайным образом нейронных сетях следы памяти расределяются по пространству мозга также случайно. След от одного объекта может иметь общие звенья нейронной сети от других объектов, поэтому нельзя указать точно, в каком месте мозга будет находиться след конкретного объекта информации – образа. Механизм доступа к информации должен базироваться не на указании места хранения информации в среде, а на анализе свойств самой искомой информации. Таким механизмом является механизм ассоциаций. Применительно

к системам обработки данных, ассоциация – это взаимосвязь между информацией на входе запоминающей среды и информацией, хранящейся в запоминающей среде.

Процесс выборки данных из биологической памяти, вероятно, более близок к работе устройств *распознавания образов*, в которых объект представляется в виде *набора сигналов* (часто именуемых *признаками*), формируемых во времени либо параллельно, либо последовательно. В результате анализа *признаков* выдается решение, позволяющее либо идентифицировать объект, либо причислить его к определенному классу. Решение принимается исходя из сравнения результатов, полученных для различных *дискриминантных функций* (для каждой категории объектов – своя функция) при подстановке в них соответствующих наборов входных сигналов.

Для дискретных значений признаков процедуры классификации можно построить на базе методов хеширования, одна из них непосредственно реализует способ выборки, названный *поиском по соответствию* [3].

Предлагаемый метод будет рассматриваться применительно к задаче идентификации слов по аргументу поиска, который представляет собой искаженный вариант одного из хранящихся в памяти слов, предполагается при этом коррекция неправильно заданных аргументов.

Процедура выделения признаков

В рассматриваемом примере ключевое слово раскладывается на ряд признаков, каждый из которых представляет собой небольшую группу букв, причем эти группы могут накладываться друг на друга. Каждая группа формируется из последовательно расположенных букв, в этом случае локальные ошибки сказываются на минимальном количестве признаков .

Возможно образование групп по *циклическому принципу* из букв, стоящих в конце и начале слова, но при этом в группе следует соблюдать тот же порядок букв, что и в исходном слове, чтобы различать слова, полученные друг из друга путем циклической перестановки букв (например *строго* и *острог*). Пусть локальные признаки имеют 3 буквы, тогда слову *строго* соответствует такой набор признаков: *стр*, *тро*, *рог*,*ого*,*сто*, *сго*.

Формируемые из n циклически последовательных символов признаки далее будут называться n-*граммами* (*монограммами*, *биграммами*, *триграммами* и *квадриграммами* при n = 1, 2, 3 и 4 соответственно).

Оптимальный выбор количества символов, образующих признак, определяется в каждом конкретном случае как содержанием ключевых слов, так и их количеством, однако следует отметить, что выбор типа признака не должен прямо зависеть от размера поискового аргумента, так как ошибки, связанные с пропуском или появлением лишних букв, могут приводить к изменению длины аргумента по сравнению с исходным ключевым словом.

Хеширование по признакам

В строке символов, из которых формируются признаки, могут быть ошибки трех видов:

1) ошибки *замены* (одна буква заменена другой);
2) ошибки *включения* (введена дополнительная буква);
3) ошибки *исключения* (буква пропущена).

В случае ошибки типа включения или исключения, позиции всех букв, расположенных справа от того места, где она произошла, меняются. Это не происходит лишь в случае, когда две ошибки противоположного типа расположены рядом и компенсируют действие друг друга, если расположены не рядом – после второй ошибки позиции букв тоже не меняются.

Для сохранения возможно большего количества правильных хеш-адресов в поисковом аргументе, несмотря на возможные искажения, могут применяться разные методы. Одним из наиболее простых является следующая комбинированная процедура. Она состоит из двух этапов:

1) вычисляются функции хеширования для локальных признаков (n-грамм), не зависящие от их позиций в поисковом аргументе;

2) осуществляется комбинационный поиск, в котором эти признаки используются как *ключевые слова*.

Затем отбрасываются слова, у которых признаки по сравнению с поисковым аргументом смещены свыше заданной величины. Например, при смещении более чем на 2 позиции сохраняется возможность идентификации слов, содержащих двойные ошибки любых типов, так как маловероятно, чтобы в ключевых словах (длина их редко превосходит 10 позиций) могли одновременно возникнуть три однотипных ошибки (включения или исключения), причем вблизи друг друга;

Пусть в качестве признаков употребляются только триграммы и используется латинский алфавит (всего 26 букв). Буквы, входящие в триграмму, обозначим x_{i-1}, x_i, x_{i+1}, а триграмме поставим в соответствие величину $v_i = 26^2 \cdot x_{i-1} + 26^1 \cdot x_i + 26^0 \cdot x_{i+1}$.

Если таблица хеширования начинается с адреса B, а ее размер равен H, то вычисляемый методом деления хеш-адрес признака определяется по формуле

$$h(v_i) = v_i \cdot \bmod H + B.$$

Использование *метода деления* для построения функции хеширования позволяет получить не только равномерное распределение хеш-адресов, но и сократить до минимума пространство, отведенное для записи идентификаторов ключевых слов. Поскольку преобразование в числовую форму производится по основанию 26, длину таблицы H следует задавать равной простому числу.

Организация таблицы хеширования

Число хеш-адресов в примере ограничено количеством различных триграмм, которое для 26-буквенного алфавита составляет лишь 17576. При использовании клеточной структуры размер таблицы хеширования будет равен этому числу, умноженному на количество позиций в каждой клетке.

Клетки могут объединяться в *связанные списки*. Хотя метод списков не обеспечивает такого быстродействия, как обычные процедуры хеширования, время выборки оказывается вполне приемлемым.

Рассмотренная структура приведена на рис. 2.15.

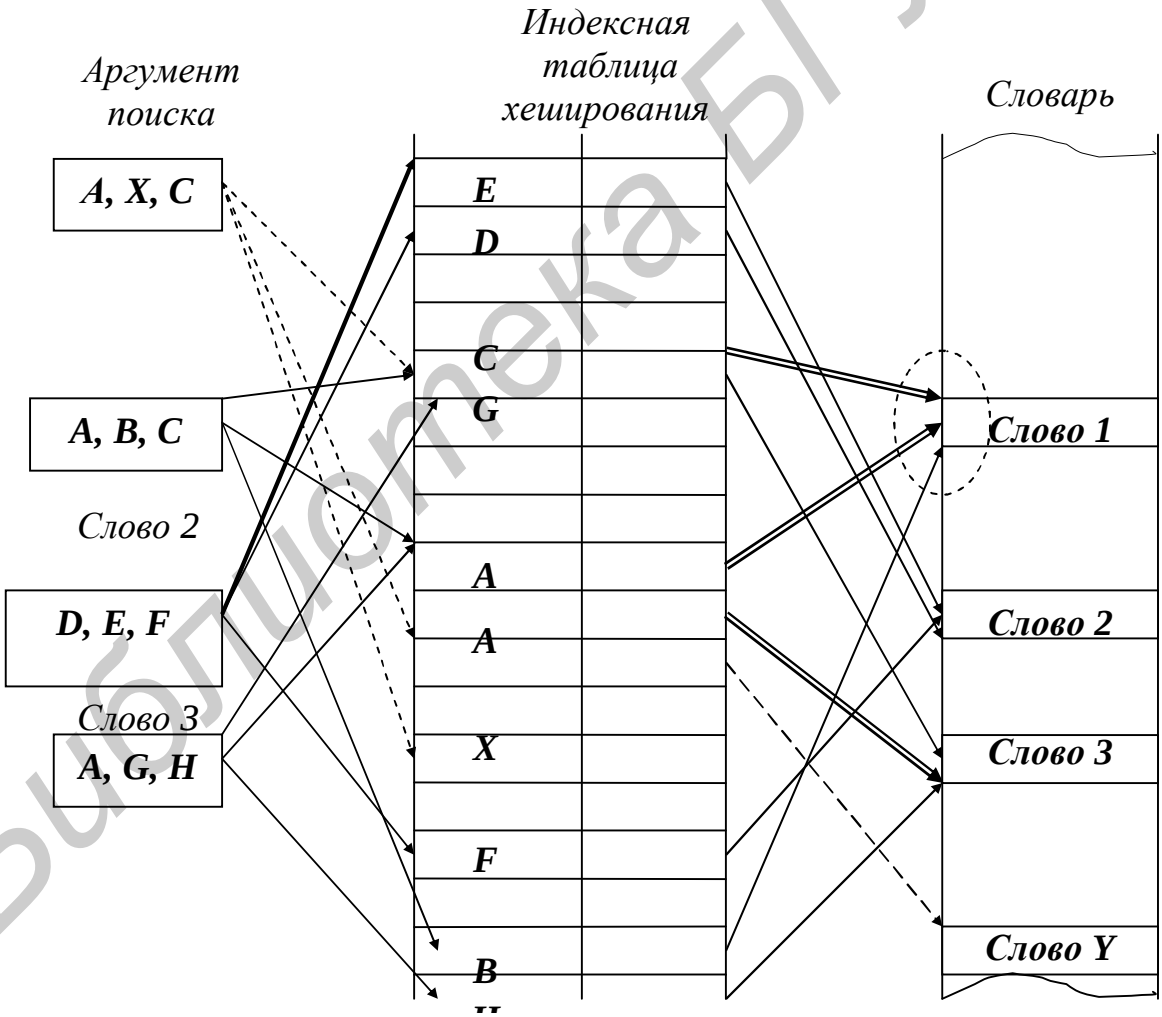


Рис. 2.15. Схема хеширования по совокупности признаков

A, B, X – признаки-триграммы. По признаку *A* имеется коллизия, второй элемент, отвечающий тому же признаку, помещен в следующую по порядку ячейку.

Каждая ячейка таблицы хеширования содержит *идентификатор некоторой триграммы,* а также указатель на область документов, в которой хранятся искомые записи. Сколько признаков содержится в заданном поисковом аргументе, столько указателей извлекается из индексной таблицы хеширования. Если в аргументе присутствует хотя бы один из признаков, отвечающих правильному ключевому слову, то адрес нужной записи дает по меньшей мере один указатель. При наличии ошибок в поисковом аргументе наряду с верными будут выбраны и неправильные указатели.

Далее производится сверка позиций, которые триграмма занимает в поисковом аргументе и найденной записи. Считается, что признаки совпадают, если относительное смещение не превышает заданной величины (в примере не превышает 2).

При вводе в рассмотренный метод *статистической обработки признаков* удается выявлять записи, наиболее похожие на поисковый аргумент (такой способ поиска называют *поиском по соответствию).* Это можно использовать, например, для коррекция неправильно произнесенных слов, сравнивая поисковый аргумент со всеми хранящимися в словаре словами и выбирая то слово, которое в наибольшей степени соответствует аргументу.

Рассмотрим этот вариант поиска подробнее.

При совпадении некоторых из признаков поискового аргумента с признаками в ключевом слове какой-либо конкретной записи связанные с этими признаками указатели дают в числе других и ее адрес. Можно подсчитать количество указателей, направленных на каждую запись-кандидата (их не больше, чем признаков, извлеченных из поискового аргумента). Далее находится запись, ключевое слово которой имеет максимальное число совпадений с аргументом. Учитываются только те указатели, у которых в ключевых словах смещение признаков относительно тех же признаков в поисковом аргументе не превышает заданной величины. Можно также каким-то образом учитывать и другие факторы, например, несовпадение размеров ключевых слов.

При случайно распределенных ошибках хорошей мерой несходства, которую можно использовать для распознавания ключевых слов *X* и *Y,* служит *признаковое расстояние FD.*

$$FD\ \ (X, Y) = max\ \ (n_x, n_y) - n_e,$$

где n_x и n_y – количества извлеченных признаков соответственно из слов X и Y,

n_e – количество не совпавших признаков (в рассматриваемом случае равно числу указателей ячейки области документов, у которых смещения признаков не выходят из установленного диапазона).

При поиске для подсчета указателей строится *таблица совпадений*. Указатель заносится в эту таблицу в случае, если в его ключевом слове обнаруживается такой же признак, как и признак в таблице хеширования, по которому был найден данный указатель. При повторных занесениях данного указателя содержимое соответствующего ему счетчика в таблице совпадений просто увеличивается на 1. Для достижения максимального быстродействия (при большом количестве поисковых аргументов) таблицу совпадений можно организовать по принципу хеширования, когда количество невелико (а обычно так и бывает), сравнение указателей, помещенных в таблицу, может осуществляться путем их *последовательного перебора*.

Опробование рассмотренного метода описано в [3]. Исследовалось действие всевозможных типов одиночных и двойных ошибок. Источниками ключевых слов служили два достаточно больших словаря; один из них содержал более тысячи наиболее употребительных английских слов, второй словарь – известные в науке фамилии. Двойные ошибки формировались при помощи датчика случайных чисел так, что вероятность появления любой неправильной буквы в любой позиции была одинакова, одиночные ошибки задавались поочередно в каждой символьной позиции каждого слова.

По аргументам поиска, содержащим ошибки, к каждому из 2-х словарей было произведено приблизительно 50 тысяч обращений.

Усредненные цифры, выражающие относительное количество успешных операций поиска, приведены в табл. 2.1.

Уровень 1 – относится к тем реализациям, в которых правильное слово имеет минимальное значение признакового расстояния *FD* среди всех кандидатов на выборку.

Уровень 2 – относится к реализациям, в которых правильному слову соответствовал либо наименьший показатель *FD*, либо ближайший к нему.

Таблица 2.1

Относительное количество успешных операций поиска

Длина слова	Исключение Замена Включение	1 0 0	0 1 0	0 0 1	1 1 0	1 0 1	0 1 1	2 0 0	0 2 0	0 0 2
3	Уровень 1	- - 96 - - - - - 39								
	Уровень 2	- - 97 - - - - - 42								
4	Уровень 1	89 93 98 - 83 47 - - 75								
	Уровень 2	93 95 98 - 84 49 - - 80								
5 . . . 10	––– " –––	Максимальное – 100 %; большинство – 90 – 99 %; наименьшие значения – для двойных ошибок исключения								

Контрольные вопросы к разделу 2

1. Что используется для поиска необходимых данных при программном способе адресации по содержанию?

2. Назовите основные особенности программного способа адресации по содержанию (хеширования).

3. В каком взаимоотношении обычно находятся пространство имен и пространство адресов при программном способе адресации по содержанию?

4. Назовите основные области применения хеширования.

5. Назовите основные методы хеширования.

6. Назовите основные функции хеширования.

7. Приведите примеры алгоритмов (способов) перевода ключевых слов в числовые значения.

8. Приведите примеры (способы) преобразования числового значения ключевого слова в хеш-адреса.

9. Что такое коллизия? Когда она возникает? Назовите основные способы (методы) обработки коллизий.

10. Что такое пробинг? При каком методе обработки коллизий он применяется? Назовите основные методы (виды) пробинга. В каких случаях они применяются?

11. Назовите особенности использования области переполнения при обработке коллизий.

12. Что такое рехеширование и когда оно применяется? Назовите основные методы рехеширования.

13. Приведите структуру хеш-таблиц. Назовите основные составляющие части хеш-таблиц, назначение флажков.

14. Приведите основные методы ускорения процедур поиска данных в хеш-таблицах.

15. Назовите особенности клеточной организации хеш-таблиц.

16. Назовите основные виды (типы) связанных списков.

17. Что такое мультисписок? Какую информацию содержит справочник в структуре мультисписка?

18. Назовите основные способы ускорения поиска данных в мультисписках.

19. Назовите некоторые варианты использования составных ключевых слов при хешировании.

20. Назовите основные особенности применения методов хеширования для поиска по соответствию.

3. ЛОГИЧЕСКИЕ ПРИНЦИПЫ ОРГАНИЗАЦИИ АССОЦИАТИВНЫХ ЗАПОМИНАЮЩИХ УСТРОЙСТВ

3.1. Основы организации ассоциативных запоминающих устройств (АЗУ)

3.1.1. Общие замечания

Обстоятельства, способствующие развитию ассоциативных средств хранения и обработки информации, рассмотрены в разд. 1.

В разд. 2 рассматривался программный подход к реализации методов адресации по содержанию, рассчитанный на применение ЭВМ универсального типа. Предполагалось, что информация хранится в запоминающих устройствах произвольного доступа со стандартной системой адресации.

В данном разделе будет рассмотрен аппаратный подход, в котором для хранения и обработки информации используется ассоциативное запоминающее устройство (память с адресацией по содержанию). Само понятие такой памяти подчеркивает аспект быстрого поиска всех информационных объектов, удовлетворяющих заданному критерию поиска. В таком контексте механизм хранения и выборки ассоциаций должен удовлетворять следующим требованиям [5]:

1) покомпонентное представление и распределенное хранение информационных объектов в ассоциативной памяти;

2) доступ аргументов поиска (поисковых объектов) одновременно ко всем объектам в памяти;

3) обеспечение выполнения требуемой операции покомпонентного сравнения;

4) индивидуальное покомпонентное сравнение аргументов поиска с каждым объектом в памяти на основе выбранной меры сходства;

5) сбор и обработка результатов совпадения поисковых объектов с объектами в памяти;

6) выборка поисковой информации.

3.1.2. Логические основы организации АЗУ

Для описания логики функционирования вычислительных устройств чаще всего используется аппарат *булевой алгебры*. Базовой операцией, выполняемой на уровне ячейки памяти с адресацией по содержанию, является *сравнение битов*. Применяется сравнение по критериям «*равно*», «*больше*» и «*меньше*».

Сравнение по критерию «равно»

Обозначим булевы значения двух логических переменных соответственно как *x* и *y*. Булева функция, которая при совпадении *x* и *y* принимает значение 1 *(истина)*, а в противном случае – 0 *(ложь)*, называется *логической эквивалентностью* и описывается выражением:

$$(x \equiv y) = (x \wedge y) \vee (\bar{x} \wedge \bar{y}). \quad (3.1)$$

Обратная по отношению к (3.1) функция ⊕ – *Исключающее "ИЛИ"* описывается выражением

$$x \oplus y = (x \wedge \bar{y}) \vee (\bar{x} \wedge y). \quad (3.2)$$

Логико-запоминающая среда обычной ассоциативной памяти состоит из ячеек, расположенных в узлах решетки, в m строках которой размещаются хранимые слова, а в n столбцах – соответствующие разряды этих слов. Каждая ячейка должна иметь логическую схему для реализации функций (3.1) и (3.2).

При *маскированном поиске* лишь часть разрядов поискового аргумента сравнивается с соответствующими разрядами всех слов памяти. Остальные разряды маскируются (блокируются). Считываются только те слова, у которых значения незамаскированных разрядов совпадают с заданными разрядами аргумента. Чтобы обеспечить выполнение маскирования, в каждой ячейке памяти также должны быть предусмотрены соответствующие средства.

Пусть j-е слово памяти имеет вид $S_j = (S_{j,n}, S_{j,n-1}, S_{j,n-2}, \ldots, S_{j,0})$, где S_{ji} – булево значение i-го разряда, а поисковый аргумент – вид $A = (a_n, a_{n-1}, \ldots, a_0)$.

Если в слове маски $C = (C_n, C_{n-1}, \ldots, C_0)$ блокируемым разрядам присвоить булево значение 1, а остальным разрядам – 0, то в качестве функции, характеризующей совпадение содержимого ассоциативной ячейки с маскированным битом поискового аргумента, можно взять

$$m_{ji} = (a_i \equiv S_{ji}) \vee C_i. \quad (3.3)$$

Если j-й разряд слова S_j замаскирован, значение $m_{ji} = 1$, так как $C_i = 1$.

*Ф*ункция, характеризующая несовпадение, имеет вид

$$\overline{m}_{ji} = (a_i \oplus S_{ji}) \wedge \overline{C}_i. \quad (3.4)$$

S_j согласуется с A во всех незамаскированных разрядах, если для всех разрядов выполняется условие совпадения (3.3), т.е.

$$m_j = \bigwedge_{i=0}^{n} m_{ji}. \quad (3.5)$$

S_j не согласуется с A при несовпадении разрядов хотя бы в одной позиции:

$$\overline{m}_j = \bigvee_{i=0}^{n} \overline{m}_{ji}. \tag{3.6}$$

При рекуррентной процедуре проверки на совпадение (например, для АЗУ с ассоциативным сравнением параллельно по словам и последовательно по разрядам) результат может быть представлен в виде:

$$e_{ji} = e_{j,i-1} \wedge m_{ji},\ e_0 = 1,\ m_j = e_{jn}. \tag{3.7}$$

Можно использовать и другую рекуррентную схему, основанную на формуле (3.6):

$$\bar{e}_{ji} = \bar{e}_{j,i-1} \vee \overline{m}_{ji},\ i = 1,2,...,n,\ \bar{e}_0 = 0,\ \overline{m_j} = \overline{e_{jn}}. \tag{3.8}$$

Сравнение по критерию «больше» и «меньше»

Другой базовой операцией в АЗУ является поиск всех слов, численные значения которых больше (меньше) величины заданного аргумента.

Как правило, в АЗУ **сравнение величин** чаще всего выполняется в несколько этапов последовательно по разрядам. Промежуточные результаты запоминаются, чтобы их можно было использовать на следующих этапах.

Пусть анализ двоичных чисел начинается **со старших** (левых) разрядов. Кроме поискового аргумента $A = (a_n, a_{n-1}, \ldots, a_0)$ и содержимого слова памяти $S_j = (S_{j,n}, S_{j,n-1}, \ldots, S_{j,0})$, в алгоритме участвуют *две вспомогательные последовательности* двоичных чисел – промежуточные результаты сравнений по критериям соответственно «больше» и «меньше»:

$$G_j = (g_{j,n+1}, g_{j,n}, \ldots, g_{j,0}) \text{ и } L_j = (l_{j,n+1}, l_{j,n}, \ldots, l_{j,0}).$$

В ЭВМ наборы L_j и G_j целиком можно не хранить, а представить их **всего двумя** машинными переменными g_{ji} и l_{ji} соответственно, значения которых вычисляются при помощи следующих рекуррентных соотношений:

$$\begin{cases} g_{ji} = g_{j,i+1} \vee \left(\overline{a}_i \wedge S_{ji} \wedge \overline{l}_{j,i+1}\right); \\ l_{ji} = l_{j,i+1} \vee \left(a_i \wedge \overline{S}_{ji} \wedge \overline{g}_{j,i+1}\right). \end{cases} \tag{3.9}$$

В качестве начальных значений задаются $g_{j,n+1} = l_{j,n+1} = 0$.

Тогда в результате всего цикла сравнения получим:

– для $S_{jn} = 1$, $a_n = 0$ будет $S_j > A$, а из формул (3.9) следует, что для всех $i < n g_{ji} = 1$, $l_{ji} = 0$;

– для $S_{jn} = 0$, $a_n = 1$ будет $S_j < A$ и для всех $i < n$ выполняется условие $g_{ji} = 0$, $l_{ji} = 1$;

– для $S_{jn} = a_n$ и $g_{ji} = l_{ji} = 0$ сравнение величин S_j и A необходимо продолжить, анализируя содержимое разрядов с меньшими номерами.

Рассмотренные выше действия повторяются с разрядами $S_{j,n-1}$ и a_{n-1} и т.д. до тех пор, пока в результате сравнения самых младших разрядов не будут вычислены значения g_{j0} и l_{j0}.

Если $g_{j0} = 1$, $l_{j0} = 0$, то $S_j > A$; при $g_{j0} = 0$, $l_{j0} = 1$, наоборот, $S_j < A$.

Из условия $g_{j0} = l_{j0} = 0$ следует, что $A = S_j$.

Комбинация $g_{j0} = 1$, $l_{j0} = 1$ принципиально **невозможна**.

Можно использовать и другой алгоритм, когда рекуррентная процедура сравнения начинается **с младших** разрядов. Это позволяет сократить количество просматриваемых разрядов, если оба сравниваемые числа начинаются с нулей.

Алгоритм описывается двумя рекуррентными формулами:

$$\begin{cases} g_{j,i+1} = \left(\overline{a}_i \wedge S_{ji}\right) \vee \left(\overline{a}_i \wedge S_{ji} \wedge g_{ji}\right); \\ l_{j,i+1} = \left(a_i \wedge \overline{S}_{ji}\right) \vee \left(\overline{a}_i \wedge \overline{S}_{ji} \wedge l_{ji}\right). \end{cases} \qquad (3.10)$$

В качестве начальных значений задаются $g_{j0} = l_{j0} = 0$.

Соотношения между сравниваемыми величинами определяются последними значениями g_{ji} и l_{ji}.

Рекуррентные соотношения (3.9) можно реализовать при помощи так называемых *итеративных цепей* (рис. 3.1).

В каждом из однотипных логических блоков, показанных в виде прямоугольника (в центре рисунка), вычисляется значение функций (3.9). Результаты вычислений последовательно передаются на разряды с меньшими номерами.

Окончательный результат определяется комбинацией сигналов, снимаемых с выходов последнего логического блока.

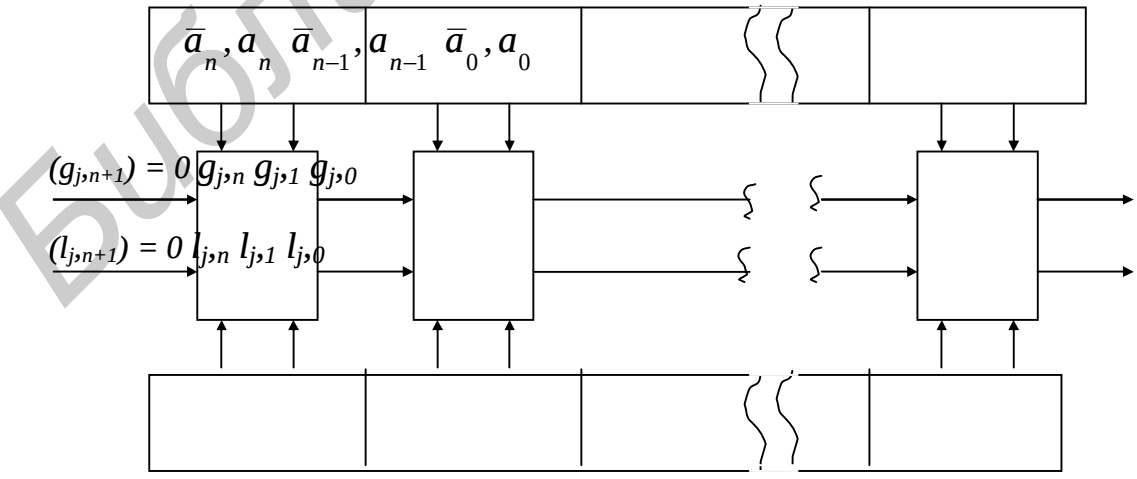


Рис. 3.1. Блок-схема итеративной логической цепи для параллельного сравнения величин

Для реализации отношений (3.9), (3.10) можно применить способ сравнения, основанный на использовании *последовательных вычитающих схем*. Одна из таких схем приведена на рис. 3.2.

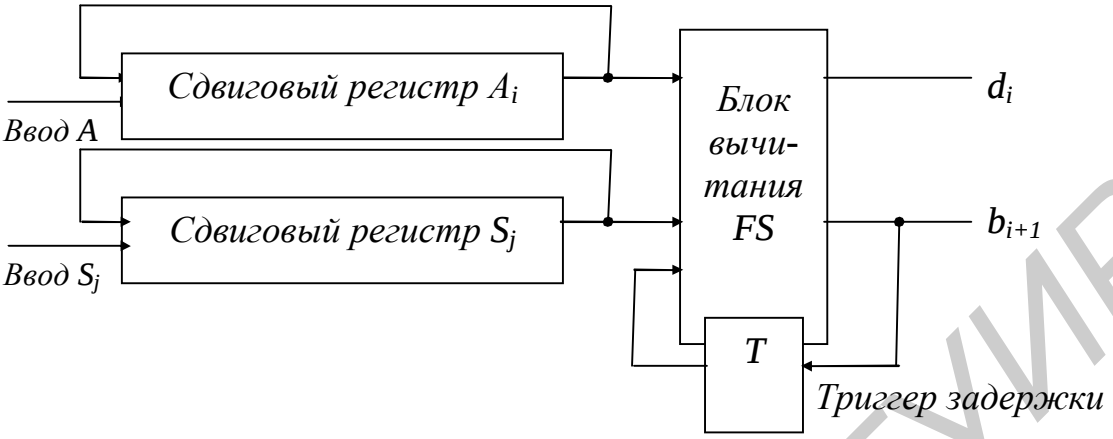


Рис. 3.2. Устройство сравнения с использованием блока вычитания последовательного типа

Содержимое младших разрядов сдвиговых регистров синхронно сдвигается на входы старших разрядов, а на выходе логического блока вычитания в каждом такте генерируются сигналы поразрядной разности d_i и переноса b_{i+1}. Пройдя через триггер задержки на 1 такт, перенос поступает на вход блока вычитания.

Результат сравнения определяется значением последнего сигнала переноса b_n. Если $b_n = 1$, то это означает, что $A < S_j$.

Работу блока вычитания можно описать при помощи *таблицы истинности* (табл. 3.1).

Таблица 3.1

Таблица истинности устройства вычитания

Сигнал разности (d_i)				Сигнал переноса (b_{i+1})			
a_i	S_{ji}	b_i		a_i	S_{ji}	b_i	
		0	1			0	1
0	0	0	1	0	0	0	1
0	1	1	0	0	1	1	1
1	0	1	0	1	0	0	0
1	1	0	1	1	1	0	1

3.2. Структура и основные функции АЗУ параллельного действия

3.2.1. Логическая структура одноразрядной ячейки АЗУ параллельного действия

Рассмотрим структуру памяти (АЗУ), рассчитанную на непосредственную схемную реализацию выражений (3.3) – (3.6). Одноразрядная ячейка такой памяти представлена на рис. 3.3.

Ячейка АЗУ включает в себя узел записи ($И_1$, $И_2$), запоминающий элемент ($И_3$, $И_4$), узел считывания ($И_5$) и узел ассоциативного сравнения по критерию «равно», реализующий функцию «Исключающее ИЛИ» ($И_6$, $И_7$), а также схемы для реализации функции « Встроенное И» (например, *схемы с разомкнутым коллектором*, если используются биполярные транзисторы).

Кроме логики, предназначенной для выполнения операции сравнения, в каждой ячейке имеются адресные линии для считывания и записи информации.

Обозначения на рис. 3.3: A_j – адресный сигнал j-го слова; W_i (1) – запись “1” в разряд i; W_i (0) – запись “0” в разряд i; C_i(1) – сравнение с “1” *i*-го разряда; C_i(0) – сравнение с “0” *i*-го разряда; M_j – сигнал совпадения *j*-го слова; $\overline{B}_i$ – выход информации (разряда *i*).

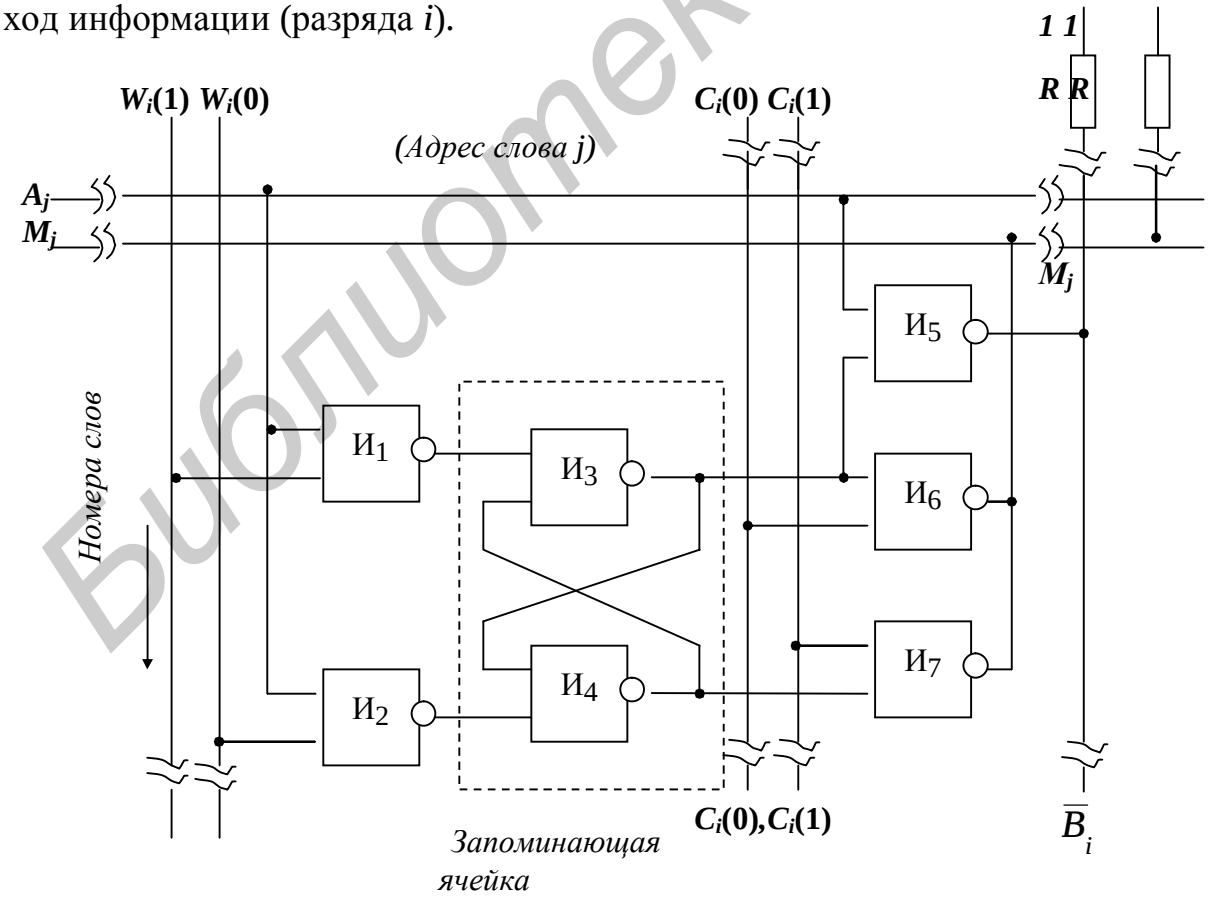


Рис. 3.3. Схема одноразрядной ассоциативной ячейки

Рассмотрим выполнение операций записи / чтения в АЗУ.

Запись

Запоминающий элемент (триггер) предназначен для хранения одного бита информации. Этот триггер устанавливается и сбрасывается сигналами W_i (1) и W_i (0), поступающими с вентилей $И_1$ и $И_2$, при условии, что на адресную линию A_j подан сигнал с логическим уровнем “1”.

Значением записанного бита является выход $И_3$. Оно равно 1, если $W_i(0) = 0$, $W_i(1) = 1$ и $A_j = 1$, а при $W_i(0) = 1$, $W_i(1) = 0$ и $A_j = 1$– равно 0.

Чтобы содержимое данного разряда в ходе записи не изменялось, запись маскируется путем задания $W_i(1) = W_i(0) = 0$.

Считывание

Считывание информации из разрядной ячейки в режиме обычной адресации производится путем подачи на линию A_j сигнала с логическим уровнем “1”, в результате инвертированное значение бита с выхода $И_5$ непосредственно поступает на общую для всех слов памяти линию $\overline{B}_i$.

Считывание в режиме адресации по содержанию (сравнение бита поискового аргумента с содержимым ассоциативной ячейки) с учетом маскирования производится при помощи линий $C_i(0)$ и $C_i(1)$, проходящих через i-й разряд всех слов памяти.

При сравнении с единицей подаются сигналы $C_i(0) = 0$, $C_i(1) = 1$, при сравнении с нулем – $C_i(0) = 1$, $C_i(1) = 0$, чтобы разряд не участвовал в операции сравнения при маскировании, на линии $C_i(0)$ и $C_i(1)$ подается “0”. При совпадении сравниваемых значений потенциалы на выходах схем $И_6$ и $И_7$ отличны от 0 (оба ключа разомкнуты) и на линии M_j появляется сигнал “1”; при несовпадении – выход одного из вентилей $И_6$ или $И_7$ замыкается на “землю” и потенциал линии M_j становится равным 0; при маскировании – выходы $И_6$ и $И_7$ размыкаются независимо от записанного бита.

3.2.2. Анализатор многократных совпадений

В АЗУ может храниться несколько слов, ассоциативные признаки которых по незамаскированным разрядам совпадают с поисковым аргументом. Задача анализатора заключается в выявлении таких слов и их приоритетной выборке.

При поступлении аргумента в этом случае одновременно возбуждается несколько линий M_j (см. рис. 3.3). Далее необходимо поочередно считать из па-

мяти содержимое слов, соответствующих управляющим сигналам на линиях $C_i(0)$ и $C_i(1)$. Именно для этого в ячейках АЗУ предусмотрены *цепи адресного считывания*. Но необходим какой-то *механизм обслуживания очереди*, при помощи которого можно было бы производить последовательный перебор активных линий M_j (обычно в порядке возрастания номеров) и подачу управляющих сигналов на адресные линии соответствующих ячеек памяти.

Для этого *анализатор многократного совпадения* имеет на каждую линию M_j массива АЗУ *буферный триггер фиксации реакции на запрос*, а также *приоритетный искатель* для выделения одной из активных линий.

Рассмотрим некоторые способы реализации анализаторов многократного совпадения.

Приоритетные анализаторы последовательного типа

Один из вариантов такого анализатора показан на рис. 3.4а. В нем логическая цепь, позволяющая выделять среди своих входов, установленных в 1, линию с наименьшим номером (N), построена по принципу *последовательного соединения разрядов*, когда каждый единичный вход M_j этой цепи блокирует действие линий с большими номерами, в результате чего в “1” устанавливается только выход, соответствующий первой активной линии.

