МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им.В.И.Ульянова (Ленина)

В. А. КИРЬЯНЧИКОВ

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ. АРХИТЕКТУРА КОМПЬЮТЕРОВ. ОРГАНИЗАЦИЯ ПРОЦЕССОРА И ОСНОВНОЙ ПАМЯТИ

Учебное пособие

Санкт-Петербург
Издательство СПбГЭТУ «ЛЭТИ»
2021

УДК 004.2(07+004.39(07)
ББК 3.973.2-02я7+3.973.2-04я7
К43

Кирьянчиков В. А.

К43 Организация ЭВМ и систем. Архитектура компьютеров. Организация процессора и основной памяти: учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2021.

ISBN 978-3-

Содержит основные сведения об архитектуре и организации компьютеров. Рассмотрены состав и взаимодействие процессора и основной памяти.

Предназначено для подготовки бакалавров по направлениям 09.03.04 – «Программная инженерия» и 01.03.02 – «Прикладная математика и информатика», также может быть полезно инженерно-техническим работникам этих областей знаний.

УДК 004.2(07+004.39(07)
ББК3.973.2--02я7+3.973.2-04я7

Рецензенты: Институт кибербезопасности и защиты информации СПбПУ (д-р техн. наук, доцент Д.С. Лаврова); д-р техн. наук, проф. Е.В. Постников (Научно-инженерный центр СПбГЭТУ)

Утверждено
редакционно-издательским советом университета
в качестве учебного пособия

ISBN 978-3-

ОГЛАВЛЕНИЕ

1. ОСНОВНЫЕ ПОНЯТИЯ АРХИТЕКТУРЫ И ОРГАНИЗАЦИИ КОМПЬЮТЕРОВ.

1.1. Состав компьютера

Компьютер, или вычислительная машина **(**ВМ) – это совокупность аппаратных и программных средств, предназначенных для управления и обработки информации. ВМ обычно содержит один основной процессор и, возможно, несколько сопроцессоров (например, графический сопроцессор), имеет фиксированный состав и универсальное применение. Вычислительные системы (ВС), в отличие от ВМ, содержат несколько процессоров, являются проблемно-ориентированными (специализированными) и имеют переменный состав. Состав аппаратных и программных средств компьютера можно пояснить с помощью рис. 1.1.

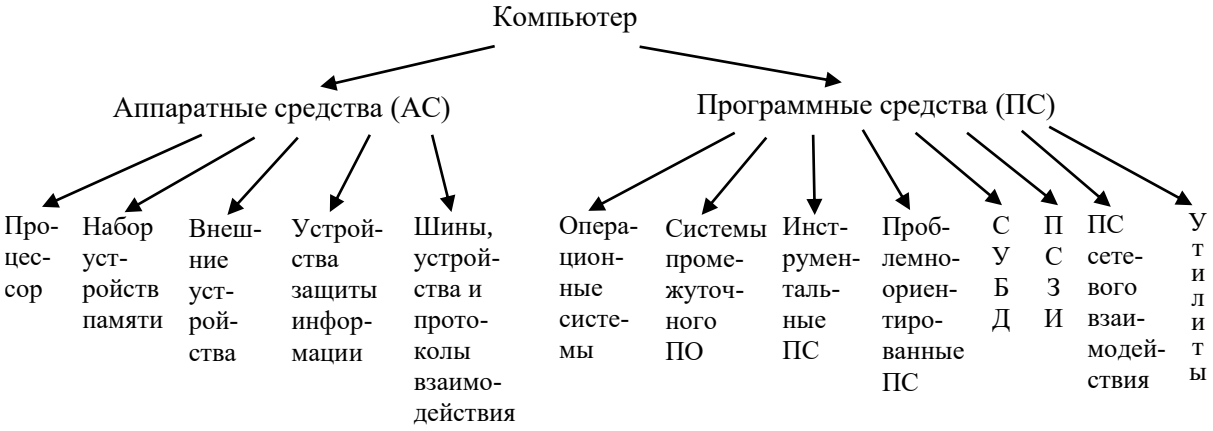


Рис. 1.1

Процессор – основное устройство управления и обработки данных. Память – набор устройств для хранения команд и данных, используемых в процессе работы машины. Внешние устройства обеспечивают взаимодействие машины с внешней средой (в том числе с пользователями). Устройства защиты информации предотвращают несанкционированный доступ или разрушение информации в компьютере. Обычно реализуются в виде аппаратно–программных модулей доверенной загрузки (АПМДЗ), включающих специальные платы и ключи доступа. Шины связи, устройства и протоколы взаимодействия реализуют физическую среду и алгоритмы обмена данными как между узлами компьютера, так и его сетевое взаимодействие с другими компьютерами. Операционные системы (ОС)

являются основными средствами управления выполнением программ и распределением ресурсов машины между задачами. Системы промежуточного ПО (Middleware) – инструментальные технологические среды (DCOM, CORBA, IntelliJ IDEA, Eclipse и др.) – позволяют реализовать определенную технологию разработки ПО для разных языков программирования. Инструментальные ПС включают все средства, необходимые для разработки программ (редакторы, компиляторы, отладчики и др.), как правило, для одного языка программирования. Проблемно-ориентированные ПС предназначены для решения задач определенной области применения: MathLab (научные вычисления), AutoCAD (конструирование), PhotoShop (графический редактор), 3D Studio (машинная графика) и др. Системы управления базами данных (СУБД) – комплекс программ, обеспечивающих управление созданием и использованием баз данных – специально организованных структур, предназначенных для хранения, изменения и доступа к взаимосвязанной информации, преимущественно больших объемов. ПС защиты информации – антивирусные программы, а также комплексы программ для идентификации пользователей, контроля доступа и шифрования информации. ПС сетевого взаимодействия предназначены для организации и поддержки совместной работы группы компьютеров в некоторой сети. Наконец, утилиты – сервисные средства решения вспомогательных задач, предназначенных для расширения возможностей ОС или для упрощения взаимодействия пользователя с компьютером. Обычно к ним относятся: архиваторы, просмотрщики, форматировщики, текстовые редакторы, ПС диагностики компьютера и многие другие.

1.2. Виды (классы) компьютеров

В настоящее время распространены следующие классы компьютеров:

1. Микрокомпьютеры - встраиваемые микропроцессоры со специальным ПО, используемые как программируемые контроллеры для бытового и промышленного оборудования (Embedded Computers), на долю которых приходится более 90 % рынка микросхем; с середины 1990-х годов широкое распространение получили микропроцессоры архитектуры ARM (Advanced RISC Machines) [1], используемые в смартфонах, интернет-планшетах и других мобильных и энергоэффективных устройствах; в настоящее время известны семейства микропроцессоров ARM: ARM7 (с тактовой частотой до 60-72 МГц) для недорогих мобильных телефонов, ARM9 (200 МГц) для продвинутых телефонов и

карманных компьютеров и ARM11 (до 1ГГц) с расширенными возможностями для цифровой обработки сигналов.

В 2019 г. российская компания «Байкал Электроникс» представила первый процессор общего назначения Baikal-M, реализованный по архитектуре ARM. В его основе лежит 28 нм техпроцесс, восемь 64-битных ядер ARM Cortex-A57 с частотой до 1,5 ГГц и 8-ядерный графический процессор Mali-T628 с аппаратным ускорением видео. Здесь имеются две проблемы:

1) необходимость использования зарубежного ядра ARM Cortex-A57;

2) в РФ налажен выпуск процессоров только по нормам до 65 нм; поэтому выпуск новых процессоров заказывают на тайваньской фабрике TSMC.

2. Персональные компьютеры (ПК) – ВМ, предназначенные для работы одного пользователя и удовлетворяющие требованиям общедоступности и универсальности, включая:

- малую стоимость, находящуюся в пределах доступности для индивидуального покупателя;
- гибкость архитектуры, обеспечивающую удобство ее применения в сфере науки, образования, и в быту;
- «дружественность» ОС и прочего ПО для пользователя;
- высокую надежность работы (более 5000 ч. наработки на отказ).

Наибольшей популярностью в настоящее время пользуются ПК на основе процессоров фирмы Intel моделей Pentium, а также многоядерных процессоров Core Duo, Core i3, i5 и i7. По конструктивным особенностям ПК делятся на стационарные и переносные (ноутбуки, планшеты);

3. Рабочие станции (Work Stations) – специализированные ВМ большей производительности, чем ПК, имеют проблемную ориентацию для выполнения научно-инженерных исследований и расчетов, обработки видео, звука, изображений, а также для построения САПР, локальных вычислительных сетей (ЛВС) и организации биржевого интернет-трейдинга.

Современная рабочая станция, как правило, имеет следующие признаки:

- многоядерный процессор, или несколько процессоров;
- мощная видеокарта серии Quadro от Nvidia или RTX;
- ECC RAM – оперативная память с коррекцией ошибок;

- твердотельные накопители (SSD) с высокой скоростью обращения, на порядок более высокой, чем у жестких магнитных дисков (HDD);
- использование жестких дисков, организованных в виде RAID-массивов.

Примерами современных рабочих станций могут служить:

- Sun SPARCstation, разрабатываемые корпорацией Sun Microsystems на основе RISC-архитектуры SPARC, наиболее мощная модель – UltraSPARC III 900 (тактовая частота 900 МГц);
- Intel NUC Hades Canyon – мини-компьютер Intel на основе процессора Core i7 и видеокарты от Radeon с тактовой частотой ядра 3.2 ГГц;
- Lenovo ThinkStation P330 – с восьмиядерным процессором Intel Core i7 8700 и видеокартой Quadro P620 от Nvidia;
- Apple iMac Pro с восьмиядерным процессором Intel Xeon и графической картой Radeon Vega;
- Dell Precision 5820 с десятиядерным процессором Intel Xeon и видеокартой от Nvidia или Radeon.

4. Большие ЭВМ (мэйнфреймы) могут содержать до 10 и более процессоров и предназначены для следующих задач:

- решения научно-технических задач со сложной обработкой данных;
- работы в вычислительных системах с пакетной обработкой;
- работы с большими базами данных;
- управления большими (глобальными) вычислительными сетями.

К мэйнфреймам относят, как правило, компьютеры со следующими характеристиками:

- производительность не менее 10 MIPS (миллионов команд в секунду);
- основная память емкостью от 64 Мб до 10Гб;
- внешняя память не менее 50 Гб;
- многопользовательский режим работы (от 16 до 1000 пользователей).

Родоначальником современных больших ЭВМ является фирма IBM, ее модели IBM 360 и 370, были взяты за основу и при создании отечественной системы больших машин ЕС ЭВМ. Среди современных разработок мэйнфреймов следует отметить: американские IBM 390, SGI, IBM ES/9000, zSeries 900, zSeries 990, а также японские компьютеры М 1800 фирмы Fujitsu.

Мэйнфреймы серии zSeries основаны на архитектуре z/Architecture, которая обеспечивает поддержку 64-разрядной реальной и виртуальной памяти, кластеризацию (до 640 процессоров) и задает новый стандарт производительности и интеграции, обеспечивая большой запас для рабочих нагрузок и приложений. Новейший сервер из этого семейства IBM zSeries 990 (z990) с кодовым названием T-Rex («Тиранозавр») считается самым мощным мэйнфреймом IBM за всю их 40-летнюю историю. Скорость 32-процессорной системы zSeries 990 составляет 9000 MIPS. Этот мэйнфрейм способен выполнять примерно втрое больше работы, чем zSeries 900 и обладает расширенным набором функций для построения центров обработки данных, обработки транзакций и интеграции приложений.

Наиболее мощные модели отечественных больших ЭВМ существенно уступают по своим характеристикам зарубежным типам этих машин:

- ЕС1087- 15 MIPS и 128 Мбайт;
- ЕС 1170 (4-процессорный вариант) - 20 MIPS и 64 Мбайта.

4. Суперкомпьютеры (СК) – высокоскоростные ВС с паралельной архитектурой векторно-матричного типа с производительностью от сотен миллионов до триллионов операций с плавающей точкой в секунду. СК превосходят мэйнфреймы по быстродействию числовой обработки, так как все их вычислительные ресурсы направлены на решение одной глобальной задачи за минимально возможное время. Тогда как мэйнфреймы, наоборот, решают сразу массу задач.

Родоначальником СК был компьютер Cray-1, спроектированный Сеймуром Крэем в США в 1976 г. с производительностью 133 Мфлопса. Наиболее мощный современный СК Cray CS-Storm с производительностью 3,57 Пфлопс находится во втором десятке рейтинга СК (Top500). Этот СК состоит из многоядерных процессоров, образующих серию кластеров, объединенных в единую сеть и предназначен для вычислений в области кибербезопасности, геопространственной разведки и распознавания образов.

Первое место в рейтинге СК Top500 в настоящее время занимает СК Summit, созданный компанией IBM для Национальной лаборатории в Окридже. Производительность СК Summit обеспечивается 9216 процессорами модели IBM POWER9 и 27648 графическими чипами Tesla V100 от Nvidia. Система имеет 512 Гбайт оперативной и 250 Пбайт постоянной памяти. Номинальная производительность – 143,5 Пфлопс.

Второе место занимает американский СК Sierra (ATS-2), использующий 2 вида процессоров – серверные ЦПУ IBM Power 9 и графические Nvidia Volta. Система из 4320 узлов со 190 тысячами ядер обеспечивает вычисления на скорости 94,64 Пфлопс;

Третье место – китайский СК Sunway TaihuLight расположенный в национальном суперкомпьютерном центре КНР, состоит из 41 тысячи процессоров SW26010 и 10,6 миллиона ядер и имеет производительность 93 Пфлопс.

Четвертое место – китайский СК Tianhe-2 («Млечный путь»), построенный на базе 80 тысяч ЦПУ Intel Xeon и Xeon Phi, содержащих более 3 миллионов вычислительных ядер, объем основной памяти – 1400 Гбайт, производительность 61,4 Пфлопс.

Пятое место – СК Японии «K Computer» с производительностью 10,51 Пфлопс, разработанный компанией Fujitsu и расположенный в Институте физико-химических исследований в городе Кобе.

Среди СК России следует привести:

1. СК Christofari, созданный специалистами Сбербанка и Sbercloud в содружестве с Nvidia, выполнен на базе 24-ядерных процессоров Xeon Platinum 8168 с тактовой частотой 2,7 ГГц и графических ускорительных модулей Nvidia DGX-2 для работы с задачами искусственного интеллекта. Кристофари имеет производительность порядка 6,7 Пфлопс, что делает его 29-м в мире, седьмым в Европе и первым в России по мощности.

2. СК «Ломоносов-2» вычислительного центра МГУ(107 место). В 2014 г. система, состоявшая из 1280 узлов на базе процессоров Xeon E5 v3 и ускорителей Nvidia K40M, заняла 22 место в мировом рейтинге СК TOP500 с производительностью 1,85 петафлопс на тесте Linpack.

3. СК Cray XC40 (занимает 465 место в Топ-500), установленный в Рос-гидромете. После модернизации, проведенной компанией «Т-Платформы» в ноябре 2018 г. включает 976 вычислительных узлов с двумя процессорами Intel Xeon E5-2697v4 (всего 35 136 вычислительных ядер) и 128 ГБ оперативной памяти на узел с общей производительностью в 1,29 Пфлопс.

4. СК «Колмогоров» группы «Тинькофф» занял 8 место в рейтинге TOP50 СК России с производительность 658,5 Тфлопс при вычислениях с плавающей точкой двойной точности – FP64. Система состоит из 10 узлов, оснащенных вычислителями NVIDIA Tesla V100 со специализированными ядрами для ускорения

задач искусственного интеллекта. Вычислительные узлы СК объединены современной высокоскоростной сетью 100 Гбит.

5. Кроме того, среди мощных российских систем можно упомянуть СК «Политехник — РСК Торнадо» (СПб политехнический университет) с производительностью 801 Тфлопс, «МВС-10П» (Межведомственный суперкомпьютерный центр РАН), «Лобачевский» в Нижегородском ГУ, «РСК Торнадо ЮУрГУ» в Южно-Уральском ГУ и другие.

1.3. Принцип программного управления и машина фон Неймана

Джон фон Нейман с группой американских специалистов в области Computer Science впервые предложил в 40-х годах XX века концепцию хранимой программы, основные принципы которой заключаются в следующем:

1. Двоичное кодирование.

Вся информация (как команды, так и данные) кодируется двоичными цифрами 0 и 1, поскольку двоичное кодирование по теории информации близко к оптимальному, а кроме того, легче реализовать элементы с двумя устойчивыми состояниями (магнитные сердечники, триггеры).

2. Программное управление.

Все вычисления, предусмотренные алгоритмом решения задачи, должны быть представлены в виде программы, состоящей из последовательности команд. Команды программы, так же, как и данные, хранятся в памяти машины и выполняются в естественной последовательности, т.е. в порядке их положения в программе. При необходимости, с помощью специальных команд, эта последовательность может быть изменена. Решение об изменении порядка выполнения команд принимается либо на основании анализа результатов предшествующих вычислений, либо безусловно.

3. Однородность памяти.

Вид хранимой информации (команды или данные) непосредственно в памяти неразличим, а зависит от последующего использования. Команды могут обрабатываться так же, как и числовые данные. Например, циклически изменяя адресную часть команды, можно обеспечить обращение к последовательным элементам массива данных. Такой приём носит название модификации команд и с

позиций современного программирования не приветствуется. Более полезным является другое следствие принципа однородности, когда команды одной программы могут быть получены как результат исполнения другой программы. Эта возможность лежит в основе трансляции — перевода текста программы с языка высокого уровня на язык конкретной машины.

4. Адресность.

Структурно основная память состоит из пронумерованных ячеек, каждая из которых доступна процессору в произвольный момент. Двоичные коды команд и данных разделяются на единицы информации, называемые словами, и хранятся в ячейках памяти, а для доступа к ним используются номера соответствующих ячеек, называемые адресами. При этом для задания данных в команде указываются не сами данные, а адреса их размещения в памяти.

Основные особенности первых машин, построенных по изложенным принципам и называемых сейчас машинами фон Неймана, состоят в следующем:

1) наличие единого вычислительного устройства, включающего один процессор, память и некоторые внешние устройства;

2) использование линейной структуры адресации памяти со словами фиксированной длины;

3) централизованный принцип управления выполнением программы по последовательному алгоритму;

4) низкий уровень машинных команд, позволяющих выполнять только элементарные операции.

Для таких машин «узким местом», ограничивающим производительность, является память и линии связи процесора и других устройств с памятью: как данные, так и команды должны последовательно выбираться из памяти и передаваться между устройствами. Для повышения производительности в фон Неймановских машинах применяются:

– увеличение разрядности данных (16 бит → 32, 64, 128 бит);

– использование конвейеризации при выборке и обработке команд;

– активное использование кэш-памяти (cache-скрытый) – быстродействующей памяти, которая является буферной между процессором и основной памятью.

Кроме того, наряду с *Принстонской архитектурой*, подразумевающей хранение команд и данных в общей памяти, сейчас применяется и *Гарвардская архитектура*, использующая раздельное хранение команд и данных.

1.4. Понятия архитектуры, организации и реализации компьютера

Архитектура – это множество ресурсов ВМ, доступных пользователю на логическом уровне, без детализации способов взаимодействия процессоров, устройств памяти, внешних устройств и программных средств. При изучении архитектуры рассматривают:

- состав и характеристики процессоров, включая системы команд;
- состав и характеристики устройств памяти и ВУ;
- состав программных средств разработки ПО;
- вид ОС и режимы обработки данных.

Организация – это способы распределения функций, установления связи и взаимодействия процессоров, устройств памяти и внешних устройств, используемые для реализации возможностей, заложенных в архитектуре. При изучении организации рассматривают:

- представление и форматы данных;
- уровни памяти и их взаимодействие;
- состав и форматы машинных команд;
- систему прерываний;
- способы обмена данными.

Реализация – способы технического исполнения конкретных устройств, линий или шин связи и протоколов взаимодействия между ними.

Обычно на уровнях организации и реализации происходит перераспределение функций между аппаратными и программными средствами. Это порождает семейство машин одной архитектуры, но разной производительности и стоимости.

1.5. Многоуровневая организация компьютера

В общем случае обработку информации на ВМ можно рассматривать в виде иерархической системы уровней [2], представленных в табл. 1.1.

Таблица 1.1

Пользователь данного уровня	Уровень	Примечания
Постановщик задач	**6 – концептуальный** (язык спецификаций)	Задаются режимы и виды обработки данных, необходимые для решения задачи, состав системных ПС
Пользователь функционального ПО, решающий задачи из конкретной предметной области	**5 – проблемно-ориентированных ПС** (входной язык пакета программ)	Уровень приложений для конкретной предметной области
Разработчик функциональных программных комплексов	**4 – промежуточного ПО** (например, язык UML)	Middleware (1 - Delphi, Visual C; 2 - DCOM, CORBA, Eclipse)
Разработчик функциональных (прикладных) программ	**3 – языков высокого уровня**	Pascal, C++, Java, Prolog, Python
Системный программист, прикладной программист	**2 – ассемблера**	Программирование фрагментов программ высокой эффективности
Системный программист	**1 – ОС**	Выполнения привилегированных команд, управление памятью
Программист/электронщик (системный архитектор)	**0 – машинных команд**	Цифровое кодирование и представление команд
Программист/электронщик (системный архитектор)	**(–1) – микрокоманд** (микроархитектурный уровень)	Описание набора элементарных операций, реализующих машинные команды
Электронщик	**(–2) – межрегистровых передач**	Реализация элементарных операций как пересылок между регистрами
Электронщик (технолог)	**(–3) – вентилей** (цифровой логический уровень)	Технологический уровень, устройства машины представляются в виде интегральных схем

К системам промежуточного ПО относятся:

– инструментальные среды программирования (Delphi, Visual C, C++ Builder)

– инструментальные технологии программирования (DCOM, CORBA, RMI, ECLIPSE).

Особенности многоуровневой организации:

1) каждый верхний уровень интерпретируется одним или несколькими нижними уровнями.

2) каждый из уровней можно проектировать независимо.

3) модификация нижних уровней не влияет на реализацию верхних.

4) чем ниже уровень реализации программы, тем более высокая производительность достижима.

1.6. Понятие семантического разрыва между уровнями

Преобразование операторов языков высокого уровня (ЯВУ) в машинный код или даже в микрокоманды требует от транслятора, во-первых, умения распознать операторы и команды различных уровней и, во-вторых, для любого оператора ЯВУ – генерировать десятки команд низкого уровня. Это приводит к усложнению транслятора, увеличению трудоемкости его разработки и снижению производительности генерируемых программ (особенно при отсутствии оптимизации). Наличие этих проблем называют *семантическим разрывом между уровнями*. Способы его преодоления зависят от архитектуры ВМ.

Для традиционных ВМ, считающихся машинами со сложным набором команд (Complex Instruction Set Computer – CISC), используется специализация машин, при которой операторы проблемно-ориентированных языков выполняются аппаратными средствами машины; платой за повышение производительности является увеличение сложности аппаратуры, особенно устройства управления и отход от универсальности. Примерами такого подхода могут служить: аппаратная реализация графических преобразований, цифровой обработки сигналов и операций с векторами и матрицами.

Для машин с сокращенным набором команд (Reduced Instruction Set Computer – RISC), характеризующихся ограниченным списком простых команд, оперирующих в основном данными, размещенными в регистрах, реализация операторов ЯВУ на основе команд RISC-процессора оказывается почти столь же эффективной, что и аппаратная реализация, но не усложняет устройства управления. При этом основные усилия направлены на повышение эффективности конвейера обработки команд и выполнения операций в формате «регистр-регистр» по возможности за 1 такт.

Дальнейшее повышение производительности ВМ основано на одновременном использовании нескольких операционных блоков процессора и создании специальных компиляторов, которые упаковывают несколько простых RISC-команд в «очень длинное командное слово» (Very Long Instruction Word – VLIW) так, чтобы в одной длинной команде было можно использовать все операционные блоки процессора, работающие параллельно.

Более подробные сведения об использовании RISC- и VLIW-архитектур изложены в разделах 5.3 и 5.4.

1.7. Организация аппаратных средств ВМ

В зависимости от способов связи между устройствами различают следующие виды организации аппаратных средств ВМ:

1. Структура ВМ с непосредственными связями показана на рис. 1.2.

ЦП – центральный процессор
ОП – оперативная память
ВУ – внешнее устройство

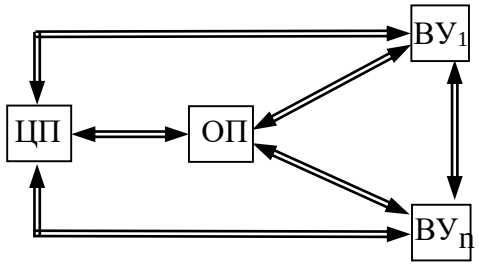


Рис. 1.2

Каждое устройство может связываться с любым другим. Причем взаимодействие пары устройств можно организовать наиболее эффективно. Этот способ применялся в двух первых поколениях машин, но с возрастанием числа устройств машины такая организация становится сложной и дорогой в реализации.

2. Структура ВМ с канальной организацией показана на рис. 1.3.

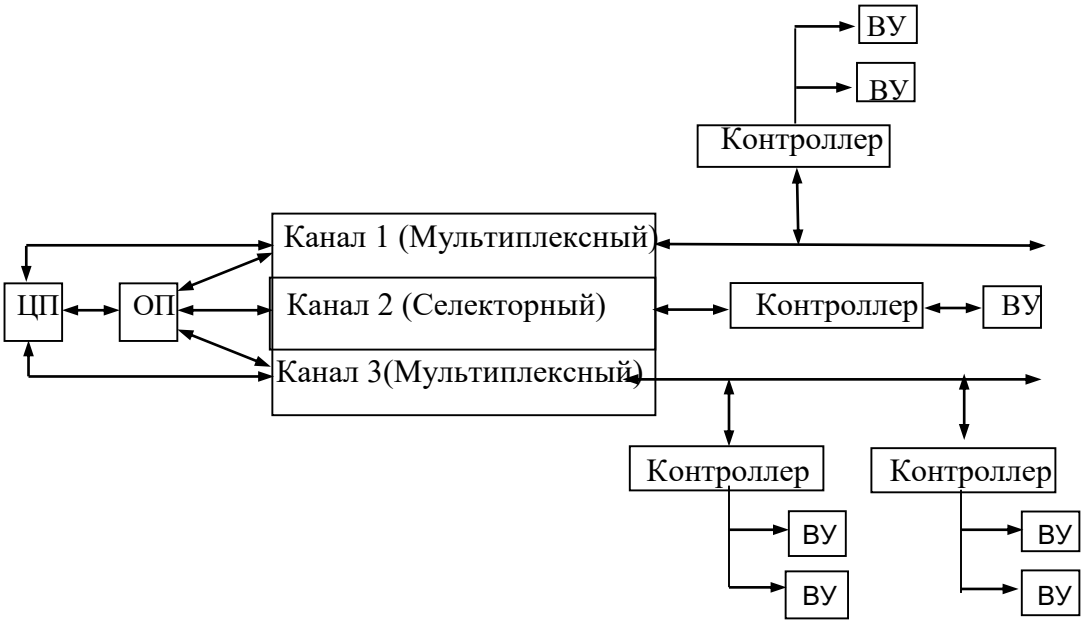


Рис. 1.3

Центральным элементом машины является основная память (ОП), которая хранит программы ЦП и программы каждого из каналов, являющихся процессо-

рами ввода-вывода, работающими параллельно с ЦП по собственной программе, выбираемой из ОП. Различают каналы: селекторный – управляет только одним ВУ и применяется для подключения быстрых устройств; мультиплексный – управляет несколькими более медленными ВУ. Для данной организации характерно большое количество связей и уровней управления взаимодействием устройств. Большая специализация процессоров различного типа затрудняет их интегральное исполнение. Такая организация применялась в машинах третьего и частично четвертого поколений.

3. Структура ВМ с магистральной организацией по типу «Общая шина» (Unibus) показана на рис. 1.4**.**

Магистрально-модульная организация компьютера предполагает выделение общего универсального канала (магистрали связи между элементами системы – модулями) и определения общих правил взаимодействия. В центре ВМ – центральный процессор, управляющий информационной связью между устройствами, подключенными к магистрали (ВУ и память).

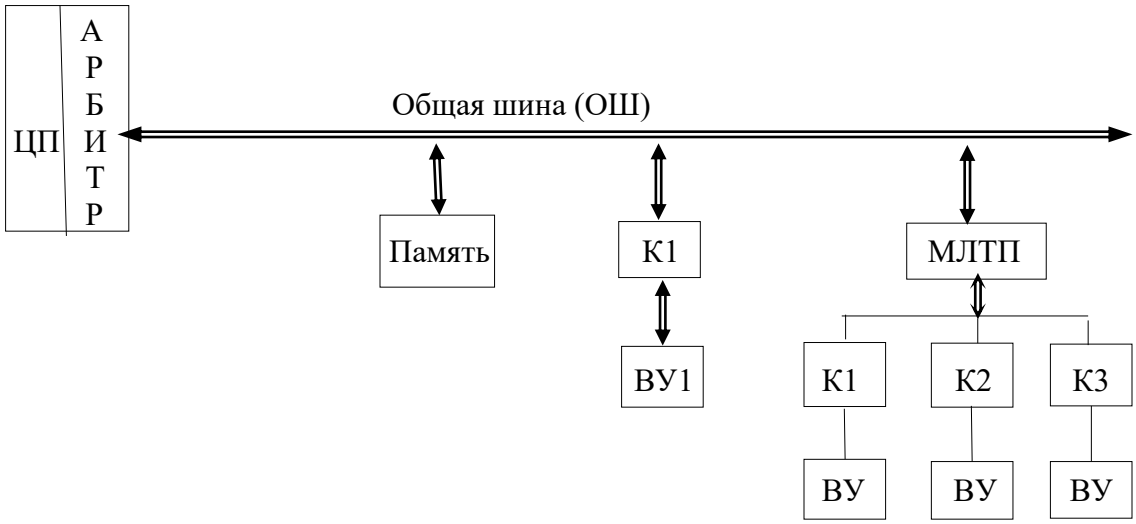


Рис. 1.4

*Магистраль***,** называемая также *Общей шиной* (ОШ), представляет собой множество проводов. По одной группе проводов (шина данных) передается обрабатываемая информация, по другой (шина адреса) – адреса памяти или ВУ, к которым обращается процессор. Есть еще третья часть магистрали – шина управления, по ней передаются управляющие сигналы (например, сигнал запуска операции в

устройстве и др.). Информация, передаваемая от процессора к другим устройствам по шине данных, сопровождается адресом, передаваемым по адресной шине. Это может быть адрес ячейки в оперативной памяти или адрес ВУ.

На схеме через МЛТП обозначен мультиплексор, обеспечивающий подключение нескольких ВУ к одному входу ОШ по нагрузочной способности, К1-К3 – контроллеры ВУ, Арбитр – аппаратная система приоритетов, разрешающая конфликты при одновременном обращении устройств к общей шине.

Типичная шинная транзакция включает в себя две части: посылку адреса и прием (или посылку) данных. Шинные транзакции обычно определяются характером взаимодействия с памятью: транзакция типа «Чтение» передает данные из памяти (либо в ЦП, либо в устройство ввода-вывода), транзакция типа «Запись» записывает данные в память.

Используется единое адресное пространство ячеек памяти и внешних устройств; следовательно, все команды обработки данных процессора могут быть применены и к внешним устройствам. Основной проблемой данной организации является то, что общая шина является «узким местом» и снижает производительность и надежность машины.

4. Структура ВМ с иерархической шинной организацией показана на рис.1.5.

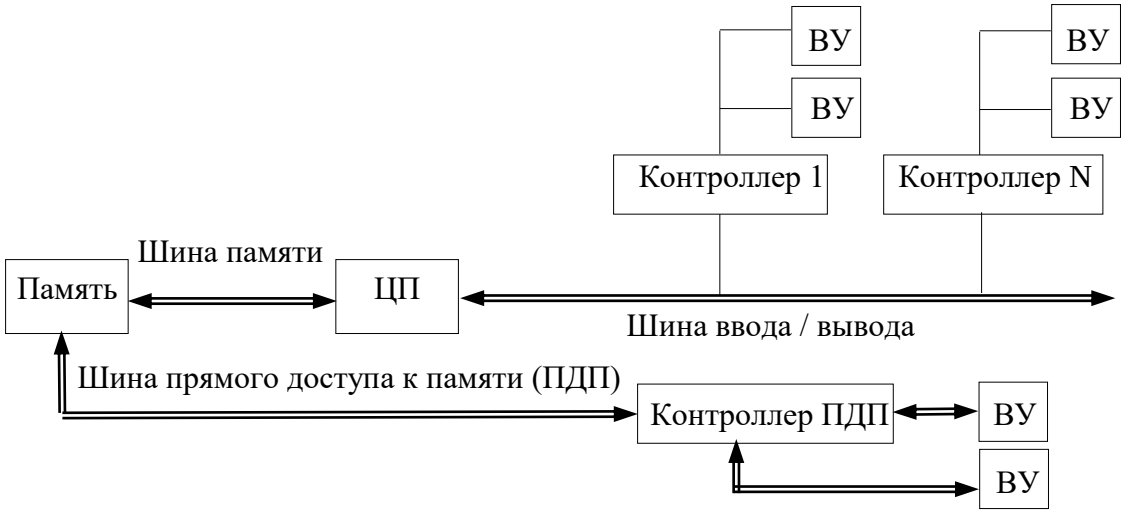


Рис. 1.5

Данная организация ВМ позволяет разгрузить шину, связывающую процессор с памятью, и как следствие – повысить производительность и надежность

работы ВМ. При этом используются различные наборы шин для связи ЦП с памятью и с внешними устройствами. Соответственно, используются различные адресные пространства для обращения к памяти и ВУ. Это требует выделения специальной группы команд ввода- вывода в системе команд процессора:

ADD AX, 100 ; адресуется ячейка памяти 100

IN AX, 100 ; адресуется внешнее устройство с номером 100

Шина прямого доступа к памяти (ПДП) используется для связи ВУ и памяти без участия процессора и позволяет повысить скорость доступа ВУ к памяти.

2. ПРЕДСТАВЛЕНИЕ И ОБРАБОТКА ДАННЫХ В ВМ

В качестве используемых и обрабатываемых в ВМ типов данных следует указать:

1) целые и вещественные числа, участвующие в арифметической обработке;
2) коды команд и адреса памяти (беззнаковые целые числа);
3) коды символов (беззнаковые целые числа), используемые в текстовых сообщениях;
4) битовые наборы (последовательность нулей и единиц), используемые в логических операциях;
5) отдельные биты, используемые в операциях условных переходов и как биты состояния устройств ВМ;
6) данные мультимедиа обработки (видео и аудио информация).

2.1. Представление и обработка целых чисел

Целые числа представляются в позиционной системе счисления в виде

$$N = \sum_{i=0}^{n-1} \alpha_i * S^i = \alpha_{n-1} S^{n-1} + \alpha_{n-2} S^{n-2} + ... + \alpha_1 S^1 + ... + \alpha_0 S^0 \quad ,$$

где S – основание системы счисления (СС), используются S = {2,8,10,16},

α_i = [0, s-1] – разрядный коэффициент.

Если аппаратные средства ВМ используют двоичную СС (S=2), то в программах для компактной записи чисел применяют S = 8, 10,16. При записи чисел в программах для указания СС применяют окончания: B (двоичный), O (восьмеричный), D или пусто (десятичный), H (шестнадцатиричный). Приведем примеры записи чисел и их десятичные значения:101B = 5, 101O = 65, 101D = 101, 101H = 257.

Для представления 16-ричных цифр, больших 9, используют буквы:

10 – A , 11 – B, 12 – C, 13 – D, 14 – E, 15 – F .

Наконец, заметим, что смешанные числа, содержащие целую и дробную часть, в позиционной СС в форме с фиксированной точкой представляются в виде

$$N = \sum_{i=-m}^{n-1} \alpha_i * S^i = \alpha_{n-1} S^{n-1} + \alpha_{n-2} S^{n-2} + ... + \alpha_0 S^0 + \alpha_{-1} S^{-1} + ... + \alpha_{-m} S^{-m}$$

Для преобразования целых чисел (ЦЧ) из одной СС в другую используется правило: для получения числа в новой СС надо его значение в старой СС разделить на основание новой СС, представленное в старой СС, и последовательно получаемые числа остатка взять в обратном порядке. При этом для преобразования большого числа в двоичную СС лучше сначала преобразовать его в 16-ричную СС и затем каждую цифру заменить на тетраду (4 бита) ее двоичного изображения.

Для преобразования дробной части смешанного числа из одной СС в другую используется правило:

для получения дробной части числа в новой СС надо его значение в старой СС умножить на основание новой СС, представленное в старой СС, целую часть частичного произведения зафиксировать, а дробную снова умножить на основание новой СС и так повторять пока дробная часть очередного произведения не станет равной 0 или не будет достигнута требуемая точность. Полученные цифры целых частей произведений записать в прямом порядке как дробную часть числа в новой СС.

При обработке целых чисел различают беззнаковое и знаковое (с учетом знака) представления. Беззнаковые двоичные ЦЧ длиной n бит изменяются в диапазоне $[0, N_{max}]$, где $N_{max} = 2^n - 1$. Например: n = 4 – диапазон [0, 15], n = 8 – диапазон [0, 255], n = 16 – диапазон [0, 65535], n = 32 – диапазон $[0, \approx 2^{32} = 4*2^{30} \approx 4*(10^3)^3 = 4*10^9]$.

Если результат арифметической обработки беззнаковых ЦЧ выходит за допустимый диапазон изменения, то возникает переполнение, которое фиксируется по наличию переноса из старшего разряда результата операции и установке бита (флага) переноса в «1».

При использовании ЦЧ с учетом знака диапазон их изменения разбивается на две части: положительные числа (для n-битного числа – $[0, 2^{n-1} - 1]$) и отрицательные числа $[-2^{n-1}, 0)$. Если результат арифметической обработки положительного числа превышает $2^{n-1} - 1$, то возникает переполнение (положительное), а для отрицательных чисел переполнение (отрицательное) возникает, если результат меньше (-2^{n-1}). ЦЧ с учетом знака могут представляться в трех формах: прямой код и два инверсных кода: обратный и дополнительный. Во всех трех формах старший разряд считается знаковым и для положительных чисел равен 0, а для отрицательных равен 1.

Положительные числа во всех трех формах представляются одинаково «0&информационные разряды числа», где &-конкатенация (присоединение). Так, для 4-битных ЦЧ с диапазоном изменения [–8,7] число 5 будет иметь двоичное представление 0101. Представления отрицательных чисел в каждой из трех форм различаются.

Прямой код (ПК) отрицательного ЦЧ имеет вид «1&информационные разряды числа». Соответственно, число [–5] в ПК будет иметь вид $[-5]_{пк} = 1101$. *Замечание:* В прямом коде существуют: положительный ноль, равный 0000, и отрицательный ноль, равный 1000.

Обратный код (ОК) отрицательного ЦЧ имеет вид «1&инверсные значения информационных разрядов числа». Соответственно, число [–5] в ОК будет иметь вид $[-5]_{ок} = 1010$. В ОК существуют: положительный ноль, равный 0000, и отрицательный ноль, равный 1111.

При работе с 16-ричными числами следует учитывать, что инверсные значения 16-ричных цифр имеют вид, показанный в табл. 2.1.

Таблица 2.1

16-ричная цифра	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Инверсная цифра	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

Дополнительный код (ДК) отрицательного ЦЧ имеет вид «обратный код числа +1». Соответственно, число [–5] в ДК будет иметь вид $[-5]_{дк} = 1011$. В ДК существует только один ноль, равный 0000.

В настоящее время при алгебраическом сложении целых чисел доминирует использование ДК по следующим причинам:

1. ДК имеет единственное изображение нуля: +0 и –0 имеют вид 0…000.
2. При возникновении переноса из старшего разряда результата алгебраического сложения ЦЧ в ДК перенос просто отбрасывается, а для ОК он должен циклически прибавляться к младшему разряду результата, что увеличивает время выполнения операции.
3. Для преобразования числа длиной n разрядов в число с большей разрядностью в ДК достаточно выполнить распространение знака в новые разряды:

   +5 для n = 4 в ДК имеет вид 0101, а для n = 8 имеет вид 00000101;

   –5 для n = 4 в ДК имеет вид 1011, а для n = 8 имеет вид 11111011.

При выполнении сложения ЦЧ с учетом знака переполнение возникает только тогда, когда операнды имеют одинаковые знаки, а знак результата противоположный.

При выполнении операций умножения и деления используется представление ЦЧ в прямом коде. Знак результата формируется как «исключающее ИЛИ» или «сумма по mod2» знаков операндов. Умножение выполняется следующим образом: $A_n * B_n = C_{2n}$. При этом переполнение не возникает. Деление выполняется следующим образом: $A_{2n} / B_n = \{C_{1n}$ (n-битное частное) и C_{2n} (n-битный остаток)$\}$. Знак частного вычисляется как «сумма по mod2» знаков делимого и делителя, знак остатка равен знаку делимого. Переполнение фиксируется, если частное не размещается в n разрядах при делении на 0 или если $A > 2^n * B$.