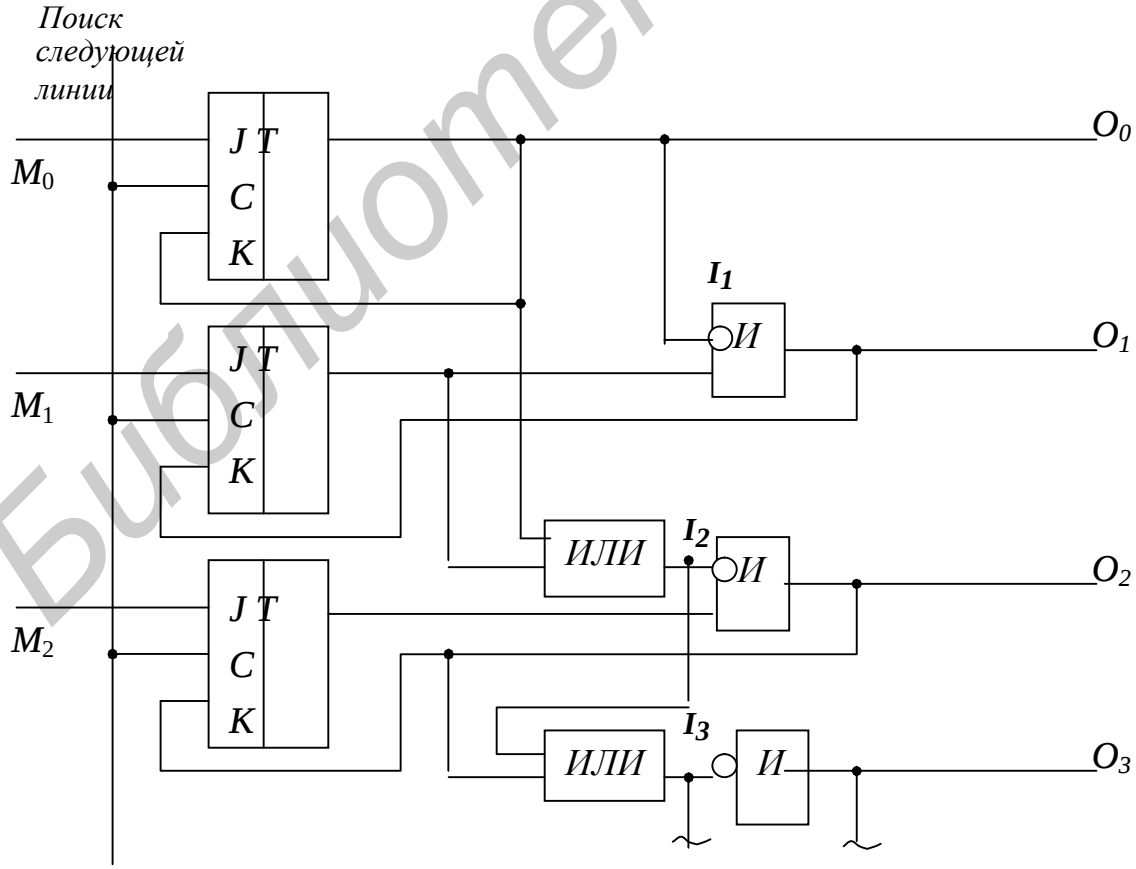


Рис. 3.4а. Схема приоритетного анализатора
с последовательным соединением разрядов

На рис. 3.4а применены следующие обозначения: M_j – сигнал совпадения; I_j – сигнал запрета; O_j – выходной сигнал; ○ — операция “НЕ”.

Чтобы установка в “0” триггера одной активной линии не повлекла за собой сброс триггера следующей линии, используются триггеры, имеющие *вход синхронизации*. Управляющий сигнал «Поиск следующей линии» не нарушает информации, записанной на других триггерах J-K. Единичный сигнал, соответствующий первой активной линии, автоматически идет на выход, а функцией управляющего сигнала «*Поиск следующей линии*» является лишь сброс первого из «ответивших» триггеров.

Более экономичная схема показана на рис. 3.4б, так как в ней применяются простейшие триггеры, что позволяет допустить некоторое снижение быстродействия, связанное с повторным выполнением операции ассоциативного считывания, тем более что они занимают не намного больше времени, чем сброс буферных триггеров.

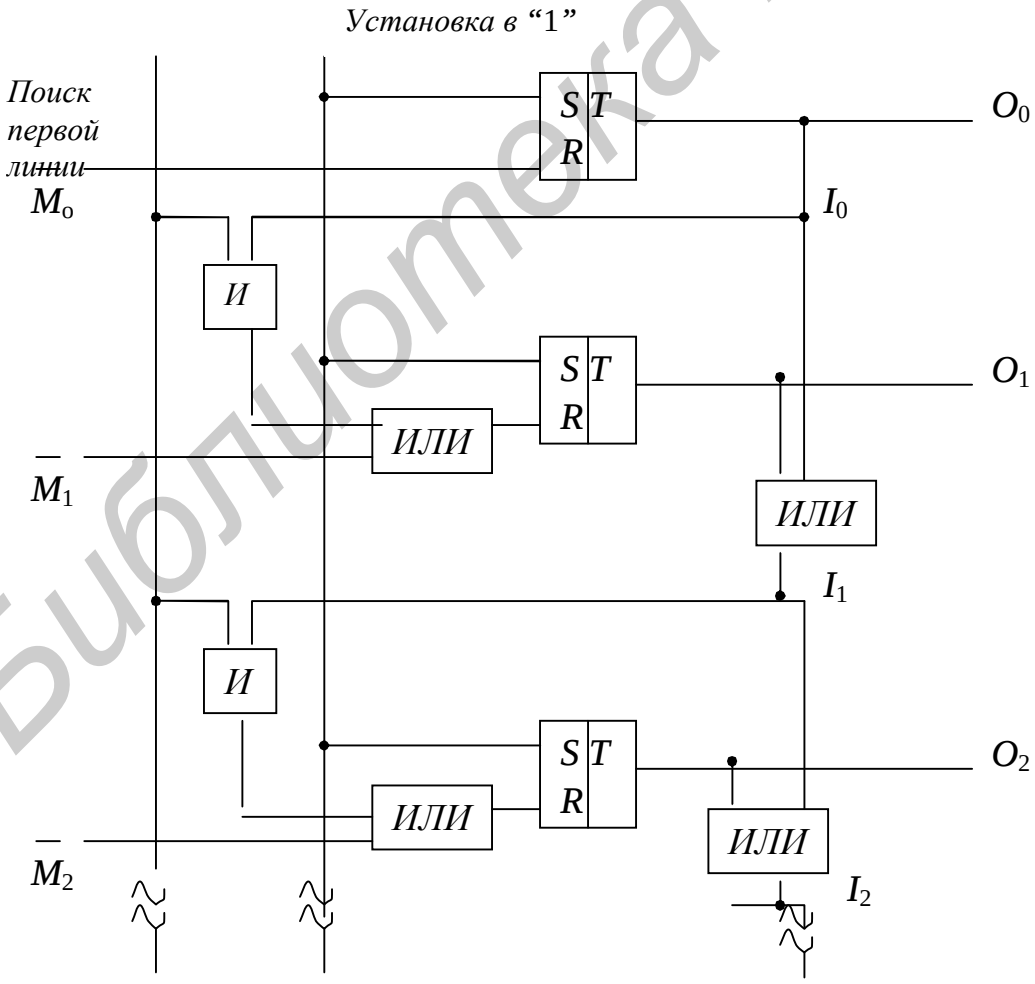


Рис. 3.4б. Схема анализатора многократного совпадения с повторением процедуры ассоциативного доступа

Сигнал управления «Поиск первой линии» подается только после того, как произойдет считывание в режиме адресации по содержанию. По нему осуществляется выделение первой из ответивших линий и сброс триггеров для линий с большими номерами. Для выборки следующих совпавших слов процедура ассоциативного доступа производится повторно. Уже обработанные ячейки помечаются, например установкой в единицу специального разряда.

Недостатком рассмотренных анализаторов последовательного типа является то, что время срабатывания анализатора при этом пропорционально числу слов в ассоциативном накопителе.

Имеются технические решения для сокращения продолжительности приоритетного анализа, использующие модульную организация приоритетных анализаторов и стандартные интегральные модули памяти.

Рассмотрим логический блок, представленный на рис. 3.5а, для формирования вектора запрета.

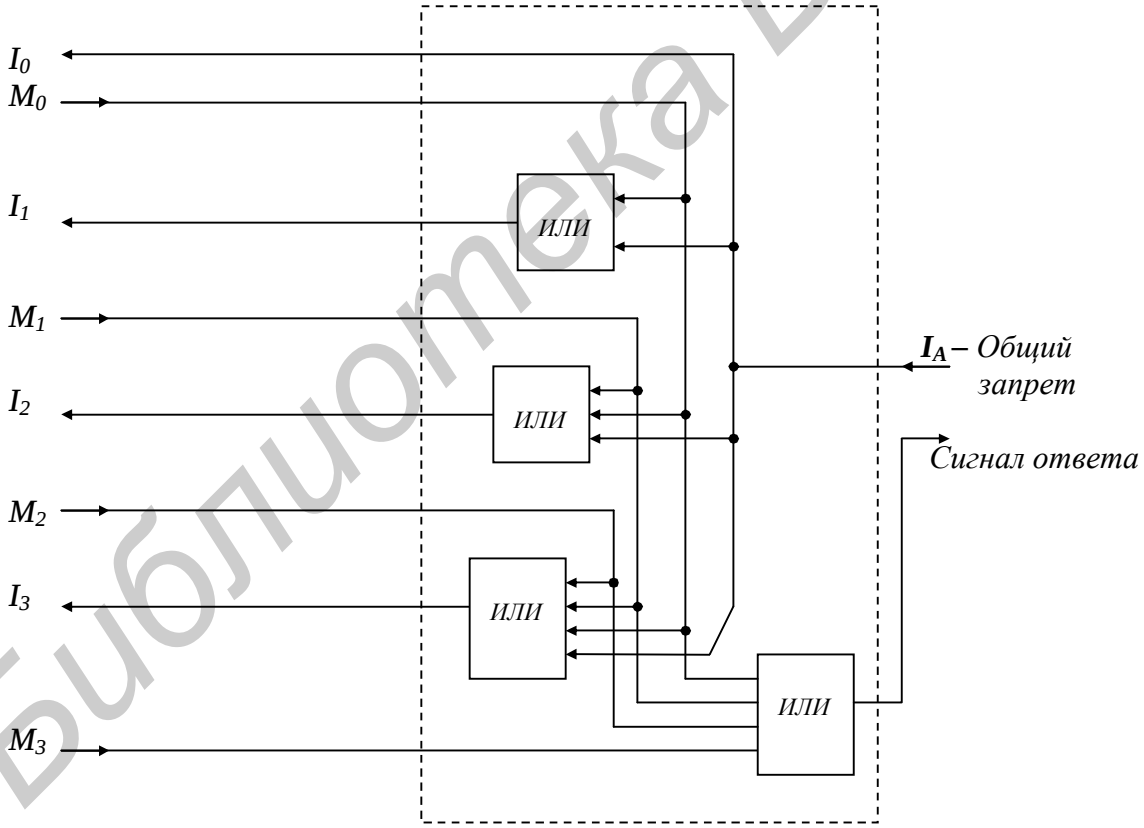


Рис. 3.5а. Отдельный модуль схемы генерации вектора запрета

Вектор запрета генерируется на выходе блока при подаче на его вход сигнала I_A, при этом все выходы I_i устанавливаются в “1” (блок «закрывается»).

Поскольку количество входов на схемах ИЛИ линейно растет с увеличением номера ячейки, блок не применяют в АЗУ большой емкости, но он вполне подходит для небольших АЗУ (например буферной памяти).

Соединяя последовательно блоки в виде древовидной структуры (как показано на рис. 3.5б), можно получить приоритетный анализатор с достаточно большим количеством входов. Сигнал I_A используется для установки всех выходов I_i заданного блока в “1” (при этом сигнал сброса проходит на все буферные триггеры, подключенные к блоку).

Предположим, что все выходные линии, за исключением линии «*Сигнал ОТВЕТА*» расположены с левой стороны (см. рис. 3.5а).

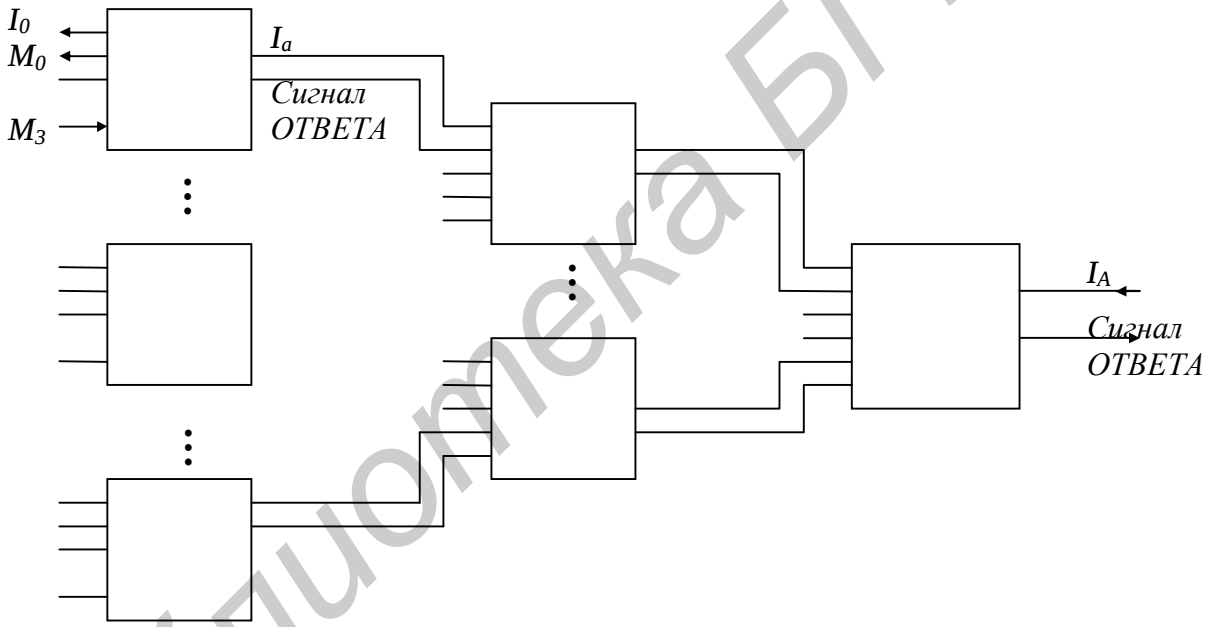


Рис. 3.5б. Древовидная структура анализатора

Сигнал ОТВЕТА (который появляется при совпадении одноименных разрядов в поисковом аргументе и в словах памяти) передается от низших уровней структуры к высшим, появление его на входе любого из блоков приводит к тому, что все его выходы I_i устанавливаются в “1”. Так как эти выходы действуют на блоки более низкого уровня аналогично входу I_A, они последовательно закрывают все блоки, расположенные левее блока с активным входом, а на выходе блоков первого уровня генерируется вектор сигналов запрета, при помощи которого сбрасываются буферные триггеры.

При любых сочетаниях сигналов на линиях M_j всегда вырабатывается правильный вектор запрета. Если имеется хотя бы одна активная линия на выходе последнего уровня структуры (корня дерева), появляется единичный сигнал ответа.

3.2.3. Формирование адреса первой ответившей ячейки АП

Анализатор можно построить таким образом, чтобы он сам генерировал адрес первой по счету ответившей ячейки памяти. Для этого достаточно схему, показанную на рис. 3.5б, дополнить несколькими логическими элементами, реализующими логическую функцию “Встроенное ИЛИ” (см. рис. 3.3)
и соединить блоки, как показано на рис. 3.6.

Можно также для генерации адреса первой ответившей ячейки использовать схему, показанную на рис. 3.7.

Эта схема отличается простотой, но не обладает высоким быстродействием.

Вначале выходные сигналы всех линий M_j от памяти фиксации результатов сравнения устанавливают соответствующие разряды *закольцованного регистра сдвига*. После этого на регистр сдвига и на *счетчик адреса (сдвигов)* подается последовательность тактовых импульсов, содержимое регистра сдвигается в сторону верхних разрядов до появления в первом из них логической “1”, в результате чего подача тактового сигнала автоматически блокируется. Содержимое счетчика непосредственно указывает адрес первого совпавшего слова (в исходном состоянии в счетчике были записаны нули), заносится в адресный регистр и производится считывание слова. Далее первый разряд регистра устанавливается в 0 и автоматически возобновляется подача тактовых сигналов на регистр сдвига и счетчик до появления “1” в первом разряде счетчика, после этого считывается следующее совпавшее слово и т.д. до тех пор, пока не будет обслужена вся очередь.

В данном варианте роль буферных триггеров выполняет регистр сдвига.

Заметим еще, что в анализаторах, как правило, предусматривается выход, указывающий на наличие одновременно нескольких совпадений (количество которых точно определить достаточно сложно), что бывает необходимо для эффективной реализации сложных видов ассоциативного поиска.

3.3. АЗУ с поиском, параллельным по словам и разрядам

3.3.1. Общие сведения о структуре

Структурная схема АЗУ с поиском, параллельным по словам и разрядам, приведена на рис. 3.8.

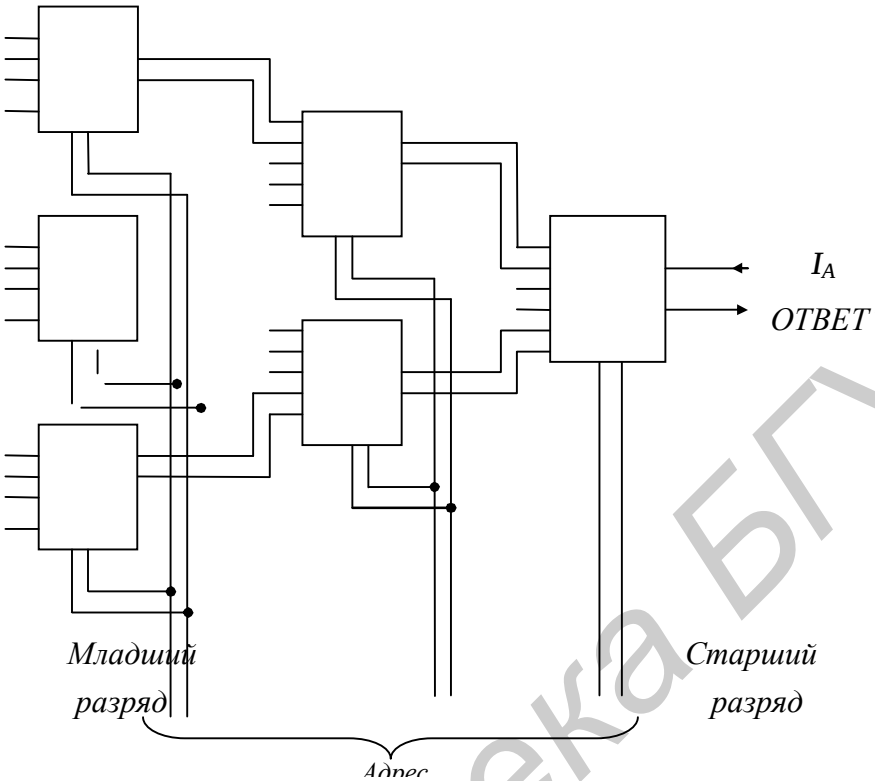


Рис. 3.6. Древовидная структура, составленная из модулей (показано три уровня)

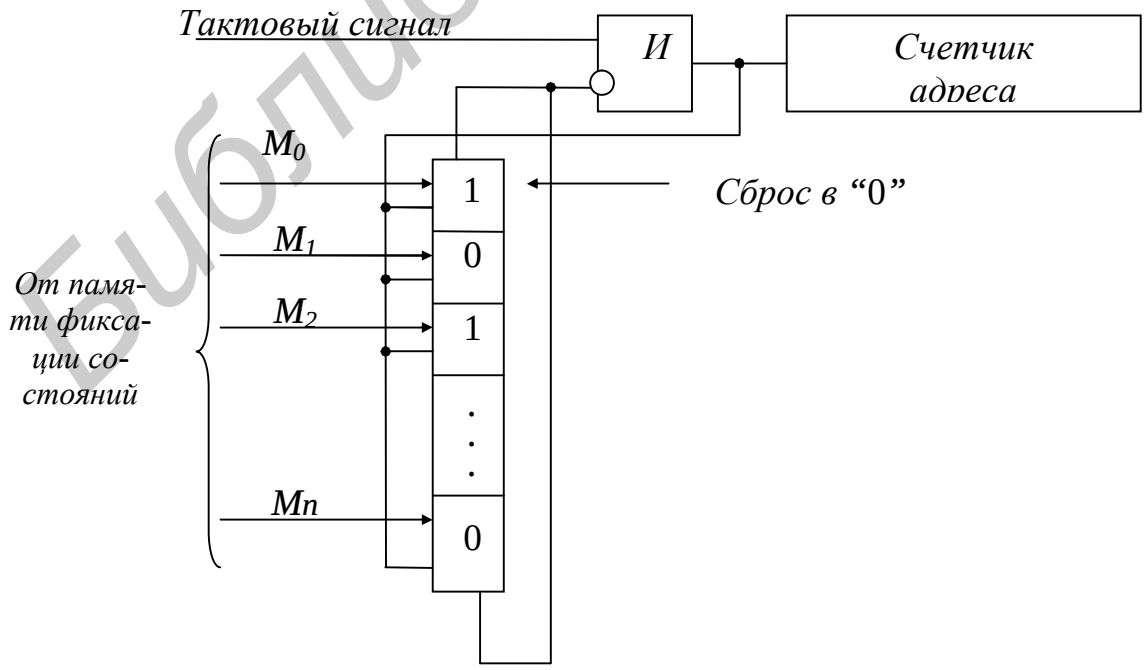


Рис. 3.7. Анализатор совпадения (на базе регистра сдвига)

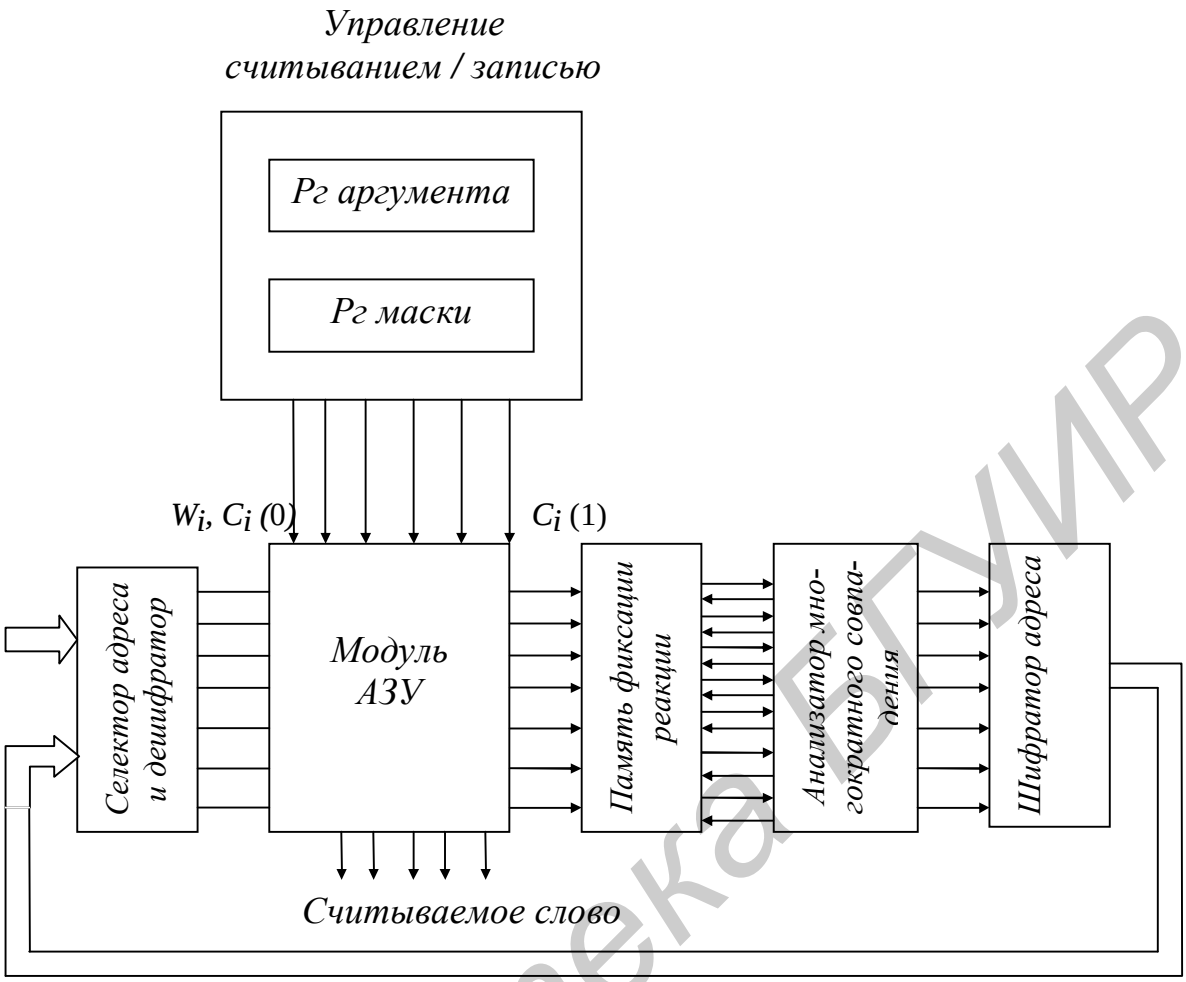


Рис. 3.8. Структура АЗУ

В состав АЗУ входят следующие основные элементы: *модуль АЗУ* (*ассоциативный накопитель*), *регистр аргумента поиска, регистр маски, память фиксации реакции,* образованная буферными регистрами линий *Mj*, *анализатор многократного совпадения, а также вспомогательные цепи управления и передачи данных.*

На выходе блока *управления считыванием/записью* формируется совокупность сигналов C_i (*1*) и C_i (*0*) (см. п. 3.2.1). При выполнении записи в память в этом блоке вырабатываются также сигналы W_i (*1*) и W_i (*0*), по которым аргумент поиска (с учетом маскирования) заносится в ячейку АЗУ с адресом, задаваемым кодом на входе дешифратора адреса.

Для модулей АЗУ, изготавливаемых из ИС и всегда снабжаемых встроенным дешифратором адреса, выходные сигналы анализатора перед подачей на дешифратор адреса предварительно кодируются шифратором, чтобы содержимое всех ответивших ячеек считывалось за один прием .

Назначение и варианты реализации составных устройств АЗУ были рассмотрены в подразд. 3.2.

К вопросу о применении средств маскирования добавим следующее:

1. З*адача поиска информации по совокупности атрибутов*, которые входят в состав поискового аргумента, является наиболее важной среди операций, в которых используются средства маскирования. Каждый из атрибутов описывается соответствующим полем слова, хранящегося в ассоциативной памяти. Путем соответствующего маскирования из этого набора выделяются заданные поля.

2. Чтобы иметь информацию о свободных ячейках памяти для записи в них новых данных, в каждом слове отводится специальный разряд (“*занято*”). В исходном состоянии в нем записан “0”, после записи в ячейку данных – устанавливается “1”*;* если элемент данных удаляется из памяти, этот разряд устанавливается в “0”.

Чтобы найти свободную ячейку, достаточно замаскировать все биты поискового аргумента, кроме разряда занятости, и произвести обычную операцию параллельного сравнения, а ее адрес определить, например, при помощи анализатора многократного совпадения.

3. Маскирование также широко используется при поиске величин, отвечающих определенным числовым соотношениям.

3.4. АЗУ с поиском, параллельным по словам и последовательным по разрядам

3.4.1. Общие сведения об АЗУ с адресной выборкой и с адресацией по содержанию

Для АЗУ с поиском, параллельным по словам и последовательным по разрядам, методы адресации во многих отношениях проще методов, применяемых при построении АЗУ параллельного действия. Кроме того, сами алгоритмы логической обработки часто требуют поочередного доступа к одноименным разрядам всех слов накопителя. Этим обусловлено широкое использование таких АЗУ, в частности, при построении ассоциативных процессоров.

Напомним особенности вариантов построения АЗУ на базе ЗУ с линейной выборкой и ЗУ с адресацией по содержанию.

Построение АЗУ на базе ЗУ с линейной выборкой

Модуль памяти с линейной выборкой представлен на рис. 3.9а.

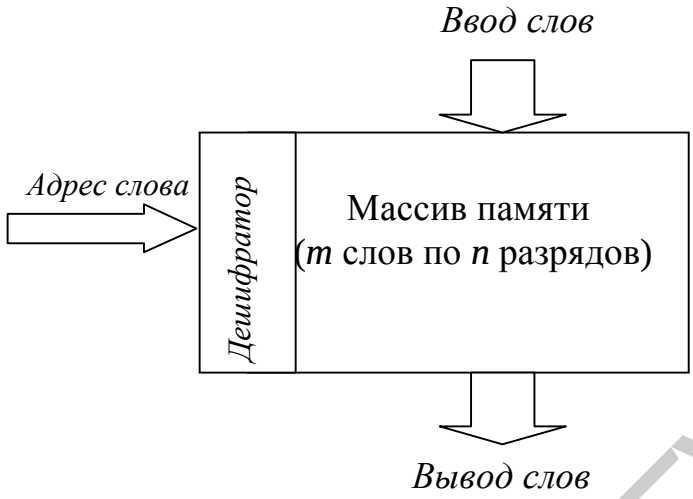


Рис. 3.9а. Модуль памяти с линейной выборкой

Дешифратор указывает адрес в массиве памяти при записи в него или считывании из него одного n-разрядного слова.

Входные и выходные информационные линии (⇓) – общие (соответственно на входе и выходе) для всех слов памяти.

Имеются также *усилители* и *цепи управления считыванием и записью*.

Адресный поиск осуществляется обычно путем последовательной выборки слов, при этом на выходных шинах модуля памяти поочередно появляются все хранящиеся в памяти слова, которые могут сравниваться с некоторой внешней информацией. Процесс поиска занимает до m операций считывания, причем, как правило, $m >> n$.

Параллельный поиск с адресацией по содержанию

Переориентировав массив из n слов по m разрядов таким образом, чтобы слова и разряды поменялись ролями (рис. 3.9б), рассмотренную выше процедуру поиска можно ускорить (правда, за счет увеличения оборудования, занятого в цепях считывания).

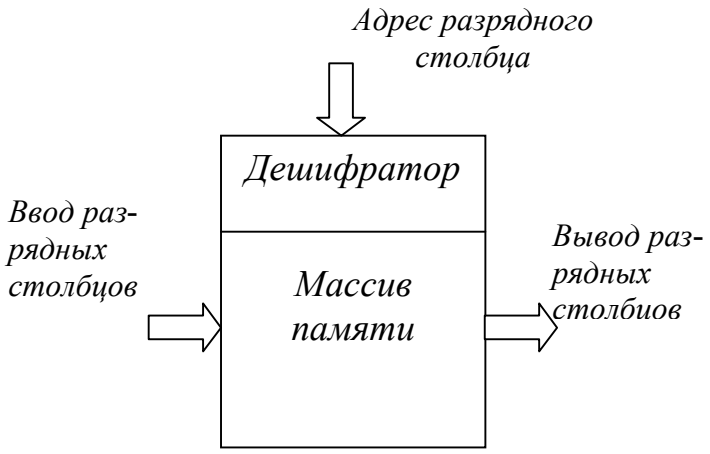


Рис. 3.9б. Модуль памяти с переменой ролями слов и разрядов

Из таких модулей можно, соединив параллельно входы их адресных дешифраторов, составить память большой разрядности для записи и считывания слов требуемой длины.

Дешифратор такой памяти осуществляет выборку *разрядного столбца*, т.е. набора одноименных разрядов всей совокупности слов. Таким образом, считывая одновременно лишь один разрядный столбец (*срез*), поиск можно производить параллельно по всем словам. Поступившее на выходные информационные линии модуля памяти содержимое столбца сравнивается с соответствующим битом внешнего поискового аргумента.

Задавая адрес разрядных столбцов при помощи счетчика, можно поочередно считать все эти столбцы и сравнить их с битами (разрядами) аргумента поиска.

Если предполагается проводить только сравнение на равенство, то для этого достаточно сформировать логическое произведение результатов поразрядных проверок, описываемое выражением (3.3):

$$m_{ji} = \left(a_i \equiv S_{ji}\right) \vee \overline{C}_i \quad .$$

В частности, можно применить и рекуррентную формулу (3.7):

$$e_{ji} = e_{j,i-1} \wedge m_{ji}, \; i = 1,2,...,n, \; e_0 = 1,$$

тогда

$$m_j = e_{jn}.$$

Сравнение слов с аргументом в АЗУ с поиском, параллельным по словам и последовательным по разрядам, чаще всего производится с использованием *памяти результатов* (рис. 3.10), представляющей собой набор *буферных триггеров*, по одному на каждое слово памяти. При помощи сигнала, подаваемого по общей линии, триггеры перед началом поиска устанавливается в “1”. Каждый триггер в ходе поиска может быть сброшен в “0” сигналом, поступающим

с выхода логической схемы "*Исключающее ИЛИ*", в случае, если значение очередного бита, считанного из памяти, не совпадает с соответствующим битом аргумента. После проведения поиска по всему содержимому массива единицы остаются только в тех триггерах, которые соответствуют словам, в точности совпадающим с поисковым аргументом. Весь поиск требует выполнения n операций считывания, что отнимает меньше времени, чем в случае адресного поиска, так как $n << m$.

Заметим, что анализ разрядов поискового аргумента (и слов памяти) можно проводить необязательно по порядку, а в любом другом порядке или лишь по отдельным разрядам (эквивалентно маскированию).

АЗУ, параллельные по словам и последовательные по разрядам, могут функционировать в двух основных режимах: поисковом и вычислительном.

1. В *режиме поиска* обычно требуется лишь определить и считать из памяти слова, удовлетворяющие определенным условиям: например, =, >, < некоторого аргумента, быть в заданном диапазоне чисел, обладать максимальной (минимальной) абсолютной величиной среди всех чисел, хранящихся в памяти, при этом содержимое памяти, как правило, не изменяется.

2. В *режиме ассоциативных вычислений*, как правило, слова подвергаются обработке, а полученные результаты вновь поступают в массив АЗУ, причем часто они заменяют ранее хранившиеся там слова (или фрагменты этих слов). Возможность работы в различных режимах обеспечивается благодаря специальной организации памяти фиксации реакций АЗУ.

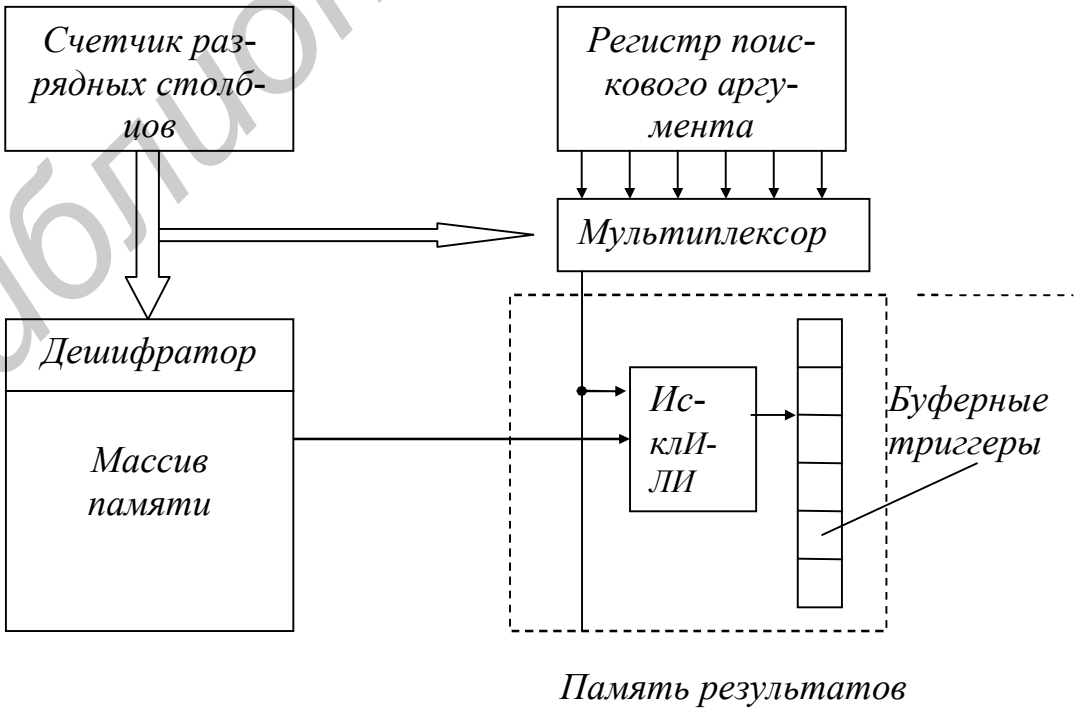


Рис. 3.10. Структура памяти результатов

Вначале рассмотрим простейшую структуру, предназначенную для поиска слов > или < *A*, либо =*A*. Важнейшим элементом этой структуры является *память результатов для проведения операций сравнения величин*.

Для иллюстрации выберем выражение (3.9).

$$\begin{cases} g_{ji} = g_{j,i+1} \vee \left(\overline{a}_i \wedge S_{ji} \wedge \overline{l}_{j,i+1} \right); \\ l_{ji} = l_{j,i+1} \vee \left(a_i \wedge \overline{S}_{ji} \wedge \overline{g}_{j,i+1} \right). \end{cases}$$

Схема ячеек *промежуточных результатов*, используемых при сравнении величин, приведена на рис. 3.11. Для обслуживания слова памяти S_j имеются два триггера g_j и l_j. Логические выходы этих ячеек обозначим как g_{ji} и l_{ji}.

Значения одноименных разрядов a_i и S_{ji} в процессе сравнения содержимого заданного слова с аргументом поиска последовательно и синхронно передаются на входы ячеек промежуточных результатов.