При решении экономических задач часто возникает потребность в обработке десятичных чисел. Для этого используется их двоично-десятичное представление, при котором каждый десятичный разряд числа представляется двоичной тетрадой (4 битами) цифр с весами 8421. Значения тетрады 1010 ÷ 1111, соответствующие 16-ричным цифрам A ÷ F, считаются запрещенными и их получение при сложении данных требует коррекции результата. Коррекция выполняется путем прибавления к результату так называемого «десятичного заполнителя» $6_{10} = 0110_2$. Например, при сложении десятичных чисел 8 + 5 =13, представленных в двоично-десятичном формате:

$1000 + 0101 = 1101$ (запрещенный код) $+0110 = (0001\ 0011)_{2\text{-}10} = 13_{10}$.

Для выполнения такой коррекции в ВМ с архитектурой Intel X86 предусмотрены специальные команды типа DAA, исполняемые после обычной команды сложения ADD. Аналогичные корректирующие операции и команды предусмотрены для выполнения вычитания.

2.2. Представление и обработка вещественных чисел

Вещественные числа (ВЧ) в компьютерах обычно представляются в форме с плавающей точкой в виде

$$N = m * S^{P} ,$$

где S – основание СС, m – мантисса, p – порядок.

Такая форма позволяет, задавая различные значения m и p, получить разные представления одного и того же числа:

$$N = 5.41 * 10^{0} = 0.541 * 10^{1} = 541 * 10^{-2} = 0.00541 * 10^{3}$$

Для повышения точности ВЧ обычно используют их нормализованное представление, при котором на мантиссу накладывается ограничение $1/S \leq m < 1$ (представление $N = 0.541 * 10^{1}$ – нормализованное) .

Основные форматы представления ВЧ имеют следующие распределения двоичных разрядов (1 – 8 – 23) – короткий формат и (1 – 11– 52) – длинный формат. Короткий формат поясняется на рис. 2.1.

Рис.2.1

На рис. 2.1 S_m – знак мантиссы или знак числа, S_p – знак порядка.

При использовании короткого формата имеем диапазон изменения ВЧ $[-N_{max}, N_{max}]$, где $N_{max} = 1 * 2^{Pmax} = 2^{128} = (2^{10})^{12.8} \approx (10^{3})^{12.8} \approx 10^{38}$

При этом ошибка представления ВЧ, определяемая весом младшего разряда мантиссы, будет равна:

$$\delta = 2^{-23} = (2^{10})^{-2.3} \approx (10^{3})^{-2.3} \approx 10^{-7} ,$$

т. е. обеспечивается точность на уровне 7 десятичных разрядов.

Для длинного формата ВЧ соответственно получим диапазон $[-10^{308}, 10^{308}]$ и точность – 17 десятичных разрядов.

В целом следует отметить, что форма представления ВЧ с плавающей точкой позволяет значительно увеличить диапазон обрабатываемых ВЧ за счет некоторой потери точности из-за того, что часть разрядов числа отводится под порядок. Короткий формат при невысокой точности значений позволяет размещать в памяти большее число операндов и выполнять операции с небольшими затратами времени, а длинный формат обеспечивает возможность вычислений с повышенной точностью.

Выполнение операций алгебраического сложения над ВЧ состоит из трех этапов:

1. Выравнивание порядков операндов выполняется в сторону большего порядка: увеличение на 1 меньшего порядка сопровождается сдвигом на 1 раздряд вправо мантиссы числа с меньшим порядком; этот процесс продолжается пока порядки операндов не станут равны.

2. Суммирование мантисс выполняется по правилам сложения целых чисел.

3. Нормализация результата.

При суммировании мантисс может произойти выход суммы за диапазон разрядной сетки, но это не переполнение, а нарушение нормализации влево. Оно происходит только на 1 разряд и исправляется сдвигом мантиссы на 1 разряд вправо и увеличением порядка на 1. Переполнение возникает только в случае, если перед этим $P = P_{max}$.

Нарушение нормализации вправо возникает при сложении близких по абсолютной величине чисел с разными знаками. При этом часть старших разрядов суммы принимает значения 0 (для положительной суммы) или 1 (для отрицательной суммы в ДК). Исправляется последовательным сдвигом мантиссы суммы на нужное число разрядов влево, сопровождающееся вычитанием 1 из порядка. Если при этом потребуется вычесть 1 из $P = -P_{max}$, то возникает отрицательное переполнение, называемое «потерей значимости» и результат приравнивается к нулю.

При выполнении операций умножения или деления порядки операндов учавствуют в алгебраическом сложении как целые числа (суммируются при

умножении или вычитаются при делении), а мантиссы перемножаются или делятся также как целые числа. Считаем, что исходные операнды представлены в нормализованной форме. Тогда при умножении может возникнуть нарушение номализации вправо на 1 разряд, которое исправляется сдвигом мантиссы результата на 1 разряд влево и вычитанием 1 из порядка. Также при умножении может возикнуть переполнение, если $P_1 + P_2 > P_{max}$ или потеря значимости, если $P_1 + P_2 < -P_{max}$. При делении может возникнуть переполнение при нулевой мантиссе делителя или если $P_1 - P_2 > P_{max}$, а также потеря значимости при исправлении нарушения номализации влево или если $P_1 - P_2 < -P_{max}$.

2.3. Логические операции над битовыми наборами

Логические операции над битовыми наборами выполняются поразрядно в соответствии с табл. 2.2. Более подробно логические операции и логические выражения рассматриваются в разделе 3.

Таблица 2.2

X	Y	X AND Y	X OR Y	X XOR Y	NOT X	NOT Y
0	**0**	**0**	**0**	**0**	**1**	**1**
0	**1**	**0**	**1**	**1**	**1**	**0**
1	**0**	**0**	**1**	**1**	**0**	**1**
1	**1**	**1**	**1**	**0**	**0**	**0**

В случае многоразрядных битовых наборов операнды удобно представлять в компактной форме как целые числа в 16-ричной СС, учитывая, что одна цифра представляет тетраду (4 бита), и в таком же виде выполнять логические операции.

2.4. Представление и обработка символов

Представление символьной информации основано на сопоставлении каждому символу определенной числовой (двоичной или 16-ричной) комбинации. Совокупность символов и назначенных им числовых кодов образует *таблицу кодировки.* В настоящее время известно много таблиц кодировки, но их объединяет весовой принцип, также называемый принцип лексикографического упорядочения кодов символов. Суть его состоит в том, что коды цифр возрастают по мере увеличения цифры, а коды букв увеличиваются в алфавитном порядке.

Так, код буквы «Б» на 1 больше кода буквы «А». Это упрощает обработку символьной информации, в частности, облегчает сортировку, упорядочивание и поиск символьных данных.

Наиболее известной таблицей кодировки является ASCII – код (American Standard Code for Information Interchange), использующий для кодирования символа 7 информационных бит + 1 бит контроля четности. Он позволяет представить 128 символов, включающих латинские буквы, знаки пунктуации и знаки математических операций. Его отечественный аналог называется КОИ-7. Позже появилась его европейская модификация Latin 1 (стандарт ISO 8859), использующий для кодирования все 8 бит и позволяющий представить буквы европейских языков, а также математических и финансовых символов. В зависимости от набора символов различают разные версии стандарта: ISO 8859-1 (западно-европейские языки), ISO 8859-2 (языки стран центральной и восточной Европы), ISO 8859-5 (языки славянских стран с кириллицей) и т.д. В операционной системе MS-DOS стандарт ISO 8859 реализован в виде кодовых страниц (Code Page), каждая из которых имеет свой идентификатор. Так, кодовая страница России имеет идентификатор CP866.

Недостаточное количество кодовых комбинаций в стадарте ISO 8859 привело к тому, что в 1993 году ряд ведущих IT-компаний (IBM, Microsoft, Apple, DEC, Hewlett-Packard) разработали 16-битовый стандарт ISO 10646, названный UCS (Universal Character Set), или Unicode, который позволяет кодировать до 65536 символов. Для букв русского языка выделены коды 1040 ÷ 1093.

Следует заметить, что при вводе символов путем нажатия клавиш на клавиатуре ВМ для повышения надежности получения кода символа используются, так называемые скан-коды, различающие код нажатой и отпущенной клавиши, например, по старшему биту однобайтного кода символа: 1 – клавиша нажата, 0 – отпущена. Для современных клавиатур существуют различные системы скан-кодов и способов сообщения об отпускании клавиши.

2.5. Представление видеоинформации и аудиоинформации

Рассмотренные ранее виды информации имели чисто статический характер. В современных ВМ также используются виды информации динамического или смешанного типа.

Видеоинформация. Видеоинформация (ВИ) бывает статической и динамической. К статической ВИ относятся рисунки, графики, чертежи, таблицы и т. п. К динамической ВИ: видео- и слайд-фильмы. Динамическая ВИ используется либо для передачи движущихся изображений (анимация), либо для последовательной демонстрации отдельных кадров.

При этом существует два способа представления графических изображений: матричный (растровый) и векторный. В матричных форматах изображение представляется матрицей точек – *пикселов* (picture element), положение которых в матрице соответствует координатам точек на экране. Каждый пиксел характеризуется большим количеством атрибутов, определяющих цвет, фон, яркость и другие свойства и может занимать до 30 бит. Учитывая, что количество пикселов на экране может быть более миллиона, основным недостатком матричной графики является требование большого объема памяти для хранения изображений и необходимость сжатия данных. К распространенным матричным форматам относятся BMP, GIF, JPEG,PNG.

Векторное представление описывает изображение не пикселами, а кривыми – сплайнами. Сплайн – гладкая кривая, проходящая через несколько (2-4) опорных точек, задающих форму сплайна. В векторной графике распространены сплайны на основе кривых Безье [3]. Любую элементарную кривую можно построить, зная четыре коэффициента P0, P1, P2, P3, соответствующих четырем точкам на плоскости. Перемещение точек влечет изменение формы кривой. Для многих видов изображений их описание с помощью математических формул является более простым способом благодаря легкости масштабирования и существенно меньшим требованиям к памяти. Недостатком векторных изображений является необходимость разбиения изображений на множество составляющих его примитивов (линия, прямоугольник, эллипс) и более низкое качество. К распространенным векторным форматам относятся CDR, DXF, PS, SVG, VSD.

Аудиоинформация. Аудиоинформация имеет чисто динамический характер и связана с представлением звуковых сигналов, частоты которых лежат в диапазоне 15 Гц ÷ 20 КГц, являющихся по своей природе непрерывными (аналоговыми). Поэтому для использования в ВМ аудиоинформация должна быть представлена в виде значений (выборок), взятых через определенные интервалы времени, и преобразована в цифровую форму с помощью аналого-цифрового преоб-

разователя (АЦП). После обработки в ВМ для воспроизведения звуковые сигналы должны быть восстановлены в аналоговую форму с помощью цифро-аналогового преобразователя (ЦАП). Для обеспечения высокого качества воспроизведения требуется большое количество выборок и соответственно большая емкость памяти для хранения аудиоинформации. Это приводит к необходимости использования различных методов сжатия данных для сокращения требуемого объема памяти. К распространенным форматам представления аудиоинформации следует отнести AVI, WAV, AIF, MPEG.

3. ЛОГИЧЕСКИЕ ЭЛЕМЕНТЫ И УЗЛЫ КОМПЬЮТЕРОВ. ОСНОВНЫЕ ПОНЯТИЯ БУЛЕВОЙ АЛГЕБРЫ

3.1. Простейшие цифровые (логические) схемы

Простейшие цифровые схемы, также называемые логическими вентилями (logic gates), служат для реализации основных логических операций:

– конъюнкция – И (AND) обозначается « $\wedge$ / • / & » ;

– дизъюнкция – ИЛИ (OR) обозначается « $\vee$ / + / !) » ;

– отрицание – НЕ (NOT) обозначается « $\neg$ » ;

– исключающее ИЛИ (XOR) - сумма по mod2, неравнозначность обозначается « $\oplus$ » ;

– штрих Шеффера – НЕ-И обозначается « $|$ » ;

– стрелка Пирса – НЕ-ИЛИ обозначается « $\downarrow$ » .

Простейшие цифровые схемы получают один или более двоичных сигналов на входе и производят новый двоичный сигнал на выходе. Вентиль NOT имеет один вход, вентиль XOR – обычно два входа, а остальные вентили могут иметь любое число входов. Взаимосвязь между входными сигналами и выходным сигналом логического вентиля может быть описана с помощью *таблицы истинности (truth table),* приведенной для указанных выше вентилей с двумя входами X и Y в виде табл. 3.1, или с помощью уравнений булевой алгебры.

Таблица 3.1

X	Y	$\neg$ X	X & Y	X ! Y	X $\oplus$ Y	X $\mid$ Y	X $\downarrow$ Y
0	0	1	0	0	0	1	1
1	0	0	0	1	1	1	0
0	1	1	0	1	1	1	0
1	1	0	1	1	0	0	0

Отличительной особенностью вентилей штрих Шеффера и стрелка Пирса является то, что с их помощью могут быть реализованы любые другие логические операции.

3.2. Цифровые схемы с комбинационной логикой

Цифровая схема – это модуль с двоичными значениями входов и выходов и спецификацией, описывающей его функциональные и временные характеристики. Цифровые схемы разделяются на комбинационные (combinational) и последовательностные (sequential). Выходы комбинационных схем зависят только от текущих значений на входах, т. е. такие схемы комбинируют текущие значения входных сигналов для вычисления значения на выходе. Например, логический вентиль – это комбинационная схема (КСх). Выходы последовательностных схем зависят как от текущих, так и от предыдущих значений на входах, т. е. зависят от последовательности изменения входных сигналов. У комбинационных схем, в отличие от последовательностных схем, память отсутствует. В данном разделе рассматриваются КСх.

Спецификация КСх определяет ее функциональные и временные характеристики. Функциональная спецификация КСх описывает логическую взаимосвязь между ее входами и выходами и может быть задана таблицей истинности или логическим (булевым) выражением. Временная спецификация описывает задержку между изменением сигналов на входе и откликом выходного сигнала.

Существует много способов реализации одной и той же логической функции из простейших логических элементов. Их выбор осуществляется исходя из проектных ограничений, включающих в себя: занимаемую на кристалле микросхемы площадь, скорость работы, потребляемую мощность и время разработки. Схема является комбинационной, если выполнены следующие правила композиции [4]:

– каждый элемент схемы сам является комбинационным;

– каждое соединение схемы является или входом, или подсоединено к одному единственному выходу другого элемента схемы;

– схема не содержит циклических путей: каждый путь в схеме проходит через любое соединение не более одного раза.

3.2.1. Терминология логических выражений

Логическая переменная A, представленная в прямой форме, или ее отрицание ¬A, называемое комплементарной формой, являются литералами. Операция «И» над одним или несколькими литералами называется конъюнкцией, произведением или импликантой. Импликанта, включающая все входы некоторой логической функции, называется минтермом. Так, импликанта A¬B¬C – это минтерм функции трех переменных A, B и C. Операция «ИЛИ» над одним или несколькими литералами называется дизъюнкцией или суммой. Сумма всех входов некоторой логической функции, называется макстермом. Выражение A + ¬B + C – это макстерм функции трех переменных A, B и C.

Таблица истинности для функции N переменных содержит 2^N строк, по одной для каждой комбинации значений входов. Каждой строке в таблице истинности соответствует минтерм, который имеет значение ИСТИНА для этой строки. Табл. 3.2– таблица истинности функции двух переменных *A* и *B*, в каждой строке которой показан минтерм, принимающий для нее значение 1.

Таблица 3.2

A	B	Y	минтерм (МТ)	имя МТ
0	0	0	¬A & ¬B	m0
0	1	0	¬A & B	m1
1	0	1	A & ¬B	m2
1	1	1	A & B	m3

Для любой таблицы истинности можно написать булево выражение путем суммирования всех минтермов, для которых выход *Y* имеет значение ИСТИНА. Для нашей таблицы это выражение имеет вид: *Y* = (A &¬B) + (*A* & *B)*. Такая сумма минтермов называется *совершенной дизъюнктивной нормальной формой* (СДНФ) функции.

Альтернативный способ выражения булевых функций – это *совершенная конъюнктивная нормальная форма* (СКНФ). Каждой строке таблицы истинности соответствует макстерм, который имеет значение ЛОЖЬ для этой строки. Тогда для любой схемы, заданной таблицей истинности, можно записать ее булево выражение как произведение всех макстермов, для которых выход имеет значение ЛОЖЬ.

Для табл. 3.3 это выражение имеет вид: *Y* = (A + B) (¬A + B).

Таблица 3.3

A	B	Y	Макстерм (МаксТ)	Имя МаксТ
0	0	0	A + B	M0
0	1	1	A + ¬B	M1
1	0	0	¬A + B	M2
1	1	1	¬A + ¬B	M3

Представления логической функции (выражения) в виде ДНФ или в виде КНФ логически эквивалентны. ДНФ дает более короткое выражение, когда в таблице истинности мало строк, для которых выход имеет значение ИСТИНА, а КНФ проще, когда в таблице истинности мало строк, для которых выход имеет значение ЛОЖЬ.

3.2.2. Основные понятия булевой алгебры.

Логические выражения, получаемые из таблицы истинности в виде СДНФ или СКНФ, не обязательно имеют простейший вид или приводят к простейшему набору логических элементов. Для упрощения логических выражений можно использовать булеву алгебру точно так же, как обычная алгебра используется для упрощения математических выражений.

Правила булевой алгебры состоят из набора аксиом и ряда теорем (табл. 3.4 – 3.6). Аксиомы и теоремы булевой алгебры подчиняются принципу двойственности. Если взаимно заменить символы 0 и 1 , а также взаимно заменить операторы • (И) и + (ИЛИ), то булево выражение останется верным.

В булевой алгебре доказательство теорем с конечным числом переменных является простым: нужно показать, что теорема верна для всех возможных значений этих переменных. Этот метод называется *совершенной индукцией*, и может быть выполнен с использованием таблицы истинности.

Таблица 3.4

	Аксиома	Двойственная аксиома	Название
A1	¬0 = 1	¬1 = 0	НЕ
A2	0 • 0 = 0	1 + 1 = 1	И/ИЛИ
A3	1 • 1 = 1	0 + 0 = 0	И/ИЛИ
A4	0 • 1= 1 • 0 =0	0 + 1= 1 + 0 = 1	И/ИЛИ

Таблица 3.5

	Теорема	Двойственная теорема	Название
T1	B • 1 = B	B + 0 = B	Идентичность
T2	B • 0 = 0	B + 1= 1	Нулевой элемент
T3	B • B = B	B + B = B	Идемпотентность
T4	¬ ¬B = B	-	Инволюция
T5	B • ¬B = 0	B + ¬B = 1	Дополнительность

Таблица 3.6

	Теорема	Двойственная теорема	Название
T6	B • A = A • B	B + A = A + B	Коммутативность
T7	(A•B) • C = A • (B•C)	(B + C) + D = B + (C + D)	Ассоциативность
T8	(A • B) + (A • C) = A • (B + C)	(A + B) • (A + C) = A + (B • C)	Дистрибутивность
T9	A • (A + B) = A	A + (A • B) = A	Поглощение
T10	(A • B) + (A • ¬B)= A	(A + B) • (A + ¬B) = A	Склеивание
T11	¬(A0 • A1 • A2) = (¬A0 + ¬A1 + ¬A2)	¬(A0 + A1 + A2) = ¬A0 • ¬A1 • ¬A2	Теорема де Моргана

Теорема *де Моргана* T11 является особенно мощным инструментом при разработке цифровых устройств. Эта теорема поясняет, что отрицание результата умножения всех термов равно сумме отрицаний каждого терма. Аналогично отрицание суммы всех термов равно произведению отрицаний каждого терма. В соответствии с теоремой де Моргана, элемент И-НЕ эквивалентен элементу ИЛИ с инвертированными входами, а элемент ИЛИ-НЕ эквивалентен элементу И с инвертированными входами.

Упрощение логических выражений с помощью теорем булевой алгебры приводит к минимизации выражения, если оно включает в себя минимально возможное количество импликант. Если есть несколько выражений с одинаковым количеством импликант, минимальным будет то, в котором меньше литералов. Полное упрощение выражений может потребовать нескольких попыток, некоторые из которых могут быть ошибочными. Целью упрощения является уменьшение количества элементов, используемых при физическом воплощении логической функции в аппаратуре, тем самым делая схему меньше, дешевле и, возможно, быстрее.

Минимизация логических выражений на основе булевой алгебры без соблюдения должной аккуратности может давать решения, отличные от требуемых.

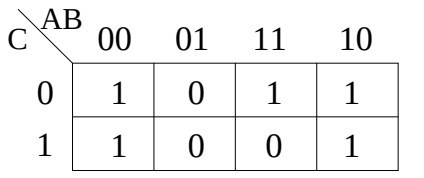
Более простой и наглядный способ минимизации, основанный на визуальном подходе, используется в картах Карно [5]. Карты Карно – это графический способ представления булевых функций с целью их наглядной минимизации, обеспечивающий упрощение сложных логических функций многих переменных. Преобразование представления логической функции, заданной в виде таблицы истинности, в карты Карно и обратно выполняются просто.

Известно, что основным методом минимизации логических функций, представленных в виде СДНФ или СКНФ, является применение операций попарного склеивания и элементарного поглощения. Операция попарного склеивания осуществляется между двумя термами (членами), содержащими одинаковые переменные, вхождения которых (прямые или инверсные) совпадают для всех переменных, кроме одной. В этом случае все переменные, кроме одной, можно вынести за скобки, а оставшиеся в скобках прямое и инверсное вхождение одной переменной, подвергаются склейке. Таким образом, главной задачей при минимизации СДНФ и СКНФ является поиск термов, пригодных к склейке с последующим поглощением, что для больших форм может оказаться достаточно сложной задачей. Карты Карно предоставляют наглядный способ отыскания таких термов. Логические термы, к которым могут быть применены операции попарного склеивания и элементарного поглощения. группируются в карте Карно в виде массивов соседних ячеек, содержащих единицы или нули, которые сразу наглядно видны.

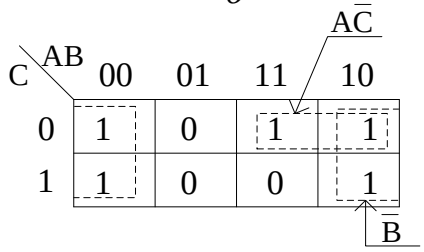
На рис.3.1 показаны таблица истинности (а) и карта Карно (б) для функции трех переменных. Верхняя строка карты Карно дает 4 возможных значения для переменных *A* и *B*. Левая колонка дает два возможных значения переменной *C*. Каждая клетка карты Карно соответствует строке таблицы истинности и содержит значение функции *Y* из этой строки. Как и каждая строка в таблице истинности, каждая клетка карты Карно представляет собой отдельный минтерм. Каждая клетка или минтерм отличается от соседней изменением только одной переменной. Это значит, что соседние клетки различаются только в значении одного литерала, значение которого «истинно» в одной клетке и «ложно» в соседней. Например, клетки, представляющие минтермы $(\neg A \bullet \neg B \bullet \neg C)$ и $(\neg A \bullet \neg B \bullet C)$ – соседние и различаются только в переменной *C*. Также можно заметить, что значения переменных *A* и *B* комбинируются в верхней строке в особом порядке:

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

а



б



в

$$Y=A\bullet \neg C + \neg B$$

Рис. 3.1

00, 01, 11, 10. Этот порядок называется *кодом Грея*. В отличие от битового порядка по возрастанию величины (00, 01, 10, 11), в коде Грея соседние записи отличаются только на один разряд, что обеспечивает требуемые свойства соседних ячеек, которые должны различаться только в одной переменной. Код Грея может быть любой разрядности. Например, трехбитный код Грея выглядит так: 000, 001, 011, 010, 110, 111, 101, 100. Также отметим, что карты Карно «закольцованы»: клетка с самого правого края таблицы является соседней с самой левой, так как они отличаются только в одной переменной (*A*).

Карты Карно обеспечивают простой способ минимизации логических выражений, обводя единицы в соседних клетках пунктирными элементами, как показано на рис. 3.1, *в*. Для каждого из них мы пишем соответствующую ему импликанту, являющуюся произведением одного или нескольких литералов. Переменные, для которых прямая и комплементарная формы попадают в один элемент, исключаются из импликанты, так что карта Карно дает то же самое выражение, которое получается с использованием булевой алгебры.

Для минимизации логических выражений следует обвести все прямоугольные блоки с единицами на карте, используя наименьшее возможное число элементов. Каждый элемент должен быть максимально большим, и тогда он является

первичной импликантой. Логические выражения являются минимальными, если записаны как сумма наименьшего числа первичных импликант.

Пример. Пусть требуется минимизировать функцию трех переменных, заданную таблицей истинности на рис.3.1, *а* при помощи карты Карно.

Решение. Обведем единицы на карте Карно, используя наименьшее возможное количество пунктирных элементов, как показано на рис. 3.1, *в*. Каждый элемент на карте Карно представляет собой первичную импликанту, а его размер кратен степени двойки (2 × 1 и 2 × 2).

Сформируем первичную импликанту для каждого выделенного элемента, выписывая только те переменные, которые появляются в нем только в прямой или в комплементарной формах. Например, элемент размером 2 × 1 включает в себя прямую и комплементарную формы переменной *B*, так что мы не включаем *B* в первичную импликанту. Однако в этом элементе есть только прямая форма переменной *A* и комплементарная форма $\neg C$ переменной *C*, так что мы включаем эти переменные в первичную импликанту A• $\neg C$. Подобным же образом овал размером 2 × 2 покрывает все клетки, где B = 0, так что первичная импликанта будет ¬B. Обратите внимание, что правая верхняя клетка (минтерм) используется дважды, чтобы сделать элементы первичных импликант как можно большими, а также элемент, покрывающий четыре клетки, оборачивается через края карты Карно. В результате можно записать минимальное выражение для функции: Y = A¬C+¬B .

Булева алгебра и карты Карно – два метода упрощения логических выражений. В конечном счете, целью является нахождение наименее затратного метода реализации конкретной логической функции. В современной инженерной практике компьютерные программы, называемые синтезаторами логики (logic synthesizers), проводят упрощение схем по описанию логический функций (см. раздел 3.5). Для больших задач программы логического синтеза намного эффективнее людей. Хотя в реальных проектах карты Карно используются редко, но понимание принципов, лежащих в основе карт Карно, и их применение при создании простых логических схем вполне оправданы.

До сих пор мы концентрировались на упрощении логических выражений для разработки логических схем, использующих наименьшее число элементов.

Однако одна из самых сложных задач в разработке схем – это учет всех ограничений, накладываемых на временные характеристики работы схемы, ведь хорошая схема должна работать предельно быстро и при этом без сбоев.

Изменение выходного сигнала схемы в ответ на изменение входов занимает время. Основные причины задержек в схемах заключаются во времени, требуемом для перезарядки емкостей цепи, а также в конечной скорости распространения электромагнитных волн в среде. Комбинационная логика характеризуется задержкой реакции или отклика и задержкой распространения. Задержка реакции *tcd* – это минимальное время от момента, когда вход изменился, до момента, когда любой из выходов начнет изменять свое значение. Задержка распространения *tpd* – это максимальное время от начала изменения входа до момента, когда все выходы достигнут установившихся значений. Из-за необходимости вникания в физические уровни создания цифровых схем, вычисление *tpd* и *tcd* выходит за рамки этого пособия.

3.3. Последовательностная логика. Конечные автоматы

В отличие от комбинационной логики, выходные сигналы которой зависят только от текущего состояния входных сигналов, выходные сигналы последовательностной логики зависят как от текущего, так и от предыдущего состояния входных сигналов [4]. Другими словами, последовательностная логика помнит информацию о входных сигналах в предыдущие моменты времени. Эта память называется состоянием последовательностной логической схемы (ПЛС) и представляет собой набор бит, которые содержат всю информацию о прошлом, необходимую для определения будущего поведения ПЛС.

Одними из простейших последовательностных схем являются триггеры – класс логических схем, обладающих способностью длительно находиться в одном из двух устойчивых состояний и чередовать их под воздействием внешних сигналов. Отличительной особенностью триггера как функционального устройства является свойство запоминания двоичной информации. Под памятью триггера подразумевают способность оставаться в одном из двух состояний и после прекращения действия переключающего сигнала. Приняв одно из состояний за «1», а другое за «0», можно считать, что триггер хранит один бит информации. При включении питания триггер непредсказуемо принимает одно из двух состояний. Это приводит к необходимости выполнять первоначальную установку

триггера в требуемое исходное состояние, то есть подавать сигнал сброса на вход триггера или на входы создаваемых из триггеров компонентов цифровых операционных устройств: счетчиков, регистров, статических запоминающих устройств (ЗУ) и т.д.

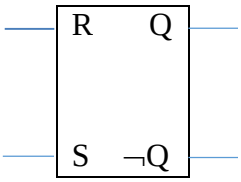
Все разновидности триггеров представляют собой ПЛС, включающие собственно элемент памяти (ЭП) и комбинационную схему (КС), которая является входной логикой. К основным типам триггеров относятся: RS-триггер, D-триггер и Т-триггер.

RS-триггер (от англ. Reset и Set), обычно строится из двух перекрестно включенных элементов ИЛИ-НЕ. Выход триггера, обычно обозначаемый буквой *Q*, определяется значениями входов *R* и *S*, которые отвечают за сброс и установку значений выхода. Работу RS-триггера можно пояснить с помощью таблицы истинности, приведенной на рис. 3.2, *а*, а его обозначение на схемах показано на рис.3.2, *б*.

Поясним таблицу истинности. Предположим, что у *Q* есть какое-то определенное значение, установленное до подачи входов, которое назовем $Q_{пред}$, $Q_{пред}$ может быть либо 0, либо 1 и отражает исходное состояние системы. Когда *R* и *S* равны 0, на выходе *Q* будет сохраняться старое значение $Q_{пред}$, а $\neg Q_{пред}$ будет его логической инверсией.

Входы		Выход	
S	R	Q	¬Q
0	0	$Q_{пред}$	$\neg Q_{пред}$
0	1	0	1
1	0	1	0
1	1	0	0

а



б

Рис.3.2

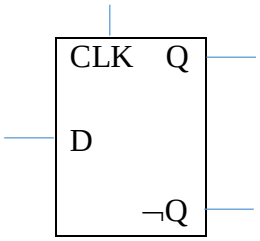
Когда поступает команда сброса *R*=1, выход *Q* принимает значение 0, а выход ¬Q – инверсное значение 1. Когда поступает команда установки *S*=1, выход *Q* становится единицей, а ¬Q – нулем. Подача на входы одновременно *R*=1 и *S*=1 не имеет смысла, так как это означает, что выход должен быть одновременно и

установлен в 1, и сброшен в 0, что невозможно. Часто в этом случае как на прямом, так и на инверсном выходе выставляется логический 0, хотя такое состояние триггера является неустойчивым и лучше его считать неопределенным.

D-триггеры также называют триггерами задержки (от англ. Delay). У D-триггера есть два входа: вход данных D, определяющий его следующее состояние, и вход тактового сигнала CLK, определяющий, когда оно изменится. Таблица истинности D-триггера показана на рис. 3.3*а*, а его обозначение на схемах показано на рис.3.3*б*.

Входы		Выход	
CLK	D	Q	¬Q
0	X	$Q_{пред}$	$¬Q_{пред}$
1	0	0	1
1	1	1	0

а



б

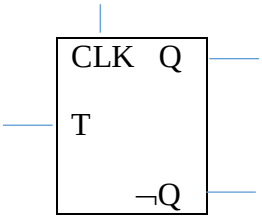
Рис.3.3

Как видно из таблицы истинности одноступенчатого D-триггера все изменения информации на входе D передаются на выход Q сразу по тактовому сигналу. Там где это нежелательно, нужно применять двухступенчатые (двухтактные) D-триггеры.

T-триггер (от англ. Toggle - *переключатель*) часто называют счётным триггером, так как он является простейшим счетчиком по модулю 2. Таблица истинности T-триггера показана на рис.3.4*а*, а его обозначение на схемах показано на рис.3.4*б*.

Вход		Выходы	
T	CLK	Q(t)	Q(t+1)
0	1	0	0
0	1	1	1
1	1	0	1
1	1	1	0

а



б

Рис.3.4

Как видно из таблицы истинности синхронного T-триггера, при единице на входе T по каждому такту на входе синхронизации (CLK) он изменяет своё логическое состояние на противоположное и не изменяет выходное состояние

при нуле на входе T. Таким образом, изменение выходного сигнала происходит с частотой вдвое меньшей, чем входного. Это позволяет использовать T-триггер для синтеза двоичных счетчиков, в которых каждый триггер соответствует одному двоичному разряду.

Один из эффективных подходов к проектированию последовательностных схем основывается на использовании конечных автоматов (КА). Под конечным автоматом понимается абстрактный дискретный процесс, который определяется тремя множествами и двумя функциями. К используемым множествам относятся [6]:

1. Множество состояний процесса $S = \{s_1, s_2, \ldots, s_k\}$.

Это множество конечно, и поэтому автоматы также называются конечными.

2. Конечное множество входных воздействий $X = \{x_1, x_2, \ldots, x_n\}$ автомата, поступающих в дискретные моменты времени t_i, разделяемые длительностью такта Δ.

3. Конечное множество выходных сигналов $Y = \{y_1, y_2, \ldots, y_m\}$ автомата, также формируемых в дискретные моменты времени t_i.

Часто множество X называют входным алфавитом автомата, а множество Y – его выходным алфавитом.

Функциями, описывающими поведение автомата являются следующие:

1. Функция переходов $\delta(t)$, определяющая правила перехода автомата из одного состояния в другое:

$$T: S \times X \rightarrow S \mid S(t + \Delta) = \delta[S(t), X(t)], \text{ где } \Delta > 0 \text{ – длительность такта.}$$

2. Функция выходов $\lambda(t)$, определяющая правила формирования выходных сигналов автомата. Вид функции выходов зависит от класса (типа) автомата. Существует два основных класса автоматов: автомат *Мили*, в котором выходной сигнал вырабатывается в момент перехода автомата из одного состояния в другое и является импульсным, и автомат *Мура*, в котором выходной сигнал зависит только от состояния автомата и является потенциальным. Соответственно, функцию выходов КА можно описать как

$$R: S * X \rightarrow Y \mid Y(t+\Delta) = \begin{cases} \lambda[S(t), X(t)], & \text{для автомата Мили} \\ \lambda[S(t+\Delta)], & \text{для автомата Мура} \end{cases}$$

Между автоматами Мили и Мура существует соответствие и взаимнооднозначное преобразование в том смысле, что им соответствует одинаковая зависимость между входной X и выходной Y последовательностями. Каноническое представление структурной схемы КА представлено на рис. 3.5.

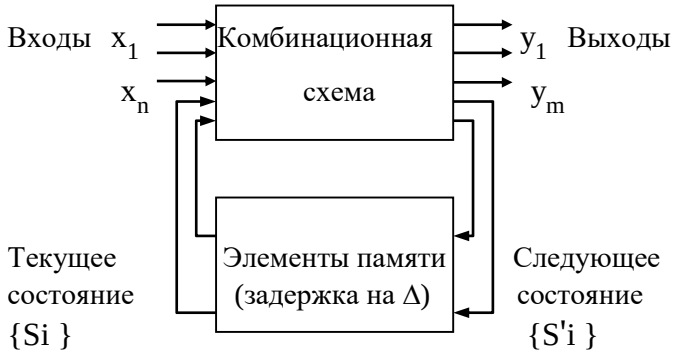
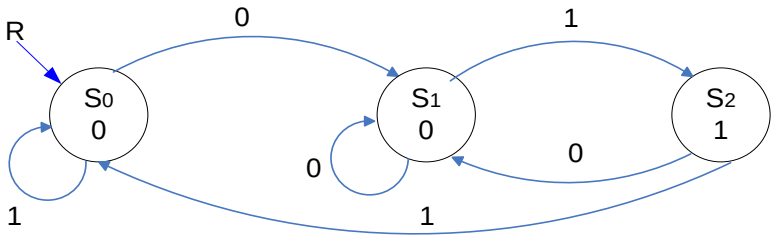


Рис.3.5

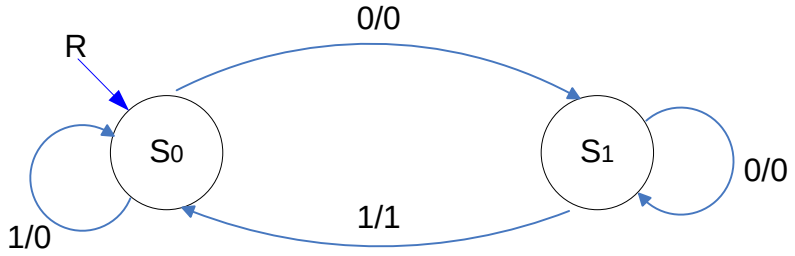
В целях упрощения анализа автомат условно разделяется на две части: комбинационную, формирующую в соответствии с функциями $\delta(t)$ и $\lambda(t)$ следующее состояние $S'=\{s'_1,\ldots,s'_n\}$ и выходные сигналы $Y=\{y_1,\ldots,y_m\}$ автомата и запоминающую (элементы памяти), обеспечивающую задержку состояния автомата на один такт Δ. Комбинационная схема формирует выходные сигналы автомата в текущий момент времени, а переменные следующего состояния $\{s'_i\}$ определяют текущее состояние S в следующий момент времени $t + \Delta$.

Закон функционирования КА может быть задан в форме ориентированного графа. При этом состояния автомата отображаются вершинами графа, а переходы между состояниями – дугами. В автоматах Мура выходные сигналы связаны только с состояниями автомата, которым соответствуют вершины графа, а дуги помечаются входными сигналами, вызвавшими соответствующий переход. В автоматах Мили дуги помечаются как входными, так и выходными сигналами, формируемыми в момент перехода автомата из одного состояния в другое. Примеры графов автоматов Мура и Мили, формирующих одинаковые последовательности выходных сигналов для заданной последовательности входных, показаны на рис. 3.6, *а* и рис. 3.6, *б* соответственно. Следует отметить, что у автомата Мура обычно больше состояний, чем у автомата Мили, решающего ту же задачу.



$S=\{S_0, S_1, S_2\}$ $X=\{x_0 = 0, x_1 = 1\}$ $Y= \{ y_0 = 0, y_1 = 1\}$

а



$S=\{S_0, S_1\}$ $X=\{x_0 = 0, x_1 = 1\}$ $Y= \{ y_0 = 0, y_1 = 1\}$

б

Рис.3.6

Наряду с использованием графов часто используется табличная форма описания закона функционирования КА, состоящая из таблицы переходов (рис.3.7, *а*) и таблицы выходов (рис.3.7, *б*) для автомата Мура, или совместной таблицы переходов-выходов автомата Мили, показанной на рис. 3.8. На рис. 3.9 показан пример последовательности входных сигналов и полученные в соответствии таблицами переходов и выходов последовательности выходных сигналов автоматов Мура и Мили.

Текущее состояние	Вход	Следующее состояние
S_0	0	S_1
S_0	1	S_0
S_1	0	S_1
S_1	1	S_2
S_2	0	S_1
S_2	1	S_0

а

Текущее состояние	Выход
S_0	0
S_1	0
S_2	1

б

Рис.3.7

Текущее состояние	Вход	Следующее состояние	Выход
S_0	0	S_1	0
S_0	1	S_0	0
S_1	0	S_1	0
S_1	1	S_0	1

Рис. 3.8

Входная последовательность	x_1	x_0	x_1	x_1	x_0	x_0	x_1
Выходная последователь-ность автомата Мура	y_0	y_0	y_1	y_0	y_0	y_0	y_1
Выходная последователь-ность автомата Мили	y_0	y_0	y_1	y_0	y_0	y_0	y_1

Рис. 3.9

Следует отметить, что выход автомата Мили опережает выход автомата Мура на один такт, так как он реагирует на вход, а не ждет изменения состояния. Если на выходе автомата Мили поставить триггер, добавив тем самым задержку на такт, то по временным параметрам такая конструкция станет эквивалентной автомату Мура.

Для создания КА следует использовать следующую последовательность действий:

- определить входы и выходы;
- нарисовать граф или диаграмму переходо;
- для автомата Мура составить таблицу переходов и таблицу выходов;
- для автомата Мили составить объединенную таблицу переходов и выходов;
- выбрать метод кодирования состояний, влияющий на схемную реализацию;
- составить булевы выражения для комбинационной схемы, формирующей следующие состояния и выходные сигналы автомата;
- разработать принципиальную схему автомата.

В заключение отметим, что проектирование сложных конечных автоматов часто упрощается, если их разбить на несколько более простых автоматов, взаимодействующих друг с другом таким образом, что выход одних автоматов является входом других. Такое применение принципов иерархической организации и модульного проектирования называется *декомпозицией* конечных автоматов.