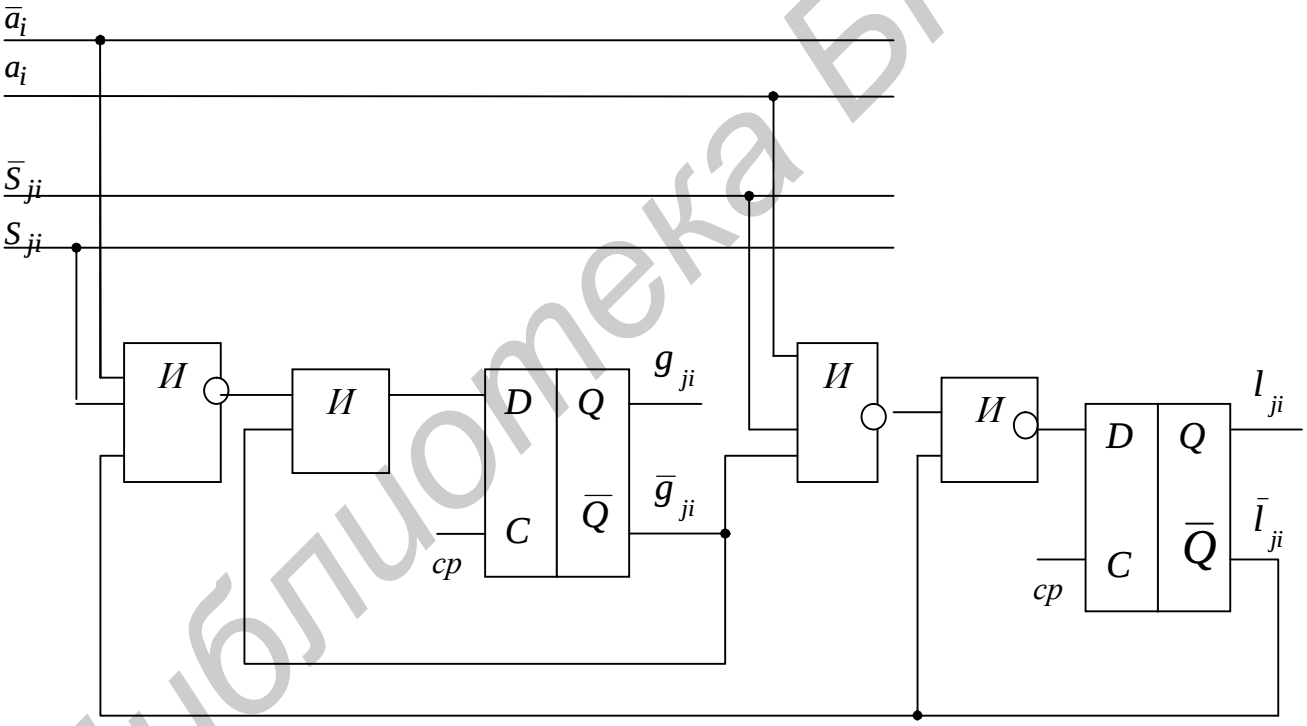


Рис. 3.11. Ячейки промежуточных результатов, используемые при сравнении величин

D-триггеры в каждом такте переходят в новые состояния, которые соответствуют значениям переменных g_{ji} и l_{ji} в формулах (3.9). По совокупности логических значений, зафиксированных на выходах триггеров после завершения поразрядного анализа содержимого слова S_j и аргумента *A*, определяется окон-

чательный результат сравнения. Если $g_{j0} = l_{j0} = 0$, то слова совпадают с аргументом.

По сигналам с выходов памяти результатов, поступающих на *анализатор многократного совпадения*, осуществляется считывание слов, удовлетворяющих заданным соотношениям.

Память результатов, построенная с использованием последовательных (рекуррентных) алгоритмов, позволяет производить поиск информации, основанный на более сложных критериях, в частности:

1) поиск величин, заключенных в заданном интервале;
2) поиск максимального (минимального) значения;
3) поиск ближайшего снизу (сверху) значения;
4) упорядоченную выборку (сортировку);
5) поиск на основе булевых функций;
6) поиск по соответствию.

Алгоритмы выполнения указанных выше поисков рассмотрены ниже, в подразд. 7.3.

3.4.2. Проблема адресного считывания и записи. Методы решения проблемы

Рассмотренная выше базовая конструкция АЗУ имеет ряд недостатков:

а) чтобы запись данных в массив памяти производить по разрядным столбцам, все содержимое массива должно быть предварительно занесено в некоторый буфер (например в отдельную область оперативной памяти);

б) модификация данных вызывает большие трудности, так, изменение лишь одного слова требует **перезаписи всего содержимого массива;**

в) практически невозможно осуществить считывание слов по заданному адресу (эта операция необходима для того, чтобы просмотреть целиком всё слово, определенная часть которого совпала с аргументом поиска).

Таким образом, базовый вариант обладает ограниченными возможностями с точки зрения реализации вычислительных и управляющих функций и применяется только в качестве простейшей памяти АЗУ (например памяти-каталога, см. рис.1.1).

Известны более совершенные варианты описанной базовой конструкции АЗУ, рассчитанной также на использование стандартных модулей памяти [3], в которых благодаря применению специальных методов формирования содер-

жимого массивов обеспечивается адресное считывание и запись как по словам, так и по разрядным столбцам – это память с *диагональной адресацией* на основе сумматора, на основе схем «Исключающее ИЛИ» и др. В таких вариантах элементы исходного массива данных перед записью в память перераспределяют так, чтобы разрядные столбцы размещались по диагоналям, образованным запоминающими ячейками.

Рассмотрим *диагональную систему адресации.*

Эта система (рис. 3.12) была принята за основу при построении ЗУ ряда ассоциативных процессоров, рассматриваемых далее, в разд. 5 данного пособия.

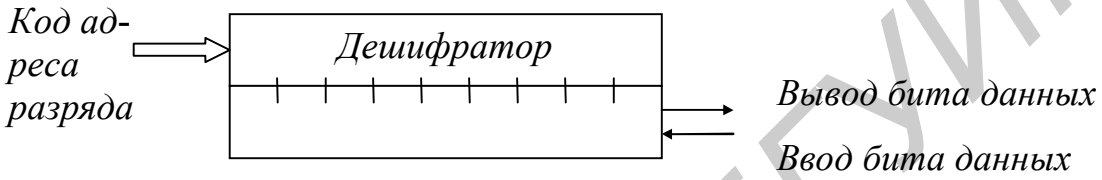


Рис. 3.12. Типовой модуль памяти с диагональной адресацией

В рассматриваемых далее примерах **массив памяти считается квадратным.** На рис. 3.13 показан массив памяти объемом 256 слов по 256 разрядов. Для массива памяти большего объема несколько таких массивов объединяются в один (можно предусмотреть возможность маскирования заданных массивов). Выборка разрядного столбца в этом случае производится параллельно во всех массивах.

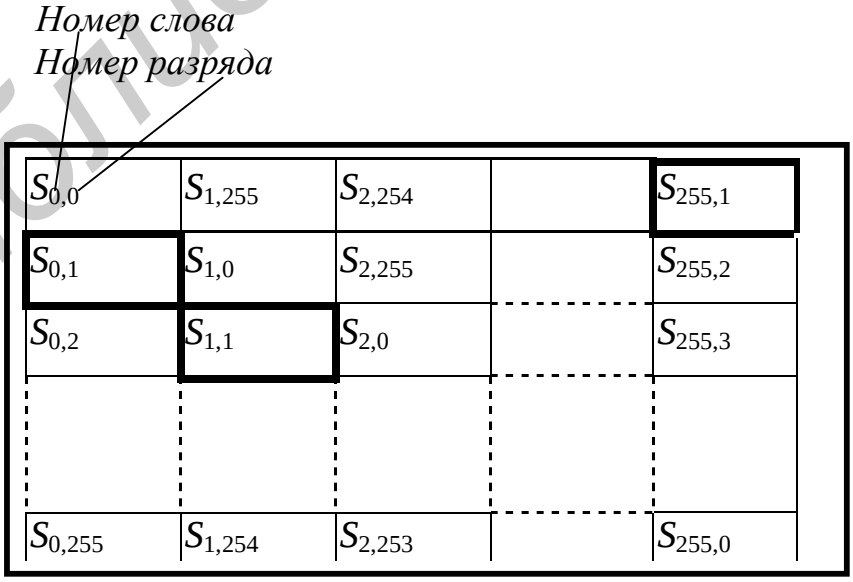


Рис. 3.13. Расположение данных в памяти с диагональной системой адресации

Преобразование исходного массива данных размером 256 х 256 в содержимое памяти с диагональной системой адресации на основе сумматора показано на рис. 3.14.

Рассмотрим, как в памяти с диагональной системой адресации выполняются операции считывания и записи слов и разрядных столбцов.

Каждый дешифратор этой памяти оснащен *блоком арифметического сложения* (+) – полным сумматором, в котором формируется сумма адресного кода, заданного извне, и постоянного схемно реализованного числа, равного N (номеру строки).

Сумма берется по модулю m, где m – количество разрядов слова.

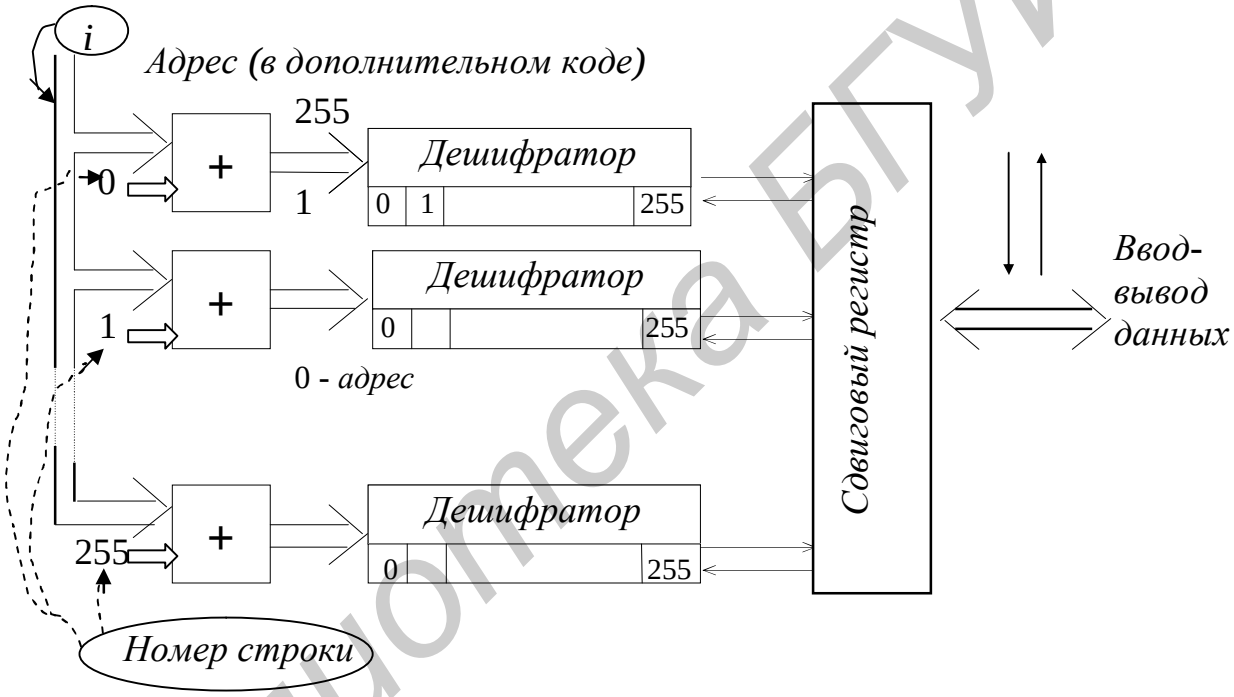


Рис. 3.14. Память с диагональной адресацией
на основе сумматора

Часть сумматоров можно блокировать (применяя соответствующие средства управления) и в результате на их выходах получить требуемый фрагмент адреса разрядного столбца.

Данные, которые необходимо записать в память или считать из нее, помещаются в специальный буферный циклический регистр сдвига. Направление сдвига содержимого этого регистра зависит от типа выполняемой операции. Рассмотрим эти операции.

Процедура записи в память разрядного столбца

Представленный в дополнительном коде адрес разрядного столбца подается на вход сумматора, производится операция сложения, а подлежащие записи данные подаются в буферный регистр, где производится их циклический сдвиг **вниз** на число разрядов, определяемое кодом, полученным в сумматоре.

Далее элементы преобразованного разрядного столбца по команде записи заносятся в соответствующие ячейки памяти. Например, первый разрядный столбец записывается в ячейки, выделенные на рис. 3.13 жирными линиями.

Считывание разрядного столбца

Операция считывания разрядного столбца выполняется аналогично: на вход сумматора подается в дополнительном коде адрес считываемого столбца, а по команде считывания разрядный столбец с переставленными элементами пересылается в буферный регистр, где путем циклических сдвигов **вверх** на то же количество разрядов, что и при считывании, восстанавливается прежний порядок элементов.

В *процессе записи – считывания слова* на вход сумматора подается адрес слова, в буферный регистр памяти – содержимое слова, где перед записью в память или после считывания из памяти сдвигается соответственно **вниз** или **вверх** на количество разрядов, задаваемое адресом; после записи в ячейки памяти i-е слово оказывается в i-м столбце массива памяти.

Другой способ реализации диагональной адресации основан на использовании для формирования адреса вместо сумматора логических схем – «*Исключающее ИЛИ*» (рис. 3.15). Этот способ позволяет работать не только со словами или разрядными столбцами, но и с другими комбинациями битов массива данных, соответствующий код которых сформирован в *регистре режима адресации*. Например, можно сделать так, чтобы опрашивался k-й разряд каждого k-го слова, где k – четное число.

При равенстве нулю всех разрядов регистра режима производится считывание или запись разрядных столбцов, при равенстве единице – считывание или запись слов.

Перестановка разрядов слова данных перед записью его в память на схемах «*Исключающее ИЛИ*» выполняется в соответствии со следующим правилом: если W – номер слова, b – номер разряда, то позиция этого разряда в памяти (номер строки в результирующем массиве) задается равной $W \oplus b$.

Рассмотрим способ реализации ассоциативной памяти на сдвиговых регистрах (рис. 3.16).

Содержимое памяти в каждом такте синхронно сдвигается на 1 разряд вправо (циклически), при этом значения самых правых разрядов всех ячеек (слов) сравниваются с соответствующими битами маскированного аргумента поиска в *памяти результатов*.

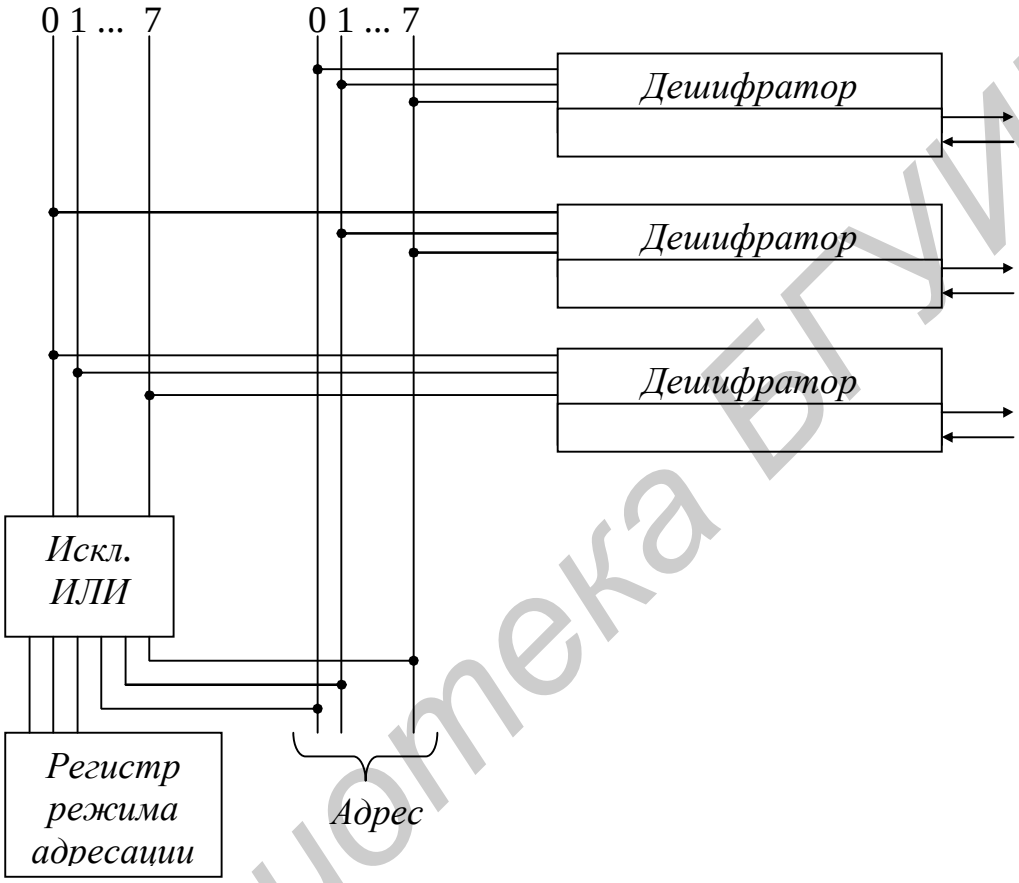


Рис. 3.15. Схема управления памяти с диагональной адресацией на основе функции "*Исключающее ИЛИ*"

Память результатов в данной схеме необходима, поскольку операции сравнения выполняются с отдельными разрядами, и конечный результат вычисляется рекурсивно.

Разряды хранимых в ячейках памяти слов могут нумероваться как слева направо, так и справа налево, в зависимости от алгоритма сравнения, начиная со старших или с младших разрядов слов.

3.5. АЗУ с поиском, последовательным по словам и параллельным по разрядам

Применение АЗУ такого типа при циркуляции информации в логико-запоминающей среде целесообразно в случае, если время получения результатов поиска сравнимо с продолжительностью других вычислительных операций, например, при использовании АЗУ в качестве фильтров данных в каналах передачи цифровой информации в темпе ее подачи. Причем все операции выполняются автоматически, без использования программных средств.

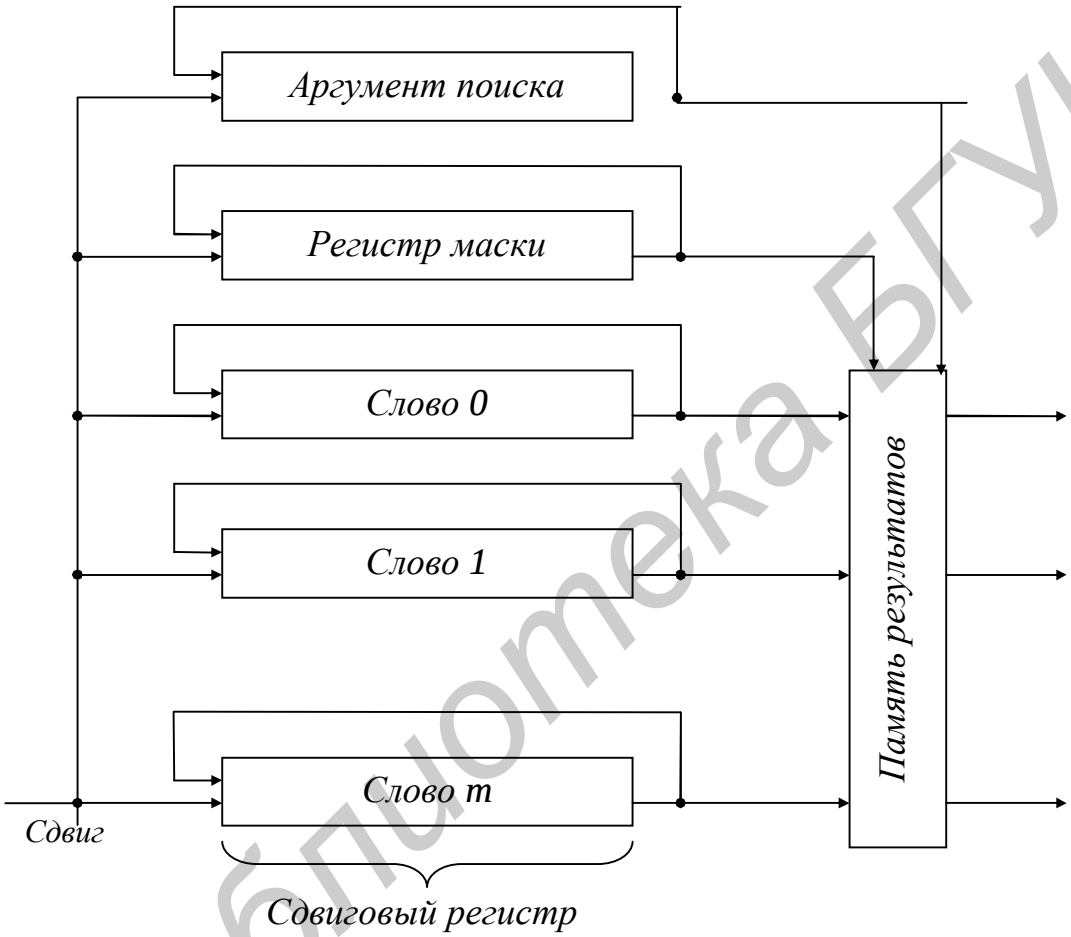


Рис. 3.16. АЗУ на сдвиговых регистрах, параллельное по словам и последовательное по разрядам

Принцип действия АЗУ, последовательного по словам и параллельного по разрядам, рассмотрим по схеме, приведенной на рис. 3.17.

В рассматриваемой схеме для хранения информации используется набор линий задержки (ЛЗ), по которым синхронно циркулируют серии импульсов. К концам линий подключены специальные приемные и передающие цепи для управления процессом циркуляции.

Для ЛЗ могут использоваться различные материалы: магнитострикционная проволока, плавленый кварц, стекло, сапфир и др. Эти материалы могут передавать акустические колебания на частотах в сотни мегагерц с очень малым коэффициентом ослабления. Могут также использоваться приборы с зарядовой связью (ПЗС), память на цилиндрических магнитных доменах (ПЦМД) или обычные *регистры сдвига в интегральном исполнении.*

ЛЗ служит *запоминающей ячейкой*, способной хранить тысячи битов информации (есть импульс – “1”, нет импульса – “0”).

Если разряды слов хранить на отдельных ЛЗ, то слова на выходе системы появляются последовательно друг за другом, а их разряды снимаются с выхода **параллельно**.

Управление работой осуществляется импульсами синхронизации (ИС). Они, в частности, подаются на *адресный счетчик*, текущее значение которого равно номеру слова, присутствующего в данный момент на выходах *усилителей считывания*, т.е. счетчик задает адрес ячейки памяти.

По сигналу «*СТАРТ*» на начало **поиска** слов содержимое *счетчика текущего адреса* передается в *счетчик конечного адреса* и снимается блокировка с *цепей сравнения* (компаратора). При появлении в *регистре считывания* слова, совпавшего с аргументом поиска по всем незамаскированным разрядам, формируется сигнал *сравнения* и производится выборка содержимого *регистра считывания*. При совпадении содержимого *счетчиков текущего* и *конечного* адресов цикл работы памяти заканчивается и формируется сигнал «*СТОП*».

Запись слов в память на ЛЗ производится в режиме обычной адресации при появлении заданного числа (адреса) в *регистре текущего адреса.*

Рассмотренная система при добавлении необходимых цепей может осуществлять и операции сравнения величин.

3.6. АЗУ, параллельные по записям и последовательные по байтам

Память такого типа удобна с точки зрения организации информационного поиска, в частности, поиска документов *по запросам* – для отбора библиотечной информации, где побайтовый просмотр данных является стандартной операцией, а целью – обнаружение в потоке данных **строки** символов (идентификаторов), совпадающей с заданным поисковым аргументом.

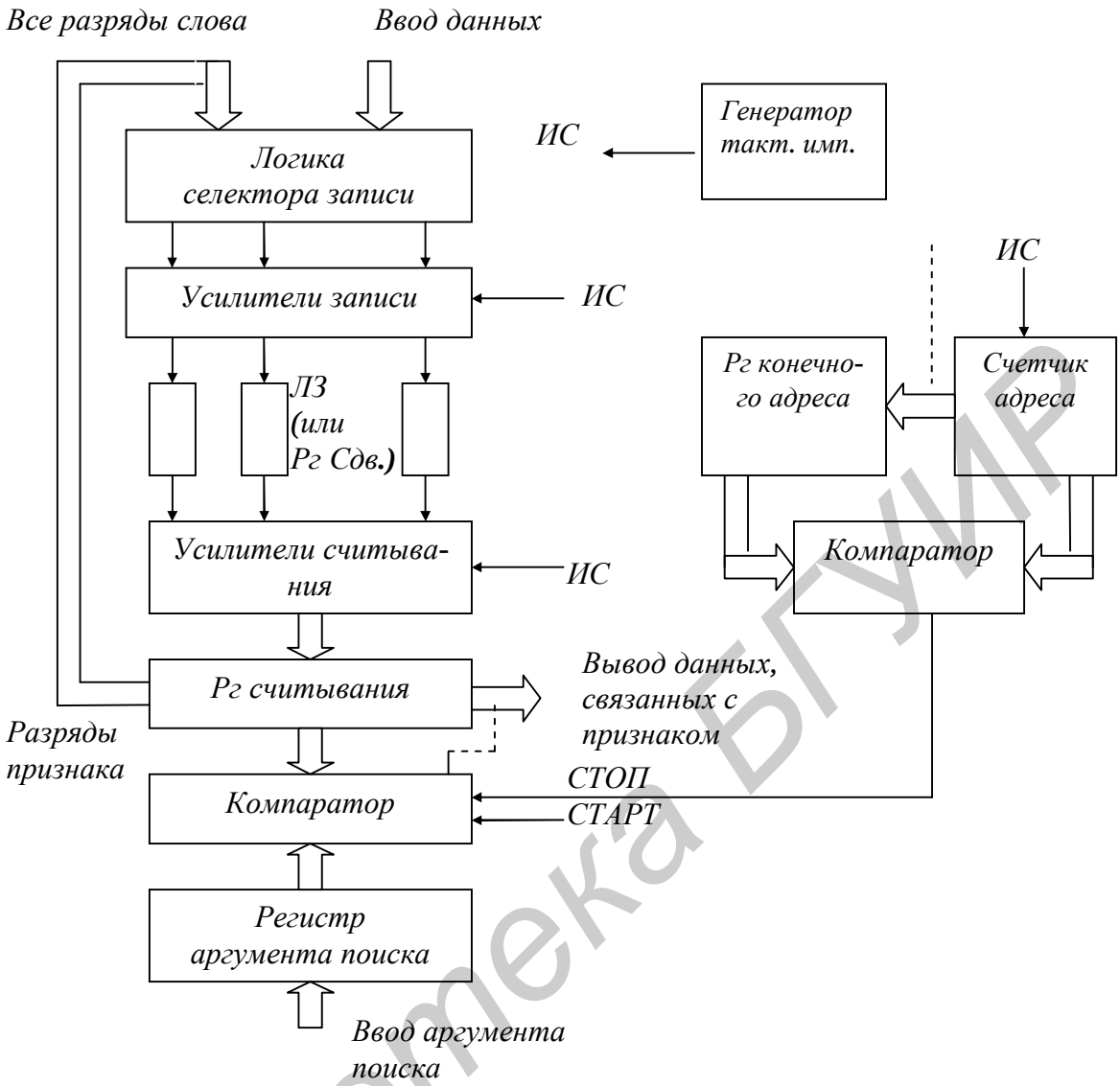


Рисунок 3.17. Структура АЗУ, последовательного по словам и параллельного по разрядам

Обозначения на рис. 3.17:

– отдельная линия;

– параллельные линии;

– сигнал управления;

ИС – импульс (сигнал) синхронизации

Документы обычно содержат большое количество идентификаторов, которые именуются *дескрипторами*. Их наличие или отсутствие в документе можно связать с логической переменной, принимающей одно из двух значений: “*истина*” или “ложь”, а условие поиска заданного подмножества документов мож-

но сформулировать в виде некоторой булевой функции, определенной на наборе таких логических переменных.

Подлежащие идентификации в процессе поиска документов элементы данных представляют собой не слова, а записи переменной длины, содержащие по несколько идентификаторов. Это значит, что стандартные массивы памяти не очень подходят для их хранения. Поиск необходимо проводить параллельно по большой совокупности записей, а сравнение строк – посимвольно (побайтно), так как эта операция обычно требует выполнения многочисленных частных проверок. Соответственно память должна иметь ряд особенностей, отличающих ее от других АЗУ, рассмотренных ранее в данном разделе:

1) большую емкость, которая может наращиваться путем добавления новых модулей;

2) возможность работать с записями переменной длины, состоящими из строк символов;

3) наличие средств для поиска по различным спецификациям и их сочетаниям.

В качестве относительно недорогих и достаточно быстродействующих носителей информации, используемых в архивной памяти, использовались магнитные ленты, барабаны, диски, информация в которых кодируется по байтам, а хранится в виде записей (блоков). Эти ЗУ, конечно, медленнее ЗУ на линиях задержки ЛЗ, но зато они имеют значительно большую емкость.

Пример формата записи в АЗУ (магнитный диск) с последовательной обработкой байтов приведен на рис. 3.18.

В каждой позиции строки (см. рис. 3.18) помещается код символа из восьми битов и один маркерный бит (используется при проведении поиска).

Значения символов для представления данных указаны в табл. 3.2

Поиск строки, совпадающей с аргументом, начинается с его первого символа. При обнаружении в потоке данных такого же символа следующий за ним отмечается маркером. На втором шаге отыскивается маркер, при совпадении его символа с соответствующим символом аргумента устанавливается новый маркер. Рекуррентная процедура поиска продолжается столько шагов, сколько символов содержится в поисковом аргументе.

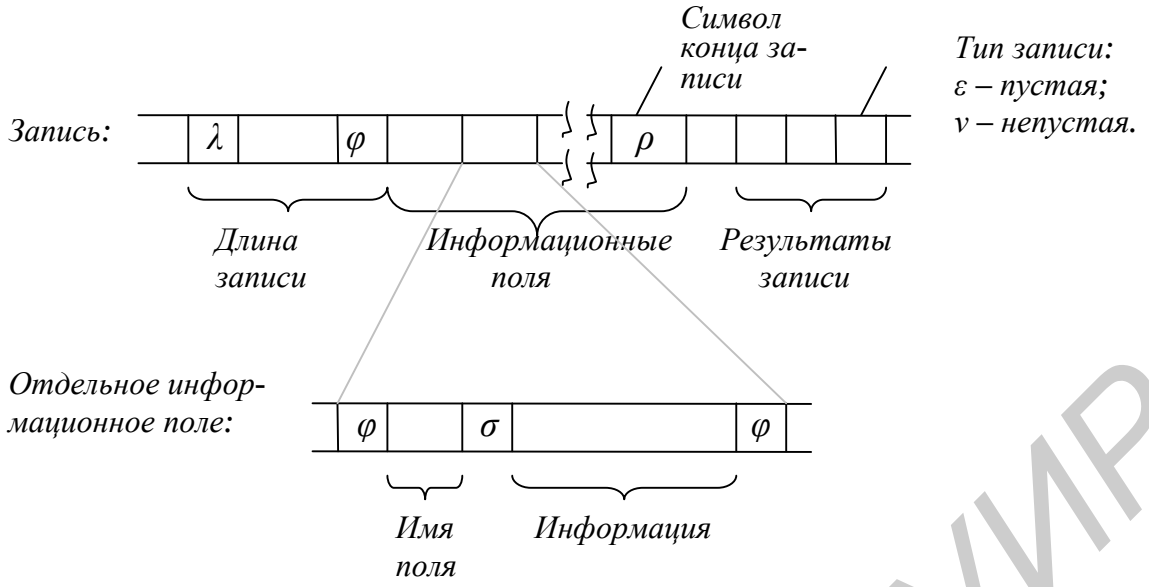


Рис. 3.18. Формат записи в АЗУ с последовательной обработкой байтов

Таблица 3.2

Символы и ограничители для представления данных

Символ	Значение символа
λ (лямбда)	Начало записи
ρ (ро)	Конец записи
σ (сигма)	Разграничитель имени и связанных с ним данных (например, значений атрибутов, размещенных в одном информационном поле)
φ (фи)	Конец любого поля
ε (эпсилон)	Конец пустой записи
ν (ню)	Конец непустой записи
β (бета)	Позиция пустого символа

3.7. Многокоординатные ассоциативные запоминающие устройства

Известен ряд параллельных структур, которые с разной степенью эффективности решают различные задачи обработки информации – это SIMD (один поток команд, множество потоков данных), конвейерная, MIMD (множество потоков команд, множество потоков данных), мультипроцессорная, потоковая, пирамидальная, нейронная, ассоциативная и др.[5].

Архитектура на базе АЗУ наиболее эффективна при преобладании символьных вычислений, разного рода переводных операций, но недостаточно эф-

фективна для локальных и глобальных, линейных и нелинейных и многих других вычислений. Актуальной становится задача создания систем ассоциативной памяти, которые помимо обеспечения возможности выполнения стандартных операций ассоциативной обработки информации должны удовлетворять ряду требований для эффективного выполнения различных вычислительных и нечисловых операций.

На основании проведенных исследований в [5] сделан вывод, что ключевыми свойствами таких АЗУ являются многокоординатный доступ и выполнение параллельной ассоциативной обработки информации одновременно по всем направлениям доступа непосредственно в логико-запоминающей среде. Разработаны принципы организации и функционирования *многокоординатных ассоциативных запоминающих устройств* (*МКАЗУ*), совмещающих функции ассоциативного коллективного доступа к информации с возможностью ее распределенного хранения и параллельной обработки.

Рассмотрен ряд многокоординатных ассоциативных ЗУ:

– с зависимым маскированием, позволяющим, например, определять маскирование ассоциативных ячеек для строчного поиска по маскированию для столбцового поиска и наоборот, что делает возможной локальную обработку растрово представленной информации;

– с фиксацией в ассоциативном накопителе результатов ассоциативного сравнения;

– с использованием ассоциативных ячеек накопителя в качестве источников поисковых аргументов;

– с реконфигурацией функциональных узлов внутри ассоциативных ячеек накопителя, например, с переключением строчного и столбцового каналов ассоциативного сравнения;

– с ассоциативным сравнением по разным направлениям в накопителе (поиск одного и того же поискового аргумента одновременно по строкам и столбцам накопителя);

– с блочной организацией ассоциативного накопителя;

– с матричным расположением поисковых аргументов;

– иерархические ассоциативные АЗУ.

В [5] также разработаны принципы организации и функционирования *многокоординатной ассоциативной среды* хранения и обработки информации. Под ассоциативной средой понимается определенным образом организованная совокупность множеств упорядоченных ассоциативных ячеек, обладающих свойствами отражения, накопления, хранения, анализа, преобразования и обмена информацией.

3.8. Схемотехническая база АЗУ

Для реализации АЗУ применялись многие виды схемотехнической базы. В частности:

1) для построения АЗУ параллельного типа применялись:

– активные электронные схемы, биполярные полупроводниковые элементы (ТТЛ, ЭСЛ, МОП и других технологий);

– сверхпроводящие переключательные элементы – *криотроны* (при этом возникала серьезная проблема с охлаждением);

– элементы на переходах Джозефсона (сверхпроводящие квантовые интерферометры с использованием квантово-механического туннельного эффекта);

– арсенид-галлиевые переходы;

2) для построения АЗУ последовательного типа применялись:

– цилиндрические тонкие магнитные пленки (ЦТМП);

– БИС;

– магнитные сердечники;

– сдвиговые регистры;

– приборы с зарядной связью (ПЗС);

– цилиндрические магнитные домены (ЦМД).

Кроме перечисленных типов схемотехнических АЗУ, были разработаны магнитооптические АЗУ, в частности:

– лазерные;

– с использованием принципов голографии.

В данном пособии мы ограничимся только перечислением типов схемотехнической базы, применяемых для построения АЗУ.

Подробно эти варианты схемотехнической базы АЗУ рассмотрены в [3].

Контрольные вопросы к разделу 3

1. Назовите основные задачи АЗУ, реализующих поиск по содержанию.

2. Какая логическая операция является базовой в ЗУ с адресацией по содержанию и какие логические функции используются при ее выполнении?

3. Приведите логические формулы для определения совпадения и несовпадения аргумента поиска (A) и слова в памяти (S_j).

4. Назовите способы реализации операции сравнения A и S_j.

5. Назовите основные функции АЗУ.

6. Назовите основные типы приоритетных анализаторов.

7. Приведите примеры способов формирования адреса первой по порядку ответившей ячейки АЗУ.

8. Назовите основные блоки АЗУ.

9. Назовите основные особенности применения маскирования при выполнении различных операций в АЗУ (считывания, поиска, записи).

10. Объясните, почему ассоциативный поиск, параллельный по словам и последовательный по разрядам, эффективнее обычного адресного поиска.

11. Какой метод чаще всего применяется при выполнении проверки на совпадение в памяти результатов?

12. Назовите основные типы поисковых операций, выполняемых в АЗУ, параллельных по словам и последовательных по разрядам.

13. В чем заключаются основные проблемы адресного считывания и записи в АЗУ базовой структуры с поиском, параллельным по словам и последовательным по разрядам?

14. В чем заключаются основные особенности диагональной адресации АЗУ?

15. Назовите известные вам виды АЗУ.

16. Дайте определение многокоординатного ассоциативного запоминающего устройства и ассоциативной среды.

4. МЕСТО АССОЦИАТИВНОЙ ПАМЯТИ В ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

4.1. Основные направления применения АЗУ в вычислительных системах

В учебном пособии по дисциплине «Организация и функционирование традиционных и интеллектуальных компьютеров» [9] рассмотрено несколько примеров применения ассоциативной памяти (АЗУ) в ЭВМ, в частности, такие направления применения АЗУ, как:

– организация виртуальной памяти;

– динамическое распределение памяти;

– ассоциативная буферная память.

В данном разделе рассмотрены другие случаи применения АЗУ:

– реализация программируемой логики;

– выполнение различных управляющих функций.

4.2. Программируемая логика

Замена фиксированной логики программной всегда привлекала внимание разработчиков вычислительной техники как эффективный метод построения универсальных схем и упрощения процесса их проектирования и изготовления документации. Создание арифметико-логических устройств (АЛУ), управление которыми осуществляется при помощи микропрограмм, явилось важным этапом на этом пути. Для хранения микроопераций обычно используется ЗУ с произвольным доступом и достаточно большой длиной слов – это память типа ROM (постоянная, только чтение) или обычная (чтение/запись). Проблемы потери информации при выключении электропитания в случае ROM исключаются, при использовании обычной памяти загрузка (перезагрузка) управляющей программы выполняется аналогично загрузке программ на языках высокого уровня (при наличии программы начальной загрузки).

Программирование логики при помощи разных видов памяти заслуживает внимания. Поэтому кратко рассмотрим это направление [3].