3.4. Цифровые функциональные узлы

Комбинационные и последовательностные логические схемы используются для создания цифровых функциональных узлов, составляющих базис операционных элементов, реализующих обработку данных в компьютере.

Основными компонентами этого базиса являются следующие элементы:

3.4.1. Регистры

Регистр – упорядоченная последовательность запоминающих элементов (ЗЭ), предназначенных для хранения битов информации. В качестве ЗЭ обычно используются триггеры. В регистре, состоящем из n ЗЭ, элемент с номером i называется i-ым разрядом регистра и хранит i-й разряд двоичного слова. Число n разрядов регистра определяет его длину. Каждому из 2^n состояний n-разрядного регистра может сопоставляться одно из целых чисел в диапазоне $[0, 2^n-1]$, либо другие виды информации. Применительно к регистрам обычно применяются операции сброса (установки в 0 всех разрядов регистра), чтения и записи информации, инвертирования и сдвига. Регистр с возможностью инвертирования, как правило, строится на триггерах со счетным входом. Регистры с возможностью сдвига часто называют сдвигающими регистрами, их разряды дополняются специальными цепями сдвига. Сдвиг может выполняться влево или вправо, на один или несколько разрядов, а также как циклический сдвиг при соединении старшего и младшего разрядов регистра.

3.4.2. Счетчики

Счетчик – операционный элемент, обеспечивающий хранение слова информации, а также выполнения над ним операции счета. В зависимости от вида операции счета различают суммирующие (увеличение на 1), вычитающие (уменьшение на 1) или реверсивные (реализует обе операции) счетчики.

Счетчики строятся также на основе триггеров, каждый из которых хранит i-й бит n-разрядного двоичного слова.

В зависимости от требований к быстродействию различают счетчики с последовательным, сквозным и групповым переносом между разрядами. При последовательном переносе время операции счета может достигать $n*(\tau + \tau_T)$, где τ – задержка логического элемента, τ_T – время переключения триггера, причем обычно $\tau << \tau_T$.

В случае сквозного переноса время операции счета составляет $(n*\tau + \tau_T)$, что на $(n - 1)*\tau_T$ меньше, чем при последовательном переносе. Наконец, в случае группового переноса время счета составляет $(\tau*n/m + \tau_T)$, где m - число разрядов в группе. Естественно, затраты оборудования при этом возрастают в обратном порядке.

3.4.3. Сумматоры

Сумматор – операционный элемент, предназначенный для выполнения операции сложения чисел. Если операнды и результат представляются в двоичной системе счисления, то сумматор называется двоичным. Сумматоры подразделяются на два типа: комбинационные и накапливающие. Комбинационные не содержат ЗЭ и реализуют сложение в виде C=A+B, а накапливающие содержат регистр, на котором до операции находится один из операндов, а затем результат – сумма, формируемая в виде C= C+A. Как и в счетчиках, в сумматорах могут возникать переносы, удлиняющие время сложения, поэтому для повышения быстродействия применяются сумматоры со сквозным и групповым переносом.

3.4.4. Схемы сравнения

Схема сравнения – операционный элемент, на котором вычисляется значение отношения между двумя двоичными словами. Результат вычисления отношения может принимать значения Истина (1) или Ложь (0). В зависимости от типа вычисляемых отношений различают: 1) схемы сравнения на равенство, вычисляющие отношения A=B или $A \neq B$; 2) схемы сравнения на больше (меньше), вычисляющие отношения A >B, A < B, $A \geq B$ или $A \leq B$.

3.4.5. Дешифраторы и шифраторы

Дешифратор – операционный элемент, преобразующий двоичный n-разрядный позиционный код $X = x_1 x_{2...} x_n$ в 2^n – разрядный унитарный двоичный код $Y = y_1 y_{2...} y_m$ ($m = 2^n$), каждое значение которого представляет собой минтерм, соответствующий набору X, на котором переменная y_i принимает значение 1.

Шифратор – операционный элемент, преобразующий 2^n - разрядный унитарный двоичный код $Y = y_1 y_{2...} y_m$ ($m=2^n$) в двоичный n-разрядный позиционный код $X= x_1 x_{2...} x_n$. Так, 8-разрядный унитарный код Y = {0,1,2,3,4,5,6,7} преобразуется в 3-разрядный двоичный код X со значениями {000, 001,010,011,100,101,110,111}.

3.5. Языки описания аппаратуры

Разработка комбинационных и последовательностных цифровых схем на уровне упрощения логических таблиц или выражений и перевода конечных автоматов в вентили вручную является очень трудоемким процессом. В 1990-е годы разработчики обнаружили, что их производительность труда резко возрастала, если они работали на более высоком уровне абстракции, определяя только логическую функцию и предоставляя создание оптимизированных логических элементов *системе автоматического проектирования* (САПР).

Были разработаны два основных языка описания цифровой аппаратуры (Hardware Description Language, HDL) – это SystemVerilog и VHDL. Основные цели использования HDL – логическая симуляция и синтез. Во время симуляции на входы проектируемого модуля подаются некоторые воздействия и проверяются выходы, чтобы убедиться, что модуль функционирует корректно. Это позволяет провести тестирование цифровой схемы и исправить ошибки до того, как система будет выпущена.

Логический синтез преобразует код на HDL в описание цифровой аппаратуры (т. е. логические элементы и соединяющие их проводники). Логический синтезатор также может выполнять оптимизацию для сокращения количества необходимых элементов.

Описание схем на HDL напоминает программный код. Однако следует помнить, что код на HDL предназначен для описания аппаратуры. Одна из главных ошибок начинающих разработчиков заключается в том, что они думают о коде на HDL как о компьютерной программе, а не как о подспорье для описания цифровой аппаратуры.

SystemVerilog и VHDL построены на похожих принципах, но их синтаксис весьма различается. Подавляющее большинство коммерческих систем сейчас строится с использованием одного из этих языков описания цифровой аппаратуры, а не на уровне упрощения логических таблиц или выражений. Поэтому в случае активного участия в разработке цифровых схем на современном уровне, следует выучить один из языков описания аппаратуры. Основную информация об особенностях этих языков, можно получить по описаниям спецификаций VHDL и SystemVerilog, изданным IEEE, или многочисленным учебникам [7],[8].

4. ОРГАНИЗАЦИЯ ПРОЦЕССОРА И ОСНОВНОЙ ПАМЯТИ ВМ

В разделе идет речь о машинах с контроллерным управлением, в которых порядок выполнения команд явно задается программой. Машины с потоковым управлением и с запросным управлением в данном курсе не рассматриваются.

Процессор выполняет два основных набора функций:

- обработка данных в соответствии с заданной программой;
- управление всеми устройствами машины.

Обработка данных выполняется программой в виде последовательности команд, представленных в цифровой форме. Структура каждой команды состоит из двух частей: операционной и адресной. Операционная часть задает код операции и режим ее выполнения. Адресная часть содержит сведения о размещении операндов (входные данные и результат операции) в виде:

- непосредственно самих значений данных;
- адресов расположения данных в памяти;
- сведений для определения адресов размещения данных в памяти.

В адресной части могут быть сведения об отсутствии операндов (нуль-адресная или безадресная команда) и адреса от одного (одноадресная команда) до трех операндов (трехадресная команда).

4.1. Типовая структура процессора и основной памяти

Типовая структура центральной части ВМ – процессора и основной памяти– представлена на рис.4.1.

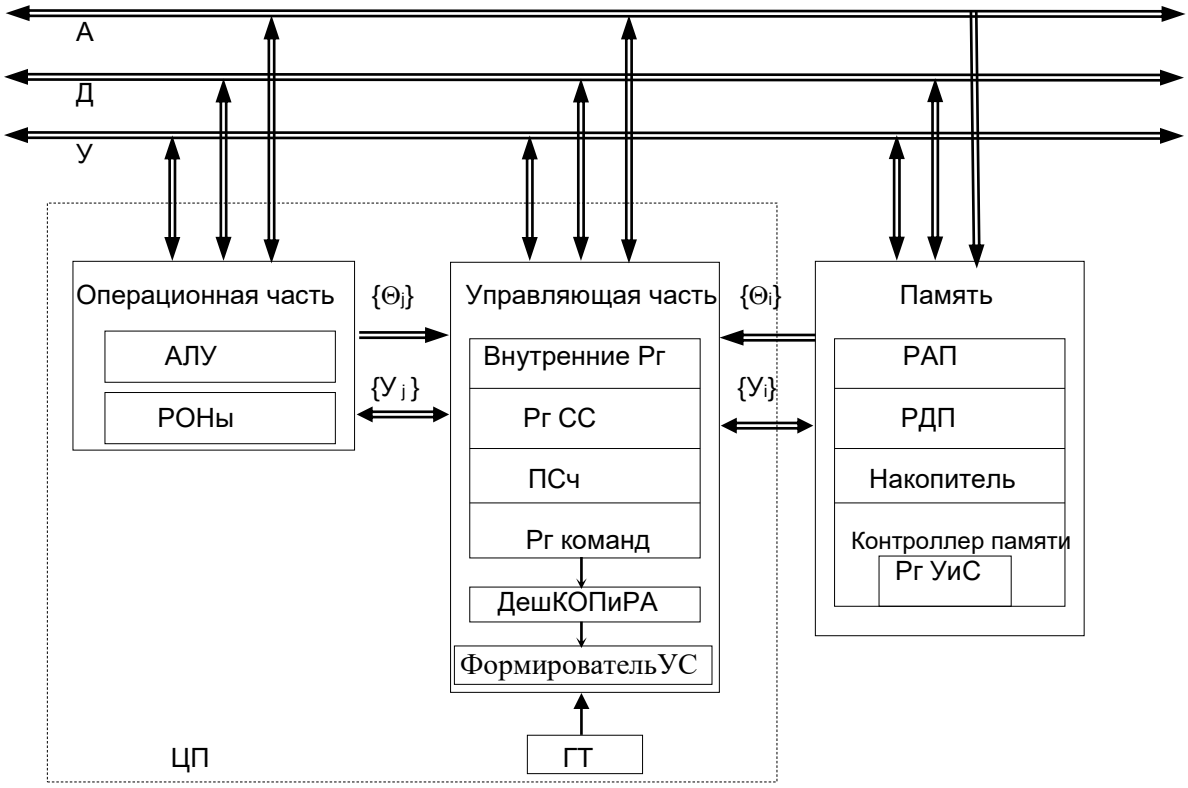


Рис.4.1

В схеме на рис. 4.1 использованы следующие компоненты процессора и памяти:

– АЛУ – арифметико-логическое устройство выполняет операции по обработке данных;

– РОНы – регистры общего назначения (от 8 до нескольких сотен штук) – сверхбыстрая память малой емкости для хранения операндов;

– Рг СС – регистр слова состояния. Содержит текущее состояние процессора, в который входит уровень приоритета текущей программы, биты условий $\{\Theta_j\}$завершения последней команды, режим обработки текущей команды. Возможны следующие режимы обработки (в порядке возрастания уровня приоритета):

1) User Mode – режим пользователя; в этом режиме не могут выполнятся системные команды (команды изменения состояния процессора и команды ввода-вывода);

2) SuperVisor Mode – режим супервизора; обеспечивается выполнение всех команд ввода- вывода;

3) Kernel Mode – режим ядра; в нем возможно выполнение всех команд процессора;

- ПСч – программный счетчик. Содержит адрес текущей команды и автоматически наращивается для подготовки адреса следующей команды (исключение составляет команда перехода);
- Рг Команд – регистр команд. Содержит код исполняемой в данный момент команды;
- ДешКОПиРА – дешифратор кода операции и режимов адресации;
- Формирователь УС – формирователь управляющих сигналов { $У_i$ };
- РАП – регистр адреса памяти; РДП - регистр данных памяти;
- Рг УиС – регистр управления и состояния контроллера памяти.

Одной из основных частей процессора является тракт обработки данных, показанный на рис.4.2 и включающий операционные элементы: набор РОНов, два входных регистра АЛУ, блоки выполнения вычислений в АЛУ и выходной регистр АЛУ. Результат вычислений может помещаться в один из РОНов или сохраняется в памяти. Быстродействие тракта данных во многом определяет производительность компьютера.

4.2. Основной цикл работы процессора. Конвейерное исполнение команд

В основной цикл, реализуемый процессором при выполнении машинной команды, входят следующие основные этапы:

1. Выборка команды (Instruction Fetch).
2. Декодирование команды (Instruction Decoding).
3. Выборка операндов из памяти (Operand Fetch).
4. Исполнение операции (Execution).
5. Сохранение результата (Store).

Обязательными для любой команды являются этапы 1 и 2. Большинство этапов для выполнения требуют как минимум один цикл памяти. После выполнения основного цикла процессора, как правило, происходит проверка запроса программного прерывания. Вообще, в процессе выполнения команд программы возможно появление запросов прерываний двух видов:

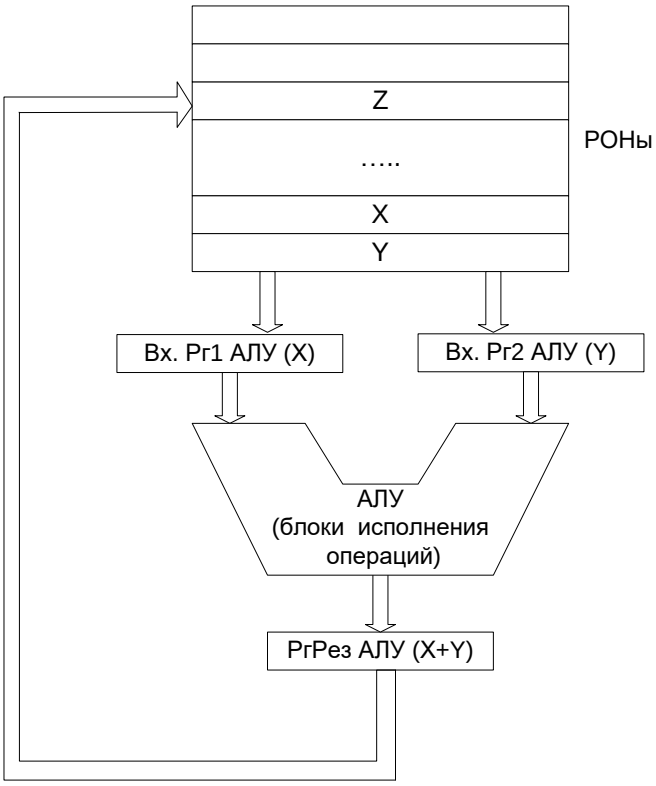


Рис.4.2

– запрос программного прерывания, который обслуживается процессором путем выполнения специальной программы – обработчика прерываний, требует сохранения текущего состояния управляющих и операционных узлов процессора и поэтому проверяется и обслуживается только после завершения выполнения очередной команды;

– запрос аппаратного прерывания обслуживается специальными аппаратными средствами без участия программы процессора и не требует сохранения его состояния, поэтому может поступать и обслуживаться после завершения любого этапа цикла процессора, прерывая выполнение текущей команды.

Подробнее прерывания будут рассмотрены в разделе 4.3.7.

Для ускорения выполнения команд программы используются выборка команд с упреждением и конвейерный способ выполнения этапов цикла процессора, когда при выполнении очередного этапа текущей команды одновременно происходит выполнение предыдущего этапа следующей команды, как это схематично показано на рис. 4.3. Выборка с упреждением означает предварительное размещение группы команд в буферном наборе регистров или в быстрой памяти, чтобы они имелись в наличии при необходимости.

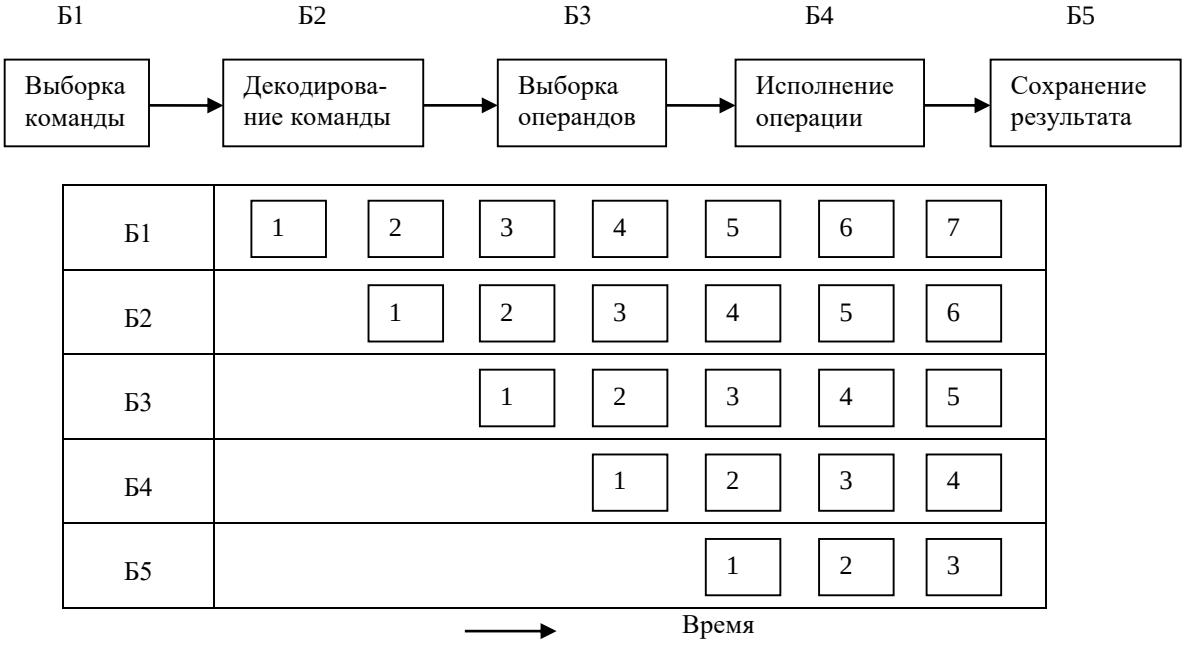


Рис.4.3

Конвейер позволяет найти компромисс между латентностью или временем $T_к$ ожидания выполнения одной команды и пропускной способностью $C_п$ процессора (числом команд, выполняемых в 1 с), существенно повышая его производительность. В идеале, если время выполнения одного этапа цикла процессора, называемого стадией конвейера, равно T_c и конвейер содержит n стадий, то время $T_к = nT_{c,}$ а $C_п = 1/ T_c$ команд в секунду. В действительности идеальная эффективность конвейера, особенно для компьютеров с CISC-архитектурой, недостижима по целому ряду причин:

– на разных стадиях конвейера должно использоваться различное оборудование и завершение текущей стадии у одной команды должно освобождать оборудование для выполнения следующих команд, а это не всегда выполняется – стадии Б1, Б3 и Б5 требуют использования средств работы с памятью;

– длительности выполнения стадий для разных команд могут существенно различаться из-за разницы в длине команды, различий в способах адресации операндов и из-за разных времен выполнения операций;

– возникают конфликты *при чтении данных после записи*, когда одна из команд зависит от результата предыдущей команды, еще не записанного в память или регистровый файл, и это требует приостановки конвейера, пока данное не будет сохранено;

– возникают *конфликты по управлению* при выполнении условных переходов, когда нужно выбрать из памяти команду, а решение о том, какую именно, еще не принято.

Конфликты при чтении данных после записи можно разрешить при помощи ***байпаса*** (обход, шунт), позволяющего обойти избыточные элементы на пути доставки данных. Если выход одной из стадий конвейера требуется на входе другой стадии, данные могут быть переданы через байпас-шину за время в один такт [4].

Конфликты по управлению разрешаются путем предсказания того, какая именно команда должна быть выбрана, и очисткой конвейера от ошибочно выбранных команд, если предсказание не сбылось. Количество команд, от которых придется очистить конвейер в случае неправильно предсказанного перехода, минимизируется путем перемещения логики вычисления условия переходов в начало конвейера.

Для дальнейшего повышения эффективности конвейера можно применять следующие способы:

1. Поделить конвейер на большее количество стадий, каждая из которых будет содержать меньше логики и, следовательно, может работать быстрее. В наши дни нередко используются конвейеры в 10–20 и более стадий. Максимальное количество стадий ограничено увеличением зависимостей внутри процессора и соответственно числа конфликтов. Кроме того, увеличивается и стоимость разработки, так как приходится добавлять регистры между стадиями и логику для разрешения более сложных конфликтов.

2. Использовать не один, а два конвейера, как это сделано в Pentium [2] и показано на рис. 4.4. Здесь общий блок выборки команд берет из памяти сразу две команды и помещает в свои конвейеры. Однако, не любые команды можно

«спаривать» на разных конвейерах. Например, если одна команда использует результат, получаемый другой, то их «спаривать» нельзя. Есть и другие ограничения. В процессоре Pentium два конвейера не являются равноправными. Один из них (U) может принять любую команду, а другой (V) – только удовлетворяющую условиям «спаривания». Если эти условия не выполняются, следующая команда тоже помещается на U-конвейер, а V-конвейер пропускает такт. По этой причине производительность двухконвейерного процессора выше, чем у одноконвейерного, но менее, чем в 2 раза.

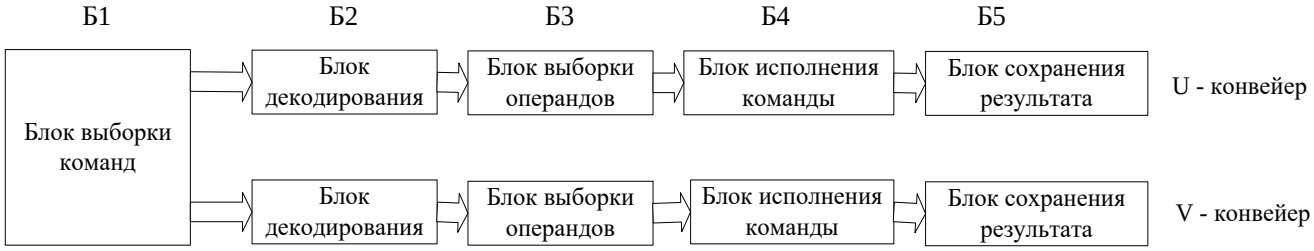


Рис.4.4. Двойная конвейеризация в Pentium

3) Еще более кардинальный способ – использовать один конвейер с большим количеством операционных блоков (ОБ), как показано на рис.4.5.

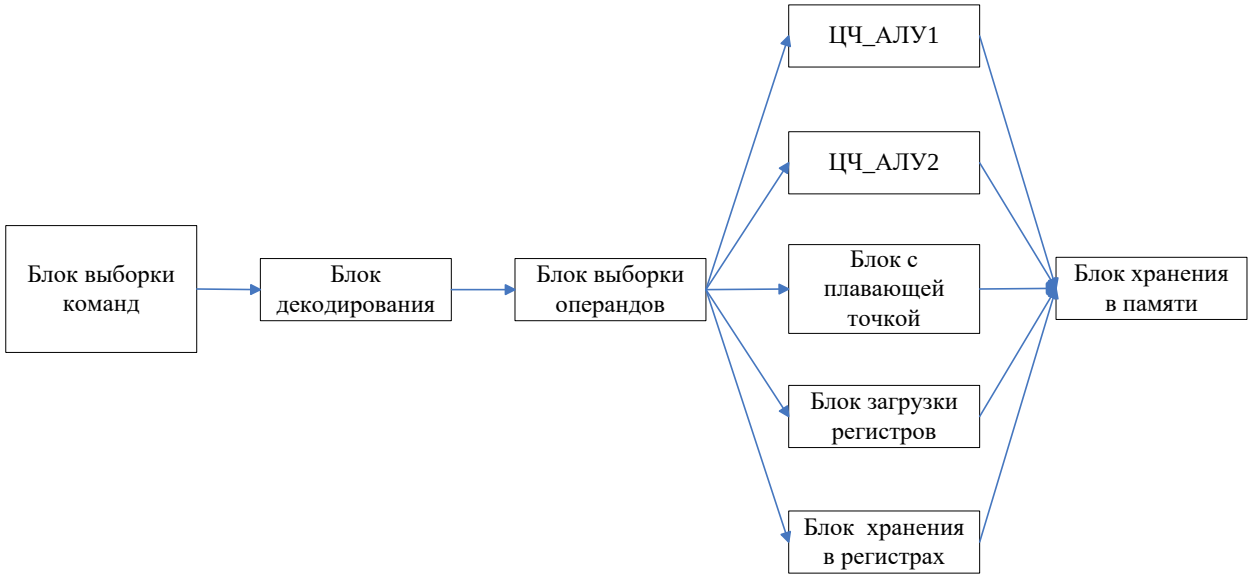


Рис.4.5. Суперскалярная архитектура в Pentium II

Этот подход назван суперскалярная архитектура. Его использование связано с тем, что на стадии исполнения операций большинству ОБ требуется

гораздо больше времени, чем для более простых стадий. На этой стадии может использоваться несколько АЛУ и блоков доступа к памяти.

4.3. Организация процессора и памяти в архитектуре Intel X86

В процессоре Intel 8086 длина слова составляет 16 бит или 2 байта. Минимально адресуемой и обрабатываемой единицей информации является байт, при этом адрес слова совпадает с адресом младшего байта и является четным. При разрядности адреса 16 бит максимальная емкость прямо адресуемой памяти составляет 2^{16} = 64 Кбайт. Для расширения адресуемого пространства памяти используется ее разбиение на блоки (сегменты), называемое сегментированием памяти. Каждый сегмент имеет произвольную длину, не превышающую 64 Кбайт. Адрес байта в сегментированной памяти задается двумя составляющими: сегментная часть (16 бит), определяющая адрес начала сегмента, и смещение (16 бит) байта в пределах сегмента. При этом адрес представляется в виде пары Сегмент (Segment) : Смещение (Offset). Начало размещения сегмента выравнивается на границу блока памяти из 16 байт, называемого «параграфом». Соответственно, физический адрес памяти получается путем суммирования сегмента, сдвинутого на 4 бита влево, со значением смещения. Результатом такого суммирования является 20-битный физический адрес, чем обеспечивается адресация 1 Мбайт памяти.

Пример. Пусть Segment = DCBA, Offset = 5678.

Физический адрес Adr будет определяться суммой

Adr = DCBA0 + 5678 = E2218

Начиная с модели Intel 386, используется 32-битная архитектура IA-32 с длиной слова 32 бит и адресуемой емкостью памяти 4 Гб. Эта и последующие модели процессоров объединены названием Intel X86. В начале двухтысячных годов были разработаны 64-битные модели процессоров AMD 64 и Intel 64 (Itanium), соответствующие архитектуре IA-64.

4.3.1. Регистры процессора

Процессор 8086 имеет 14 целочисленных регистров, используемых для управления выполняющейся программой, для адресации памяти и для обеспечения арифметических вычислений. Каждый регистр имеет длину в одно слово (16 бит) и адресуется по имени. Кроме того, для обработки вещественных

чисел в математическом сопроцессоре используются еще 11 регистров, которые будут рассмотрены в разделе 4.3.6.

Регистры общего назначения (РОН)

При программировании на ассемблере регистры общего назначения являются «рабочими лошадками». Все команды так или иначе изменяют значения регистров, и всегда быстрее и удобнее обращаться к регистру, чем к памяти. Особенность РОН состоит в том, что возможна их адресация как целого слова, так и любой однобайтовой части. Левый байт является старшей частью (high), а правый – младшей частью (low). Например, двухбайтовый регистр CX состоит из двух однобайтовых: CH и CL, и ссылки на регистр возможны по любому из этих трех имен. В состав РОН входят:

1. Регистр AX (AH | AL) является основным сумматором (аккумулятором) и применяется для всех операций ввода-вывода, арифметических операций и некоторых операций над строками. Например, команды умножения и деления предполагают использование регистра AX.

2. Регистр BX (BH | BL) является базовым регистром. Этот РОН может использоваться в качестве «базы» для расширенной адресации при доступе к структурам данных. Другое его общее применение – участие в вычислениях.

3. Регистр CX (CH | CL) является счетчиком. Он необходим для управления числом повторений циклов и для операций сдвига влево или вправо. Регистр CX также используется для вычислений.

4. Регистр DX (DH | DL) является регистром данных. Он применяется для вычислений, для некоторых операций ввода/вывода и тех операций умножения и деления над большими числами, которые используют регистровую пару DX и AX.

Регистры AX, BX, CX и DX могут использоваться без ограничений для любых целей – временного хранения данных, аргументов или результатов различных операций, а также для сложения и вычитания как 8- , так и 16-битовых значений.

Другие четыре регистра общего назначения: SI (индекс источника), DI (индекс приемника), BP (указатель базы), SP (указатель стека) – имеют более конкретное назначение и могут применяться для временного хранения любых переменных,

только когда они не используются по назначению. Индексные регистры примняются для расширенной адресации при доступе к структурам данных, а регистровые указатели SP и BP обеспечивают доступ к данным в сегменте стека.

5. Регистр SI применяется для некоторых операций над строками и в данном контексте регистр SI связан с регистром DS.

6. Регистр DI такжприменяется е для строковых операций, но в данном контексте регистр DI связан с регистром ES.

7. Регистр SP указывает на верхушку стека, для адресации стека связан с регистром SS и обеспечивает использование стека в памяти для временного хранения адресов или данных.

8. Регистр BP используется для доступа к параметрам процедур: данным и адресам, передаваемым через стек.

Начиная с процессора Intel 386, все регистры общего назначения являются 32-битными [9], называются расширенными (Extended), обозначаются EAX, EBX, ECX, EDX, ESI, EDI, EBP и ESP и могут использоваться как универсальные РОН для любых целей.

Сегментные регистры (CS, DS, SS и ES)

Как уже отмечалось, сегментом называется область, которая начинается на границе параграфа, т. е. по любому адресу, который делится на 16 без остатка. Хотя сегмент может располагаться в любом месте памяти и иметь размер до 64Кбайт, он требует столько памяти, сколько необходимо для выполнения программы. При сегментировании памяти для формирования любого адреса применяются два числа – адрес начала сегмента, хранящийся в соответствующем сегментном регистре, имеющем длину 16 бит, и смещение искомого байта относительно начала сегмента. В любой программе имеется три главных сегмента:

1. Сегмент кода – содержит машинные команды, которые будут выполняться. Обычно первая выполняемая команда находится в начале этого сегмента, и операционная система передает управление по адресу данного сегмента для выполнения программы. Регистр сегмента кода (CS) содержит начальный адрес сегмента кода. Этот адрес плюс величина смещения в указателе команд (IP) определяет адрес команды, которая должна быть выбрана для выполнения.

2. Сегмент данных – содержит определенные данные, константы и рабочие области, необходимые программе. Регистр сегмента данных (DS) содержит

начальный адрес сегмента данных. Этот адрес плюс величина смещения, определенная в команде, указывают на конкретную ячейку в сегменте данных.

3. Сегмент стека. Стек содержит адреса возврата как для возврата в операционную систему после завершения программы , так и для возврата в главную программу после завершения подпрограмм. Регистр сегмента стека (SS) содержит начальный адрес в сегменте стека.

4. Дополнительный сегмент данных – используется в операциях над строками для управления адресацией памяти. Для этого регистр дополнительного сегмента (ES) должен быть связан с индексным регистром DI.

Смещение следующей выполняемой команды всегда хранится в специальном регистре – IP (указатель команды), принудительная запись в который приведет к тому, что следующей будет исполнена не команда, расположенная сразу за данной, а команда, смещение которой задано в IP. В общем случае при передаче управления в другой сегмент команды безусловного перехода, вызова подпрограммы и т. п. – осуществляют запись в регистры CS и IP.

Регистр флагов

В процессоре 8086 девять из 16 битов регистра флагов (Flags) являются активными и определяют текущее состояние машины и результатов выполнения операций. Многие арифметические команды и команды сравнения изменяют состояние флагов. Формат Flags и значения его битов следующие:

X	X	X	X	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF
15							8	7							0

CF – флаг переноса (CARRY) – содержит перенос из старшего бита после арифметических операций, а также последний бит при сдвигах или циклических сдвигах;

PF – флаг четности (PARITY) – показывает число единиц в младшем байте результата (1 – четное и 0 – нечетное число);

AF – дополнительный флаг переноса (AUXILARY) – устанавливается при переносе или заёме из бита 4 результата, он используется в двоично-десятичной арифметике;

ZF – флаг нуля (ZERO) – характеризует значение результата арифметических операций и операций сравнения (0 – ненулевой, 1 – нулевой результат);

SF – знаковый флаг (SIGN) – показывет знак результата арифметических операций и операций сравнения (0 – плюс, 1 – минус);

TF – флаг слежения, ловушка (TRAP) – обеспечивает возможность работы процессора в пошаговом режиме с остановкой после выполнения команды;

IF – флаг прерываний (INTERRUPTION) – указывет на возможность внешних прерываний;

DF – флаг направления (DIRECTION)) – указывет направление пересылки или сравнения строковых данных;

OF – флаг переполнения (OVERFLOW) – указывает на переполнение результата при выполнении арифметических команд.

В старших моделях процессоров Intel X86 используются расширенные флаговые регистры EFLAGS (32 бита) и RFLAGS (64 бита).

4.3.2. Организация стека в архитектуре Intel X86

Стек – специальная область памяти, доступная для записи (заполнения) и выборки (выталкивания) данных только с одного конца, называемого верхушкой стека. В процессорах Intel X86 на верхушку стека указывает РОН указатель стека (SP – stack pointer), стек может работать только со словами, заполнение стека происходит в сторону уменьшения адресов с помощью команды PUSH, а освобождение стека (выталкивание данных из стека) – в сторону увеличения адресов командой POP.

По существу команды PUSH AX (сохраняющая содержимое РОН AX в стеке) и POP AX (восстанавливающая содержимое РОН AX из стека) выполняются в два приема:

PUSH AX:	POP AX:
(SP)-2→SP	[SP]→AX
(AX)→[SP]	(SP)+2→SP

При выполнении программ в процессорах Intel X86 стек используется в следующих случаях:

- промежуточного хранения содержимого регистров;
- обмена содержимого регистров;
- сохранения адресов возврата при вызове подпрограмм;
- передачи параметров между вызываемой и вызывающей программами;

– сохранения адресов возврата и регистра флагов при обработке прерываний.

4.3.3. Организация выполняемых программ в MS DOS

Поскольку выполнение лабораторных работ по курсу происходит в среде ОС MS DOS, рассмотрим основные типы исполняемых файлов в MS DOS:

*.com – файлы исполняемых программ типа com;

*.exe – файлы исполняемых программ типа exe.

Файлы типа *.com содержат только исполняемый код без дополнительной информации о программе, формируются в загрузочном виде и не требуют настройки по памяти. Весь код, данные и стек такой программы располагаются в одном сегменте, имеют длину не более 64 Кб и служат для организации простых модулей, ориентированных на модели памяти tiny и small [9].

Файлы типа *.exe могут иметь произвольную длину, они содержат заголовок, в котором описывается размер файла, требуемый объем памяти и таблица загрузки – список команд с абсолютными адресами, требующих настройки при загрузке в зависимости от размещения программы в памяти.

Структура размещения в памяти файла типа COM показана на рис. 4.6.

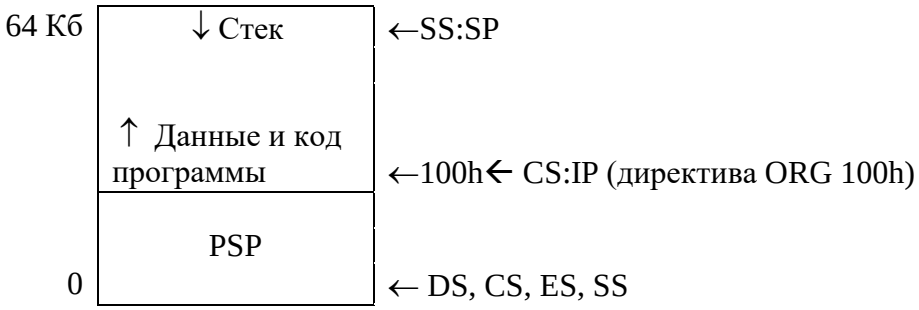


Рис. 4.6

В начальной части com-файла программы (а также и exe-файла) MS DOS размещает специальный блок – префикс сегмента программы ПСП (PSP – program segment prefix), который содержит информацию для доступа программы к параметрам командной строки, к среде окружения, для реакции программы на критические ошибки и управляющие команды типа Ctrl+C. В начальной части PSP размещена команда вызова обработчика прерывания для завершения программы и возврата в DOS. Так как после загрузки все сегментные регистры, включая CS, указывают на начало PSP, а IP = 0, то программа не может исполняться,

начиная с этого адреса, и первой командой делают ORG 100h , устанавливающую CS:IP на конец PSP.

Структура размещения в памяти файла типа EXE показана на рис. 4.7 , а состав ряда полей ПСП – в табл. 4.1.

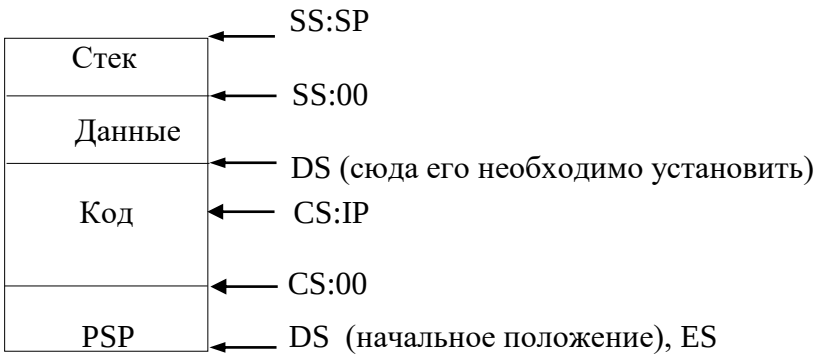


Рис.4.7

Таблица 4.1

0h	DW	Команда INT 20 (16-ный код: CD 20) - вызов прерывания DOS для завершения программы и возврата в DOS
2h	DW	Размер доступной для программы памяти в параграфах
0Eh	DD	Адрес обработчика прерывания по Ctrl^Break (INT 23h)
12h	DD	Адрес обработчика прерывания по критической ошибке (INT 24h)
2Ch	DW	Значение сегментного адреса среды окружения
80h	64W	DTA–буферная область данных / начало командной строки прораммы

4.3.4. Режимы адресации памяти в архитектуре Intel X86

Большинство команд процессора Intel X86 выполняются с аргументами, которые принято называть *операндами.* Операнды в программе могут задаваться следующим образом:

- в регистрах общего назначения;
- непосредственно в коде команды;
- в ячейках памяти, задаваемых в команде прямо или косвенно;
- в портах ввода-вывода.

Для указания места расположения операнда используются 8 режимов адресации (табл. 4.2).

1. Регистровая адресация.

Операнды могут располагаться в любых регистрах общего назначения и сегментных регистрах. В этом случае в операторе программы (на языке ассемблера) указывается название соответствующего регистра.

2. Непосредственная адресация.

Некоторые команды (пересылки, все арифметические команды, кроме деления) позволяют указывать один из операндов непосредственно в операторе программы.

3. Прямая адресация.

Если известен адрес операнда, располагающегося в памяти, можно использовать этот адрес. В реальных программах обычно для задания статических переменных используют директивы определения данных, которые позволяют ссылаться на статические переменные не по адресу, а по имени.

Если селектор сегмента данных находится в DS, имя сегментного регистра при прямой адресации можно не указывать, так как DS используется по умолчанию. Прямая адресация иногда называется адресацией по смещению.

4. Косвенная адресация.

Адрес операнда в памяти можно не указывать непосредственно, а хранить в любом регистре. До процессоров i386 для этого можно было использовать только регистры BX, SI, DI и BP, но потом эти ограничения были сняты и адрес операнда разрешили считывать из EAX, EBX, ECX, EDX, ESI, EDI, EBP и ESP (но не из AX, CX, DX или SP напрямую – надо использовать EAX, ECX, EDX, ESP соответственно или предварительно скопировать смещение в BX, SI, DI или BP). Как и в случае прямой адресации, DS используется по умолчанию, но не во всех случаях: если смещение берут из регистров ESP, EBP или BP, то в качестве сегментного регистра используется SS. В реальном режиме можно свободно пользоваться всеми 32-битными регистрами, надо только следить, чтобы их содержимое не превышало границ 16-битного слова.

5-6. Базовая или индексная адресация.

Такая форма адресации используется в тех случаях, когда в регистре находится адрес начала структуры данных, а доступ надо осуществить к какому-нибудь элементу этой структуры. Другое важное применение адресации по базе со сдвигом – доступ из подпрограммы к параметрам, переданным в кадре стека, используя регистр BP (EBP) в качестве базы и номер параметра в качестве смещения.