4.2.1. Программирование логики при помощи памяти с произвольным доступом

Рассмотрим пример, приведенный в табл. 4.1, где x, y, z – независимые двоичные переменные, а f_1, f_2, f_3 – соответствующие булевы функции. Независимые булевы переменные полностью определяют таблицу истинности (для N переменных 2^N строк).

Таблица 4.1

Таблица истинности

	x	y	z	f_1	f_2	f_3	
Адреса	0	0	0	0	0	1	*Содержимое памяти*
	0	0	1	0	1	1	
	0	1	0	0	0	1	
	0	1	1	1	0	1	
	1	0	0	0	0	1	
	1	0	1	0	1	0	
	1	1	0	0	0	1	

Считая двоичное число (x, y, z) адресом в памяти с произвольным доступом, число (f_1, f_2, f_3) можно теперь считать двоичным словом, помещенным по этому адресу.

Записав содержимое правой половины табл. 4.1 в память емкостью 8 слов по 3 разряда, логические значения f_1, f_2 и f_3 можно получить, задавая адрес (x, y, z) на входе дешифратора.

4.2.2. Программирование логики при помощи ассоциативной памяти

Если логические функции имеют значение “1” в небольшом числе строк, а многие из комбинаций логических переменных запрещены или недоопределены (что часто бывает на практике), стандартный модуль памяти с 2^N ячейками будет иметь малый коэффициент заполнения.

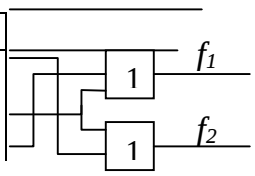
Использование ассоциативного массива устраняет этот недостаток. Рассмотрим пример, приведенный в табл. 4.2. В данном примере функции заданы следующими соотношениями:

$$\left.\begin{aligned} f_1 &= \left(\overline{A} \wedge B \wedge \overline{C} \wedge \overline{D} \wedge E\right) \vee \left(A \wedge \overline{B} \wedge C \wedge D \wedge E\right), \\ f_2 &= \left(\overline{A} \wedge \overline{B} \wedge C \wedge \overline{D} \wedge E\right) \vee \left(\overline{A} \wedge B \wedge \overline{C} \wedge D \wedge E\right). \end{aligned}\right\} \quad (4.1)$$

Таблица 4.2

Таблица истинности для единичных значений f_1 и f_2

A	*B*	*C*	*D*	*E*	f_1	f_2
0	0	1	0	1	$_1$0	1 $_1$
0	1	0	0	1	$_2$1	1 $_2$
1	0	1	1	1	$_3$1	0 $_3$



Считая (A, B, C, D, E) аргументами поиска, двоичные значения левой половины таблицы (части *A–E*) можно запомнить в АЗУ из трех слов по 5 разрядов.

Предположив, что каждой строке соответствует выход слова, для формирования функции f_1 необходимо подать второй и третий выходы на схему ИЛИ, а для формирования функции f_2 – первый и второй выходы на другую схему ИЛИ.

4.2.3. Программирование логики при помощи функциональной памяти

Участки поискового аргумента, которые используются в операциях сравнения с записанными в памяти словами, в АЗУ обычного типа задаются при помощи маски. Маска исключает из процесса сравнения одни и те же разряды всех слов (ячеек). Но если ввести дополнительные управляющие символы, то можно определять по отдельности для каждой записи индивидуальную последовательность исключаемых символов.

Если же в обычных АЗУ использовать наряду с двоичными значениями “0” и “1” знак произвольного значения “*Ø*” и устанавливать для каждого слова индивидуальное сочетание маскируемых разрядов, то отдельные фрагменты информации, рассматриваемые как «неопределенные», при выборке могли бы пропускаться.

Такие АЗУ (с маскированием внутри памяти) могут применяться для реализации логических функций, используемых при проведении вычислений и в операциях управления.

АЗУ, имеющие встроенные средства маскирования слов памяти, получили название *функциональной памяти* (*ФП*).

Рассмотрим структуру ячейки ФП.

Для записи трех символов “0”, “1” и “∅” каждая одноразрядная ячейка памяти должна состоять не менее чем из двух двоичных запоминающих элементов.

Три символа можно закодировать при помощи двух двоичных цифр четырьмя способами, например один из них:

$0 \leftrightarrow (1,0)$; $1 \leftrightarrow (0,1)$; $\varnothing \leftrightarrow (0,0)$.

В аппаратно реализованной ячейке ФП, кроме двух бистабильных схем, должны быть также логические цепи сравнения.

Один из вариантов реализации одноразрядной ячейки ФП приведен на рис. 4.1.

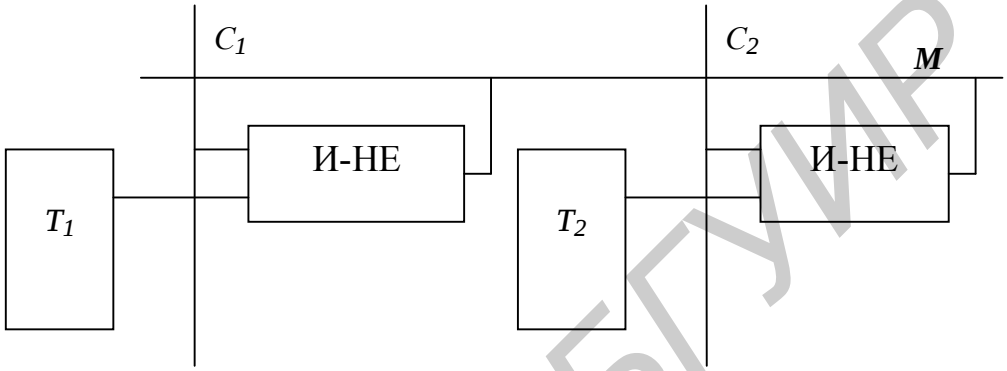


Рис. 4.1. Структура ячейки ФП

По структуре ячейка ФП напоминает ячейку АЗУ, показанную на рис. 3.3, но в отличие от нее не имеет цепей адресного считывания.

Ячейка ФП работает следующим образом.

Предположим, что данный разряд поискового аргумента не маскируется. Если значение этого разряда равно “0”, то $C_1 = 0$, $C_2 = 1$; если оно равно “1”, то $C_1 = 1$, $C_2 = 0$.

Пусть оба триггера (T_1 и T_2) находятся в состоянии ”0”, что соответствует значению “$\varnothing$”, так как к линии M в данной ячейке подключены только вентили И-НЕ, сигнал несовпадения на эту линию не подается, какое бы сочетание сигналов ни присутствовало на линиях C_1 и C_2.

Применение ФП позволяет реализовать сжатые таблицы истинности, полученные при помощи известных методов Квайна – Мак-Класки и карт Карно – Вейча.

Отметим, что использование специальной памяти для реализации логических функций оправдано лишь в случае ее применения для реализации большинства логических структур системы, например, замены управляющего блока программируемой логики.

Для примера рассмотрим типичную мини-ЭВМ, которая использует приблизительно 50 независимых логических переменных и приблизительно 50 управляющих функций, для которых эти переменные являются аргументами.

Обычно в булевы выражения входят различные наборы, состоящие из небольшого количества переменных. Полная таблица истинности в этом случае имела бы около 10^{15} строк, что практически исключает возможность ее реализации при помощи памяти прямого доступа и АЗУ. Поэтому остается единственно приемлемым способом реализация логики на основе функциональной памяти.

Условия, определяющие логику управления процессом вычислений, можно представить набором булевых выражений в дизъюнктивной нормальной форме, т.е. в виде логических сумм логических произведений.

Все присутствующие в выражениях сомножители произведения помещаются в соответствующие строки объединенной таблицы истинности, которая включает, кроме обычной, и сжатую таблицу. Строки сжатой таблицы содержат также «безразличные» значения логических аргументов (∅), они помещаются в массив функциональной памяти. Позиции строк этого массива, которые соответствуют единичному значению соответствующей функции, объединяются при помощи набора логических схем ИЛИ (по одной на каждую функцию), как это было показано в п. 4.2.2.

Рассмотрим пример реализации при помощи функциональной памяти следующих логических выражений [3]:

$$\left.\begin{aligned} f_1 &= \left(A \wedge \bar{B} \wedge C\right) \vee \left(\bar{D} \wedge E\right) \vee \left(F \wedge \bar{G}\right), \\ f_2 &= \left(B \wedge D \wedge F\right), \\ f_3 &= \left(A \wedge \bar{B} \wedge C\right) \vee \left(K \wedge L\right). \end{aligned}\right\} \qquad (4.2)$$

Содержание массива ФП представлено в табл. 4.3 (знаком “∅” отмечены безразличные значения).

Таблица 4.3

Реализация выражений (4.2) при помощи ФП

	A	B	C	D	E	F	G	K	L	f_1	f_2	f_3
	1	0	1	∅	∅	∅	∅	∅	∅	1	0	1
	∅	∅	∅	0	1	∅	∅	∅	∅	1	0	0
	∅	∅	∅	∅	∅	1	0	∅	∅	1	1	0
	∅	1	∅	1	∅	1	∅	∅	∅	0	1	0
	∅	∅	∅	∅	∅	∅	∅	1	1	0	0	1

Заметим, что порядок строк в табл. 4.3 может быть произвольным.

Рассмотрим ФП, ориентированную на реализацию булевых функций, содержащих только операции $\wedge$ и $\vee$.

Такая функция в общем случае имеет вид

$$F = V_p\left(A_1^{P_1} \wedge A_2^{P_2} \wedge ... \wedge A_n^{P_n} \right) \Big| \left(p_1, p_2, ..., p_n\right) \in P, \qquad (4.3)$$

где $p_i \left(i = 1,2,...,n\right)$ может принимать одно из 3-х значений: 1, 0, $\varnothing$;

P– множество всевозможных сочетаний значений p_i.

Оператор $A_i^{P_i}$ задает функцию, которая в зависимости от значения p_i определяется следующим образом:

$$A_i^0 = \overline{A}_i, \; A_i^{\varnothing} = 1, \; A_i^1 = A_i.$$

Выражения $\overline{A}_i$ и A_i называются литералами, соответствующими независимой переменной A_i.

Функция, записанная в форме (4.3), имеет непосредственное отношение к так называемым сжатым таблицам истинности. Для построения последних можно использовать, как уже было сказано ранее, известный метод Квайна – Мак-Класки.

Соотношение между обычной и сжатой таблицами истинности показано в табл. 4.4.

Таблица 4.4

Два вида таблиц истинности

Обычная				Сжатая			
A	*B*	*C*	*F*	*A*	*B*	*C*	*F*
0	0	0	0	I 0	1	$\varnothing$	1
0	0	1	0	I 1	$\varnothing$	0	1
0	1	0	1				
0	1	1	1				
1	0	0	1				
1	0	1	0				
1	1	0	1				
1	1	1	0				

левая часть *правая часть*

Символ ∅ означает, что в данной позиции можно записать произвольную информацию. Можно комбинировать строки, содержащие ∅, а также формировать новые строки с бóльшим количеством символов ∅.

Выполнив в булевом выражении (4.3) преобразования, аналогичные тем, которые необходимы для перехода от обычной таблицы истинности к сжатой, можно поставить в соответствие каждой строке таблицы истинности (p_1, p_2,…, p_n) единственный член логической суммы (4.3).

Очевидно, что при сравнении независимой комбинации переменных со всеми строками таблицы истинности совпадение по всем разрядам достигается лишь для строк, приведенных в правой части табл. 4.4 (значения переменных, стоящих в той же позиции, что и ∅, при этом не учитываются).

Операция сравнения подобного типа отличается от обычных операций маскированного сравнения, используемых в АЗУ, тем, что маски (символы ∅) фактически устанавливаются не на поисковом аргументе, а на строке таблицы.

При реализации сжатой таблицы истинности при помощи ФП в ее ячейки записывается содержимое левой части табл. 4.4. Как уже отмечалось, ячейка ФП может быть в одном из трех состояний (1, 0, ∅), причем те разряды, которые = ∅, в операциях сравнения не участвуют.

В ФП могут храниться одновременно несколько различных булевых функций. В этих случаях память делится на две части, именуемые входной и выходной таблицами.

В качестве примера рассмотрим таблицу истинности сложения по модулю 16 чисел, представленных в коде 8421 с единицей. Булевы функции, описывающие значение битов результата R_i и переноса C для всех сочетаний исходных двоичных цифр S_i, имеют следующий вид:

$$\left.\begin{aligned} C &= S_3 \wedge S_2 \wedge S_1 \wedge S_0, \\ R_3 &= \left(\overline{S}_3 \wedge S_2 \wedge S_1 \wedge S_0\right) \vee \left(S_3 \wedge \overline{S}_2\right) \vee \left(S_3 \wedge \overline{S}_1\right) \vee \left(S_3 \wedge \overline{S}_0\right), \\ R_2 &= \left(\overline{S}_2 \wedge S_1 \wedge S_0\right) \vee \left(S_2 \wedge \overline{S}_1\right) \vee \left(S_2 \wedge \overline{S}_0\right), \\ R_1 &= \left(\overline{S}_1 \wedge S_0\right) \vee \left(S_1 \wedge \overline{S}_0\right), \\ R_0 &= \overline{S}_0. \end{aligned}\right\} \qquad (4.4)$$

Комбинированная таблица истинности – табл. 4.5. Позиции, в которых стоят ∅ во входной и в выходной таблицах, оставлены пустыми.

Таблица 4.5

Комбинированная таблица истинности сложения чисел в коде 8421 с единицей

Строка	Входная таблица					Выходная таблица					
		S_3	S_2	S_1	S_0		C	R_3	R_2	R_1	R_0
1					0						1
2				0	1					1	
3				1	0					1	
4			0	1	1				1		
5			1	0					1		
6			1		0				1		
7		0	1	1	1			1			
8		1	0					1			
9		1		0				1			
10		1			0			1			
11		1	1	1	1		1				

В схемном исполнении входная часть комбинированной таблицы истинности строится из ячеек ФП, показанных на рис. 4.1. Выходом каждой строки (слова памяти) является сигнал совпадения, поступающий по линии М. Логические суммы (по одной для каждой булевой функции) формируются при помощи схем ИЛИ. Входами для этих схем служат сигналы совпадения от тех слов, которым соответствует значение “1” в столбцах выходной таблицы.

Напомним, что основной целью введения ФП являлась реализация программируемых логических операций. В таком применении выходная таблица ФП должна обеспечивать подключение линий, идущих от слов памяти, ко входам различных схем ИЛИ в зависимости от типа логической операции.

Для построения программируемой выходной таблицы каждая ячейка, т.е. пересечение ее столбца с линией слова входной таблицы, должна иметь обычный триггер. Выходы всех триггеров одного столбца объединяются при помощи функции “Встроенное ИЛИ”.

В эти триггеры в соответствии с таблицей истинности реализуемой логической функции заносятся “1” и после этого выходная таблица действует аналогично рассмотренной выше аппаратной таблице.

Введение в ФП специального признака позволяет выполнять различные операции поиска и считывания, а также более сложные задачи и команды.

4.2.4. Другие способы реализации программируемой логики

Среди других способов можно назвать, в частности, программируемый логический массив (PLA), программируемые логические матрицы в виде полупроводниковых БИС, универсальные логические модули, которые реализуют необходимые управляющие функции при помощи внешних управляющих сигналов.

Рассмотрим аргументы (критерии), которые используются при сравнении методов реализации программируемой логики.

1. Одним из наиболее важных аргументов при выборе способа реализации управляющей логики является быстродействие. Программируемая постоянная память и программируемые логические массивы оказываются менее быстродействующими, чем аппаратурное решение схем управления (на порядок и более). Поэтому если временные характеристики – критичны, то выбирают аппаратную реализацию схем управления.

2. Другая проблема связана с емкостью памяти устройств, необходимых для замены отдельных логических цепей. Количество ячеек ФП равно произведению количества независимых переменных и логических функций. В постоянном ЗУ емкость растет по экспоненте с увеличением числа переменных.

3. Количественная оценка общей стоимости различных способов реализации логики ЦП крайне сложна, и, кроме того, она зависит от уровня технологии и стоимости элементной базы. По проведенным оценкам при количестве переменных ≥ 50 обычную память с произвольным доступом уже можно исключать из рассмотрения.

4. Вероятно, наиболее важный аргумент в пользу применения ФП состоит в ее гибкости, которая проявляется в процессе логического проектирования (кроме того, требуется меньше документации даже для больших систем).

4.3. Применение АЗУ для выполнения различных управляющих функций

Кроме названных случаев применения, АЗУ используется также для выполнения управляющих функций, в частности:

– в качестве управляющей микропрограммной памяти;

– при обработке данных;

– в телекоммуникационных системах;

– для синхронизации работы страничной памяти;

– обработки символов;

– трансляции кодов;

– реализации алгоритмов поиска линий связи.

Более подробно применение АЗУ для выполнения различных управляющих функций рассмотрено в цитируемых в [3] источниках.

Контрольные вопросы к разделу 4

1. Приведите основные варианты использования АЗУ в вычислительных системах.
2. Приведите варианты реализации программируемой логики.
3. Что является основной особенностью функциональной памяти?
4. Назовите основные критерии, применяемые при сравнении методов реализации программируемой логики.
5. Для выполнения каких управляющих функций применяется АЗУ?

5. АССОЦИАТИВНЫЕ ПРОЦЕССОРЫ

5.1. Основные тенденции развития ассоциативной памяти

В работе Т. Кохонена [3] прогнозировались основные пути дальнейшей разработки структур аппаратной реализации АЗУ и способы ее применения для выполнения функций не только чистого поиска, но и для других целей. Выделены три основных фундаментальных направления, ориентированных на достижение большего параллелизма и гибкости при выполнении поисковых операций:

1. Расширение количества логических функций, выполняемых отдельными ячейками АЗУ, и введение локальных связей между ячейками, что позволяет распределить алгоритмы обработки данных по всему устройству памяти и выполнять их параллельно.

2. Построение матричных процессоров из большого количества блоков высокого уровня (например микропроцессоров) с использованием АЗУ для определения условий управления отдельными блоками обработки данных и организации их взаимодействия.

3. Увеличение гибкости памяти хранения результатов в обычных АЗУ, особенно в параллельных по словам и последовательных по разрядам.

Т. Кохонен считал наиболее перспективным создание ВС с высоким уровнем параллелизма, построенных по принципу « параллельность по словам и последовательность по разрядам», что позволяет использовать слова увеличенного размера. Большинство численных алгоритмов также выполняет последовательную поразрядную обработку данных, поэтому последовательная выборка информации из памяти является для них приемлемой.

5.2. Общие сведения об ассоциативных процессорах. Классификация ассоциативных процессоров

Ассоциативным процессором (*АП*) называют АЗУ, дополненное логикой и микропрограммным управлением. АП обладает бóльшими возможностями обработки данных, чем простой поиск с использованием АЗУ, хотя иногда АЗУ также называют ассоциативным процессором. Так как параллельная обработка при выполнении одной и той же операции (например операции сравнения) присуща уже самому АЗУ, то АП относят к классу *SIMD-архитектур* (*или ОКМД – одиночный поток команд*. *Множественный поток данных*) [10]. Большие АЗУ из-за высокой стоимости не получили широкого распространения. АЗУ не-

большого объема используются в современных ЭВМ с архитектурой фон Неймана и применяются в тех случаях, когда нужен быстрый просмотр кэш-памяти, поиск страниц в системах с виртуальной памятью и др.

Существует классификация ассоциативных процессоров *по уровню распределенности аппаратной поддержки* (*логики*). Логика может быть распределена до уровня бита, байта, слова или более крупного блока памяти. Чем ниже этот уровень, тем выше уровень параллелизма (и соответственно – стоимость).

5.3. Ассоциативные процессоры с высоким уровнем параллелизма

К АП с высоким уровнем параллелизма относятся следующие типы систем [6]:

1) *АЗУ, организованные по словам, с параллелизмом на уровне бита* (*разряда*);

2) *программируемые АЗУ*;

3) *системы памяти с распределенной логикой*.

К системам *первого типа* относятся АЗУ, параллельные по словам и битам. В этих системах время поиска по всей памяти близко по времени сравнения одного бита. Поскольку реализация такой разработки обходится дорого, такие системы рассматривались только в некоторых ранних работах.

Примером *программируемого АЗУ* может служить *ассоциативная память с расширенными возможностями – ACAM* (*Augmented Content Addressed Memory*). Она представляет собой двумерный массив программируемых ячеек, способных, в частности, выполнять арифметические операции. Каждая ячейка содержит приблизительно 40 эквивалентных вентилей *ИЛИ-НЕ*.

Активация ячеек производится комбинациями сигналов, подаваемых по строкам и столбцам. Структура может настраиваться по строкам на сдвиги, ввод-вывод, поиск равенства, а по столбцам – выполнять ввод-вывод и другие операции.

Система *ACAM* предполагает сложное управление, что потребовало бы в другом исполнении большого количества оборудования.

Память с распределенной логикой DLM (*Distributed Logic Memory*) – это еще один пример ассоциативной системы с высоким уровнем параллелизма. В этой системе каждый элемент представляет собой небольшой процессор, называемый *ячейкой*, который может хранить и обрабатывать один символ.

Данные представлены в виде пар: “*Имя атрибута* (*метка*) – *значение*”.

Система *DLM* особенно удобна для обработки *данных переменной длины*, так.как в ней не ограничивается количество используемых ячеек. Поэтому такая архитектура предназначалась для поиска информации.

Структура системы *DLM* приведена на рис. 5.1.

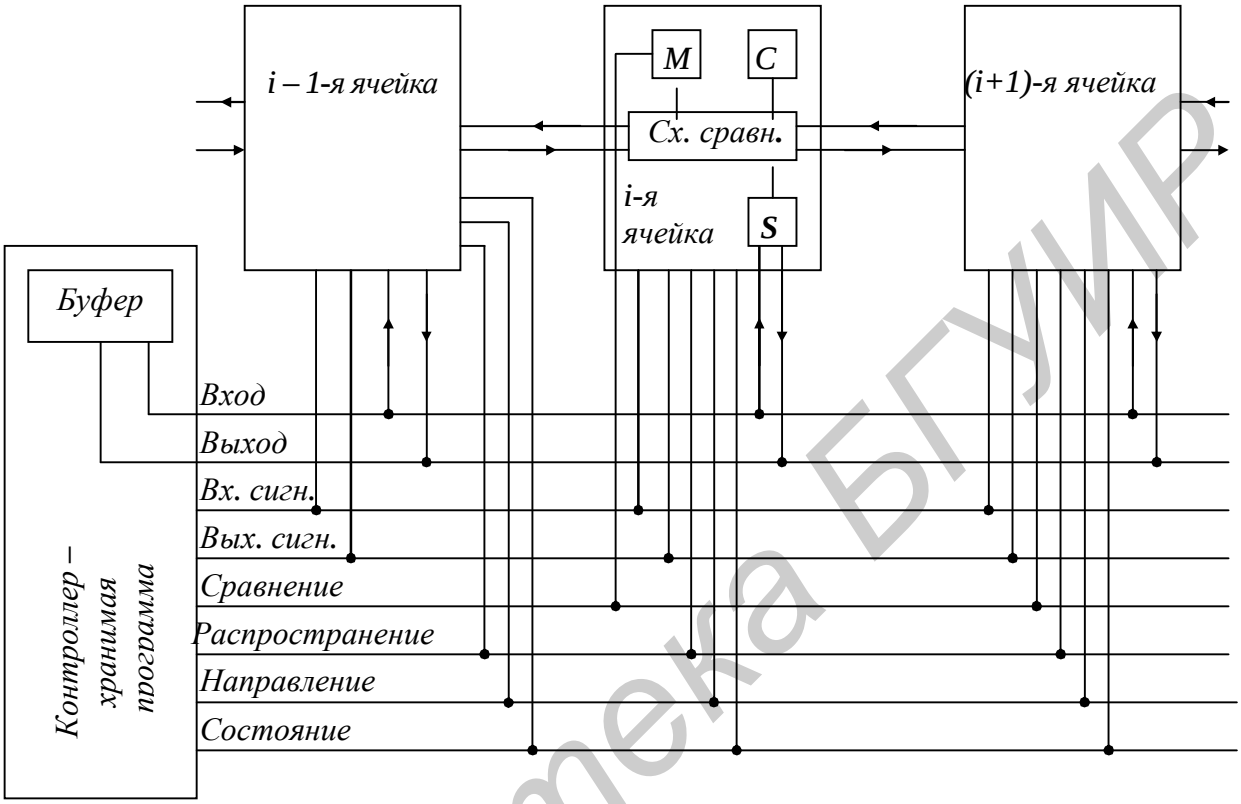


Рис. 5.1. Память с распределенной логикой *DLM*

Ассоциативность достигается при помощи цепочки ячеек, работающих параллельно под управлением общих сигналов. *Цепочки текста,* которые хранятся в *DLM,* могут иметь произвольную длину и размещение. Данные можно загрузить в ячейки *DLM*. Нужные ячейки могут выдавать содержимое своих регистров *S,* выполнять сравнение данных, изменять свои состояния и посылать сигналы соседним ячейкам.

Ячейка содержит *регистр данных S, схему сравнения* и *регистры M* и *C*.

В регистре *M* (*Match*) может быть установлена *метка совпадения*, а в регистре *C* (*Connection*) – *метка условия*.

Когда поступает сигнал сравнения, каждая ячейка сравнивает свое содержимое с данными, поступающими на вход. В случае совпадения ячейка посылает сигнал в соседнюю ячейку и активирует ее. Если активированная ячейка

получит *сигнал распространения* – она включает соседнюю ячейку, а сама выключается.

Сигнал *направление* указывает направление распространения (вправо или влево).

При выполнении поиска сначала отыскиваются метки, затем производится последовательное сравнение значений вдоль цепочки ячеек, по одному символу за один раз, причем каждая ячейка, где обнаруживается совпадение, активирует соседнюю.

При поиске можно задавать метки и выбирать соответствующие им значения (*параметры*) или, наоборот, необходимые операции программировать при помощи команд *DLM* и хранить их в контроллере.

Команда *DLM* имеет следующий формат:

<*код операции*> [<*предикат*>]

В поле *предиката* (который может и отсутствовать) указываются состояния регистров *C* и *M*:

MATCH A (*C = 1*) – эта команда устанавливает регистр *M* в состояние “1” только в тех ячейках, где *C* = 1.

Команда *MATCH A* действует безусловно на все ячейки.

Шаг распространения выполняется следующим образом:

$$\begin{cases} C \neg 0 & \text{(очистить все } C\text{)} \\ RIGHTC \leftarrow 1\ (M = 1) & \text{(установить метку } C = 1 \text{ во всех ячейках, расположенных непосредственно } справа \text{ от тех ячеек, где в результате сравнения } M = 1\text{)} \\ M \neg 0 & \text{(очистить все } M\text{)} \end{cases}$$

Рассматривая эти три команды как макрокоманду, которую назовем MARKRIGHTC, рассмотрим выполнение поиска на следующем примере [6].

В соответствии с принятой в *DLM* структурой данных в ячейках хранятся последовательные пары *МЕТКА – ЗНАЧЕНИЕ*, например:

 # HE $ 300 *# YOU $* 050 *# HIS $* 120,

где # и *$* – ограничители.

Эта цепочка занимает в памяти 24 ячейки.

Пусть требуется найти ключ (метку) *HIS* и выбрать соответствующие данные. Ниже приведена программа, составленная для данного примера (цепочки ячеек и состояния регистров *M* и *C*, соответствующие этой программе, показаны на рис. 5.2).

MARKRIGHTC /* макрокоманда: очистить все *C*, затем установить *C* = 1 в ячейках непосредственно справа от ячеек с *M* = 1, затем очистить все *M*. Теперь первые символы всех меток маркированы */

MATCH H (*C* = 1) /* искать *H* в активных ячейках */

MARKRIGHTC /* активировать соседние справа ячейки */

MARKRIGHTC /* макрокоманда: очистить все *C*, затем установить *C* = 1 в ячейках непосредственно справа от ячеек с *M = 1*, затем очистить все *M*. Теперь первые символы всех меток маркированы */

MATCH H (*C* = 1) /* искать *H* в активных ячейках */

MARKRIGHTC /* активировать соседние справа ячейки */

MATCH I (*C* = 1)

MARKRIGHTC

MATCH S (*C* = 1)

MARKRIGHTC /* активировать ячейки, в которых начинаются данные*/

RETRIVVE: READ (*c* = 1) /* читать один символ */

if # then go to END

M ¬ *1* (*C* = 1) /* установить *M = 1* для использования макрокоманды*/

MARKRIGHTC /* активировать следующий символ */

go to RETRIVVE

END: Stop

В системе *DLM* в дополнение к *READ* имеется команда *WRITE*, а в дополнение к *RIGHTC* и *RIGHTM*-команды *LEFTC* и *LEFTM* соответственно.

Из-за ряда недостатков (большие размеры, невысокая скорость поиска) этот проект не был практически применен, но послужил основой для нескольких практически реализованных систем (например *PEPE*, а также линейный АП – *ALAP* фирмы *Hughes Aircraft*).

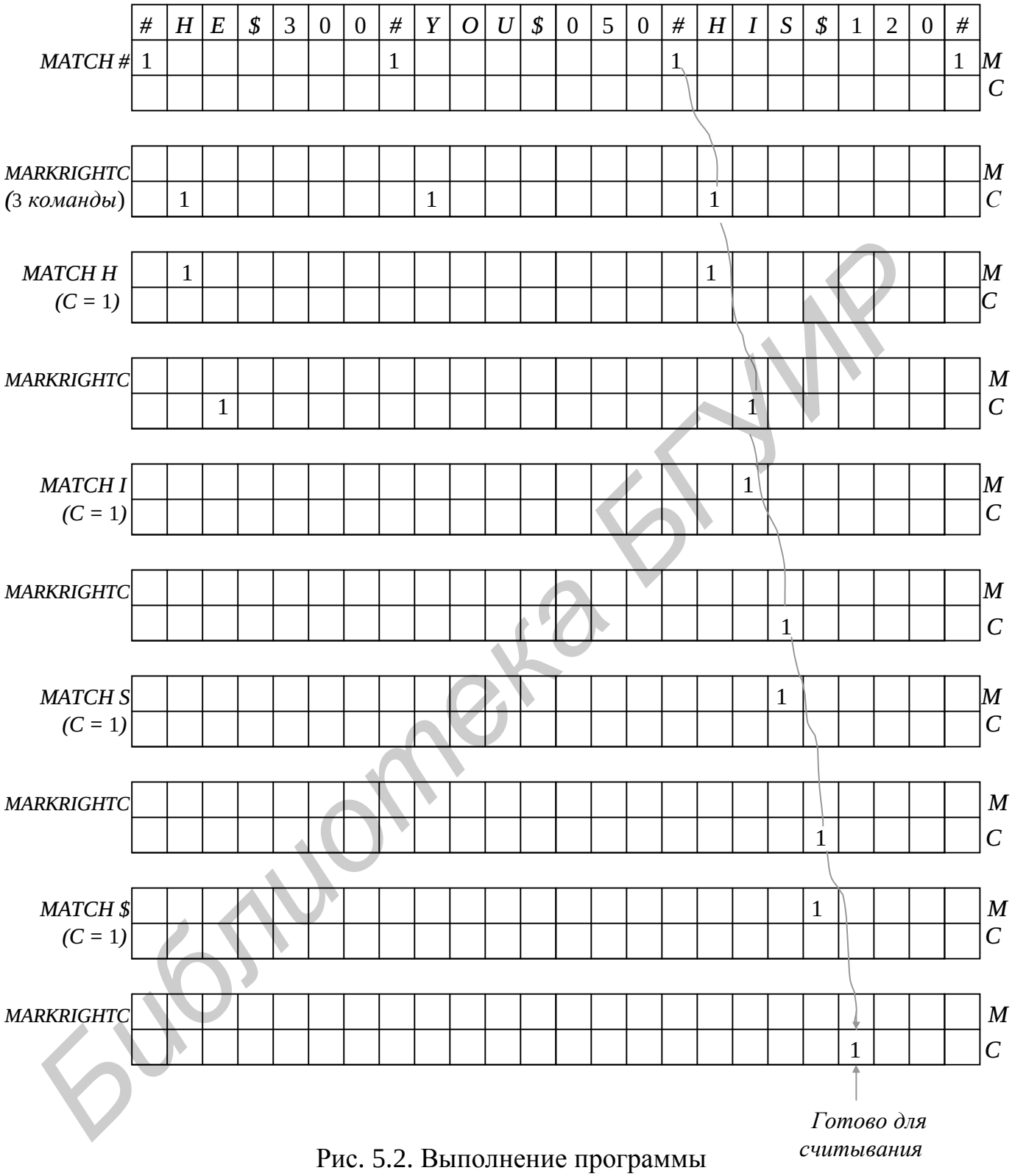


Рис. 5.2. Выполнение программы примера в системе *DLM*

5.4. Ассоциативные процессоры с операционными устройствами высокого уровня

Параллелизм в рассмотренной выше *памяти с распределенной логикой* (*ПРЛ, или DLM*) реализован на уровне выполнения поразрядных операций (если говорить об обработке числовой информации), для которых последовательности микроопераций задаются при помощи внешних устройств глобального управления.

Другой подход характеризуется практически независимым выполнением операций в отдельных ячейках. Параллельный процессор формируется из массива ячеек, хранящих каждая в своей локальной памяти (ЛП) небольшое количество (до нескольких сотен) операндов. Процессор снабжен гибким АЛУ.

Если в ПРЛ логика управления концентрируется в каждой ячейке памяти, в результате чего ячейка может оперировать только с разрядной информацией, то в рассматриваемых далее примерах другого подхода каждый модуль, используя локальную систему управляющих сигналов, может схемно (аппартно) выполнять арифметические и логические операции. Кроме того, связь между ячейками рассматриваемого типа осуществляется при помощи машинных команд и числовых переменных, что важно с точки зрения программирования при решении сложных задач.

Представителями такого типа ассоциативных процессоров являются матричные процессоры.

5.4.1. Базовая структура матричного процессора

Структура простейшего матричного процессора приведена на рис. 5.3.

В зависимости от конфигурации сети линий связи матричный процессор (МП) может быть *бесструктурным*, *линейным*, *прямоугольным* и т.д. Если связь между ячейками (элементарными процессорами – ЭП) осуществляется только при помощи *управляющей шины* или *шины ввода-вывода*, систему называют *групповым процессором*. В каждый ЭП входит *операционный блок* (*ОБ*) и *память операционного блока* (*ПОБ*) для хранения данных в процессе вычислений. Работа всех ЭП осуществляется под общим программным управлением, содержащимся в памяти, управление ОБ – при помощи содержащихся в нем фиксированных микропрограмм.

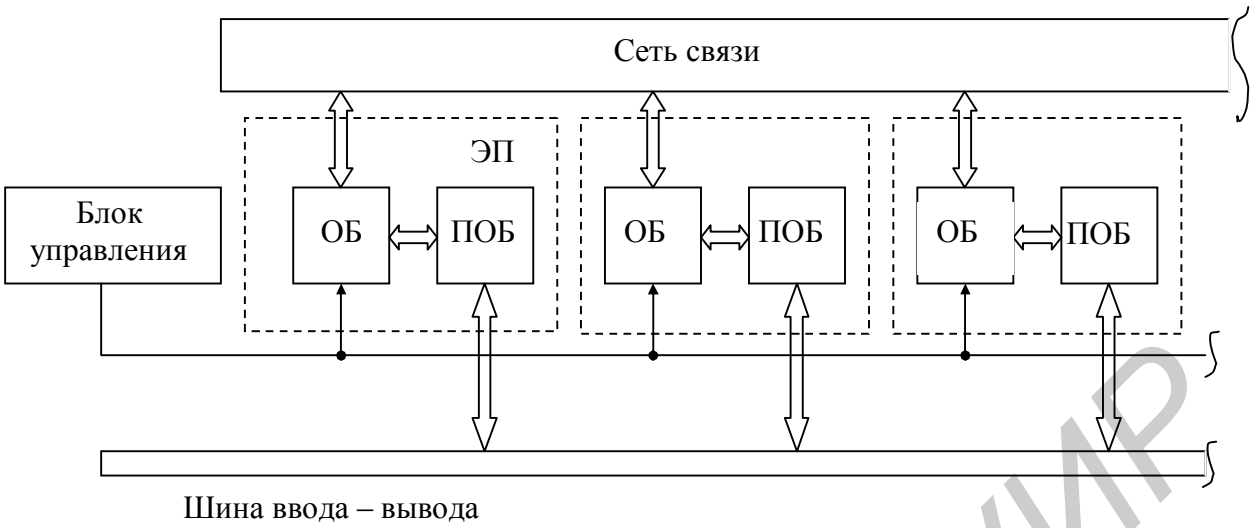


Рис. 5.3 Упрощенная структура матричного процессора

Для управления МП можно также использовать универсальную ведущую ЭВМ, для которой матричный процессор является периферийным устройством (ПФУ). В приведенной на рис. 5.3 структуре МП можно применить АЗУ небольшого объема в каждом устройстве обработки для получения ответа на символически заданные команды или данные.

Рассмотрим связи между ячейками МП.

При независимом друг от друга функционировании ЭП взаимодействуют лишь с ведущей ЭВМ по каналам ввода-вывода.

При решении пространственных задач, а также задач, требующих вычисления многократных интегралов и задач решения дифференциальных уравнений, для которых переменные задаются в узлах *решетчатой функции*, вычислительные алгоритмы ориентируются на описание взаимосвязи между соседними узлами.

Для решения **специальных** задач можно задавать фиксированную топологическую структуру массива, для решения **широкого круга задач** связь между ячейками может задаваться программно, как, например, в процессоре *RADCAP*, который будет рассмотрен ниже.

Ячейки могут связываться непосредственно или через линии связи, образованные при помощи мультиплексоров.

Отметим, что структура микропроцессоров часто соответствует топологии решаемых вычислительных задач.

5.4.2. Трехканальный процессор

В процессорах, представляющих собой очень большие массивы ячеек, задержка при передаче информации между отдаленными ячейками с использованием *пошаговых операций* (как например в ПРЛ) может оказаться ощутимой. Один из вариантов решения, позволяющего значительно сократить задержки, приведен на рис. 5.4.

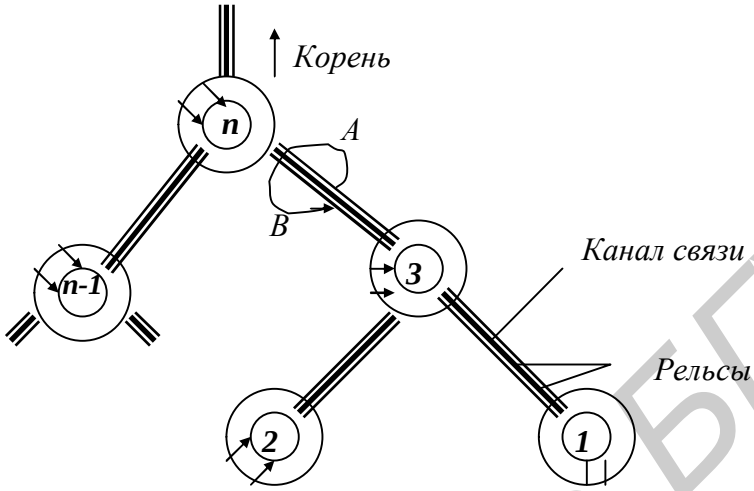


Рис. 5.4. Структура связей в трехканальном процессоре – древовидная

Для ассоциативной выборки каждая ячейка этого процессора, названного трехканальным, должна содержать *регистр сравнения, цепи сравнения и триггер совпадения*. Для определения режима работы ячейки (передачи, приема, обработки информации или свободного состояния) необходимы другие *триггеры состояния.*

Для глобальных операций передачи информации, например, при одновременной передаче аргумента поиска во все ячейки, используется *канал связи.* Для передачи промежуточных результатов и локального взаимодействия между ячейками предназначены две вспомогательные линии связи, называемые *рельсами*. Рельсы всегда используются для двухсторонней связи ячеек, имеющих последовательные индексы.

Чтобы сократить время передачи сигналов, особенно в случае, когда массив разбит на небольшие поддеревья, можно ввести *сокращающие линии связи* для сигналов, например, линию, соединяющую точки *A* и *B* на рис. 5.4.

5.4.3. Ассоциативный управляющий переключатель

В мультипроцессорных системах «перекрестные переключения» процессоров и памятей являются обычными явлениями. Для повышения эффективности матричного процессора также вводится переключение операционных блоков (ОП) с помощью *ассоциативного управляющего переключателя* (*АУП*), структура которого показана на рис. 5.5 .

В структуре МП, предназначенного для фильтрации радиолокационной информации, каждый ОБ можно подключить к одному из независимых программируемых блоков управления. Такая система является совокупностью независимых МП.

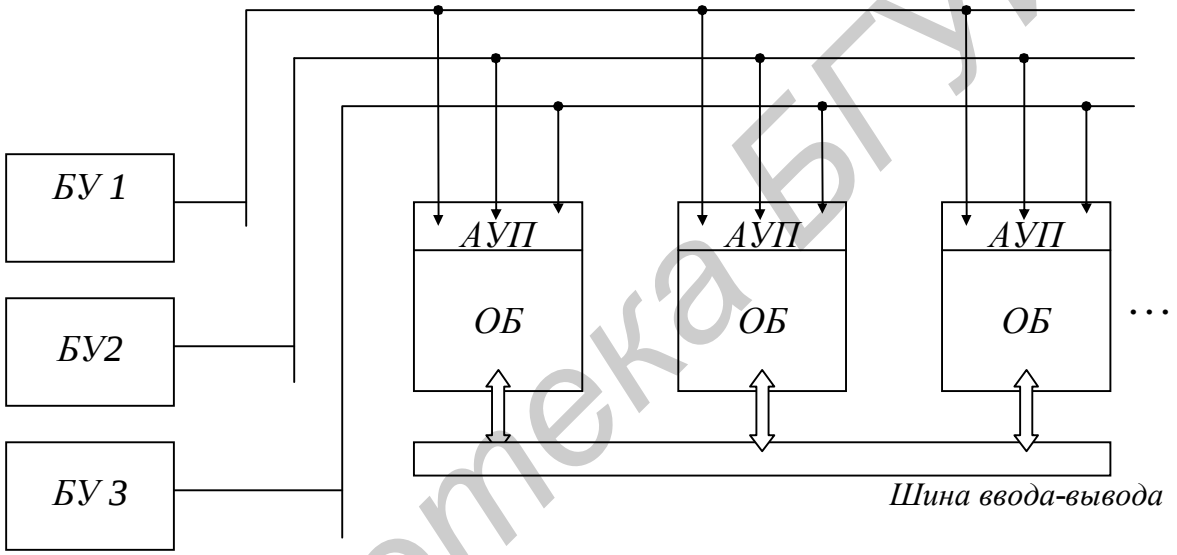


Рис. 5.5. Ассоциативный управляющий переключатель (АУП)

Управляющий переключатель назван *ассоциативным*, так как соединения устройств выполняются на основании *ассоциативной адресации*. Один из регистров каждого *ОБ* играет роль *регистра сравнения*, в котором содержатся промежуточные результаты. Его содержимое автоматически сравнивается с аргументами поиска, которые формируются в блоке управления. От результата сравнения зависит подключение *ОБ* к одному или другому блоку управления.

5.4.4. Ассоциативный матричный процессор RADCAP

АП *RADCAP* (позже переименованный в *SIMDA*) с 1024 ОБ высокого уровня был разработан фирмой Texas Instruments Corp.для фирмы Romc Air De-

velopment Centre (RADC), для проводимых ею параллельных вычислительных экспериментов [7].

Отметим его основные особенности (рис. 5.6).

В базовую ячейку ***RADCAP*** входит локальная память операционного блока с произвольным доступом (ПОБ) емкостью 256 х 5 слов, ОБ с 4-разрядным АЛУ, два 4-разрядных арифметических регистра (*A* и *D*), два регистра для управления микропрограммами (B и C) и два регистра состояния. Связь между ячейками и шинами ввода-вывода осуществляется поразрядно. Один разряд каждого слова памяти зарезервирован для контроля по четности.

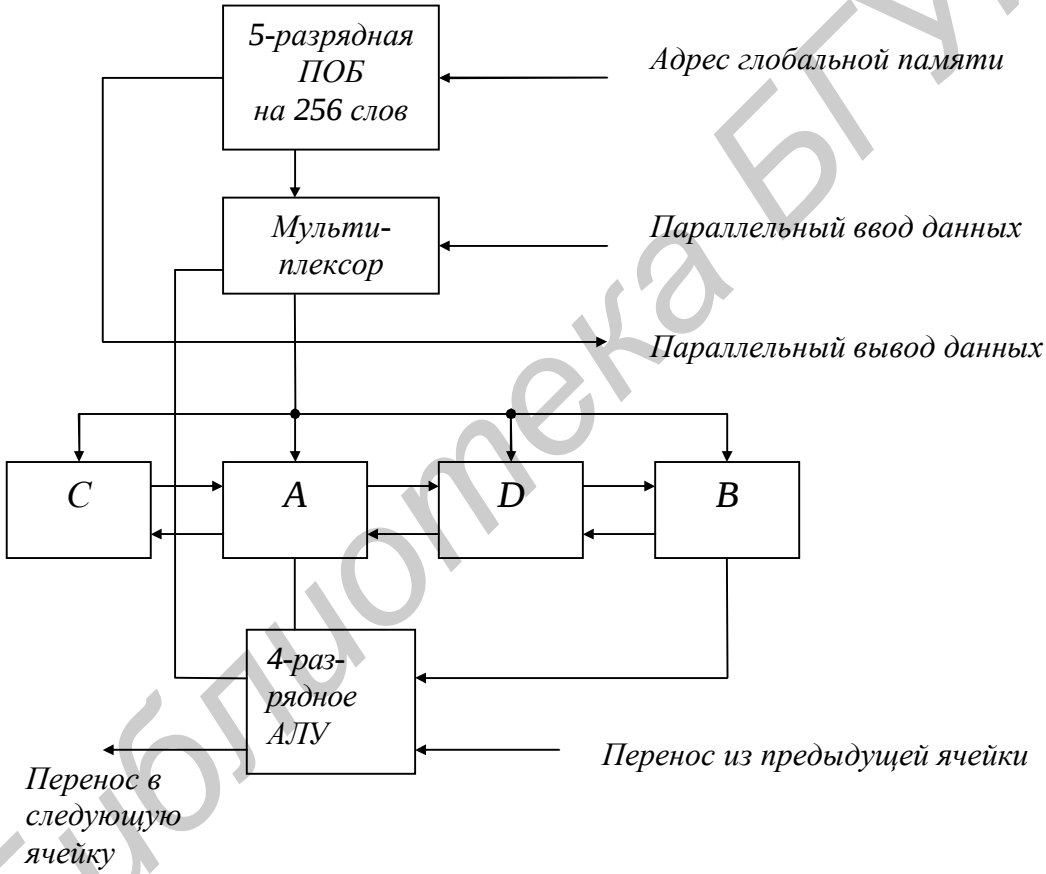


Рис. 5.6. Схема базовой ячейки системы ***RADCAP*** (***SIMDA***)

Каждый ОБ выполняет 48 арифметических и логических команд, а также передачи между регистрами. Арифметические команды выполняются с 16-разрядными словами, а это значит, что все операнды необходимо разделить на 4-разрядные группы. Для выборки из памяти можно использовать как *обычную*, так и *ассоциативную адресацию*. Индексирование слов в ПОБ выполняется при помощи *глобальной адресации.*

Регистры состояния (на рис. 5.6 они не показаны) используются для указания активности ОБ, фиксации результатов сравнения (>, <, =), переполнения, отказа и состояния микропрограмм.

Максимальное (минимальное) значение, содержащееся в определенной адресной позиции всех активных ячеек, можно определить при помощи встроенной логики, выполнение многих других операций (например, установление приоритетов и подсчет количества активных элементов) – при помощи дополнительных логических цепей.

Матричный процессор *RADCAP* разделен на 32 группы по 32 ОБ. В каждом блоке имеется *переключатель* – мультиплексор на 2 направления для соединения ячеек блока с *блоком управления* или *каналом ввода-вывода*. К блоку управления можно подключить любое количество блоков, а остальные в это время могут одновременно выполнять операции ввода-вывода.

Для взаимодействия между ОБ в *RADCAP* к каждой i-й ячейке подключено (на рис. 5.6 они не показаны) 3 мультиплексора X_i, Y_i, Z_i: Z_i – для непосредственного соединения i-й ячейки с ячейками под номерами $(i-8)-(i+7)$, Y_i – для соединения ячейки i с мультиплексорами Z_i, Z_{i+16}, Z_{i+32},…, а мультиплексор X_i – для соединения i-й ячейки с мультиплексорами Y_i, Y_{i+128},… . Общее число связей равно 1024.

5.4.5. Ассоциативный групповой процессор PEPE

Ассоциативный параллельный процессор *PEPE* (*Parallel Element Processing Ensemble*) был создан для обработки данных при радиолокационном сопровождении баллистических ракет (рис. 5.7).

PEPE содержал группу из 288 *операционных устройств* (*ОУ*), каждое из которых предназначалось для хранения и обработки данных для одной опорной траектории.

В каждое операционное устройство входит:

– *блок корреляции, или корреляционное устройство* (*КУ*)– для первичной обработки и сравнения радиолокационных сигналов с заданной траекторией, находящейся в памяти;

– *арифметическое устройство* (*АУ*);

– *запоминающее устройство* (*ЗУ*);

– *«ассоциативный» выходной блок* (*АВУ*) – для параллельной выдачи команд управления на радиолокационную станцию.

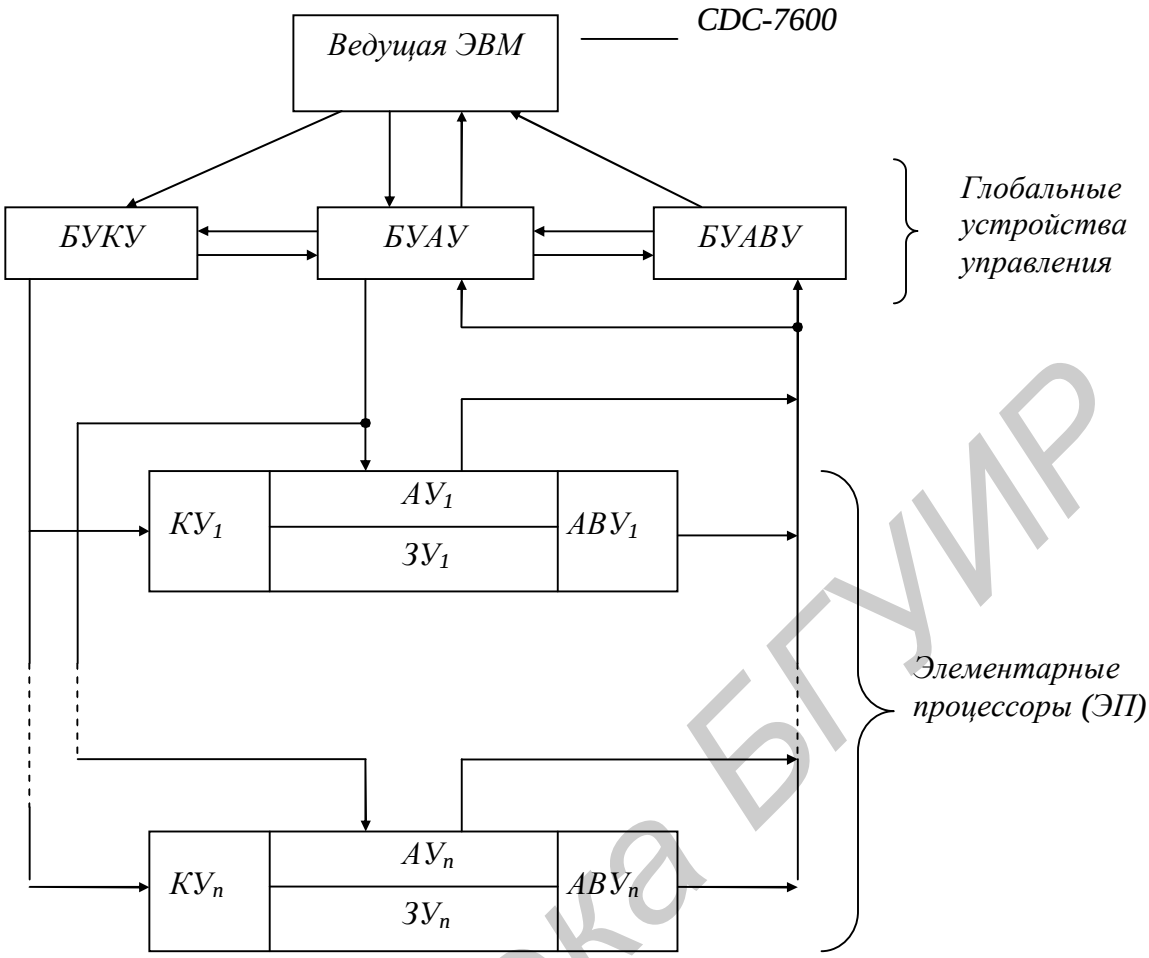


Рис. 5.7. Ассоциативный параллельный процессор *PEPE*

Для общего управления в процессор также входят:

CDC – ведущая ЭВМ,

БУАУ – блок управления *АУ*,

БУКУ – блок управления *КУ*,

БУАВУ – блок управления *АВУ*.

Арифметические операции выполняются с 32-разрядными словами с плавающей точкой, с обычной и двойной точностью.

В процессоре между ОУ не существует непосредственной связи, поэтому он считается *групповым процессором*.

PEPE считают ассоциативным процессором по двум причинам:

1) управление операциями в блоке корреляции основано на принципе групповой ПРЛ, который позволяет выполнять как ассоциативный выбор ОУ, так и параллельную обработку их содержимого;

2) выходной блок процессора – ассоциативный.

5.5. Ассоциативные процессоры с последовательной обработкой разрядов

Считается, что рассмотренные ранее матричные и групповые процессоры оказываются весьма эффективны при выполнении групповых вычислений для специальных задач, таких, как *фильтрация* радиолокационных сигналов. Вычислительные системы, построенные на основе параллельных АЗУ с последовательной выборкой разрядов, наиболее эффективны по стоимости и гибкости при решении разносторонних задач, связанных с параллельными вычислениями и поиском информации. Такие системы называют *ассоциативными процессорами с последовательной обработкой разрядов*. Структура и перечень операций их основных элементов (АЗУ и памяти результатов) были рассмотрены ранее.

На рис. 5.8 для сравнения показаны структура организации «полностью параллельного процессора», например ПРЛ, ориентированного на групповое выполнение операций, и система на основе параллельного АЗУ с последовательной выборкой [3].

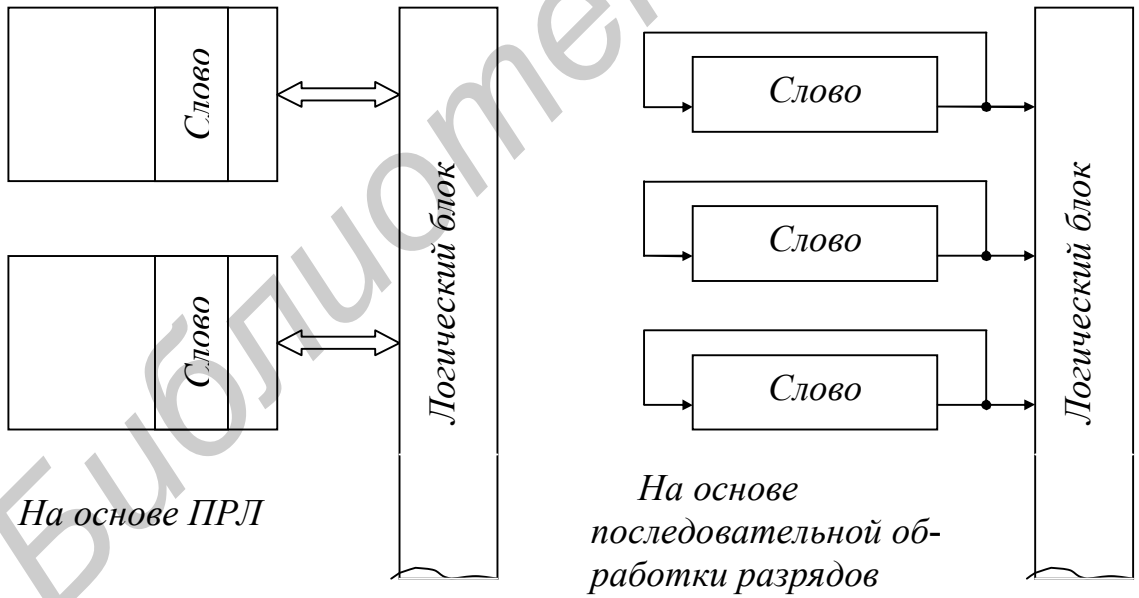


Рис. 5.8. Структуры на основе ПРЛ и на основе АЗУ с последовательной обработкой разрядов

Если используется память для операндов, изображенная слева на рис. 5.8, то для выполнения арифметических операций необходимо обеспечить

взаимодействие между *X*-ячейками. Если же используется АЗУ, показанное справа на рис. 5.8, и выполняется сдвиг операндов в памяти результатов, логику итеративного взаимодействия ячеек можно заменить простыми последовательными логическими цепями для каждого слова.

Таким образом, если сравниваемые системы предназначены только для выполнения арифметических операций, то отличие их в быстродействии будет несущественным. Но с другой стороны, очевидно, что аппаратная реализация ячеек второй системы значительно проще, особенно для операндов с большим количеством разрядов. При этом в памяти выполняются только сдвиги.

Можно расширить возможности параллельной обработки данных по сравнению с ПРЛ, если в памяти результатов АЗУ обеспечено взаимодействие между словами. Обычно такое взаимодействие осуществляется *программно*.

Машинные команды, определяющие передачу информации, могут использовать усложненные принципы адресации слов и разрядных столбцов, как, например, это сделано в ассоциативной системе *STARAN*.

5.5.1. Вычислительная система STARAN

Наиболее характерным представителем группы ассоциативных вычислительных систем является система *STARAN*, разработанная в США фирмой *Good Year Aerospace Corp.* [3, 7, 8, 19, 20].

От матричных систем, описанных выше, она отличается не только наличием ассоциативной памяти, но и другими особенностями:

– ассоциативная память является памятью с многомерным доступом, т.е. в нее можно обратиться как поразрядно, так и пословно;

– операционные процессорные элементы предусмотрены для каждого слова памяти;

– имеется уникальная схема перестановок для перегруппировки данных в памяти.

Основным элементом системы *STARAN* является многомерная ассоциативная матрица – ассоциативный модуль (АМ), который представляет собой квадрат из 256 слов на 256 разрядов, т. е. содержит в общей сложности 65536 битов данных. Для обработки информации имеется 256 процессорных элементов, которые последовательно, разряд за разрядом, обрабатывают слова (рис. 5.9). Все ПЭ работают одновременно, по одной команде, выдаваемой уст-

ройством управления. Таким образом, сразу по одной команде обрабатываются все слова, выбранные по определенным признакам из памяти.

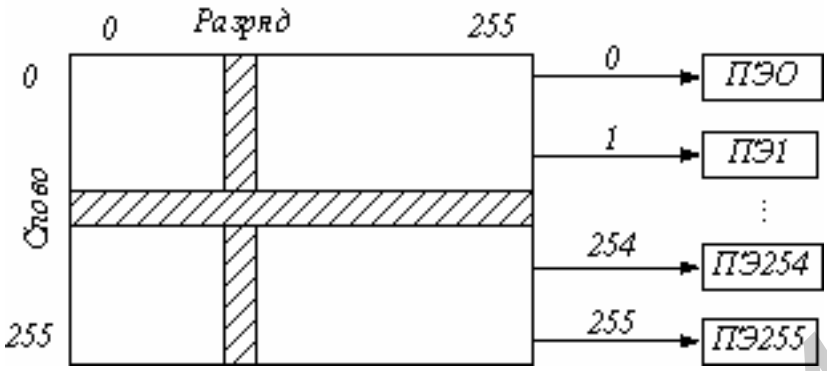


Рис. 5.9. Процессорная обработка в системе STARAN

Схема перестановок позволяет сдвигать и перегруппировывать данные так, чтобы над словами, хранящимися в памяти, можно было выполнять параллельно арифметические и логические операции. Большая часть операций выполняется над каждым из 256-разрядных слов. Операции, в которых участвуют несколько слов, используются достаточно редко. Обычно 256-разрядное слово ассоциативной матрицы разбивается программистом на поля переменной длины, и в процессе обработки именно над этими полями производятся и арифметические, и логические действия.

Базовая конфигурация системы STARAN содержит один АМ. Однако число таких модулей может варьироваться в системе от 1 до 32. Таким образом, при максимальной комплектации в системе может подвергаться ассоциативной обработке 256 кбайтов информации. Скорость поиска и обработки информации 256 процессорными элементами высока, и остальные элементы системы спроектированы так, чтобы поддерживать эту скорость.

Устройство управления ассоциативными модулями организует выполнение операций над данными по командам, хранящимся в управляющей памяти. Оно может выбирать несколько рабочих подмножеств из общего множества данных, хранимых в АМ, и выполнять над этими подсистемами операции, не затрагивая остальную информацию.

Управляющая память разделена на шесть секций:

– первая (емкостью 612 слов) – память библиотеки подпрограмм;

– вторая и третья (512 слов) – память команд;

– четвертая (512 слов) – быстродействующий буфер данных;

– пятая (16384 слов) – основная память;

– шестая (10720 слов) – область памяти для прямого доступа. Длина одного слова – 32 разряда.

Первые четыре секции выполнены на интегральных схемах и имеют высокое быстродействие с длительностью цикла памяти около 200 нс. Вторая и третья секции (память команд) работают попеременно: одна выдает команды в устройство управления, а другая в это время загружается от страничного устройства и наоборот. Пятая и шестая секции выполнены на ферритовых сердечниках, длительность цикла примерно 1 мкс. При необходимости емкость пятой секции может быть удвоена. Страничное устройство загружает первые три секции памяти информацией из быстродействующего буфера, основной памяти или памяти прямого доступа.

Последовательный контроллер ассоциативной системы является обычной однопроцессорной ЭВМ типа РДР-11 и обеспечивает работу в режиме трансляции и отладки программ, первоначальную загрузку управляющей памяти, связь между оператором и системой, управление программами обработки прерываний по ошибкам, а также программами технической диагностики обслуживания. Последовательный контроллер снабжен памятью (емкость 8К слов), печатающим устройством, перфоленточным вводом – выводом и имеет интерфейс, обеспечивающий связь с другими элементами системы.

Подсистема ввода – вывода обеспечивает возможность подключения к системе STARAN других вычислительных устройств и разнообразного периферийного оборудования. Имеются четыре вида интерфейсов: прямой доступ к памяти, буферизованный ввод – вывод, параллельный ввод – вывод, логическое устройство внешних функций. Прямой доступ к памяти позволяет использовать память внешней (несистемной) ЭВМ как часть управляющей памяти системы. Эта память становится таким образом доступной как для внешней ЭВМ, так и для системы STARAN. При этом нет необходимости в буферизации передаваемой между ними информации.

Интерфейс прямого доступа может использоваться и для подключения внешней памяти. Буферизованный ввод – вывод используется для связи системы со стандартными периферийными устройствами. Обмен производится блоками данных или программ. Этот интерфейс может использоваться и для связи с несистемной ЭВМ, однако прямой доступ там все-таки предпочтителен, так как обмен производится быстрее и нет необходимости формирования информации в блоки перед передачей. Параллельный ввод – вывод, который включа-

ет в себя по 256 входов и 256 выходов для каждой матрицы, является важной составной частью подсистемы ввода – вывода. Он позволяет увеличить скорость передачи данных между матрицами, обеспечить связь системы с высокоскоростными средствами ввода – вывода и непосредственную связь любого устройства с ассоциативными модулями. С помощью параллельного ввода–вывода можно, в частности, подключать к ассоциативным матрицам накопители на магнитных дисках, что позволяет быстро вводить и выводить большие объемы информации.

Совокупность всех перечисленных средств, входящих в систему STARAN, позволяет выполнять одновременно сотни и тысячи одинаковых операций.

Рассмотрим подробнее особенности системы *STARAN*, связанные с организацией программной параллельной работы памяти (рис. 5.10).

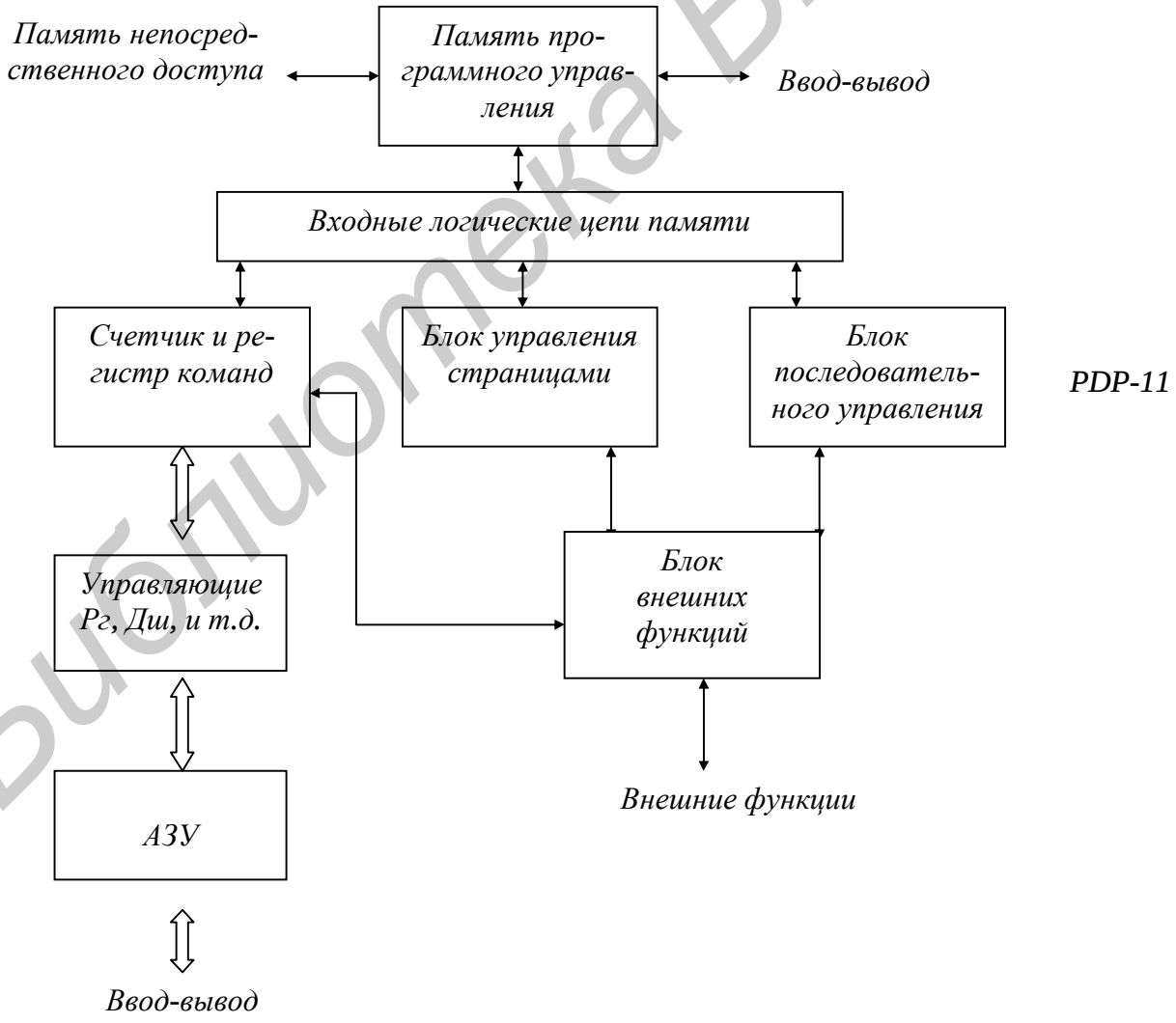


Рис. 5.10. Структурная схема системы *STARAN*

Аппаратно реализованная ассоциативная память в системе построена на принципе *диагональной адресации* (см. п. 3.4.2) , адресное чтение и запись слов выполняется параллельно. Используемые для диагональной адресации цепи *«Исключающее ИЛИ»*, построенные на базе серийно выпускаемых модулей, названы в этой системе *перебрасывающими цепями*.

Для программиста диагональная адресация остается просто адресацией к словам или разрядным столбцам. Схемы *«Исключающее ИЛИ»* можно также запрограммировать для адресации к другим типам разрядных столбцов массива АЗУ, для чего можно использовать специальный аргумент в адресном регистре (см. рис. 3.15).

Так как для диагональной адресации длина слов и разрядных столбцов должна быть одинаковой, в *STARAN* используются **квадратные** модули памяти – блоки емкостью 256 х 256. Массив памяти может содержать 1 – 32 блока, причем максимальное количество блоков ограничено индексной емкостью машинных команд, которые управляют работой всей системы.

Считывание информации производится параллельно из 256-разрядного регистра по каналам шириной 256 битов.

Работа памяти результатов и логика чтения / записи в памяти такого типа уже рассматривалась ранее (см. п. 3.4.1).

Реализация памяти результатов

Память хранения результатов *STARAN* (рис. 5.11) содержит в каждом блоке три 256-разрядных регистра, составляющих *модуль обрабатывающих элементов* (*ОЭ*):

X – обычно используется для хранения текущих результатов при выполнении рекуррентных операций;

Y – используется как регистр результатов поисковых операций, а также арифметических и логических операций;

M – используется как маскирующий регистр для слов и как регистр выборки слов при записи разрядных столбцов в АЗУ.

Для хранения аргумента поиска или в качестве буферного регистра при адресном чтении и записи данных в виде слов или разрядных столбцов в управляющих цепях АЗУ применен 256-разрядный регистр, называемый *общим регистром F*.

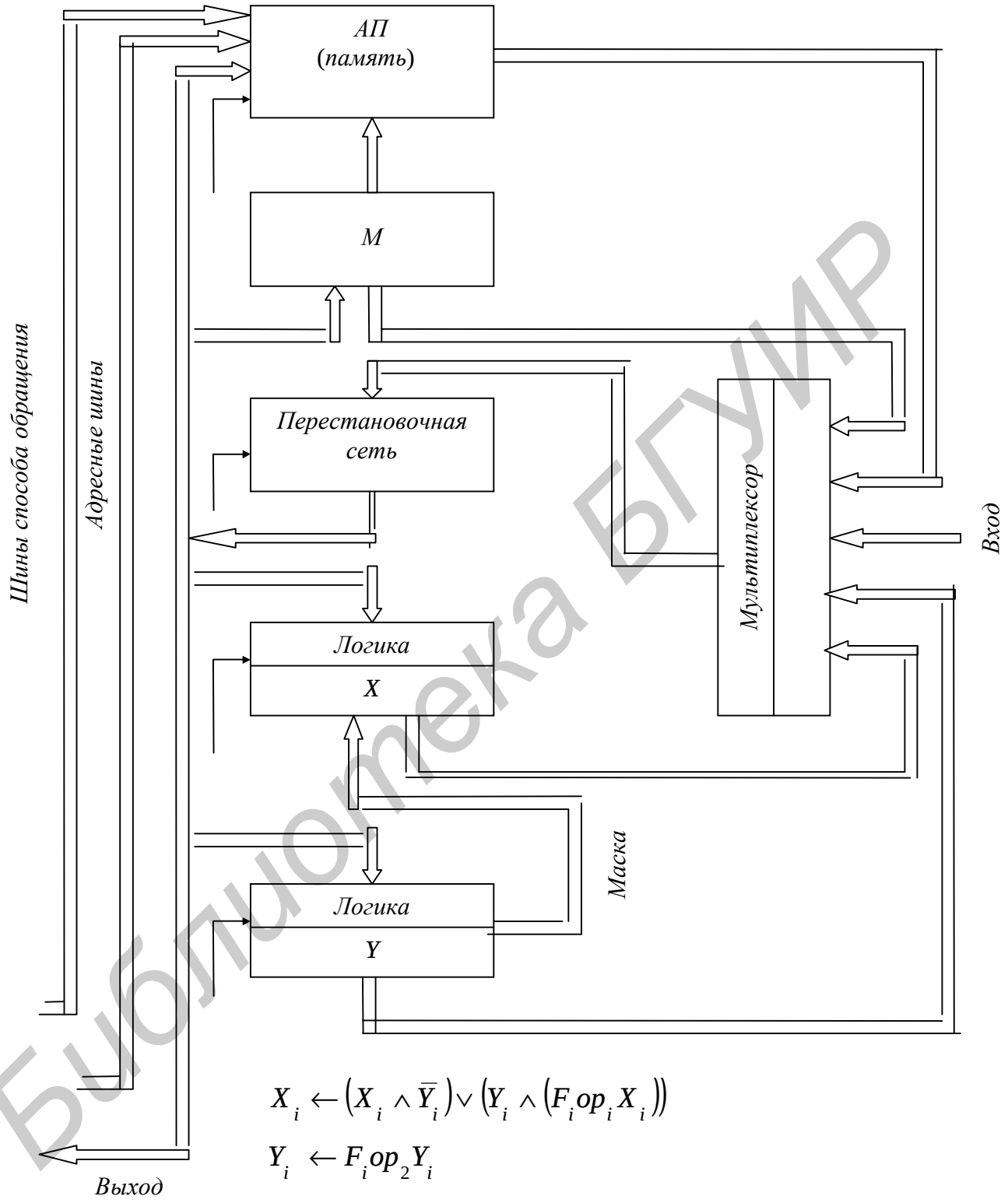


Рис. 5.11. Память хранения результатов системы *STARAN*

Логические цепи в каждой ячейке *X*-, *Y*-регистров позволяют выполнять различные операции над содержимым ячейки (в том числе сложение и вычитание).

Для формирования 8-разрядного адреса первого совпавшего слова в каждом блоке имеется *анализатор многократных совпадений*, который, кроме того, указывает на наличие в блоке хотя бы одного совпадения.

В состав каждого ЭП входит по одному разряду из названных регистров.

Регистр маски *M* позволяет маскировать 256 элементов (разрядов).

Регистры *X, Y* и связанная с ними логика предназначены для выполнения логических операций над двумя переменными.

Мультиплексор обеспечивает поочередное использование перестановочной сети различными устройствами модуля: памятью, регистрами, магистралями (шинами) ввода – вывода. Перестановочная сеть (*Flip*) играет важную роль при реализации операций перестроения данных в АЗУ. Она осуществляет взаимное соединение ОЭ для передачи информации из одного элемента в другой. Это достигается управляемой перестановкой разрядов (одиночных или группами) при передаче слов через сеть. Многократной перестановкой (передачей) через *Flip*-сеть можно добиться сложных перестроений как в вертикальном, так и в горизонтальном направлении.

Описание работы ОЭ ассоциативного матричного модуля можно выполнить, введя следующую систему обозначений:

Пусть x_i и y_i – состояние i-го разряда регистров X и Y;

f_i – состояние i-го выхода *Flip*-сети ;

F – логическая функция двух переменных.

В системе *STARAN* имеется 3 варианта операций, реализуемых обрабатывающими элементами:

1) совместная независимая реализация одинаковых логических функций

$$F(x_i, f_i) \rightarrow x_i, \quad F(y_i, f_i) \rightarrow y_i, \quad i = 0, 1, \ldots, 255;$$

2) селективная реализация указанных функций

$$F(x_i, f_i) \rightarrow x_i \text{ при } y_i = 1; \; x_i \rightarrow x_i \text{ при } y_i = 0 \quad i = 0, 1, \ldots, 255;$$

3) селективная и независимая их реализация

$$F(x_i, f_i) \rightarrow x_i \text{ при } x_i = 1; \; x_i \rightarrow \bar{x}_i \text{ при } y_i = 0;$$

$$F(y_i, f_i) \rightarrow y_i, \quad i = 0, 1, \ldots, 255.$$

В последнем случае для маскирования операций на регистре X используется состояние регистра Y, в котором он находился перед модификацией при помощи F.

Рассмотрим основные типы команд, реализованных в системе *STARAN*.

1. *Команды регистра*

Операции над данными, не находящимися в АЗУ, приведены в табл. 5.1.