До процессора i386 в качестве базовых и индексных регистров можно было использовать только регистры BX, BP, SI или DI и сдвиг мог быть только байтом или словом (со знаком). Начиная с процессоров iX86, можно дополнительно использовать EAX, EBX, ECX, EDX, EBP, ESP, ESI и EDI, так же как и для обычной косвенной адресации. С помощью этого метода можно организовывать доступ к одномерным массивам байт: смещение соответствует адресу начала массива, а число в регистре – индексу элемента массива, который надо использовать. Очевидно, что если массив состоит не из байт, а из слов, придется умножать базовый регистр на два, а если из двойных слов – на четыре. Для этого предусмотрен следующий специальный метод адресации.

7. Индексная адресация с масштабированием

Этот метод адресации полностью идентичен предыдущему, за исключением того, что с его помощью можно прочитать элемент массива слов, двойных слов или учетверенных слов, просто поместив номер элемента в регистр:

```
mov ax, [esi*2]+2  .
```

Множитель, который может быть равен 1, 2, 4 или 8, соответствует размеру элемента массива – байту, слову, двойному слову, учетверенному слову. Из регистров в этом варианте адресации можно использовать только EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, но не SI, DI, BP или SP, которые можно было использовать в предыдущих вариантах.

8. Адресация по базе с индексированием и масштабированием.

Это самая полная возможная схема адресации, в которую входят все случаи, рассмотренные ранее, как частные. Смещение может быть байтом, словом или двойным словом. Если регистры EBP или ESP используются в роли базового

регистра, селектор сегмента операнда берется по умолчанию из регистра SS, во всех остальных случаях – из DS.

Таблица 4.2

Режим адресации		Описание в Ассемблере	Регистр сегмента	Пример использования
Код	Название			

0	Регистровая	EAX/AX/AL	–	MOV EAX, EDX; DEC CX
				MOV AL, CH ; PUSH DS
				IN AL, DX

1	Непосредственная	Константа или номер (порта ввода-вывода)	–	MOV AX, 100h;
				AND EAX, 0000FFFFh
				ADD AL, -30h ; OUT 21h,

2	Прямая	Имя (метка) смещения в памяти	DS	MOV BL, Mem_B1 MOV AX, OFFSET Table

3	Косвенно-регистровая	[BX], [SI],
		[BP],
		[DI],
		[EAX] и др.
			DS
			SS
			DS(ES)
			DS
				MOV AL, [SI]
				MOV CX, [BP]
				MOV AX, ES:[BX]
				ADD EAX, [EDX]

4	Базовая	[BX] + смещение, [BP] + смещение	DS SS	MOV AX, CS:[BX] + 4 MOV CX, PAR_TAB[BP]
5	Индексная	[SI] + смещение, [DI] + смещение	DS DS(ES)	CMP ‘A’, STR1[SI] MOV AL, BYT_TAB[DI]
6	Базово-индексная	BX] [SI] + смещ., BX] [DI] + смещ., [BP] [SI] + смещ., [BP] [DI] + смещ.	DS DS SS SS	MOV AX, ES:VALUE[BX][DI]
7	Индексная с масштабированием	[ERg* m] + смещ.	DS SS – для EBP,ESP	MOV CX, WORD_TAB[ESI*2]
8	Базово-индексная с масштабированием	[ERg1] [ERg2* m] + смещение	DS SS – для EBP,ESP	MOV BX, [EDX][ECX*2] + 8

Примечания.

Базовая адресация применяется для работы со строками или записями, при этом в базовый регистр заносится начало структуры или записи, а смещение задает начало некоторого поля записи или структуры (регистр BP используется по умолчанию для доступа к параметрам процедур через кадр стека). Индексная адресация применяется для доступа к элементам однородных (обычно одномерных) массивов, смещение задает начало элемента этого массива. Базово-индексная адресация используется при работе с элементами полей записи и для работы с двумерными массивами.

4.3.5. Краткая характеристика системы команд процессоров Intel X86

Формат команды представлен в табл. 4.3.

Таблица 4.3

Префиксы	Переопределение сегмента	Код операции (КОП)	Режим адресации	МИБ	Смещение	Данные
0/1/2*)/3*)	0/1	1/2*)	0/1	0/1*)	0/1/2/4*)	0/1/2/4*)

В первой строке табл. 4.3 указывается имя поля команды, во второй – длина поля в байтах. Знаком *) помечены значения длины поля, используемые, начиная с процессора i386. Максимальная длина команды для процессора i8086 равна 6 байт.

Поле префиксов в процессоре i8086 состоит из 1 байта и может задавать префикс повторения команды, который в ассемблере обозначается как REP (повторять), REPE (повторять по равно), REPNE (повторять по неравно) или префикс LOCK – запрет доступа к шине на время выполнения команды. Начиная с процессора i386, также добавляются однобайтовые префикс размера адреса и префикс размера операнда.

Поле переопределения сегмента используется для жесткого задания регистра сегмента, участвующего в формировании исполнительного адреса, вместо регистра, принятого по умолчанию.

Поле КОП – единственное поле, которое не может быть пустым, оно определяет код операции, которая должна быть выполнена процессором, и тип обрабатываемых данных (байт, слово, двойное слово). Для процессора i8086 поле является однобайтовым, а начиная с процессора i386 – двухбайтовым.

Поле режима адресации служит для задания режима адресации и места размещения операнда: в регистре или в памяти.

Поле МИБ (масштаб, индекс, база) является расширением поля режима адресации, используемым начиная с процессора i386 и задающим при формировании исполнительного адреса операнда масштабный коэффициент, индексный регистр и регистр базы.

Поле «смещение» содержит смещение адреса памяти при использовании прямой адресации.

Поле «данные» содержит значение операнда при непосредственной адресации.

Все команды процессоров iX86 можно разделить на следующие группы:

- команды передачи данных;
- команды арифметических операций над целыми числами;
- логические команды;
- сдвиговые команды;
- команды обработки строк;
- команды передачи управления;
- команды прерываний;
- команды управления флагами;
- команды управления состоянием процессора;
- команды плавающей арифметики;
- команды мультимедийных расширений (MMX – MultiMedia eXtension);
- команды потокового расширения (SSE – Streaming SIMD Extension).

4.3.6 . Арифметическая обработка чисел с использованием математического сопроцессора

Арифметическая обработка числовых данных обычно выполняется с использованием математического сопроцессора (FPU – Floating Point Unit, NPR – Numeric PRocessor), который сначала выполнялся в виде отдельной микросхемы (i8087 – i80387), а начиная с процессора i486DX встраивается в состав основного процессора. Сопроцессор называется так потому, что может работать параллельно с процессором после инициализации. Языки высокого уровня используют его непосредственно, а ассемблер порождает ESC команды.

Независимо от наличия сопроцессора, выполнение команд FPU может быть осуществлено с использованием библиотеки эмулятора: автоматически определяется наличие сопроцессора, и если он есть, то выполняет команды FPU, а иначе их выполнение эмулируется (моделируется на уровне микроопераций) основным процессором. Эмуляция FPU не всегда совместима с некоторыми резидентными программами (одни и те же прерывания используются для эмулятора и резидентных программ).

Назначение математического сопроцессора

Расширение вычислительных возможностей основного процессора – выполнение арифметических операций над целыми и вещественными числами с точностью до 18 десятичных разрядов, вычисление основных математических функций (экспоненты, логарифмы и тригонометрические) и т. д. Применение сопроцессора повышает производительность вычислений в сотни раз.

Типы данных математического сопроцессора

Сопроцессор поддерживает 7 типов данных: три целых (16 бит – Word Integer, 32 бит – Short Integer и 64 бит– Long Integer), 80-битные двоично-десятичные целые (Packed Decimal) и три формата с плавающей запятой, представленые в табл. 4.4.

Таблица 4.4

Разрядность (порядка(p) / мантиссы(m))	Точность в десятичных разрядах	Диапазон изменения
32 (8/24) Короткое вещественное	6 / 7	10^{-38} - 10^{37}
64 (11/53) Длинное вещественное	15 / 16	10^{-308} - 10^{307}
80 (15/65) Расширенное вещественное	18 / 19	10^{-4932} - 10^{4931}

Сопроцессор выполняет все вычисленя в 80-битном раширенном формате, а 32-битный и 64-битный форматы используются для обмена данными с основным процессором и памятью. Кроме обычных чисел сопроцессор использует специальные данные, получаемые при выполнении операций:

	sgn	p	m
Положительная бесконечность	0	11 …1	00 … 0
Отрицательная бесконечность	1	11 …1	00 … 0
Неопределенность	1	11 …1	10 … 0
Не число	0	11 …1	XX…X ,

где sgn – знаковый бит, X…X $\neq 0$

Регистры математического сопроцессора

Сопроцессор предоставляет для хранения и обработки данных восемь дополнительных 80-битных регистров R0–-R7, организованных в виде закольцованного аппаратного стека ST(0) – ST(7), вершина которого обозначается ST, ST(0) или TOP, а более глубокие регистры ST(1) – ST(7). Так, например, если ST = R5, то ST(1) = R6, ST(2) = R7, ST(3) = R0 и т. д.

Кроме того, сопроцессор использует пять вспомогательных регистров:

1. Регистр управления CR – задает режим обработки данных: маскирование ошибок (некорректная операция, деление на 0, переполнение и т. д.), точность обработки (расширенная, двойная, одинарная), способы округления (к ближайшему числу, к нулю, к $+\infty$ или к $-\infty$).

2. Регистр состояния SR – содержит: флаги особых случаев, возникающих в результате выполнения операций (IE – некорректная операция, DE – денормализованный операнд, ZE – деление на ноль, OE – переполнение, UE – антипереполнение, PE – неточный результат); флаги условий, возникающие при операциях сравнения; поле указателя вершины стека ST или TOP; бит B занятости сопроцессора.

3. Регистр тегов TW – содержит двухбитовое поле для каждого из восьми числовых регистров сопроцессора

TW(i)	ST(i)
00	вещественное число, не равное нулю
01	вещественное число, равное нулю
10	не число
11	не инициализировано

4. Регистр указателя команды FIP – содержит адрес последней выполненной команды.

5. Регистр указателя операнда DIP – содержит адрес операнда последней выполненной команды.

Два последних регистра используются обработчиком исключений для анализа.

При программировании обработки выражений в сопроцессоре они представляются в виде польской инверсной записи (ПОЛИЗ или RPN – Reversed Poland Notation), в которой любое выражение преобразуется к постфиксному виду (бесскобочная запись).

Примеры:

f:= (a + b) * pi – c преобразуется к виду ab + pi * c – f :=

f:= atan (sqrt (x^2 / (x^2 - 1)) преобразуется к виду xx * xx * 1 - / sqrt atan f :=

Фрагмент программы на Ассемблере, реализующей с помощью FPU вычисление выражения из первого примера, имеет вид.

```
Data   Segment
a        DQ    2.5
b        DQ    6.0
c        DQ    3.5
f        DQ     ?
Data  EndS
Code Segment
Assume cs : Code, ds : Data
Expression Proc Far
   Fld    a              ;       a → st(0)
   Fld    b              ;       b → st(0),  a → st(1)
   Fadd                  ;       st(0) + st(1) → st(0)   или   (a+b) → st(0)
   Fld    pi             ;       pi → st(0),  (a+b) → st(1)
   Fmul                  ;       st(0) * st(1) → st(0)   или   (a+b)*pi  → st(0)
   Fld    c              ;       c → st(0),  (a+b)*pi  → st(1)
   Fsubr                 ;       st(1) – st(0) → st(0)   или   (a+b)*pi - c  → st(0)
   Fstp   f              ;       f:= st(0);  st(1) → st(0)
   Ret
Expression EndP
Code EndS
```

4.3.7. Организация прерываний в процессорах Intel X86

В архитектуре процессоров Intel X86 предусмотрены особые случаи, когда процессор прерывает выполнение текущей программы и передает управление программе, обрабатывающей возникшую ситуацию. Такие особые ситуации назыаются прерываниями и используются как механизм асинхронного взаимодействия процессов, одновременно выполняемых в вычислительной системе, и как еще один способ вызова подпрограмм, называемых обработчиками прерываний. Весь программный интерфейс прикладных программ с функциями операционной системы DOS и сервисами BIOS реализуется именно на основе прерываний. Все системы ввода/вывода в компьютере работают с использованием прерываний. Прерывания вырабатываются контроллером диска, адаптером локальной сети, портами последовательной передачи данных, звуковым адаптером и другими устройствами.

Прерывания бывают двух видов:

1) *программные*, вызываемые по команде INT процессора, и называемые *исключениями;*

2) *аппаратные*, вызываемые внешним по отношению к программе сигналом (например, поступающим от внешнего устройства) и называемые просто *прерываниями*.

Исключения делятся на три типа: ошибки, ловушки и остановы.

Ошибка происходит до выполнения команды, например, если такой команды не существует, или выполняется обращение к запрещенной для доступа области данных. Адрес возврата в этом случае указывает на ошибочную команду.

Ловушка – прерывание, возникающее после выполнения команды (например, для организации режима пошагового выполнения программы под управлением отладчика). Адрес возврата указывает на следующую команду.

Останов – ситуация с неопределенным результатом. Возврат может вообще не происходить.

Команды вызова прерываний **INT N**, получившие свое название до деления особых ситуаций на прерывания и исключения, по существу являются исключениями типа ловушки, поскольку сохраняемый по ним адрес возврата указывает на следующую команду прерываемой программы.

Механизм вызова обработчика прерываний с заданным номером N, основан на использовании специального массива, называемого таблицей векторов прерываний (ТВП), объемом 1 Кб, размещающегося в памяти по адресам 0000h:0000h – 0000h:03FFh. Каждый элемент этого массива представляет собой дальний адрес обработчика прерывания, задаваемый в формате сегмент:смещение (CS:IP) и имеет размер 4 байта. Соответственно, адрес вектора прерываний определяется путем умножения номера N вызываемого обработчика на 4. Всего в ТВП задается 256 векторов прерываний.

Команда INT сохраняет в стеке регистр флагов и дальний адрес возврата, указывающий на следующую команду прерываемой программы, и загружает в CS:IP дальний адрес обработчика прерываний. Для завершения обработчика используется команда IRET, которая восстанавливает путем выталкивания из стека адрес следующей команды прерванной программы и содержимое регистра флагов.

Аппаратные прерывания – прерывания, поступающие от внешних устройств или аппаратных средств процессора асинхронно по отношению к выполняемой программе. Делятся на следующие группы:

1. Прерывания низкого уровня.

Номера веторов прерываний $N_{ВП}$ = 0…1Fh :

– прерывания от схем процессора 00..07 (0 – деление на ноль; 1 – пошаговый режим; 2 – немаскируемое прерывание; 3 – точка останова; 4 – прерывание по переполнению; 5 – печать содержимого экрана, либо прерывание по команде BOUND; 6 – прерывание по отсутствию команды; 7 – прерывание по отсутствию FPU).

– прерывания от контроллера прерываний PIC (Programmable Interrupt Controller - микросхема 8259) 08…0Fh (8 – системный таймер; 9 – клавиатура; 0Ah – обслуживание видеоадаптера; 0Eh – обслуживание жесткого диска). В конце данного раздела рассматриваются особенности использования расширенного контроллера прерываний (APIC) в старших моделях Intel X86.

– прерывания BIOS 10h…1Fh (10h – обмен данными с дисплеем; 13h – обмен данными с диском; 14h – последовательный порт ввода вывода и т.д.).

2. Прерывания среднего уровня.

$N_{ВП}$=20h…5Fh

Здесь прерывание с номером 21h представляет набор функций ОС для работы с файлами управления задачами, выделения и освобождения памяти, а также для работы с виртуальной памятью.

Прерывания пользователей.

$N_{ВП}$=60h…7Fh (прерывания от 70h используются для различных устройств).

3. Прерывания языков высокого уровня.

Используются интерпретатором BASIC и в настоящее время неактуальны.

Уровни приоритета

Прерывания, обслуживаемые контроллером прерываний 8259, имеют 16 уровней приоритета с запросами IRQ0 – IRQ15. Самый высокий уровень приоритета имеет запрос IRQ0.

IRQ0 (INT 8) – прерывания от системного таймера;

IRQ1 (INT 9) – прерывания от клавиатуры;

IRQ2 – размножитель приоритетных уровней;

IRQ8 (INT 70) – прерывания от часов реального времени;

IRQ9 (INT 0Ah) – прерывание обратного хода луча и звуковой карты;

IRQ10- IRQ12 – резерв для дополнительных устройств;

IRQ13 (INT 2) – прерывания по ошибке FPU;

IRQ14 (INT 76h) и IRQ15 (INT 77h) – прерывания от контроллеров жесткого диска IDE1, IDE2.

IRQ3 (INT 0Bh) – прерывания от последовательного порта COM2;

IRQ4 (INT 0Ch) – прерывания от последовательного порта COM1;

IRQ5 (INT 0Dh) – прерывание от LPT2 используется дополнительным устройствами;

IRQ6 (INT 0Eh) – прерывания от магнитного диска;

IRQ7 (INT 0Fh) – прерывания от LPT1 используется дополнительным устройствами.

Маскируемые прерывания

Целью маскирования или запрета прерываний являются запрещения прерывания выполнения критических (по доступу к ресурсам) частей программы или запрещение на некоторое время долго обслуживаемых прерываний, например, в системах реального времени.

Способы реализации.

1. Общее маскирование.

Управление реализуется командами CLI, STI, которые соответственно сбрасывают в ноль или устанавливают в единицу флаг разрешения прерывания IF.

2. Выборочное маскирование.

Засылка определенного кода в регистр маски контроллера 8259. В этом контроллере имеется три основных регистра:

IRR – регистр запроса прерывания;

ISR – регистр обслуживания прерывания (порт 20h);

IMR – регистр маскирования прерывания (порт 21h.

К регистру IRR подключены все линии запросов IRQ0..IRQ15. Регистр ISR хранит приоритет текущего обслуживаемого прерывания. Если запрос прерыва-

ния, поступивший в регистр IRR, не замаскирован, то происходит сравнение приоритетов PR_{IRR} и PR_{ISR} . Если $PR_{IRR} > PR_{ISR}$, то поступивший запрос принимается на обслуживание.

Например, запрет прерываний от жесткого диска обеспечивается командами

```
mov al, 01000000b;  маскируются запросы прерывания от жесткого диска
out 21h
------
mov al, 0
out 21h
```

Разработка собственных прерываний

Причины разработки: необходимость создания собственной подпрограммы, резидентной в памяти и доступной из любой программы; необходимость дополнения функций существующих прерываний; использование холостых прерываний, телом которых является IRET.

Существует два способа заполнения вектора прерывания адресом своего обработчика:

- низкоуровневый – командой mov записать по адресу вектора прерывания адрес сегмента и смещения обработчика;

- с использованием функций операционной системы: функции 25 и 35 прерывания Int 21h позволяют устанавливать новое и получать старое значение адреса обработчика вектора прерывания.

Пример. Пусть нам требуется установить собственный обработчик прерывания по номеру N=60 :

```
Data Segment
    old_cs dw 0; буфер для хранения сегмента и
    old_ip dw 0; смещения старого вектора прерывания (ВП)
Data ENDS
Code Segment
;сохранение адреса старого обработчика
    mov ax, 3560h
    int 21;      старый_ВП_cs→es, старый_ВП_ip→bx
    mov old_cs, es
```

```
    mov old_ip, bx
;задание адреса нового обработчика в ВП 60h
    push ds
    mov dx, offset New_sub
    mov ax, seg    New_sub
    mov ds, ax
    mov ax, 2560h
    int 21h
    pop ds
;новый обработчик прерывания 60h
New_sub proc far
    push ax
    ; тело нового обработчика
    pop ax
    mov al, 20 ; загрузка в регистр ISR кода 20
    out 20h    ; для разрешения прерываний более низкого уровня
    iret
New_sub ENDP
;восстановление старого ВП
    lds dx, DWORD PTR OLD_CS
    mov ax, 2560h
    int 21h
Code ENDS
```

Возможные проблемы:

– если данные передаются через память, нужно тщательно следить за содержимым регистра DS; лучше данные передавать через регистры или стек;

– если возможно прерывание обработчика через Ctrl+Break, необходимо предусмотреть восстановление адреса старого обработчика;

– нужно минимизировать код обработчика прерывания, так как на время его выполнения может быть запрещено выполнение других прерываний.

Перекрытие обработчика прерываний

Существующие обработчики прерываний DOS и BIOS сложно поддаются модификации на уровне исходных кодов и для добавления новых функций требуется реализовать следующий механизм:

– создать новый обработчик прерываний (реализующий дополнительные функции), который вызывает старый (системный), размещенный по новому неиспользованному ранее вектору в диапазоне 60h…70h;

– перенести адрес старого обработчика прерываний в новый вектор прерывания;

– изменить вектор прерывания с системным номером таким образом, чтобы он указывал на новый обработчик прерывания;

– завершить программу установки нового обработчика и оставить ее резидентной в памяти.