Таблица 5.1

Команды регистра

Источник информации	Назначение	Комментарий
X, Y или M	X или Y	
$r(X)$, $r(Y)$, $r(F)$	X, Y или F соответствен-	
f_α (F, X, Y)	но	$\alpha = 0, 1, \ldots, 15$
g_β (F, Y)	X	$\beta = 0, 1, \ldots, 15$
h (F)	Y	
k (X, Y)	X или Y	
m (F)	F	

Функции r, f_α, g_β, h, k определяются следующим образом:

r – вращение содержимого регистра;

f_α – в управляющем блоке содержится указатель ($FP1$), который загружается из управляющей памяти и указывает определенный разряд регистра F.

Команда определяет две логические функции, одна из которых выбирается, если в указанном разряде находится “1”, а другая – если в указанном разряде – “0”. Возможными функциями являются

$$(X_i \wedge \overline{Y}_i) \vee \left[Y_i \wedge \left(F_i op_a x_i\right)\right],$$

где нижний индекс i определяется указанным разрядом регистров, а op_α – одна из 16 булевых функций от двух переменных;

g_β – указатель $FP1$, аналогично рассмотренному выше, определяет одну из двух функций, имеющих вид $F_i\ op_\beta\ Y_i$.;

h – содержимое регистра F циклически сдвигается вправо на 32 N разрядов, где $N = 0, 1, \ldots, 7$, в результате чего все разряды номеров 32–255 должны быть равны 0;

k – если все $Y_i = 0$, значение этой функции будет равно X, в противном случае ее значение равно Y;

m – изменение порядка разрядов регистра *F* на противоположное.

Отметим, что в общий регистр *F* необходимо предварительно загрузить слово, разрядный столбец из АЗУ или управляющий код.

2. *Команды обработки многократных откликов*

В управляющем блоке имеются два указателя – *FP*1 *и FP*2, которые определяют соответственно адрес блока и слово или разрядный столбец.

Команда *"Найти первую откликнувшуюся ячейку"* предназначена для определения адресного кода старшей значащей "1" регистра *Y* (внутри блока), для чего используется анализатор многократных совпадений, и загрузки его в (*FP*1, *FP*2).

Команда *"Сбросить первую откликнувшуюся ячейку"* сбрасывает значащую "1" в регистре *Y*.

Команда *"Сбросить другие откликнувшиеся ячейки"* выбирает старшую значащую "1" в регистре *Y* и сбрасывает все остальные.

3. *Команды обращения к ассоциативной памяти*

В управляющем блоке имеются различные регистры для адресации к ассоциативной памяти. Эти регистры можно загрузить из управляющей памяти, их содержимое вместе с кодом команды поступает на общую шину управления.

Регистр выбора блоков, каждый разряд которого соответствует одному блоку. Для указания конкретного блока, к которому производится обращение, в соответствующем разряде регистра необходимо записать "1".

При *адресном чтении или записи из АЗУ* регистр *F* служит источником или приемником данных соответственно. Содержимое регистра *F* можно разделить на 8 полей по 32 разряда каждое, причем только выбранное поле можно записать и прочитать независимо.

Задаваемый в команде режим адресации указывает, производится обращение к слову или разрядному столбцу блока. При очень часто применяемом режиме адресации используются два указателя:

*FP*1 определяет адрес блока,

*FP*2 определяет слово или разрядный столбец.

Существуют и другие режимы, при которых применяются еще два дополнительных указателя. Возможные типы инструкций (команд) приведены в табл. 5.2.

Таблица 5.2

Команды обращения к ассоциативной памяти

Источник	Назначение	Комментарий
АЗУ	*F*	Все или только отдельные поля регистра *F* могут получить новое значение
F	*АЗУ*	Все или только отдельные поля АЗУ могут получить новое значение
АЗУ, *X*, *Y* или *M*	*X*, *Y* или *M* *АЗУ*	Можно использовать маскирование регистров *X* или *Y* при помощи *M*

Различные сдвиговые и перестановочные операции можно выполнить путем модификации указателей *FP*1, *FP*2 (например, увеличить или уменьшить их значения) между парами команд чтения и записи.

Все рассмотренные выше команды предназначены **только** для взаимодействия с ассоциативной памятью.

Кроме таких команд, в системе *STARAN* имеются (выполняются) также и другие команды:

– команды условных и безусловных переходов;

– команды управления программными циклами заданной длины;

– команды обработки приоритетов;

– команды связи с ПФУ;

– команды арифметических операций и др.

Рассматривать все команды системы *STARAN* нет необходимости, так как они во многом аналогичны командам обычных ЭВМ. Однако некоторые особенности, характерные для *STARAN*, следует отметить.

Кроме обычных для универсальных ЭВМ условий переходов в зависимости от наличия нескольких *флажков-признаков* (*“переполнение”*, *“= 0” или “знак результата”*), здесь может существовать большое количество других условий, определяющих переходы, т. е. состояния полей регистров, которые могут применяться в процессе вычислений:

1) *поля указателей* *FP*1, *FP*2, *FP*3, *FP*4;

2) *поля регистров длины FL*1, *FL*2, используемых в качестве счетчиков программных циклов;

3) *поля регистров переходов и связи*, которые используются для хранения содержимого программных счетчиков в режиме прерывания.

Программное обеспечение системы STARAN

Программное обеспечение *STARAN с*остоит из следующих программ:

– *Ассемблер Apple* (см. ниже);

– *пакет обслуживающих программ;*

– *отладочный пакет;*

– *диагностические программы;*

– *библиотека подпрограмм*.

Все эти программы выполняются *последовательным контроллером*, за исключением библиотеки подпрограмм и некоторых других программ, выполняемых устройством управления ассоциативного процессора.

Система команд содержит приблизительно 100 инструкций, разделенных по способу адресации на следующие группы:

– *с адресацией слов* (указывает на память АЗУ);

– *с адресацией путем указателя данных* (указывает на память АЗУ);

– *с непосредственной адресацией*;

– *с адресацией массива* (разрядный срез или слово адресуется либо непосредственно, либо с помощью указателя поля).

Язык APPLE (*Associative Processor Programming Language*)

Хотя программист должен определять форматы записей при помощи *операторов описания данных*, он не должен заботиться о физическом размещении записей в массиве. Кроме обычных, выполняемых последовательно команд загрузки, тестирования, ветвления и управления, язык *APPLE* содержит также команды ассоциативного поиска и арифметические команды. В системе *STARAN* записи различных логических файлов могут быть физически перемешаны в массиве и логически не упорядочены. Операции могут быть условными в зависимости от определенных меток полей, так что нет необходимости упорядочивать записи.

5.5.2. Ортогональная ЭВМ

Используемые при вычислениях переменные, записанные в виде слов, в системе *STARAN* и аналогичных ЭВМ, обрабатываются независимо с помощью одновременного выполнения аналогичных программных шагов для всех или части слов (групповая обработка). Однако, например, при выполнении операций векторной алгебры и различных преобразований, при которых последовательность операций должна выполняться с наборами данных памяти, таких, как элементы векторов или матриц или переменные в функциях и преобразованиях, взаимосвязанные переменные необходимо обрабатывать параллельно. При этом используются принципы *ортогональной обработки информации* [3]. Напомним, например, для *евклидова векторного пространства ортогональное преобразование – это линейное преобразование, сохраняющее неизменным длины или (что эквивалентно этому) скалярное произведение векторов*. Так, в двумерном пространстве, т.е. в плоскости, ортогональное преобразование определяет поворот на некоторый угол вокруг начала координат или зеркальное отражение относительно некоторой прямой, проходящей через начало координат (если определитель соответствующей ортогональной матрицы преобразования равен минус 1).

На рис. 5.12 приведена упрощенная схема такой вычислительной системы на примере ЭВМ *OMEN* (*Orthogonal Mini Embedment*).

Для сохранения возможности работы со словами (горизонтальная обработка), присущей универсальным ЭВМ, а также использования возможности применения операций с разрядными столбцами (вертикальная обработка), что позволяет одновременно выполнять операции с большим числом данных, применяются два арифметических блока: один из них – обычный арифметический блок ЭВМ, другой – вертикальный арифметический блок, в значительной степени аналогичный памяти результатов, рассмотренной выше.

Функциональный генератор вертикального арифметического блока содержит логические цепи для каждого разряда, при помощи которых можно сформировать любую из 16 булевых функций двух переменных, которые являются соответствующими разрядами буферных регистров вертикального блока или одного разряда из регистра и одного разряда, считываемого из памяти.

(*OMEN*) – *США*

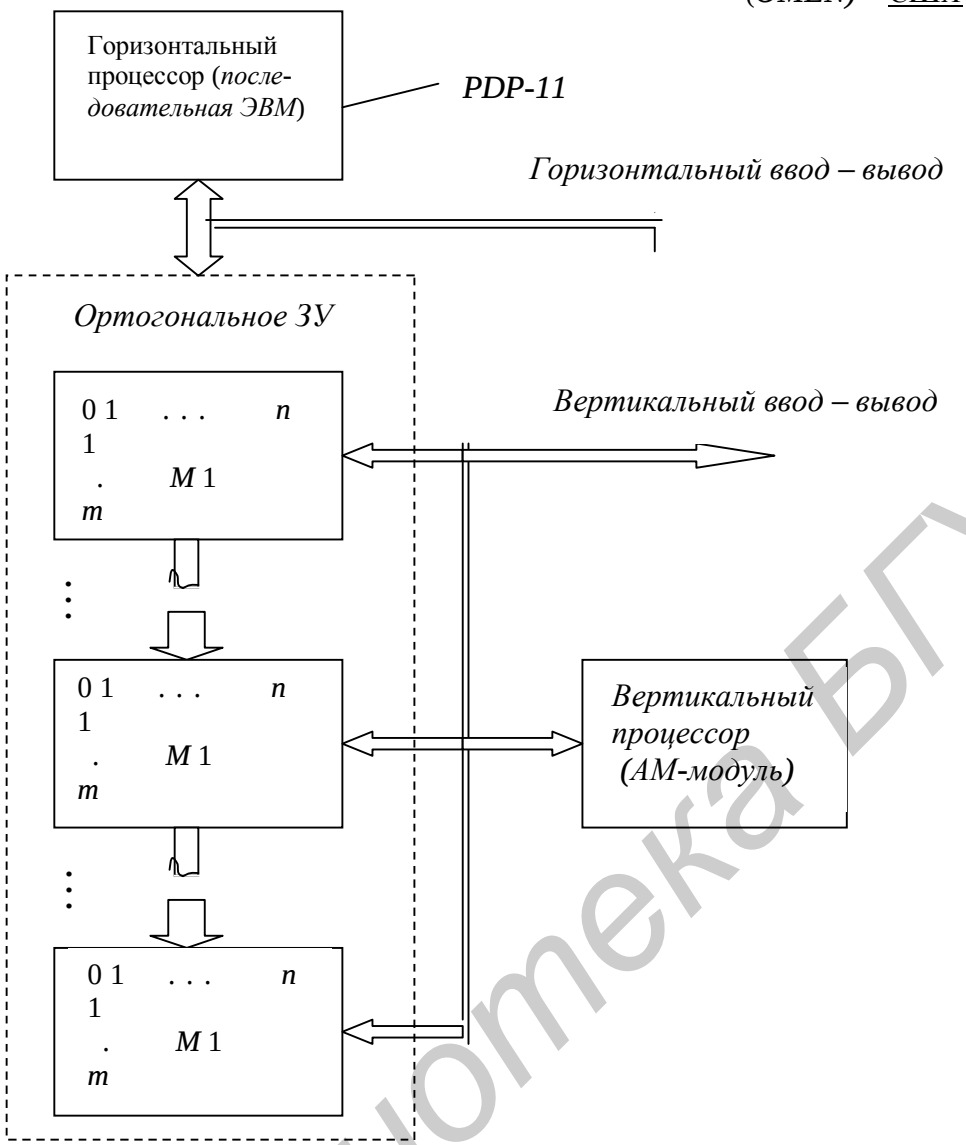


Рис. 5.12. Структурная схема ортогональной ЭВМ OMEN

Результат применения функции можно поместить в любой вертикальный регистр или разрядный столбец. Каждый разряд в вертикальном арифметическом блоке дополнен *схемами полного суммирования* и *памятью* для бита переноса. В устройстве содержится несколько *регистров переноса,* при помощи которых обеспечивается одновременное выполнение различных арифметических операций.

Вертикальный арифметический блок, кроме разрядных столбцов, может выполнять операции с вертикальными слоями разрядов (состоящими из двух разрядов горизонтальных слов). В системе *OMEN* имеется 8 буферных регист-

ров для операндов. Модуль ассоциативной памяти реализован на микросхемах типа *Intel* 1103 и позволяет выполнять диагональную адресацию. Емкость АЗУ составляет 128 К 16-разрядных слов. В качестве горизонтального арифметического блока использована ЭВМ типа *PDP*-11, позволяющей выполнять все 16 булевых операций с использованием АЗУ в качестве модуля памяти.

Вычислительная система *OMEN* в основном предназначена для выполнения матричных операций, дискретных преобразований, например, быстрого преобразования Фурье и других задач обработки сигналов.

Контрольные вопросы к разделу 5

1. Назовите основные направления развития АЗУ, ориентированные на повышение параллелизма и гибкости при выполнении поисковых операций.

2. Приведите классификацию АП по уровню распределенности аппаратной поддержки (логики).

3. Назовите основные типы АП с высоким уровнем параллелизма.

4. Приведите основные особенности матричных процессоров и примеры реализации таких процессоров.

5. Приведите основные особенности процессоров *RADCAD* и *PEPE*.

6. Приведите основные особенности АП с последовательной обработкой разрядов.

7. Назовите основные особенности системы *STARAN* (структуры, построения памяти хранения результатов, системы команд, программного обеспечения).

8. Приведите основные особенности ортогональных ЭВМ.

6. ПРИМЕНЕНИЕ ПРИНЦИПОВ АССОЦИАТИВНОГО ПОИСКА И ОБРАБОТКИ ИНФОРМАЦИИ В ИНТЕЛЛЕКТУАЛЬНЫХ КОМПЬЮТЕРАХ

6.1. Основные особенности интеллектуальных компьютеров

Основные особенности интеллектуальных компьютеров (ИК), которыми должны были стать ЭВМ пятого поколения, рассмотрены в [9, 14].

Если переходы от первого ко второму, от второго к третьему и от третьего к четвертому поколению ЭВМ характеризовались в основном ростом производительности и расширением сервисных возможностей для пользователей, то переход к ЭВМ пятого поколения должен был означать резкий рост *«интеллектуальных»* возможностей (способностей) компьютера.

В частности, интеллектуальный компьютер должен непосредственно «понимать» задачу, поставленную перед ним человеком. Следовательно, отпадает необходимость в составлении программы как средства «общения» с ЭВМ при решении той или иной задачи.

Предполагается, что интеллектуальные компьютеры будут иметь соответствующие технические и программные средства, которые позволят им вести диалог с непрофессиональным пользователем *на естественном языке:*

– в речевой форме;

– в виде графической информации (чертежи, образы, текст).

Следовательно, будет реализован интеллектуальный интерфейс *«человек-машина»*, под которым понимается система программных и аппаратных средств, обеспечивающих для конечного пользователя, не имеющего специальной подготовки в области вычислительной техники, использование ЭВМ для решения задач, возникающих в сфере его профессиональной деятельности, либо вообще без посредников-программистов, либо с незначительной их помощью.

Процесс внедрения интеллектуального интерфейса в ВТ можно определить как *интеллектуализацию ЭВМ*. При этом термины *«интеллектуальный»* и *«интеллектуализация»* не предполагают обязательного определения понятия *«интеллект»*, а только подчеркивают, с одной стороны, нетривиальность функций, выполняемых интеллектуальным интерфейсом в общей системе, а с другой – то, что выполнение этих функций до последнего времени являлось прерогативой человека.

Функционирование интеллектуального интерфейса опирается на *методы работы со знаниями* – их представление, хранение, преобразование и т.д.

Под термином *«знание»* при этом понимается вся совокупность информации, необходимой для решения задачи, включающей в себя в том числе информацию:

1) о системе понятий предметной области, в которой решаются задачи;
2) о системе понятий формальных моделей, на основе которых решаются задачи;
3) о соответствии систем понятий, указанных в пп. 1 и 2;
4) о текущем состоянии предметной области;
5) о методах решения задач.

При этом система знаний должна быть организована в ЭВМ таким образом, чтобы обеспечить взаимодействие ВС с пользователем в системе понятий и терминов предметной области.

Знания о предметной области, организованные на основе тех или иных методов и средств представления знаний, называются *моделью предметной области* (*МПО*).

Средства поддержки такой технологии должны обеспечивать представление информации в естественном для пользователя виде – текстов, изображения, речи.

Основное требование к ВС для такой технологии – это превращение машины в удобного партнера конечного пользователя. Удобство программного продукта для пользователя в общем случае является более важным качеством, чем *производительность* ЭВМ.

Из неподготовленности пользователя, его случайного характера вытекает требование обеспечения работы ВС в условиях ошибок, допускаемых пользователем (необходимо в процессе общения иметь возможность указывать пользователю ошибки и непонятные места в ходе решения задач).

Отметим, что средства интеллектуального интерфейса образуют новый уровень в ВС – *уровень конечного пользователя* (*КП*).

Отличие этого уровня от предыдущих:

1) средства уровня КП ориентированы на приспособление к пользователям, имеющим различную (в том числе низкую) квалификацию; средства предыдущих уровней направлены на приспособление пользователя к этим средствам, требующим приобретения специальной квалификации;

2) средства уровня КП должны обеспечивать весь процесс решения задачи, начиная от постановки задачи, в то время как традиционные средства были направлены на обеспечение выполнения программы;

3) средства уровня КП должны обеспечивать решение задач совместно ЭВМ и КП.

Напомним еще раз основные функции, которые должны быть реализованы в интеллектуальных ЭВМ:

– решение задач и получение выводов;

– управление базами знаний и переработка знаний;

– обеспечение интеллектуальных интерфейсов.

Соответственно ожидается, что в состав аппаратных средств компьютеров пятого поколения войдут три основные функциональные компоненты, обеспечивающие выполнение приведенных выше функций. Особого внимания заслуживает компонента управления базами знаний, которая включает механизмы накопления и обновления информации в памяти ЭВМ, являющиеся ключевыми при манипулировании знаниями, а компоненты распорядителя базой знаний и управления процессом получения выводов составляют в совокупности машину баз знаний. На эту машину возлагаются следующие функции:

– накопление большого объема хорошо структурированной информации и обеспечение доступа к любой ее части;

– поиск необходимых сведений в ответ на запрос от подсистемы получения выводов с последующей передачей их этой подсистеме;

– накопление в базе знаний сведений, поступающих от подсистемы получения выводов.

Рассмотрим возможности и перспективы применения принципов ассоциативного поиска и обработки информации для решения в интеллектуальных ЭВМ основных задач обработки знаний, создания интеллектуального человеко-машинного интерфейса и др.

6.2. Применение принципов ассоциативного поиска и обработки информации для решения задач обработки знаний

Напомним особенности фон-неймановской логической организации ВС, которые существенно затрудняют создание машин обработки знаний на основе традиционных ЭВМ:

– последовательная обработка, ограничивающая эффективность физическими возможностями элементной базы;

– низкий уровень доступа к памяти, т.е. сложность и громоздкие процедуры ассоциативного поиска нужного фрагмента знаний. Так как в интеллектуальных системах операции информационного поиска носят массовый характер, то эти системы, если они построены на базе современных ЭВМ, затрачивают больше времени (на несколько порядков) на информационный поиск, чем собственно на переработку знаний. Устранение этого недостатка требует создания ассоциативной памяти, обеспечивающей ассоциативный доступ к произвольным фрагментам хранимых знаний, к фрагментам, имеющим произвольные размеры и произвольную структуру. При этом аппаратная реализация информационно-поисковых операций в силу их массовости требует глубокого распараллеливания;

– линейная организация памяти и простой вид конструктивных объектов, непосредственно хранимых в памяти. Это приводит к тому, что в интеллектуальных системах, построенных на базе современных ЭВМ, манипулирование знаниями осуществляется с большим трудом;

– представление информации в памяти современных ЭВМ имеет уровень, весьма далекий от семантического, что делает переработку знаний довольно громоздкой, требующей учета большого количества деталей, касающихся не смысла перерабатываемой информации, а способа ее представления в памяти;

– в современных ЭВМ имеет место весьма низкий уровень аппаратно-реализуемых операций над числовыми данными и полностью отсутствует аппаратная поддержка логических операций над фрагментами знаний, имеющих сложную структуру, что делает манипулирование такими фрагментами весьма сложным;

– в современных ЭВМ громоздко реализуются даже простейшие процедуры логического вывода.

Отметим, что бурный прогресс электроники со времен появления первых ЭВМ (появление транзисторов, БИС, СБИС) создает большие возможности для значительного развития архитектуры ЭВМ, повышения уровня машинного языка, ассоциативной обработки и т.д.

Дальнейшую эволюцию средств ВТ применительно к созданию машин обработки знаний можно разделить на три линии. Первая связана с появлением

большого числа новых классов ВТ, направленных на более эффективное решение отдельных типов задач; вторая – с попытками создания машин переработки знаний; третья – с развитием не-фон-неймановских парадигм обработки информации.

Трудности, связанные с использованием универсальных ЭВМ, включая и многопроцессорные системы, послужили толчком к развитию теоретических исследований в области параллельного программирования. Возникло множество подходов, связанных с отдельными изменениями принципов логической организации ЭВМ в зависимости от классов задач и предметных областей, на которые ориентируется тот или иной класс ЭВМ. Эти тенденции позволили определить некоторые ключевые принципы логической организации машин обработки знаний.

Среди этих тенденций основными являются:

– переход к нелинейной организации памяти;

– аппаратная реализация ассоциативного доступа к информации;

– реализация параллельных асинхронных процессов над памятью, в частности, разработка вычислительных машин, управляемых потоком данных;

– аппаратная интерпретация языков высокого уровня;

– разработка аппаратных средств ведения баз данных (процессоров баз данных).

На пересечении этих тенденций возникли различные классы вычислительных устройств, среди которых можно отметить:

– машины с развитой ассоциативной памятью;

– ассоциативные параллельные матричные процессоры;

– системы, осуществляющие переработку информации непосредственно в памяти путем равномерного распределения функциональных средств по памяти;

– вычислительные машины со структурно-перенастраиваемой памятью;

– ассоциативные однородные среды и др.

Применительно к проблематике машин переработки знаний представляет интерес рассмотрение перечисленных выше устройств в эволюционном аспекте, приведенное в [13]. В данном разделе пособия кратко рассмотрены только ассоциативные процессоры и процессоры баз данных.

Ассоциативные процессоры (АП) подробнее рассмотрены в разд. 5 данного пособия. Напомним, что преимущества АП связаны прежде всего с про-

стотой и эффективностью реализации поисковых операций в больших информационных массивах. Ассоциативный характер переработки информации снимает проблему адресации ячеек памяти.

Использование традиционных АП имеет ограничения, связанные с невысокими возможностями внутренней структуризации информации в АЗУ, вызванными традиционной линейной организацией их памяти. Это накладывает значительные ограничения на вид информационных массивов, в которых производится поиск. Традиционные АП являются удобными средствами поиска и простой переработки информации в массивах регулярной структуры. В то же время для работы с массивами нечисловой сложноструктурированной информации они оказываются неудобными. Потребности же в организации эффективного поиска в массивах нечисловой информации, имеющей сложную нерегулярную структуру, растут и связаны в основном с подходами к созданию машин обработки знаний.

Появление процессоров баз данных (ПБД) можно в определенной мере рассматривать как переход к качественно новой ступени развития АП. Общей особенностью ПБД и АП является ориентация на операции поиска в массивах регулярной структуры. Их различие заключается в том, что в ПБД реализуются гораздо более сложные поисковые операции, по отношению к которым операция ассоциативного поиска представляет собой «строительный блок». ПБД основаны на моделях данных, среди которых наибольшее распространение получили реляционная, сетевая и иерархическая. По мере развития технических средств реляционная модель данных занимает доминирующее положение благодаря своей внутренней простоте и однородности [13].

Функционирование процессоров реляционных баз данных сводится к выполнению операций реляционной алгебры над реляционными таблицами. Операции реляционной алгебры (выборка, объединение, проекция) требуют значительного информационного поиска в реляционных таблицах, поэтому при создании ПБД используются научные и технологические результаты, связанные с разработкой АП, в частности такие, как принцип динамической реализации АЗУ с «логикой на дорожке», набор операций специализированных однородных процессоров, относящихся к классу матричных АП, концепция крупноблочной обработки и др. [4].

ПБД представляют собой важную ступень в процессе развития технических средств переработки информации. Данные в реляционной модели уже обладают определенными чертами знаний, в частности, внутренней интерпретируемостью, что подтверждается ориентацией реляционной модели данных на довольно сложные механизмы переработки и информационного поиска. Усложнение структур перерабатываемых данных приводит к необходимости дальнейшего наращивания мощности механизмов информационного поиска и переработки знаний.

6.3. Применение принципов ассоциативного поиска и обработки информации для создания интеллектуального интерфейса

Как уже было отмечено раньше, одной из важнейших особенностей интеллектуального интерфейса является обеспечение возможности общения пользователя с ЭВМ на естественном языке. Проблема перевода естественного языка на машинный оказалась достаточно трудной. Было предпринято много различных попыток и создано значительное количество систем, которые существенно различаются по уровню компетентности.

Иногда полезно иметь даже плохой перевод, если он сделан достаточно быстро. Некоторые системы могут обрабатывать текст, используя усеченную форму языка, что облегчает перевод. А другие системы могут быть нацелены на решение конкретных подзадач, например, просмотр словаря, или (как в случае плохого перевода) полагаться на участие человека (редактора) в подготовке окончательной версии перевода. Следует отметить, что хороший перевод можно сделать только при условии понимания текста машиной (или человеком). Это условие стало ключевым для исследований во всех сферах искусственного интеллекта.

Наряду с переводом существуют и другие подходы к вопросу о том, каким образом машины могут получить возможность понимать естественный язык. Так, в одном из подходов [13] было предложено считать, что машина понимает текст, если она:

– на вопросы дает правильные ответы, вытекающие из текста;

– на вопросы дает правильные ответы, с одной стороны, вытекающие из текста, с другой – из ранее полученных знаний;

– правильно увязывает знания, полученные из текста, со своими прежними знаниями.

Некоторые системы перевода с естественного языка рассмотрены в [14].

Одним из важнейших свойств как человека, так и других интеллектуальных систем, имеющих решающее значение для создания человеко-машинного интерфейса, в компьютерных системах пятого поколения является эффективное распознавание образов. Необходимо не только собирать и хранить информацию, но и интерпретировать ее, а это значит, что во многих ситуациях потребуется решать задачи распознавания образов.

Люди обладают удивительной способностью распознавать образы, в то время как у компьютеров эта способность находится на более низком уровне. Люди привлекают богатую контекстуальную информацию, владеют изощренными стратегиями поиска (мы выделяем соответствующие сигналы и преобразуем их в понятную для нашего мозга форму: одни глаза могут воспринимать и передавать 4,3 млн битов информации в секунду). Можно ожидать, что развитие инженерии знаний для искусственных систем позволит расширить когнитивную основу, в рамках которой станут возможными различные виды интеллектуальной деятельности, включая эффективное распознавание образов.

ЭВМ относительно легко может выявлять отдельные фигуры (например буквы), если эти фигуры хорошо определены, имеют четкие границы и занимают ожидаемое положение. Распознавание комбинации свойств может осуществляться с помощью как последовательной, так и параллельной обработки.

Известны программы, которые могут идентифицировать точки пересечения линий (вершины), что позволяет извлекать информацию о сложных геометрических объектах; программы, которые могут распознавать линии, области и т.д.

Для распознавания символов алфавита, цифр и других фигур часто используются шаблоны: входные данные сравниваются с представленным в памяти шаблоном фигуры. Ведутся работы по созданию программ, которые позво-

ляют компьютерам распознавать естественные образы, такие, как электрокардиограммы, изображения облаков, клетки крови, отпечатки пальцев и др. [25].

Известны системы, использующие свойства ассоциативной памяти для автоматизации процесса распознавания образов. В этих системах связанные сигналы, одновременно генерируемые образом, запоминаются так, что при повторении одного из них порождается другой сигнал. Согласно этому подходу сигнал может быть воссоздан с помощью последующего сигнала, похожего на оригинальный, но не идентичного ему. Такой прием используется и в жизни, когда необходимо распознать неясное или неполное изображение.

Этот способ распознавания может послужить основой для совершенствования взаимодействия человека с машиной, помочь в принятии решения, автоматического перевода с одного языка на другой, в доступе к данным и т.д.

Большое признание получит подход, согласно которому восприятие рассматривается как вычислительный процесс [14].

Умение распознавать образы (зрительные или иные) должно стать ключевым свойством вычислительных систем, «дружелюбно» настроенных к пользователю. Техника распознавания образов как таковая, применяется ли она для обработки конструкций естественного языка или в процессе восприятия реального окружающего мира, приобретает все большее значение для компьютеров пятого поколения.

6.4. Применение принципов ассоциативного поиска в механизмах памяти интеллектуальных ЭВМ

В естественных биологических системах и системах искусственного интеллекта память имеет два уровня: активная память и пассивная память [14].

Возможности памяти в искусственных и биологических системах зависят от того, насколько эффективны активная и пассивная системы, конструкции запоминающих элементов, общие принципы и методы кодирования, способы поиска, ассоциативная память, буферы и другие подсистемы. В одной из моделей системы памяти запоминающий элемент представлен как набор независимых компонентов, управляемых центральным обрабатывающим устройством, которое вызывает информацию, когда это требуется, и загружает закодированную

информацию в ячейки памяти. Данный процесс включает *«поиск»* и *«активную обработку сообщения»*, причем каждый элемент памяти может иметь собственные вычислительные ресурсы для обработки сообщений, посылаемых таким элементам памяти, и каждый элемент может независимо решать, какое действие ему выполнить. Такая организация очень напоминает структуру человеческой памяти, где определенные ее элементы, будучи связанными друг с другом, образуют сеть.

В *активной* памяти, связанной в сеть, информация может быть найдена множеством различных способов (но простейший из них – *ассоциативный поиск*).

В *пассивной* памяти осуществляется *адресная выборка*.

Нет сомнения, что принципы ассоциативного поиска и обработки информации найдут применение в решении и других задач, возникающих при создании интеллектуальных ЭВМ.

6.5. Графодинамическая ассоциативная машина обработки знаний

6.5.1. Основные преимущества использования графодинамических ассоциативных машин

Для устранения перечисленных в подразд. 6.2 ограничивающих особенностей фон-неймановской логической организации ВС, существенно затрудняющих создание машин обработки знаний на основе традиционных ЭВМ, в работах [13, 21] предлагается использовать графодинамические ассоциативные машины.

Преимущество использования графодинамических абстрактных машин в качестве инструмента для создания интеллектуальных компьютеров (аппаратно-программных систем, основанных на знаниях) обусловлено следующими обстоятельствами:

1) в них принципиально проще реализуется ассоциативный метод доступа к перерабатываемой информации;

2) существенно проще поддерживается открытый характер как самих машин, так и реализуемых на них формальных моделей. В частности, на графодинамических машинах существенно проще реализуются семиотические модели;

3) графодинамические ассоциативные машины являются удобной основой для интеграции самых различных моделей обработки информации, ибо на базе графодинамических абстрактных машин легко реализуются не только графодинамические, но и символьные формальные модели, не только параллельные, но и последовательные модели, не только асинхронные, но и синхронные модели, не только различные модели логического вывода, но и всевозможные модели реализации хранимых процедурных программ.

Остальные достоинства графодинамических ассоциативных машин обусловлены достоинствами графовых информационных конструкций и графовых языков.

6.5.2. Структура графодинамической ассоциативной памяти

Структура памяти абстрактной машины описывается специальным языком, для графодинамической ассоциативной машины таким языком является язык SC (Semantic Code), который рассмотрен в работе [21]. Базис этого языка составляет абстрактный тип данных – графодинамические структуры, основные идеи заключаются в следующем (рис. 6.1):

1) элементами памяти является дуга или узел;
2) связь между элементами памяти не фиксирована и может меняться;
3) значение элемента памяти также может меняться.

Таким образом, обработка графодинамических структур сводится не только к изменению значения элемента памяти, но и к изменению связи между ними. Различные варианты реализации памяти абстрактной графодинамической ассоциативной машины на традиционной фон-неймановской архитектуре рассмотрены в работах [22, 23, 24]. Основной подход к реализации графодинамических структур на традиционных архитектурах – это представление их в виде списочных структур.

В вышеперечисленных работах также рассмотрены различные варианты представления графодинамических структур в виде списочных структур.

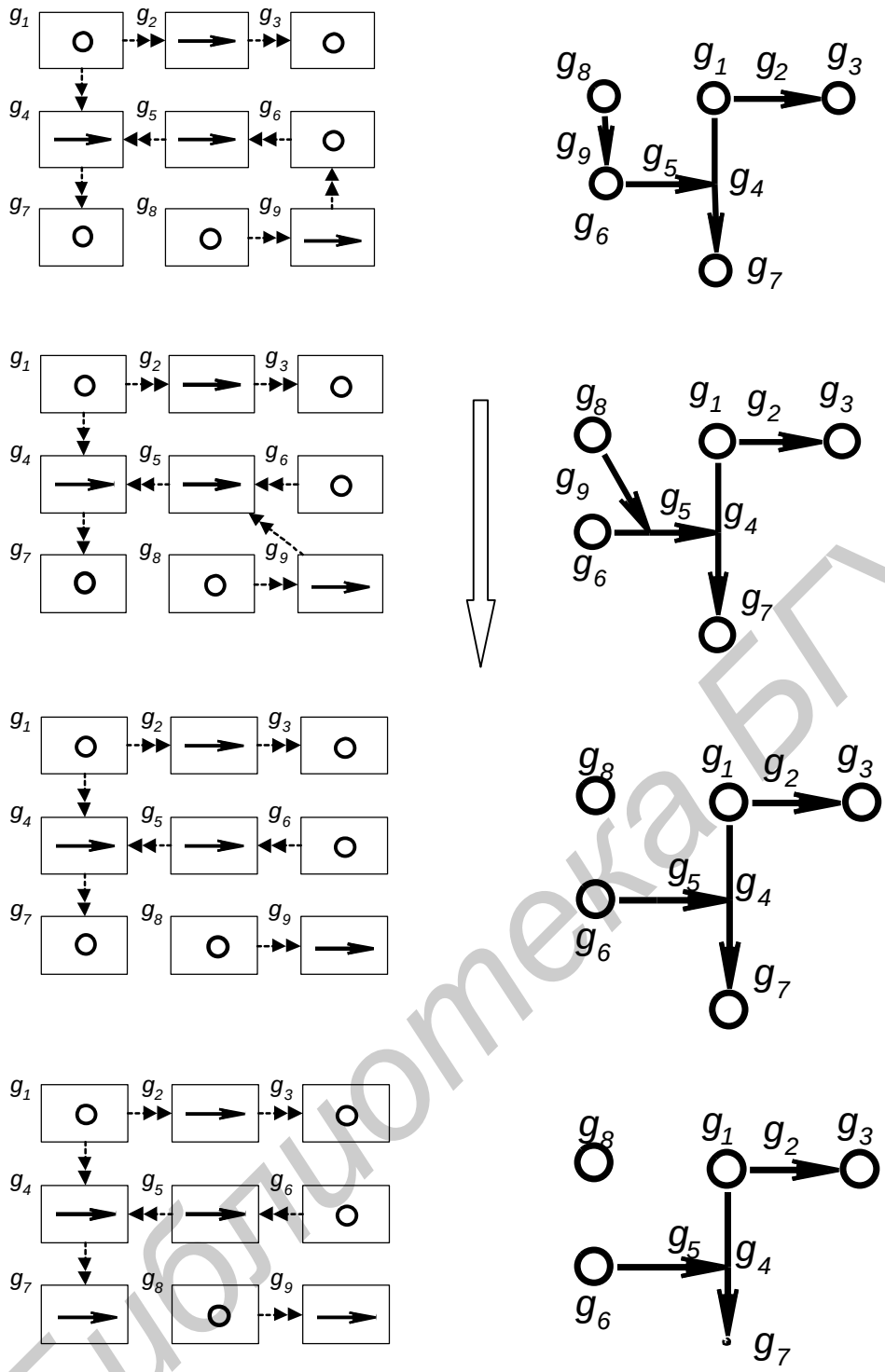


Рис. 6.1. Пример обработки графодинамических структур

Основные особенности этих вариантов следующие:

1) вся память разбивается на блоки, в которых хранится информация о первичных элементах sc-конструкции (тип, идентификатор и содержимое);

2) количество информации в блоке об инцидентных элементах и её местонахождение варьируется в каждом из методов.

Над памятью абстрактной графодинамической ассоциативной машины заданы следующие операции:

1) генерация элемента памяти дуги или узла;
2) удаление элемента памяти;
3) установка начала дуги;
4) установка конца дуги;
5) проверка, является ли элемент памяти узлом или дугой;
6) получение множества входящих дуг в узел или дугу;
7) получение множества выходящих дуг из узла.

На основе рассмотренной абстрактной графодинамической ассоциативной машины может быть построено множество абстрактных графодинамических ассоциативных машин, в которых будет уточняться структура памяти и набор операций.

Способы представления различных типов информации на языке SC рассмотрены в работе [21], сведение абстрактных машин обработки информации к абстрактным графодинамическим ассоциативным машинам рассмотрено в работе [13]. В вышеперечисленных работах показано, что абстрактная графодинамическая ассоциативная машина является достаточно мощным средством описания операционной семантики самых различных языков, а также показано соотношение между различными абстрактными графодинамическими ассоциативными машинами.

Рассмотрим примеры представления различных видов знаний в графодинамической ассоциативной памяти.

На рис. 6.2 приведен пример записи результата измерения массы физического тела, который означает, что масса физического тела p равна $66{,}5$ кг.

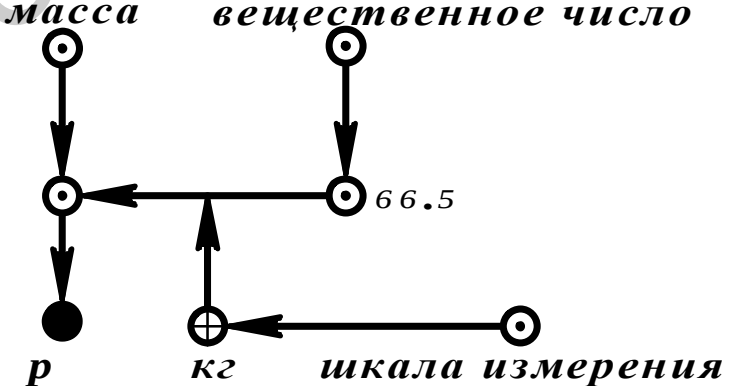


Рис. 6.2. Пример записи результата измерения

На рис. 6.3 приведён пример записи результата «измерения» арности отношения. Эта конструкция означает, что отношение r является n-арным отношением, т.е. представляет собой семейство n-арных множеств (множеств, мощность которых равна n). Заметим, что далеко не каждое отношение обладает свойством иметь арность. Этим свойством обладают те и только те отношения, каждое из которых представляет собой семейство множеств одинаковой мощности. То есть понятие «арность отношения» и понятие «семейство множеств одинаковой мощности» являются синонимами.

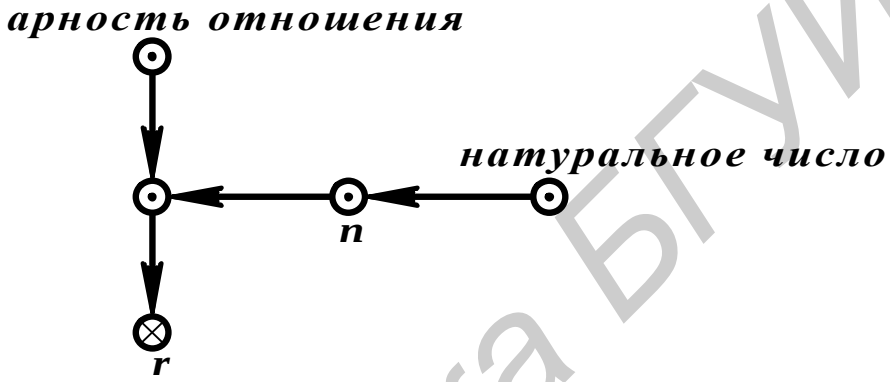


Рис. 6.3. Пример записи результата измерения арности отношения

На рис. 6.4 приведён пример записи результата измерения величины плоского угла в радианах. Измеряемым объектом здесь можно считать тернарный кортеж, состоящий из трёх геометрических фигур, лежащих на одной плоскости. При этом две из этих фигур являются либо отрезками, либо прямыми, либо лучами, а третья фигура трактуется как фигура, лежащая внутри измеряемого угла. Собственно, измеряемым углом здесь является один из четырех углов, образованных пересекающимися прямыми, на которых лежат указанные выше отрезки или лучи. Конструкция на рис. 6.4 означает, что число ***x*** есть результат измерения (в радианах) величины плоского угла, который составлен геометрическими фигурами p1 и p2 (каковыми могут быть прямые, лучи, отрезки, множества точек, лежащих на одной прямой точек) и внутри которого находится геометрическая фигура ***p***.

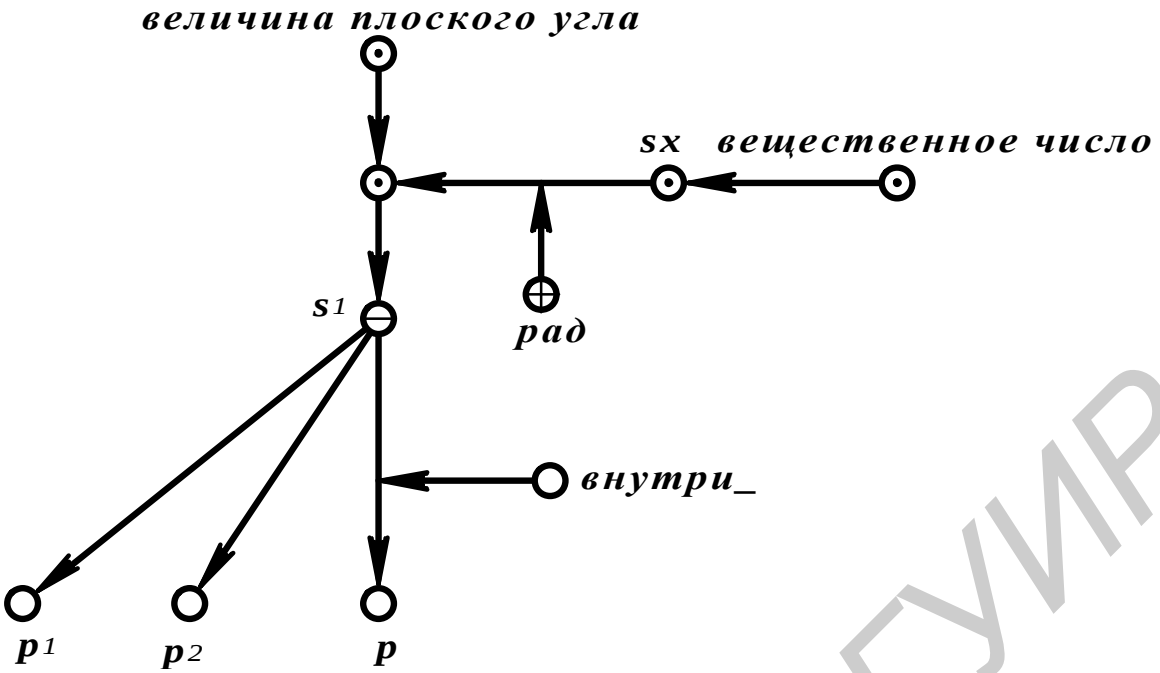


Рис. 6.4. Пример записи результата измерения величины угла

На рис. 6.5 рассмотрен пример записи результата измерения местоположения некоторого объекта ***p*** на земной поверхности в географической системе координат. Эта конструкция означает то, что местоположение объекта p в географической системе координат определяется широтой, которая задаётся кортежем.

Рис. 6.5. Пример записи результата измерения местоположения

На рис. 6.6 приведен пример записи результата измерения длительности во времени (отрезка времени). Приведенная запись означает, что некий процесс ***p*** длился 2 года 1 месяц или (по другой шкале) 765 суток (напомним, что года могут быть високосными, а месяцы могут иметь разное количество дней).

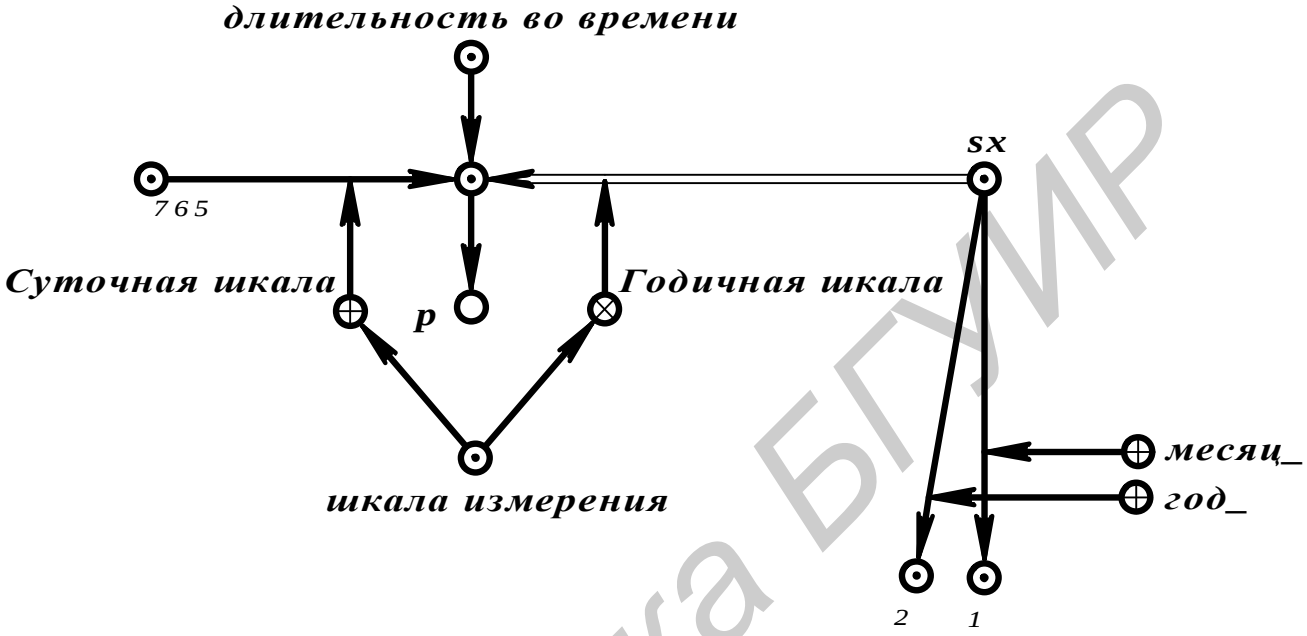


Рис. 6.6. Пример записи результата измерения длительности во времени

На рис. 6.7 приведен пример, показывающий, что некоторый объект имеет массу, равную 30 кг.

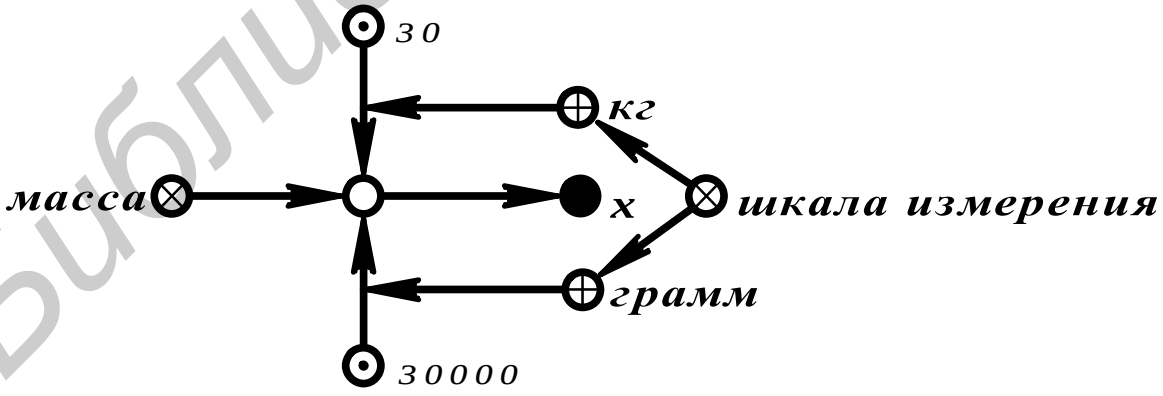


Рис. 6.7. Пример записи результата измерения массы

6.5.3. Язык записи микропрограмм обработки графодинамической ассоциативной памяти

Язык программирования, предназначенный для записи микропрограмм обработки графодинамической ассоциативной памяти, должен быть:

1) языком процедурного программирования, так как микропрограммы абстрактных машин носят процедурный характер;

2) языком параллельного и асинхронного программирования, так как реализация различных операций формальной модели в общем случае носит параллельный и асинхронный характер. Процесс реализации каждой операции также может носить параллельный и асинхронный характер, что существенно повышает производительность интеллектуальных систем. Необходимость в этом возникает при реализации сложных операций, к которым, в частности, относится большинство операций логического вывода в моделях обработки знаний;

3) языком программирования, хорошо приспособленным к переработке sc-конструкций, т.е. графовых конструкций, принадлежащих языку SC. Такая приспособленность, в частности, требует наличия средств ассоциативного доступа к элементам перерабатываемой графовой конструкции.

В качестве языка процедурного программирования, предназначенного для эффективной интерпретации графодинамических моделей различного вида, используется язык SCP (Semantic Code Programming). Язык SCP относится к классу графовых языков программирования. Особенностью этого языка является то, что не только данные, но и сами программы, написанные на языке SCP, представляются в виде sc-конструкций. Язык SCP является языком параллельного асинхронного программирования. Это необходимо для того, чтобы обеспечить адекватную интерпретацию не только последовательных, но и параллельных, не только синхронных, но и асинхронных формальных моделей.

Язык SCP относится к классу процедурных языков параллельного программирования, ориентированных на переработку нечисловой информации.

Особенностями языка SCP по сравнению с другими языками указанного класса являются:

1) приспособленность к переработке нечисловых структур, имеющих мощные выразительные возможности, т.е. обеспечивающих описание любых сложноструктурированных предметных областей;

2) использование структурно перестраиваемой графодинамической ассоциативной памяти. Как тексты языка SCP (scp-процедуры), так и перерабатываемые данные являются графовыми конструкциями, хранимыми в графовой структурно перестраиваемой ассоциативной памяти (это означает, что язык SCP относится к классу графовых языков);

3) ориентация на переработку непосредственно семантических сетей, а не структур (например списковых), с помощью которых семантические сети кодируются (это означает, что язык SCP описывает переработку информации непосредственно на уровне семантического кодирования);

4) возможность описания (с помощью scp-процедур) механизмов решения задач (логических операций, операций вывода), имеющих различный уровень сложности и поддерживающих самые различные стратегии решения задач;

5) высокий потенциал распараллеливания процессов переработки информации в графовой структурно перестраиваемой ассоциативной памяти (как на уровне реализации различных логических операций, так и на уровне реализации каждой из этих операций);

6) возможность через понятие sc-узла интегрировать процедуры над sc-конструкциями с процедурами, написанными на традиционных языках и описывающими переработку обычных информационных объектов. Это обеспечивается тем, что содержимым sc-узла может быть информационный объект любой природы.

Операторы языка SCP (scp-операторы) разбиваются на типы. Множество типов scp-операторов в свою очередь разбивается на семейства типов scp-операторов. Перечислим идентификаторы ключевых узлов, каждый из которых обозначает соответствующее семейство scp-операторов:

1) gen (семейство типов scp-операторов генерации sc-конструкций);

2) erase (семейство типов scp-операторов удаления sc-конструкций);

3) search (семейство типов scp-операторов ассоциативного поиска sc-конструкций);

4) if (семейство типов scp-операторов проверки условий);

5) change (семейство типов scp-операторов изменения свойств sc-элементов);

manage (семейство типов scp-операторов управления scp-процессами).

Семейство scp-операторов генерации sc-конструкций (такие операторы будем также называть gen-операторами) разбивается на следующие типы scp-операторов:

1) операторы генерации sc-элемента (genEl-операторы);

2) операторы генерации sc-конструкции, состоящей из трёх элементов (genStr3-операторы).

Семейство scp-операторов удаления sc-конструкции (такие операторы будем называть также erase-операторами) разбивается на следующие типы scp-операторов:

1) операторы удаления указанного sc-элемента (eraseEl-операторы);

2) операторы удаления указанных элементов sc-конструкции, состоящей из трех элементов (eraseElStr3-операторы);

3) операторы удаления нескольких указанных элементов sc-конструкции, состоящей из трех элементов (eraseSetStr3-операторы).

Семейство scp-операторов ассоциативного поиска sc-конструкций (такие операторы будем называть также search-операторами) разбивается на следующие типы scp-операторов:

1) операторы ассоциативного поиска произвольных элементов sc-конструкции, состоящей из трёх элементов (searchElStr3-операторы);

2) операторы ассоциативного поиска произвольных элементов sc-конструкций, удовлетворяющих заданным условиям и состоящих из трех элементов, и формирования из них множества (searchSetStr3-операторы);

3) операторы ассоциативного поиска произвольных элементов sc-конструкций, удовлетворяющих заданным условиям и состоящих из трёх элементов, и формирования из них множества, элементами которого остаются только sc-элементы, удовлетворяющие заданным условиям (selectYStr3-операторы);

4) операторы ассоциативного поиска произвольных элементов sc-конструкций, удовлетворяющих заданным условиям и состоящих из трёх элементов, и формирования из них множества, элементами которого остаются только sc-элементы, не удовлетворяющие заданным условиям (selectNStr3-операторы).

Семейство scp-операторов проверки условий (такие операторы будем называть также if-операторами) разбивается на следующие типы scp-операторов:

1) операторы проверки типа sc-элемента (ifType-операторы);

2) операторы проверки равенства значений двух операндов (ifEq-операторы);

3) операторы проверки строгого неравенства значений двух операндов (ifGr-операторы);

4) операторы проверки неравенства значений двух операндов (ifGrEq-операторы).

Семейство scp-операторов изменения свойств sc-элемента (такие операторы будем также называть change-операторами) разбивается на следующие типы scp-операторов:

1) операторы сложения содержимых sc-элементов (add-операторы);

2) операторы вычитания содержимых sc-элементов (sub-операторы);

3) операторы пересылки значения программной переменной (assign-операторы).

Семейство scp-операторов управления scp-процессами (такие операторы будем также называть manage-операторами) разбивается на следующие типы scp-операторов:

1) операторы порождения scp-процесса (call-операторы);

2) операторы порождения параллельных scp-процессов (callPar-операторы);

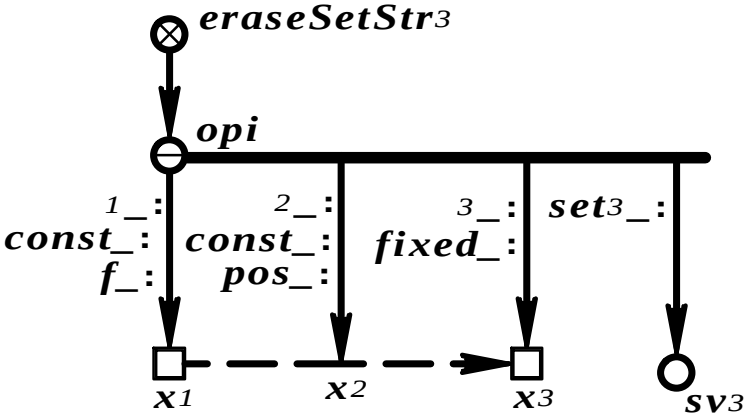
3) операторы завершения scp-процесса (return-операторы);

4) операторы ожидания завершения scp-процесса (waitReturn-операторы).

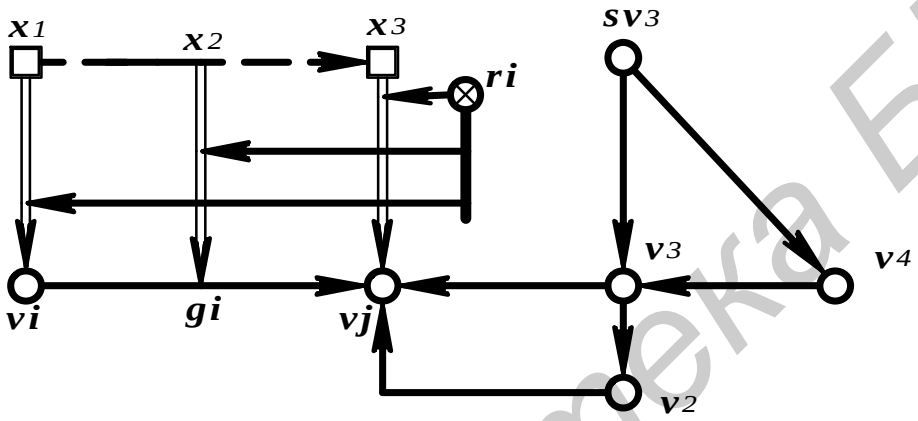
Рассмотрим примеры работы scp-операторов.

На рис. 6.8 показан пример работы оператора удаления eraseSetStr3. После успешного выполнения данного оператора (opi) будет удален sc-узел v2, который является первым элементом трехэлементной конструкции, соответствующей заданному шаблону и третий элемент которого vj входит в множество sv3(состоит из vj и v3). Также будут удалены дуги, входящие и выходящие из удаленного узла.

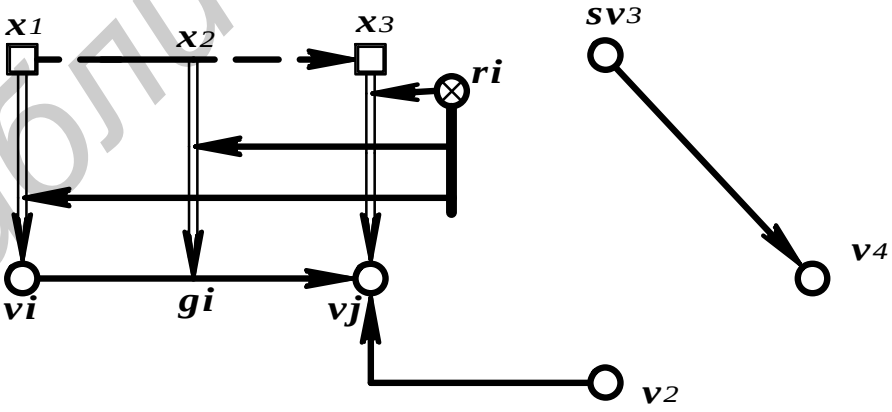
На рис. 6.9 приведен пример работы scp-оператора ассоциативного поиска sarchSetStr3.



а



б

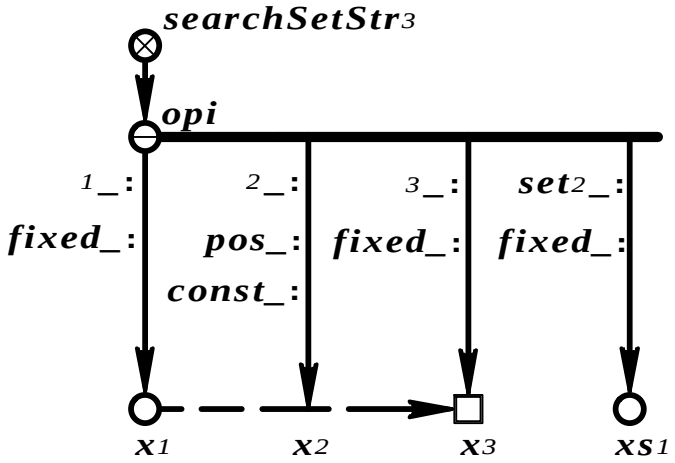


в

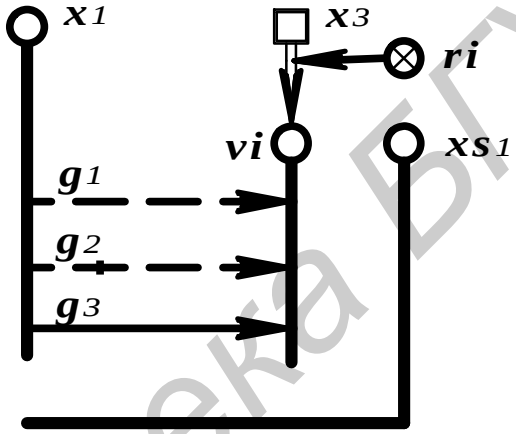
Рис. 6.8. Пример работы scp-оператора eraseSetStr3:

а – запись оператора; б – состояние памяти до выполнения оператора;

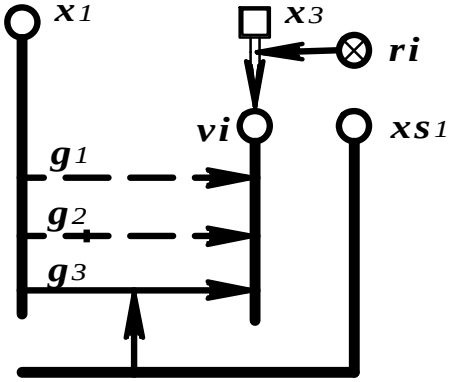
в – состояние памяти после выполнения оператора



а



б



в

Рис. 6.9. Пример работы scp-оператора searchSetStr3:

а – запись оператора; б – состояние памяти до выполнения оператора;

в – состояние памяти после выполнения оператора

В результате успешного выполнения данного оператора (opi) будет найдена константная позитивная sc-дуга (v1) и константная позитивная sc-дуга (gi), соединяющая его с sc-элементом (ei), являющимся значением операнда x3. Также будут сгенерированы дуга, связывающая операнд x2 с sc-элементом gi, и дуга, связывающая операнд x1 с sc-элементом vi, а также дуги принадлежности, связывающие указанные выше дуги со знаком соответствующего бинарного отношения (ri). Генерация этих дуг означает, что у операндов x1 и x2 сформировалось (вычислилось) значение.

6.5.4. Структура базового программного обеспечения графодинамических ассоциативных машин

Структура базового математического и программного обеспечения схематически изображена на рис. 6.10.

В структуре базового программного обеспечения выделяются две главные компоненты, которые включают остальные, это процессорный и терминальный модули. Процессорный модуль предназначен для хранения и обработки графодинамических структур, а терминальный модуль – для взаимодействия с пользователем. Традиционно в состав базового программного обеспечения включают следующие инструментальные средства: операционную среду (или систему), базовую систему программирования, средства визуализации и редактирования информации. Для графодинамических ассоциативных машин процессорный модуль выполняет функции операционной среды, а терминальный модуль – это базовая система программирования, средства визуализации и редактирования графовых структур.

В основе всего базового математического обеспечения лежит модель представления графодинамических структур, рассмотренная в работе [21].

Терминальный модуль основывается на методе визуализации графодинамических структур, который использует язык SCg (Semantic Code graphic), и на методе взаимодействия пользователя с графодинамической ассоциативной машиной. Эти методы и в целом терминальный модуль рассмотрены в работе [13].

Рассмотренные компоненты базового математического и программного обеспечения графодинамических ассоциативных машин позволяют создавать прикладные интеллектуальные системы.

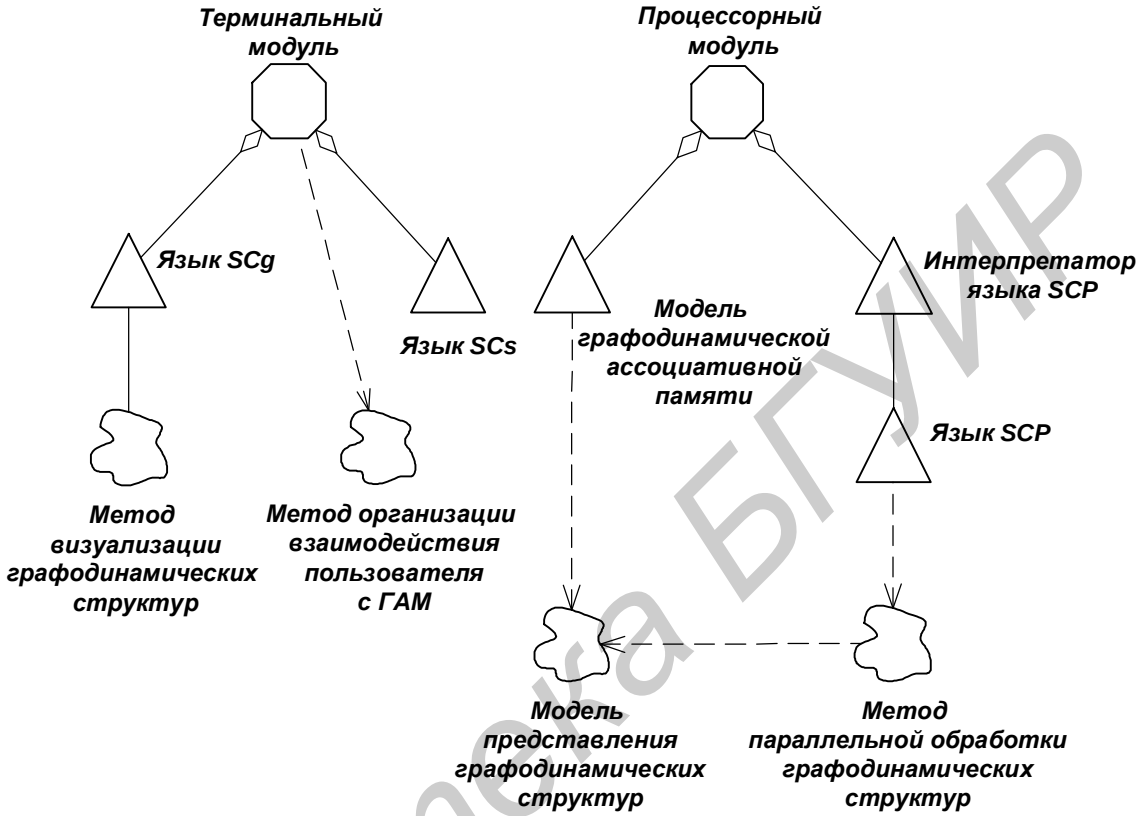


Рис. 6.10. Структура базового математического и программного обеспечения графодинамических ассоциативных машин

Контрольные вопросы к разделу 6

1. Назовите основные особенности интеллектуальных компьютеров.
2. Что понимается под термином «интеллектуальный интерфейс»?
3. Что включает в себя информация, понимаемая под термином «знание»?
4. Назовите основные особенности уровня конечного пользователя.
5. Назовите основные элементы ЭВМ пятого поколения (интеллектуальных ЭВМ).
6. Назовите основные направления применения принципов ассоциативного поиска в интеллектуальных ЭВМ.

7. Назовите особенности логической организации фон-неймановской архитектуры вычислительных систем.

8. Назовите преимущества использования графодинамических ассоциативных машин в качестве основы интеллектуальных систем.

9. Назовите основные элементы структуры графодинамической ассоциативной памяти.

10. Приведите примеры представления различных видов знаний в графодинамической ассоциативной памяти.

11. Назовите основные типы операторов языка SCP.

12. Назовите основные элементы базового программного обеспечения графодинамической ассоциативной машины.

7. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Практикум предназначен для изучения и практического освоения принципов, прикладных методик, базовых алгоритмов и технологий, используемых при разработке программных моделей устройств с адресацией данных по содержанию (ассоциативных устройств).

Практикум включает работы:

– по построению и проверке модели, обеспечивающей программные методы реализации адресации данных по содержанию при помощи таблиц хеширования (лабораторная работа №1);

– по построению и проверке программных моделей, обеспечивающих выполнение базовых поисковых и вычислительных операций ассоциативной памяти и ассоциативного процессора (лабораторные работы №2 – №4).

Каждая лабораторная работа является самостоятельной и включает цель работы, краткие теоретические сведения, необходимые для выполнения работы (более полно они даны в разд. 1–6), и индивидуальные задания.

При разработке программных моделей могут быть полезны рекомендации, приведенные в лабораторных практикумах [11,12].

7.1. Лабораторная работа №1

Тема: Построение и проверка таблиц хеширования

Цель работы: освоение навыков построения и проверки таблиц хеширования.

Краткие теоретические сведения

Назначение и применение таблиц хеширования

Таблицы хеширования применяются для обеспечения программного поиска информации по содержанию специального ключевого слова или фрагмента (части) самих данных.

Основные понятия

Для формирования таблиц хеширования используются следующие компоненты и понятия:

ID – ключевое слово (любое имя, слово, сочетание любых символов и т.д.) или идентификатор;

V – числовое значение ключевого слова;

h (V) – хеш-адрес, вычисленный по числовому значению V ключевого слова;

B – начальный адрес таблицы;

H – общее количество ячеек памяти в таблице.

Напомним, что при совпадении хеш-адресов, вычисленных по разным ключевым словам, возникает коллизия (конфликт). В этом случае для размещения второй и последующих записей необходимо использовать резервные ячейки памяти, которые размещаются либо в самой таблице хеширования (внутренняя адресация), либо в области переполнения.

Для поиска свободных резервных ячеек при внутренней адресации применяется процедура пробинга (линейного, квадратичного или случайного).

Структура ячейки (строки) хеш-таблицы

В состав ячейки (строки) хеш-таблицы должны входить:

– ключевое слово (ID) или его идентификатор;

– соответствующие этому ключу данные или указатели;

– различные флажки (признаки).

Формат ячейки (строки) хеш-таблицы приведен на рис. 7.1.

ID	C	U	T	L	D	Po	Данные или указатели (Pi)

Рис. 7.1. Формат ячейки памяти хеш-таблицы

Обозначения на рис. 7.1 [3]:

ID – идентификатор ключевого слова или само ключевое слово (К),

C – флажок коллизий,

U – флажок «занято»,

T – терминальный символ,

L – флажок связи,

D – флажок вычеркивания,

Po – указатель области переполнения (или следующей записи в цепочке),

Pi – указатель области данных

Индивидуальные задания

Вариант 1

Построить хеш-таблицу в соответствии с приведенными ниже исходными данными.

Справочные данные о группе или подгруппе (не менее 10 человек):

1) фамилия, инициалы (или только имя);

2) любые данные (например просто буквы: ааа....или ввв…).

Ключевое слово – фамилия студента, имя или инициалы.

Способ фрмирования вычисленного значения ключевого слова (V) – по первым двум буквам фамилии, имени или по инициалам.

Может быть применен следующий способ формирования вычисленного значения адреса **V** ключевого слова:

– используется русский алфавит;

– буквам присваиваются соответствующие номера:

А = 0, Б = 1, ... Я=32, т.е. всего 33 номера. Будем эту величину считать основанием позиционной системы счисления, тогда сочетание двух первых букв фамилии, имени или инициалы можно записать, например, следующим образом:

а) Вяткин $V [Вя] = 2 \cdot 33^1 + 32 \cdot 33^0 = 98$

б) Третьяк $V [Тр] = 19 \cdot 33^1 + 15 \cdot 33^0 = 644.$

Аналогично может быть использован латинский алфавит.

Структура ячейки (строки) хеш-таблицы и ее размер определяется в соответствии с исходными данными и форматом ячейки (см. рис. 7.1).

Способ формирования хеш-адреса определяется размером ячейки (строки) и вычисленным значением V (например, $h(V) = V \cdot \text{mod } H + B$, где V·modH – остаток от деления V на H).

Способы обработки коллизий:

– размещение резервных ячеек в хеш-таблице (внутренняя адресация);

– использование области переполнения.

Способ (вид) пробинга при выборе резервных ячеек – линейный (или любой другой).

Разработанная программа должна уметь выполнять следующие функции:

– определение числовых значений ключевых слов (V);

– формирование хеш-адресов [h (v)];

– формирование самой хеш-таблицы;

– обработку коллизии (то есть обеспечение нахождения резервных ячеек и запись в них);

– выборку из хеш-таблицы необходимой информации по соответствующему ключевому слову;

– занесение в хеш- таблицу новых записей по новым ключевым словам,

– удаление из хеш-таблицы отдельных записей/ячеек.

Варианты хеш-таблицы приведены в табл. 7.1.

Таблица 7.1

Варианты хеш-таблицы

№ варианта	H, ячеек	L	Способ обработки коллизий	Вид пробинга	Примечания
1	50–100	0	Внутренняя адресация	Линейный	Количество записей в таблице (т.е. занятых ячеек) – не менее 10
2	50–100	1	Внутренняя адресация	Линейный	Количество коллизий – не менее 3-х
3	50–100	0	Использование области переполнения	–	Длина цепочек резервных ячеек (при коллизии)–2–4
4	100	1	Использование области переполнения	–	

Программа должна обеспечивать индикацию (вывод на экран) содержимого всех полей и указателей ячейки хеш-таблицы, вычисленных значений V и h(V) для каждой записи, коэффициента заполнения хеш-таблицы.

Вариант 2

Построить индексную хеш-таблицу разреженной двоичной матрицы в соответствии с приведенными ниже исходными данными.

Сформировать разреженную двоичную матрицу при помощи программы генерации случайных чисел. Размер матрицы (p x g), количество нулей в матрице должно быть больше количества единиц.

В качестве ключевого слова (ID) использовать пару индексов (i, j), где

i= (1, 2, ... p) – номер столбца,

j= (1, 2, ... g) – номер строки.

Для формирования вычисленного значения ключевого слова v использовать формулу

$$v = j + p (i – 1) – 1.$$

Структура ячейки хеш-таблицы и ее размер определяются в соотвествии с исходными данными и форматом ячейки.

Способ формирования хеш-адреса определяется размером ячейки памяти и вычисленным значением v при помощи операции деления

$$H(v) = v \bmod H + B,$$

где $v \cdot \bmod H$ – остаток от деления v на H.

Способ обработки коллизий – размещение резервных ячеек в ХЕШ-таблице (внутренняя адресация).

Разработанная программа должна уметь выполнять следующие функции:

– формирование разреженной матрицы размером p x g;

– определение v для элементов матрицы (по паре индексов (i, j));

– определение хеш-адресов [h (v)];

– формирование хеш-таблицы для ненулевых элементов матрицы;

– обработку коллизий (т. е. нахождение резервных ячеек и запись в них);

– выборку из хеш-таблицы необходимых ячеек по заданному ключевому слову ID = (i, j).

Для проверки правильности выполнения всех вычислений необходимо обеспечить обратное преобразование индексной хеш-таблицы в исходную разреженную матрицу и сравнить их объемы.

Дополнительное задание к варианту 2

1. Определить, при каком соотношении p и g (p = g, p > g или p < g) возможны коллизии.

2. Определить, при каких условиях хеш-таблица ненулевых элементов разреженной матрицы занимает меньший объем памяти, чем запись этой таблицы в нормальном (обычном) виде.

7.2. Лабораторная работа №2

Тема: Моделирование алгоритмов базовых операций ассоциативного процессора

Цель работы: освоение навыков построения и верификации моделей алгоритмов базовых операций ассоциативного процессора над разрядными срезами.

Краткие теоретические сведения

Будем рассматривать ассоциативные процессоры с пословной организацией, т.е. с параллелизмом на уровне слов и обработкой их последовательно по разрядам. Множество слов образует ассоциативный массив или ассоциативное запоминающее устройство (АЗУ) ассоциативно организованного процессора (АП). Соответственно имеется по одному процессорному элементу на каждое слово, так как весь разрядный срез АЗУ может обрабатываться параллельно.

Структура АП

Базовая структура пословно организованного АП содержит обычно следующие подсистемы:

– массив ассоциативной памяти;

– регистр поискового признака;

– регистр маски;

– регистры хранения ответов;

– регистр или буфер ввода-вывода АЗУ (R);

– регистр метки выбора слов;

– контроллер (память и программы).

В данной лабораторной работе используются только следующие подсистемы базовой структуры АП, показанные на рис. 7.2:

– массив ассоциативной памяти (АМ);

– буферный регистр ввода-вывода (R);

– регистр меток выбора слов (T).

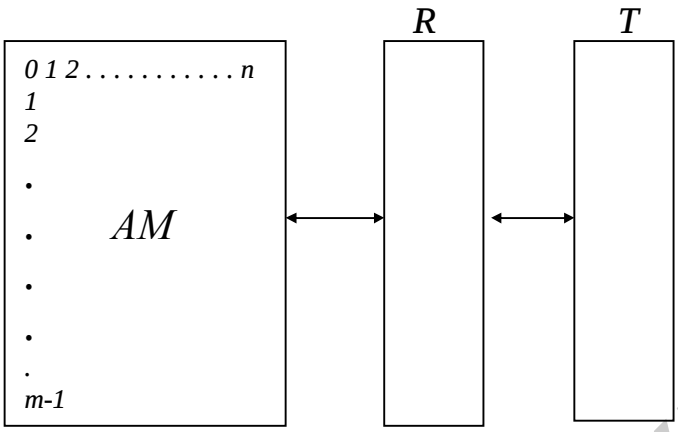


Рис. 7.2. Блоки базовой структуры АП, используемые в лабораторной работе №2

Отметим, что регистры *R* и *T* представляют собой множество триггеров, образующее одноразрядный двоичный вектор вдоль всего массива АЗУ.

Рассмотрим алгоритмы выполнения базовых операций АП.

Отметим, что разрядный срез является основной конструкцией в пословно организованных АП. Существуют операции считывания и записи разрядного среза, пересылки содержимого одного разрядного среза и т.д.

Для описания отдельных операций АП применим синтаксис одной из систем (в частности системы *LUCAS*).

Рассмотрим основные базовые операции АП.

Загрузить разрядный срез (*Load bit slice*)

Эта операция пересылает содержимое некоторого разрядного среза АЗУ в один из регистров *R* или *T*.

Введем следующие обозначения:

R_i – триггер регистра выбора – вывода;

T_i – триггер регистра меток выбора слов;

S_i – позиция источника;

D_i – позиция назначения.

Возможны 3 варианта операции "Загрузить разрядный срез":

а) *LOAD R* – Загрузить разрядный срез в регистр *R*;

б) *LOAD R* [*T*] – Загрузить в *R* отмеченные метками *T* биты разрядного среза;

в) *LOAD T [T]* – Загрузить в *T* биты разрядного среза, отмеченные прежними значениями меток *T*.

Пример реализации варианта «б» показан на рис. 7.3.

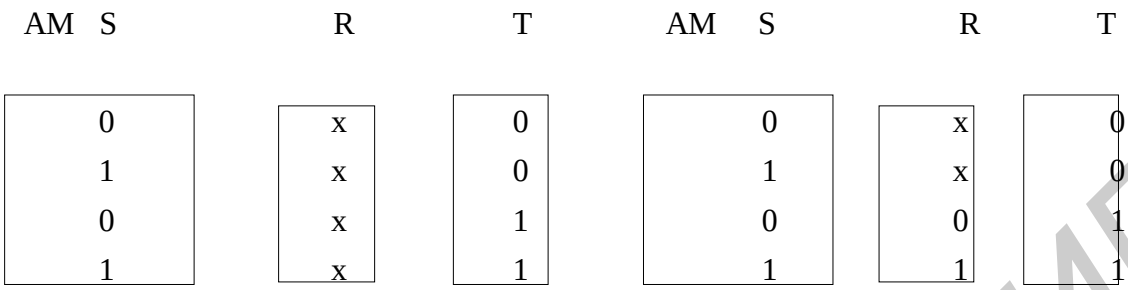


Рис. 7.3. Реализация варианта «б» операции «Загрузить разрядный срез»

Запомнить разрядный срез (*Store bit slice*)

Эта операция противоположна операции загрузки.

Переслать разрядный срез (Move bit slice)

Эта операция пересылает содержимое одного разрядного среза в позицию другого. При этом используются триггеры памяти ответов, т.е. сначала выполняется операция "Загрузить", а затем – "Запомнить".

Логические операции с разрядными срезами

В этом случае два разрядных среза подвергаются логической операции и результат помещается в позицию (адрес) некоторого третьего среза. При этом также используются триггеры памяти ответов.

Для любой операции первый из операндов загружается в какой-нибудь регистр (например *R*), заданная логическая операция выполняется между вторым операндом (т.е. соответствующим разрядным срезом) и регистром *R*, после чего результат, заменивший прежнее содержимое регистра *R*, пересылается из этого регистра в позицию результирующего разрядного среза. Логической операцией может быть любая булева функция двух переменных.

Пример выполнения операции *OR* (*ИЛИ*) приведен на рис. 7.4.

AM	S_1	S_2	D	T
	0	1	x	1
	1	0	x	1
	1	1	x	0
	0	0	x	1
	1	0	x	1

До операции

AM	S_1	S_2	D	T
	0	1	1	1
	1	0	1	1
	1	1	x	0
	0	0	0	1
	1	0	1	1

После выполнения операции

Рис. 7.4. Пример выполнения операции *OR* (*ИЛИ*)

Отметим, что к базовым операциям АП с пословной организацией относятся также следующие операции, рассмотренные в [6]:

– выбрать первого ответчика и сбросить (*Select first and reset*);

– перации с байтами (*Store I/O, Load I/O* и др.), аналогичные операциям с битовыми срезами;

– операции с полями (*Move Field, Load Field* и др.), осуществляющие пересылку полей, находящихся внутри слов.

Эти операции в данной лабораторной работе не используются.

Индивидуальное задание

Построить и проверить модели алгоритмов базовых операций с разрядными срезами АП.

Разработанная программа должна уметь:

– строить ассоциативный массив (*AM*) двоичных чисел размерностью *m* слов на *n* разрядов;

– загружать любые разрядные срезы АМ в регистры (*R* или *T*);

– запоминать и пересылать любые разрядные среды АМ (т. е. выполнять команды *Store bit slice* и *Move bit slice*);

– выполнять любые логические операции (табл.7.2) над любыми разрядными срезами (*Si* и *Sj*) *AM*, при этом адресом (номером) назначения (*D*) может быть любой срез *AM* или регистр *R*.

Таблица 7.2

Логические функции 2-х переменных

Логические аргументы	Значение истинности				Наименование функции	Запись функции в нотации основного функционально-полного набора (И, ИЛИ, НЕ)
x_1	0	0	1	1		
x_2	0	1	0	1		
f_0	0	0	0	0	Константа 0	$f_0 = 0$
f_1	0	0	0	1	Конъюнкция (И)	$f_1 = x_1 \cdot x_2$
f_2	0	0	1	0	Запрет 1-го аргумента (НЕТ)	$f_2 = x_1 \cdot \overline{x_2}$
f_3	0	0	1	1	Повторение 1-го аргумента (ДА)	$f_3 = x_1$
f_4	0	1	0	0	Запрет 2-го аргумента (НЕТ)	$f_4 = \overline{x_1} \cdot x_2$
f_5	0	1	0	1	Повторение 2-го аргумента (ДА)	$f_5 = x_2$
f_6	0	1	1	0	Неравнозначность (ИЛИ-ИЛИ) Дизъюнкция (ИЛИ)	$f_6 = \overline{x_1} \cdot x_2 + x_1 \cdot \overline{x_2}$
f_7	0	1	1	1	Операция Пирса (ИЛИ-НЕ)	$f_7 = x_1 + x_2$
f_8	1	0	0	0	Эквивалентность (И-И)	$f_8 = \overline{x_1 + x_2}$
f_9	1	0	0	1	Отрицание 2-го аргумента (НЕ)	$f_9 = x_1 \cdot x_2 + \overline{x_1} \cdot \overline{x_2}$
f_{10}	1	0	1	0	Импликация от 2-го аргумента к 1-му (НЕТ-НЕ)	$f_{10} = \overline{x_2}$
f_{11}	1	0	1	1	Отрицание 1-го аргумента (НЕ)	$f_{11} = x_1 + \overline{x_2}$
f_{12}	1	1	0	0	Импликация от 1-го аргумента ко 2-му (НЕТ-НЕ) Операция Шеффера (И-НЕ)	$f_{12} = \overline{x_1}$
f_{13}	1	1	0	1	Константа 1	$f_{13} = \overline{x_1} + x_2$
f_{14}	1	1	1	0		$f_{14} = \overline{x_1 \cdot x_2}$
f_{15}	1	1	1	1		$f_{15} = 1$

7.3. Лабораторная работа №3

Тема: Моделирование ассоциативного процессора с применением последовательных (рекуррентных) алгоритмов

Цель работы: освоение навыков построения и верификации (проверки) моделей ассоциативного процессора с применением рекуррентных алгоритмов.

Краткие теоретические сведения

Будем далее рассматривать ассоциативные процессоры с пословной организацией, т.е. с параллелизмом на уровне слов и с обработкой их последовательно по разрядам.

Множество слов образует ассоциативный массив или АЗУ ассоциативно организованного процессора, структура которого приведена на рис. 7.5. Соответственно имеются логические цепи на каждое слово, т.к. весь разрядный срез АЗУ может обрабатываться параллельно.

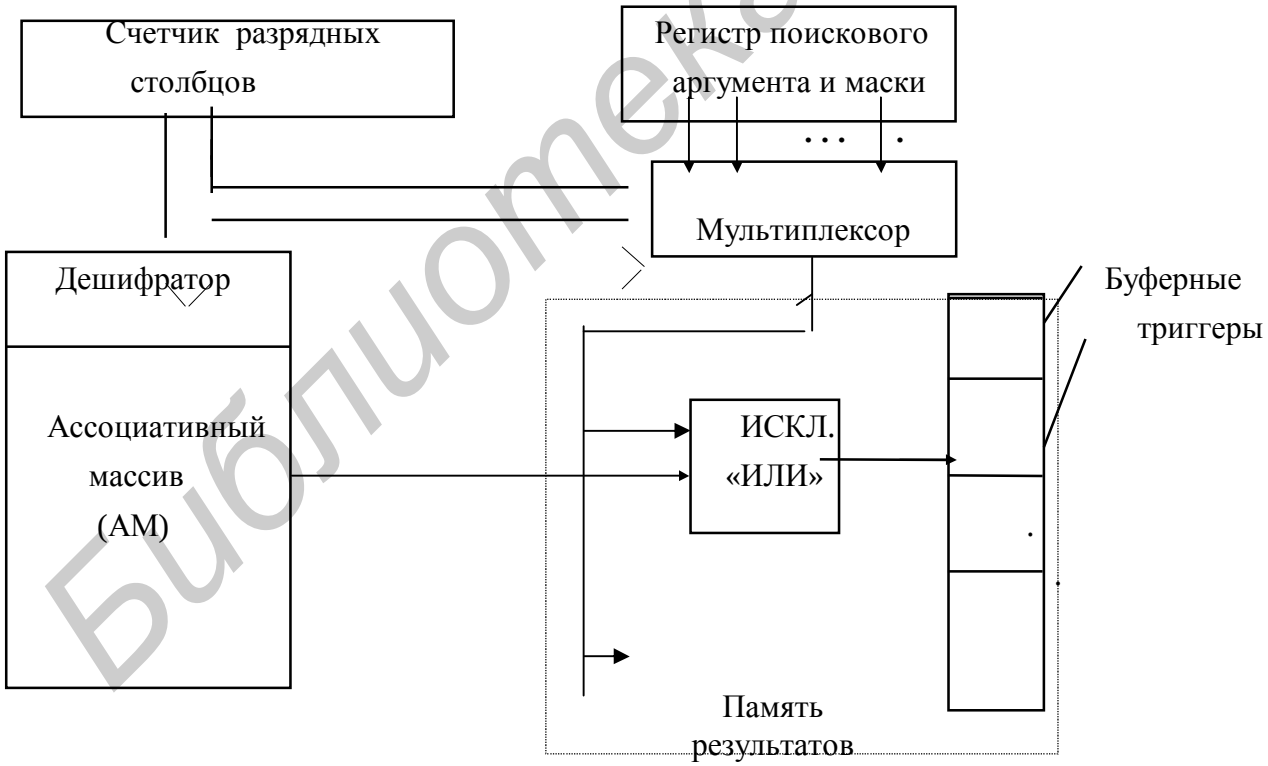


Рис. 7.5. Структура ассоциативного процессора

В АЗУ, параллельных по словам и последовательных по разрядам, для обработки данных используются последовательные (рекуррентные) алгоритмы. Это позволяет наряду с элементарными операциями сравнения на равенство применять и более сложные логические и вычислительные алгоритмы.

Отметим, что АЗУ может функционировать в двух основных режимах: поисковом и вычислительном.

В режиме поиска обычно требуется локализовать и считать из памяти слова, подчиняющиеся определенным условиям. Например, это могут быть:

– слова, равные, большие или меньшие некоторого аргумента;

– слова, лежащие в заданном диапазоне чисел;

– слова, обладающие максимальной или минимальной величиной среди чисел, хранящихся в памяти, и др.

В процессе поиска содержимое памяти, как правило, не меняется.

В режиме ассоциативных вычислений слова, наоборот, подвергаются обработке, а полученные результаты вновь поступают в массив АЗУ, причем часто они заменяют ранее хранившиеся там слова (или определенные фрагменты этих слов).

Возможность работы АЗУ в различных режимах обеспечивается благодаря специальной конструкции памяти результатов, которая состоит из логических цепей последовательного типа (по одной на каждое слово АЗУ).

Рассмотрим работу памяти результатов для проведения операций сравнения.

Алгоритмы для варианта сравнения, начиная со старших разрядов, описывается следующими рекуррентными соотношениями:

$$g_{ji}=g_{j,i+1}\acute{U}(\bar{a}_i\grave{U}S_{ji}\grave{U}\bar{l}_{j,i+1}),$$

$$l_{ji} = l_{j,\,i+1}\,\acute{U}\;\;(a_i\,\grave{U}\,\hat{S}_{ji}\,\grave{U}\hat{g}_{j,i+1}), \qquad (7.1)$$

где g_{ji}, $l_{j,i}$ – двоичные переменные, вычисляемые в процессе выполнения поразрядного сравнения;

a_i – i-й разряд аргумента поиска;

$S_{j\,i}$ – i-й разряд j-го слова, хранящегося в массиве памяти.

Эти алгоритмы являются удобными для реализации при помощи цепей последовательного типа.

Будем рассматривать алгоритм, выполняющий сравнение разрядов, начиная со старшего разряда n. В качестве начальных значений задаются

$$g_{j,\,n+1} = l_{j,\,n+1} = 0.$$

Последовательные цепи памяти результатов могут строиться либо по асинхронному, либо по синхронному принципу. Будем применять синхронные схемы, так как они более просты в управлении, имеют более высокую помехозащищенность. В качестве элементов памяти используем наиболее простые триггеры с синхронизацией – D-типа или другого типа (рис. 7.6).

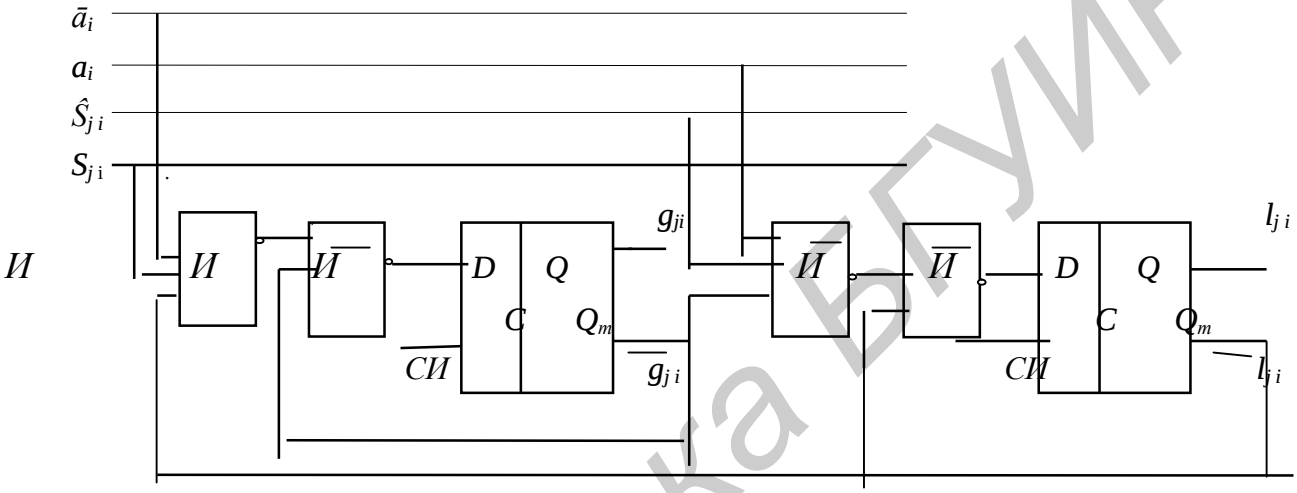


Рис. 7.6. Ячейки промежуточных результатов, используемые при сравнении величин

На схеме, обслуживающей слово j, имеются два триггера – g_j и l_j, которые будем называть ячейками промежуточных результатов. Логические значения выходов этих ячеек, как и переменные в выражении (7.1), обозначим g_{ji} и l_{ji} соответственно (на схеме имеются также вспомогательные вентили (*И*)).

При сравнении содержимого заданного слова памяти с аргументом поиска значения их одноименных разрядов a_i и $S_{j\,i}$ последовательно и синхронно передаются на входы ячеек промежуточных результатов. В каждом такте D-триггеры переходят в новые состояния, которые соответствуют значениям логических переменных g_{ji} и l_{ji} в выражении (7.1).

Окончательный результат сравнения определяется совокупностью логических значений, зафиксированных на выходах триггеров после завершения поразрядного анализа содержимого слова памяти и поискового аргумента.

Если $g_{j\,0} = l_{j0} = 0$, то $S_j = A$,

$g_{j\,0} = 1$, *а* $l_{j\,0} = 0$, то $S_j > A$,

$g_{j0} = 0$, *а* $l_{j\,0} = 1$, то $S_j < A$.

Отметим, что состояние $g_{j\,0} = l_{j\,0} = 1$ принципиально невозможно.

Сигналы с выходов памяти результатов подаются на приоритетный анализатор, при помощи которого осуществляется считывание слов, удовлетворяющих заданным соотношениям.

Основные типы операций поиска

Выше были кратко перечислены основные типы операций поиска. Рассмотрим их несколько подробнее.

Поиск величин, заключенных в заданном интервале

Для реализации операций граничного поиска в каждой логической цепи памяти результатов, кроме триггеров *g* и *l*, необходимо иметь еще один вспомогательный триггер. При помощи этих триггеров, называемых далее флажками результата, фиксируются слова, входящие одновременно в два набора, каждый из которых составлен из элементов, удовлетворяющих одному частному ограничению.

Перед началом поиска все флажки результата устанавливаются в «1». На каждом промежуточном этапе флажки, соответствующие словам, для которых частное условие поиска не выполняется, сбрасываются в «0». Таким образом, по завершении операции поиска единицы остаются только в тех флажковых триггерах, которые связаны со словами, удовлетворяющими всем ограничениям.

Поиск величин, заключенных в заданных границах, проводится в 2 этапа:

1) отыскиваются все числа, которые меньше верхней границы;

2) отыскиваются те числа, которые больше нижней границы.

После окончания поиска флажки результата отмечают только те величины, которые находятся в требуемом интервале.

Поиск максимального (минимального) значения

Для этого поиска внешний аргумент поиска не нужен. Вместо него разряды регистра аргумента, начиная со старшего, последовательно устанавливаются в значение «1». После установки в «1» очередного разряда производится опера-

ция маскированного поиска, в ходе которого отбираются все слова, содержащие "1" в соответствующей позиции.

Если на каком-то шаге ни одного такого слова не обнаружено (т.е. во всех словах памяти в этом разряде стоят «0»), содержимое памяти результатов не меняется. После каждого шага количество слов, отмеченных единичными флажками результатов, уменьшается. Процесс продолжается до тех пор, пока не будут опрошены все разряды массива АЗУ. Если в памяти записано несколько максимальных чисел, равных друг другу, все они будут зафиксированы в памяти результатов.

Очевидно, что поиск минимального значения в целом выполняется аналогично. Он также начинается со старших разрядов, но на каждом шаге выявляются слова, содержащие в соответствующих позициях не «1», а «0».

Поиск ближайшего снизу (сверху) значения

Эта операция состоит из 2-х этапов:

1) выявляются все числа, меньшие указанной величины;

2) среди выявленных на 1-м этапе чисел находится максимальное число.

Если необходимо отыскать числа, ближайшие сверху к заданной величине, производится поиск чисел, превосходящих последнюю, и среди них отбираются минимальные.

Упорядоченная выборка (сортировка)

Эта операция заключается в пошаговом поиске максимального или минимального значения в наборе чисел, отсеянных после прохождения предыдущего этапа сортировки. Эту операцию можно реализовать в виде последовательности рассмотренных выше операций.

Поиск на основе булевых функций

Введя в состав АЗУ соответствующие логические цепи, можно организовать поиск, критерием в котором будет выступать некоторая булева функция. Поиск такого типа применяется при локализации слов, для которых логическая функция, определенная на части разрядов, при подстановке в нее соответствующих битов аргумента принимает заданные значения. Чаще используются такие функции, как «Исключающее ИЛИ», «Логическая эквивалентность», ИЛИ, И.

Поиск по соответствию

В некоторых случаях требуется найти записи, двоичные представления которых содержат максимальное количество разрядов, совпадающих с разрядами поискового аргумента. Для решения этой задачи в памяти результатов АЗУ необходимо иметь набор счетчиков, при помощи которых осуществляется подсчет совпадающих значений одноименных разрядов в каждом слове памяти. Если указанные счетчики, в свою очередь, имеют средства адресации по содержанию, в них можно привести поиск максимального значения и определить таким образом слова, наиболее схожие с поисковым аргументом.

Вычислительные операции ассоциативного процессора

К вычислительным операциям ассоциативного процессора можно отнести арифметические и логические операции над разрядными срезами, часть из которых была рассмотрена в лабораторной работе №2, а также над байтами, полями, словами, которые будут рассмотрены в работе №4.

Индивидуальное задание

Построить и проверить программную модель ассоциативного процессора с применением последовательных (рекуррентных) алгоритмов.

Модель должна обеспечивать выполнение поисковых и вычислительных операций ассоциативного процессора, рассмотренных в данном учебно-методическом пособии. Варианты операций приведены в табл. 7.3.

Таблица 7.3

Поисковые и вычислительные операции ассоциативного процессора

Вариант	Поисковые операции	Вычислительные операции
1	Поиск величин, заключенных в заданном интервале	Арифметические или логические операции над заданными разрядными срезами
2	Поиск максимального (минимального) значения	– "–

3	Поиск ближайшего сверху (снизу) значения	– " –
4	Упорядоченная выборка (сортировка)	– " –
5	Поиск по основе булевых функций	– " –
6	Поиск по соответствию	– " –

7.4. Лабораторная работа №4

Тема: Моделирование ассоциативной памяти с системой адресации по разрядным столбцам и по словам

Цель работы: освоение навыков построения и верификации модели ассоциативной памяти, обеспечивающей адресное считывание и запись по разрядным столбцам и по словам и выполнение арифметических операций над полями слов.

Краткие теоретические сведения

Используемая в лабораторных работах 2 и 3 базовая модель ассоциативного процессора имеет ряд недостатков , в частности:

а) запись данных в ассоциативный массив (АМ) приходится производить по разрядным столбцам, из-за чего все содержимое массива должно быть предварительно занесено в некоторый буфер;

б) большие трудности вызывает также модификация данных (изменение лишь одного слова требует перезаписи содержимого всего массива);

в) в рамках базовой конфигурации АМ практически невозможно осуществить считывание слов по заданному адресу (эта операция необходима для того, чтобы просмотреть целиком все слово, определенная часть которого совпала с аргументом поиска).

Рассмотрим метод адресации АМ, позволяющий избежать отмеченные выше недостатки.

В рассматриваемом варианте организации ассоциативной памяти (АП) применяются специальные методы формирования содержимого массивов и введена дополнительная логика, что обеспечивает адресное считывание и запись как по разрядным столбцам, так и по словам.

Типовой модуль памяти такого типа приведен на рис. 7.7.

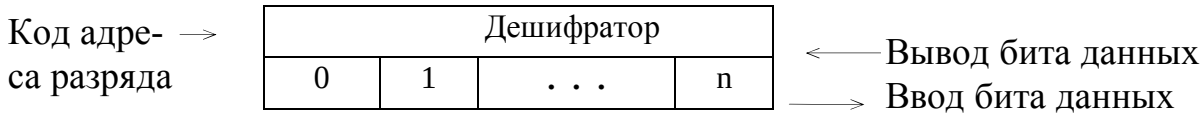


Рис. 7.7. Типовой модуль ассоциативной памяти

Типовой стандартный модуль ассоциативной памяти представляет собой массив размером 1 x m, где m –количество слов (в частности, можно использовать серийные интегральные схемы памяти 1 x N битов).

В рассматриваемом варианте массив памяти считается квадратным (например 256 x 256 разрядов). Если требуется массив большего объема, объединяются несколько таких массивов. Как видно из рис.7.7, в ассоциативной памяти такого типа используется больше аппаратуры, чем в базовой конструкции, рассмотренной ранее. В их состав входят, в частности, m дешифраторов, причем на каждый из них подается свой адресный код, вычисляемый при помощи соответствующих функциональных цепей.

Применяемый в таких ассоциативных памятях принцип формирования адресов получил название диагональной адресации. Это название связано с тем, что элементы исходного массива данных перед записью в память перераспределяются следующим образом:

– разрядные столбцы размещаются по диагоналям, образованным запоминающими ячейками;

– слова размещаются по столбцам.

Прмер размещения массива данных размером 256 x 256 разрядов приведен на рис.7.8.

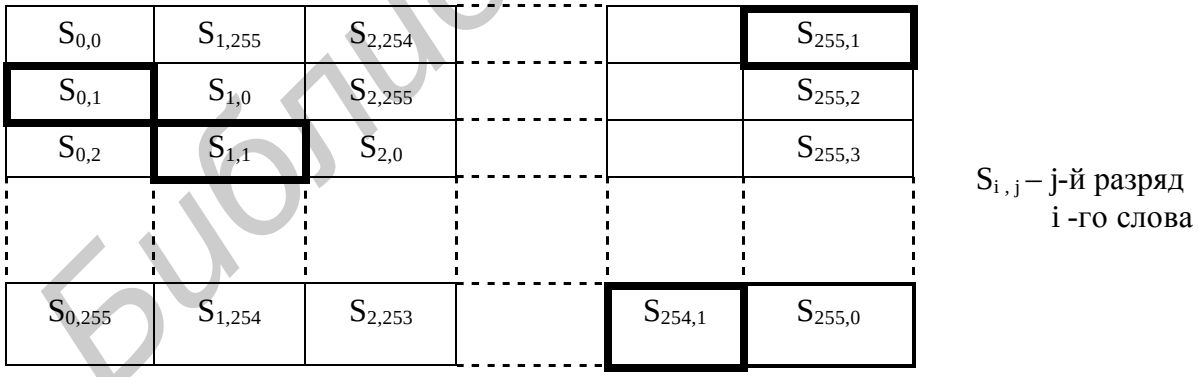


Рис. 7.8. Расположение данных в ассоциативной памяти с диагональной адресацией

Рассмотрим, как в памяти с диагональной системой адресации выполняются операции считывания и записи слов и разрядных столбцов на примере применения диагональных сумматоров.

Схема памяти с диагональной адресацией на основе сумматора приведена на рис.7.9.

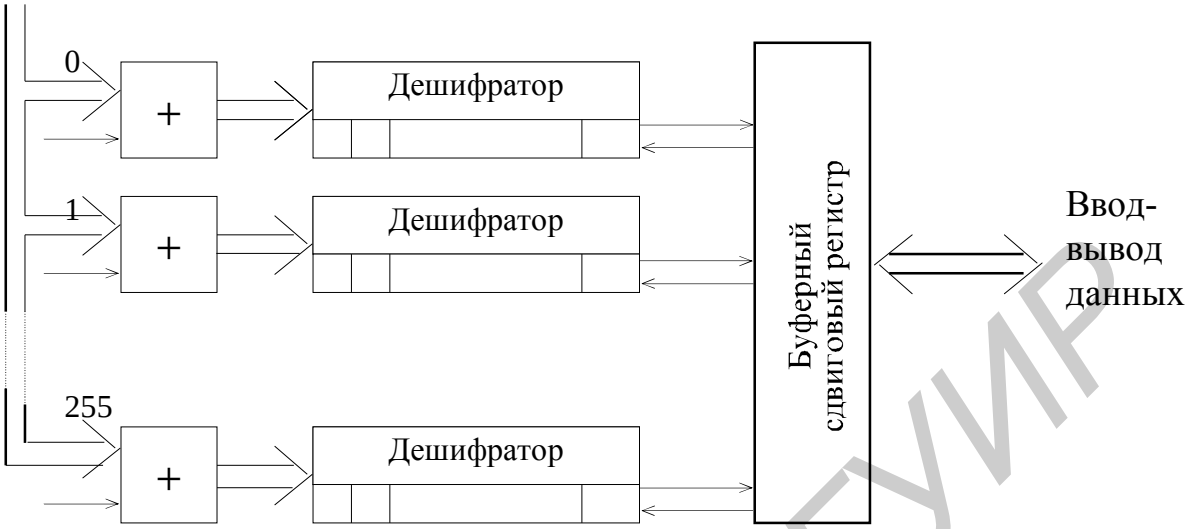


Рис. 7.9. Схема памяти с диагональной адресацией

В этой памяти каждый дешифратор имеет блок арифметического сложения с полным сумматором, в котором формируется сумма адресного кода, заданного извне, и постоянного схемно-реализуемого числа, равного номеру строки.

Применяя соответствующие средства управления, части сумматоров можно блокировать (маскировать) и в результате на их выходах получить требуемый фрагмент адреса разрядного столбца.

Данные, которые необходимо записать в память или считать из нее, помещаются в специальный буферный регистр. Содержимое этого регистра можно циклически сдвигать вверх или вниз на заданное количество разрядов. Направление сдвига зависит от выполняемой операции.

Выполнение операций записи – считывания

Запись в память разрядного столбца

Адрес разрядного столбца, представленный в дополнительном коде, подается на вход сумматора. Логические цепи последнего деблокируются и производится суммирование (поданного адреса и номера строки), а данные, подлежащие записи в память, передаются в буферный регистр. Производится

сдвиг содержимого буфера вниз, причем количество сдвигов определяется адресом столбца.

Далее выдается команда записи и элементы преобразованного разрядного столбца записываются в соответствующие ячейки памяти, определяемые кодами, полученными на выходах соответствующих сумматоров. В частности, первый адресный столбец записывается в ячейки, выделенные на рис. 7.8 жирными линиями.

Считывание разрядного столбца

Для считывания разрядного столбца на вход сумматора также подается его адрес в дополнительном коде. В сумматорах формируется результирующий адресный код.

По команде считывания разрядный столбец с переставленными (диагонально) элементами пересылается в буферный регистр. Для восстановления прежнего порядка элементов столбец сдвигается циклически вверх на то же количество разрядов, что и при записи.

Запись слова в память

Запись слова осуществляется следующим образом:

– адрес слова подается на входы сумматоров, где производится суммирование кодов адреса с нулем (т.е. для всех строк формируется одинаковый адрес – адрес столбца);

– содержимое слова пересылается в буферный регистр памяти и сдвигается вниз на количество разрядов, задаваемое адресом слова (столбца);

– по команде записи данные из буферного регистра передаются в ячейки памяти, причем так, что i-е слово оказывается в i-м столбце массива памяти.

Считывание слова из памяти

Считывание слова из ассоциативной памяти происходит в той же последовательности, что и запись:

– адрес слова подается на входы сумматора (и складывается с нулем);

– содержимое адресуемого столбца пересылается в буферный регистр, где производится сдвиг вверх на количество разрядов, задаваемое адресом слова.

Индивидуальное задание

Построить и проверить программную модель ассоциативной памяти с диагональной адресацией на основе сумматора.

Размер массива памяти не более 16х16 двоичных разрядов (битов).

Разработанная программная модель должна уметь выполнять операции считывания и записи любых задаваемых разрядных столбцов и слов, а также выполнять поисковые логические и арифметические операции над любыми столбцами и словами.

Модель должна уметь выполнять поисковые операции, аналогичные операциям, рассматриваемым в лабораторной работе №3 (поиск по совпадению, поиск в заданных пределах, поиск максимального значения, поиск ближайшего сверху и снизу, упорядоченная выборка, поиск по основе булевых функций и поиск по соответствию) с использованием рекуррентных алгоритмов.

Модель должна уметь выполнять также вычислительные операции типа сложения полей слова. Эта операция может быть организована, например, следующим образом.

Каждое слово массива ассоциативной памяти состоит из 4-х полей (общая длина слова – 16 разрядов), содержащих переменные *Vj*, *Aj*, *Bj* и *Sj*, где *j* – номер слова внутри массива памяти (размерность полей $Vj = 3, Aj = Bj = 4$ и $Sj = 5$).

Задачей является формирование суммы полей $Aj + Bj$ и запись результата в поле *Sj*. Указанные действия производятся лишь для тех слов, у которых значение переменной *Vj* совпадает с содержимым соответствующего поля аргумента поиска (остальные поля аргумента поиска при этом должны быть замаскированы).

Сложение начинается с младших разрядов полей *Aj* и *Bj*. Для запоминания значения поразрядной суммы и переноса в памяти результатов необходимо иметь дополнительные регистры.

Методические указания к лабораторной работе №4

При выполнении данной лабораторной работы рекомендуется:

1) вначале построить (сформировать) исходный двоичный массив размером не более 16 х 16 двоичных разрядов в обычном виде;

2) затем преобразовать его в массив с диагональной адресацией и сравнить соответствие столбцов и слов;

3) при выполнении операций записи разрядных столбцов и слов в массив с диагональной адресацией использовать любые (задаваемые «вручную») столбцы и слова исходного массива;

4) при выполнении поисковых операций использовать модель, выполненную в лабораторной работе №3 «Моделирование ассоциативного процессора с применением последовательных (рекуррентных) алгоритмов»;

5) для выполнения вычислительных операций над полями Aj и Bj слова память результатов необходимо дополнить (доработать) средствами, обеспечивающими выполнение операций поразрядного сложения с учетом сигналов переноса из младшего разряда.

ЛИТЕРАТУРА

1. Дейт, К. Дж. Введение в системы данных / К. Дж. Дейт ; пер. с англ. – 6-е изд. – Киев, 1998.

2. Корнеев, В. В. Параллельные вычислительные системы / В. В. Корнеев. – М. : Нолидж, 1999.

3. Кохонен, Т. Ассоциативные запоминающие устройства / Т. Кохонен ; пер. с англ. – М. : Мир, 1982.

4. Майерс, Г. Архитектура современных ЭВМ / Г. Майерс ; пер. с англ., под ред. В. К. Потоцкого. – М. : Мир, 1985.

5. Огнев, И. В. Ассоциативные среды / И. В. Огнев, В. В. Борисов. – М. : Радио и связь, 2000.

6. Озкарахан, Э. Машины баз данных и управление базами данных / Э. Озкарахан ; пер. с англ. – М. : Мир, 1989.

7. Тербер, К. Дж. Архитектура высокопроизводительных вычислительных систем / К. Дж. Тербер ; пер. с англ. – М. : Наука, 1985.

8. Фостер, К. Ассоциативные параллельные процессоры / К. Фостер ; пер. с англ. – М. : Энергоиздат, 1981.

9. Качков, В. П. Организация и функционирование традиционных и интеллектуальных компьютеров : учеб. пособие / В. П. Качков, И. Я. Доморадов; под науч. ред. проф. В. В. Голенкова. – Минск : БГУИР, 2006.

10. Шпаковский, Г. И. Параллельные процессоры для цифровой обработки сигналов и метаданных / Г. И. Шпаковский. – Минск : БГУ, 2000.

11. Гапонов, П. А. Лабораторный практикум по курсу «Конструирование программ и языки программирования» / П. А. Гапонов, В. М. Кузьмицкий. – Минск : БГУИР, 1998.

12. Приходько, Ю. Г. Лабораторный практикум по курсу «Основы алгоритмизации и программирования» / Ю. Г. Приходько, С. А. Самодумкин, О. Е. Елисеева. – Минск : БГУИР, 2000.

13. Программирование в ассоциативных машинах: монография / В. В. Голенков [и др.]. – Минск : БГУИР, 2001.

14. Сименс, Дж. ЭВМ пятого поколения. Компьютеры 90-х годов / Дж. Сименс. – М. : Финансы и статистика, 1985.

15. Учебник по программированию на языке Pascal 7.0. Занятие III.5. [Электронный ресурс]. – 2007. – Режим доступа:

http://www.59323.ocpi.ru/Pascal/Part3/edu5/edu5_5.htm.

16. Структуры и алгоритмы. Хеширование. [Электронный ресурс]. – 2005.– Режим доступа:
http://www.structur.h1.ru/hash.htm.

17. Шень, А. Программирование: теоремы и задачи / А. Шень. – М. : МЦНМО, 2004.

18 Богатырев, Р. Расстановка, или схемы хеширования / Р. Богатырев, А. Шилов // Мир ПК, 2001, №6.

19. Корнеев, В. В. Будущее высокопроизводительных вычислительных систем / В. В. Корнеев // Открытые системы. – 2003. – № 5.

20. Основы вычислительных систем. Ассоциативные системы. [Электронный ресурс]. – 2004. Режим доступа:
http://256bit.ru/education/infor2/lecture2-3.htm.

21. Представление и обработка знаний в графодинамических ассоциативных машинах: монография / В. В. Голенков [и др.]; под ред. В. В. Голенкова. – Минск : БГУИР, 2001.

22. Гулякина, Н. А. Моделирование графодинамической ассоциативной памяти: сб. науч. тр. Первой междунар. летней школы-семинара по искусственному интеллекту для студ. и асп. / Н. А. Гулякина, Р. Е. Сердюков. – Минск : БГУИР, 1997.

23. Гапонов, П. А. Модели и методы параллельной асинхронной переработки информации в графодинамической ассоциативной памяти: дис. … канд. техн. наук / П. А. Гапонов. – Минск, 2000.

24. Кузьмицкий, В. М. Принципы построения графодинамической ассоциативной памяти / В. М. Кузьмицкий. // Интеллектуальные системы: сб. науч. тр. / Ин-т техн. кибернетики НАН Беларуси; науч. ред. А. М. Крот. – Вып.2. – Минск, 1999.

25. Русын, Б. П. Структурно-лингвистические методы распознавания изображений в реальном времени / Б. П. Русын. – Киев : Наук. думка, 1986.

СОДЕРЖАНИЕ