Возврат после завершения старого обработчика может происходить либо в новый обработчик командами

push f

call old_handler ,

если дополнительные функции реализуются после выполнения старого обработчика, либо в вызывающую процедуру командой

jmp cs:old_handler ,

если дополнительные функции реализуются до выполнения старого обработчика.

Разработка резидентных обработчиков прерываний

Такие обработчики называются TSR-процедуры (Terminate and Stay Resident – завершить и оставить в памяти). Для оставления обработчика в памяти перед его завершением командой ret следует вызвать прерывания int 27h (более старая версия) или int 31h (более новая версия).

Резидентный обработчик обычно пишется в виде модуля типа com и для его разработки необходимо:

– наличие свободного вектора прерывания и указание метки конца обработчика для int 27h (определение длины обработчика +100PSP);

– для int 31h длина обработчика задается в параграфах.

Для минимизации длины кода обработчика инициализирующую часть выносят за его пределы.

```
Cod_s    Segment
Begin:   jmp short set_up; переход на инициализацию
Rezid_h: proc far
         push ds
;тело процедуры обработчика прерывания
         pop ds
         iret
Final    EQU $;          текущее значение счетчика размещения
Rezid_h  ENDP
Set_up:  mov dx, offset Rezid_h
         mov ax, 2568h;    задание свободного ВП с номером 68h
         int 21h
;завершение с оставлением в памяти
         lea dx, Final
         int 27h
         ret
Cod_s    ENDS
```

Расширенный контроллер прерываний

В 90-х годах прошлого века компания Intel разработала новую версию контроллера прерываний для использования в многоядерных/многопроцессорных системах, которая получила название контроллер APIC (Advanced Programmable Interrupt Controller – расширенный программируемый контроллер прерываний) и впервые стала применяться в компьютерах на базе процессора Pentium (P54C).

Преимущества расширенного контроллера прерываний:

- возможность реализации межпроцессорных прерываний – сигналов от одного процессора другому;
- поддержка до 256 входов IRQ, в отличие от 8-16 в 8259 PIC контроллере;
- крайне быстрый доступ к регистрам текущего приоритета прерывания и подтверждения прерывания. Контроллер 8259 PIC исполнялся как устройство шины ISA с очень медленным доступом к его регистрам (порт 0x20).

Контроллер прерываний APIC предназначен для обработки аппаратных прерываний, поступающих от устройств, и состоит из двух основных компонентов – это так называемый контроллер локального APIC (Local APIC или LAPIC), располагающийся в самом процессоре (точнее говоря, в каждом процессорном ядре) и чип контроллера ввода/вывода APIC(I/O APIC), располагающийся на материнской плате. Таким образом, количество локальных контроллеров прерываний LAPIC соответствует количеству процессорных ядер, установленных в компьютере. Контроллеров I/O APIC в системе также может быть несколько – до 8 штук. Проводники IRQ от устройств подсоединены к IO APIC. Для общения local APIC и IO APIC, а также local APIC различных ядер друг с другом, используется FSB шина многопроцессорной системы, также используемая для соединения процессоров и контроллера памяти.

Технология APIC разрабатывалась преимущественно для работы на многопроцессорных системах, где требуется надежная система распределения аппаратных прерываний, идущих от устройств к процессорам. На сегодняшний день система контроллеров LAPIC используется как на однопроцессорных, так и на многопроцессорных системных платах компьютеров. Для того, чтобы технология APIC работала, требуется поддержка со стороны программного обеспечения, прежде всего, операционных систем, начиная с Microsoft Windows 2000 и более поздних. Если процессор имеет более 1 ядра, в настройках BIOS следует включить APIC MODE, переведя его в положение Enabled.

В последнее время наблюдается тенденция к отказу от IO APIC, как и проводников IRQ, и переходу на MSI (Message Signaled Interrupts – Прерывания, инициируемые сообщениями). Это альтернативная форма прерываний, доступная в PCI версии 2.2 и более поздних, а также обязательная в PCI Express любых версий. Вместо присваивания фиксированного номера запроса на прерывание, устройству разрешается записывать сообщение по определённому адресу системной памяти, отображенному на аппаратуру локального контроллера прерываний (local APIC) процессора. Для записи сообщения используется тот же механизм захвата шины (bus mastering), что и для DMA. При этом каждое устройство, использующее MSI, может иметь от одной до тридцати двух уникальных областей памяти.

Достоинства MSI:

– возможность полного отказа от проводников IRQ от устройств и разъемов PCI до главного контроллера прерываний (IO APIC), что упрощает материнскую плату.

– в многопроцессорных и многоядерных системах устройства, использующие несколько областей MSI, получают возможность самостоятельно выбирать процессор/ядро для обработки конкретного прерывания, причем делать это полностью на уровне аппаратуры без исполнения программного кода. Это позволяет оптимизировать работу путём размещения большей части структур драйвера устройства и связанного с ним ПО (сетевых протоколов и т. д.) в кэше конкретного процессора или же в его «ближней» NUMA-памяти.

MSI поддерживается в операционных системах Microsoft Windows 7 и более поздних, а также в ядре Linux начиная с версии 2.6.8.

4.3.8. Эволюция микроархитектуры Intel X86

Первый однокристальный четырехбитный микропроцессор 4004 фирма Intel разработала в 1971 году с целью использования в качестве универсального контроллера для калькуляторов. Он содержал 2300 транзисторов на кристалле кремния площадью 12 мм^2, был изготовлен по 10 мкм технологии и работал с тактовой частотой 750 кГц. Успех процессора 4004 вдохновил Intel на создание 8-битного микропроцессора 8008, а затем и 8080, который, в свою очередь, превратился в 16-битные процессоры 8086 (1978 год) и 80286 (1982 год). В 1985 году Intel представила процессор 80386, который расширил архитектуру 8086 и превратил ее в 32-битную, ознаменовав появление архитектуры Intel X86.

В табл. 4.5 показана эволюция микропроцессоров Intel X86. За 40 лет после создания 4004, линейные размеры транзисторов уменьшились в 160 раз, число транзисторов на кристалле увеличилось на пять порядков, а рабочая частота выросла почти на четыре порядка. Ни в одной другой области техники никогда не было такого удивительного прогресса в столь короткий промежуток времени.

Таблица 4.5

Процессор	Год	Технология (мкм)	Число транзисторов (млн)	Частота (Мгц)	Микроархитектура
80386	1985	1.5–1.0	0,275	16–25	Многотактная
80486	1989	1.0–0.6	1,2	25–100	Конвейерная
Pentium	1993	0.8–0.35	3,2–4,5	60–300	Суперскалярная
Pentium II	1997	0.35–0.25	7,5	233–450	Внеочередная версия
Pentium III	1999	0.25–0.18	9,5–28	450–1400	Внеочередная версия
Pentium 4	2001	0.18–0.09	42–178	1400–3730	Внеочередная версия
Pentium M	2003	0.13–0.09	77–140	900–2130	Внеочередная версия
Core Duo	2005	0.065	152	1500–2160	Двухядерная
Core 2 Duo	2006	0.065–	167–410	1800–3300	Двухядерная
Core i3-i7	2009	0.045–	382–731	2530–3460	Многоядерная

Intel 80386 был многотактным процессором. Программируемая логическая матрица (ПЛА), содержащая *микрокод*, использовалась устройством управления для того, чтобы определить порядок перехода между состояниями автомата управления. Блок управления памятью управлял доступом к внешней памяти.
У процессора Intel 80486 производительность значительно улучшилась благодаря использованию конвейерной обработки. По-прежнему выделяются тракт данных, устройство управления и ПЛА с микрокодом.

Процессор Pentium использовал суперскалярную архитектуру и мог запускать на выполнение две команды одновременно. Pentium использовал раздельные кэши команд и данных. Он также использовал предсказание переходов для уменьшения потерь времени из-за команд условного перехода.

Процессоры Pentium Pro, Pentium II и Pentium III имели общую микроархитектуру под кодовым названием P6, обеспечивавшую внеочередное выполнение команд. Сложные команды IntelX86 разбивались на одну или несколько микроопераций (МОп), похожих на команды MIPS. Затем эти МОп выполнялись на быстром вычислительном ядре с 11-стадийным конвейером, обеспечивавшем внеочередное выполнение команд.

В Pentium III 32-битный тракт данных назван целочисленным функциональным блоком (ФБ). Тракт данных для операций над числами с плавающей запятой (ПЗ) назван блоком вычислений с ПЗ. В процессоре также имеется

ФБ SIMD для пакетной арифметики с целыми числами и числами с ПЗ. Площадь кристалла для организации внеочередного запуска команд на выполнение, оказалась больше, чем площадь для самого выполнения команд. Кэши команд и данных были увеличены до 16 Кб каждый, и на кристалл также была добавлена более большая, но и более медленная кэш-память второго уровня размером 256 Кб.

Процессор Pentium 4 был очередным процессором с внеочередным выполнением команд, но с конвейером очень большой длины (сначала конвейер имел 20 стадий, а в поздних версиях – 31 стадию), что позволило достичь частоты, превышающей 3 ГГц. Для дешифрации трех команд за один такт процессор проводил разбиение команд на простые микрооперации и сохранял их в *кэш последовательностей микроопераций*. Более поздние версии Pentium 4 также использовали многопоточность для увеличения пропускной способности процессора при выполнении нескольких потоков. Процессор Pentium M был усовершенствован за счет увеличения размера кэш-памяти команд и данных до 32 Кбайт, а размера кэш-памяти второго уровня – до двух Мбайт.

Процессор Core Duo представлял собой многоядерный процессор, содержащий два ядра Pentium M, подключенных к общей кэш-памяти второго уровня размером 2 МБ. В 2009 году Intel представила новую микроархитектуру под названием Nehalem как модернизированную версию Core. Эти процессоры, включая Core i3, i5 и i7, реализуют поддержку 64-битных команд. Они имеют от двух до шести ядер, кэш-память 3-го уровня размером 3–15 МБ и встроенный контроллер памяти. Некоторые модели поддерживают гиперпоточность (hyperthreading) – так Intel называет многопоточность с двумя активными потоками, которая, с точки зрения пользователя, удваивает количество ядер.

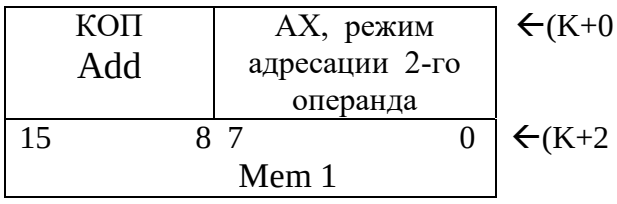
Наряду с процессорами Core фирма Intel по архитектуре Nehalem разрабатывает линейку серверных процессоров Intel Xeon. Этот ряд процессоров включает три линейки E3, E5 и E7, отличающихся от процессоров для настольных ПК:

- увеличенным объёмом кэш-памяти;
- поддержкой многопроцессорных систем большой производительности;
- поддержкой большего объёма памяти и портов ввода-вывода;
- наличием памяти с коррекцией ошибок.

Модели ряда E3 (4 ядра) предназначены для недорогих серверов, ряда E5 (4 –22 ядра) – наиболее универсальные со средней производительностью и ряда E7 – самые производительные модели.

5. УПРАВЛЕНИЕ ВЫПОЛНЕНИЕМ КОМАНД В КОМПЬЮТЕРАХ

Пример: пусть требуется описать этапы цикла процессора iX86 при реализации команды Add ax, Mem1, которая размещается в памяти в двух 16-битных словах с адресами (К+0) и (К+2) в виде



Опишем на уровне элементарных операций основные этапы выполнения команды, применительно к структурной схеме процессора с памятью (рис. 3.1).

1. Выборка и дешифрация первого слова команды:

1) ПСч → РАП ; переслать значение программного счетчика в регистр адреса контроллера памяти
2) Код “Чт” → КонтрЗУ ; загрузить код операции «Чтение» в контроллер памяти
3) “0” → ГОТЗУ, пуск ЗУ ; сбросить флаг «Готовность ЗУ», запустить операцию в памяти
4) Проверить условие ГОТЗУ = 1 ; если условие «ложно», то ожидать завершения операции в памяти, иначе продолжить дальше
5) РДП → РгКом ; переслать содержимое регистра данных контроллера памяти (РДП) в регистр команд процессора
6) (ПСч)+2 → (ПСч) ; подготовить в ПСч адрес следующего слова команды
7) РгКом.КОП→ДешКОП ; поле КОП (код операции) регистра команд подать на вход дешифратора операций
8) РгКом.РА→Деш.РА ; поле РА (режим адресации) регистра команд подать на вход дешифратора режимов адресации.

2. Выборка второго слова команды:

1) ПСч → РАП
2) Код “ЧТ” → КонтрЗУ | Эту последовательность операций
3) “0” → ГОТЗУ, пуск ЗУ | обозначим в дальнейшем
4) Ожидание ГОТЗУ = 1 | как “ЧТ ЗУ” (чтение из памяти)
5) РДП → БуфРг ; переслать содержимое РДП в буферный регистр процессора
6) ПСч + 2 → ПСч ; подготовить в ПСч адрес следующей команды.

3. Выборка операнда из Mem1:

1) БуфРг → РАП ; переслать содержимое буферного регистра процессора в РАП
2) “ЧТ ЗУ” ; чтение из памяти
3) РДП → РгВх АЛУ ; переслать содержимое РДП во входной регистр АЛУ (РгВх).

4. Исполнение операции:

1) РгАХ +РгВх → РгАХ ; суммирование аккумулятора АХ и РгВх АЛУ
2) Учет переносов
3) {Qi} → РгССП ; установка флагов в регистре слова состояния процессора по результату завершения операции.

Для описания элементарных операций, составляющих процесс выполнения команды, будем использовать язык межрегистровых передач (ЯМРП) [10], любой оператор которого описывается в следующем виде:

<ОП_ЯМРП> : : = [метка] условие : список _ действий ,

где условие – булевское выражение, содержащее временную привязку списка действий к такту процессора; список действий – набор микроопераций, выполняемых параллельно в данном такте, а микрооперация – действие, выполняемое аппаратными средствами за один такт генератора.

Совокупность микроопераций, выполняемых за один такт, называют микрокомандой. Последовательность микрокоманд, реализующих всю команду,

называют микропрограммой. Примеры описания элементарных операций, использовавшихся выше при реализации этапов выполнения команды, на языке МРП приведены в табл. 5.1.

Таблица 5.1

Выполняемые операции	Описание операций на ЯМРП
1. ПСч → РАП	ВЫБК.& ГОТЗУ & Т1: ПСч → ША (У1) ША → РАП (У2)
2. Код "Чт." → Контр.ЗУ 0 → ГОТЗУ	ВЫБК.& ГОТЗУ & Т2: "Чт" → ШУ (У3) ШУ → РгУ КонтрЗУ (У4) "0" → РгСостКонтрЗУ.ГОТЗУ (У5)
3. РДП → Рг ком. ПСч + 2 → ПСч Рг Ком. КОП → ДешКОП Рг Ком.РежА → ДешРА	ВЫБК.& ГОТЗУ & Т3: РДП → ШД (У6) ШД → Рг Ком (У7) ПСч + 2 → ПСч. (У8) Рг Ком.КОП → ДешКОП (У9) Рг Ком.РежА → ДешРА (У10)

В табл. 5.1 наряду с использовавшимися выше, применяются обозначения: ВЫБК – этап выборки команды, Ti – i-й такт генератора, Уi – i-й сигнал управления, ША – шина адреса, ШУ – шина управления, ШД – шина данных.

Для формирования сигналов Уi, управляющих выполнением элементарных операций в процессорах, используются два основных способа:

1. Аппаратный, реализуемый на основе жесткой логики (Wired Logic).

2. Микропрограммный, реализуемый на основе программируемой логики (Stored /Programmed Logic).

5.1. Аппаратный способ формирования управляющих сигналов

Аппаратный способ формирования управляющих сигналов (УС) основывается на использовании конечных автоматов, сущность и структура которых рассмотрены в разделе 3.3, в качестве автоматов управления, реализуемых схемным (аппаратным) образом. Как было отмечено в разделе 3.3, автомат управления или его отдельные компоненты (в случае применения декомпозиции автомата) могут быть реализованы в виде автомата Мура с формированием потенциальных выходных сигналов или автомата Мили с формированием импульсных выходных сигналов.

В качестве входных воздействий $\{X_i\}$ автомата для управления выполнением команд компьютера рассматриваются: коды операций, режимы адресации, тактовые сигналы T_i, а также состояния Q_i регистров процессора и других устройств компьютера. Выходные сигналы автомата упраления $\{Y_j\}$ обеспечивают выполнение элементарных операций, требуемых для реализации команд программы, в различных узлах компьютера.

Основным достоинством аппаратного способа формирования УС является максимально достижимое быстродействие управления выполнением команд программы.

В то же время имеется и ряд недостатков аппаратного формирования УС:

1. Основной недостаток – жесткость структуры автомата управления, требующая его переконструирования даже при небольшом изменении состава входных или выходных сигналов, функций переходов или выходов.

2. Синтез и реализация автомата с большим количеством состояний является сложной научно-технической проблемой.

3. Нерегулярность структуры автомата затрудняет его реализацию с помощью серийно выпускаемых интегральных схем (ИС) и требует их выполнения в заказном варианте.

5.2. Микропрограммный способ формирования управляющих сигналов

Микропрограммный способ формирования УС основан на том, что входные сигналы автомата управления можно представлять как адреса некоторой памяти, ячейки которой содержат требуемые комбинации выходных сигналов. Поэтому формирование сигналов управления в текущий момент времени можно представить как выборку по n-битному адресу, каждый бит которого соответствует текущему значению одного из n входных сигналов X_i, одного из 2^n слов памяти, содержащих требуемые m-битные комбинации значений каждого из выходных сигналов Y_j. Тогда проблема формирования сигналов управления выполнением команд процессора сводится к организации *последовательности выборок слов из памяти* по адресам, соответствующим последовательности комбинаций входных сигналов, которая обеспечит требуемую последовательность комбинаций значений управляющих сигналов.

Набор микроопераций (элементарных операий), выполняемых в текущем такте с помощью сформированных в этом такте управляющих сигналов, называется микрокомандой, а последовательность микрокоманд образует микропрограмму выполнения машинной команды. Поэтому память, хранящая набор микропрограмм для выполнения каждой команды процессора, называется микропрограммной памятью, а данный способ управления выполнением команд – микропрограммным управлением (МПУ). Идею построения МПУ впервые предложил M. Wilks (Кембридж) в 50-х годах прошлого века, но активная реализация МПУ началась только в середине 60-х годов по мере развития технологии разработки ИС (в частности, схем памяти).

Структурная схема устройства микропрограммного управления представлена на рис. 5.1.

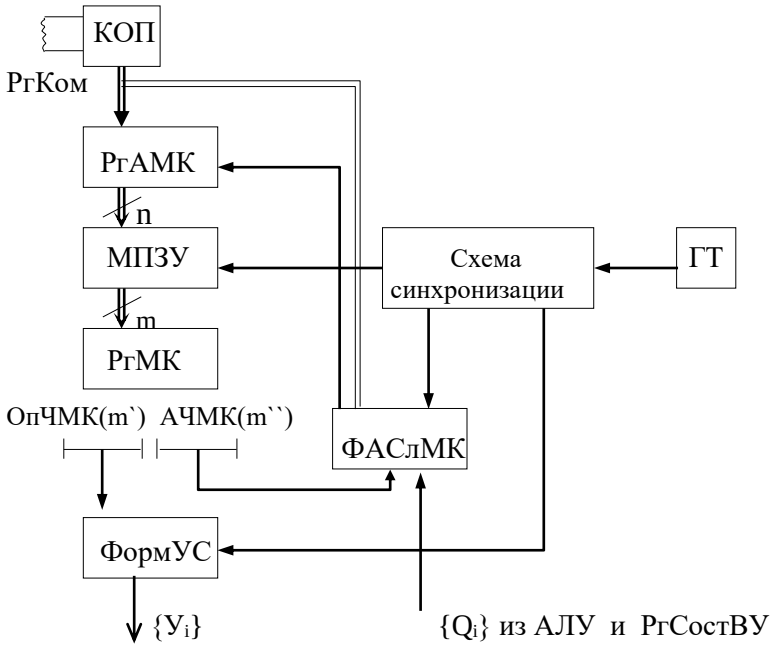


Рис. 5.1

На рисунке приняты следующие обозначения: МПЗУ – микропрограммное запоминающее устройство (память микропрограмм); РгКом и КОП – регистр команд (машинных) и его поле кода операции; РгАМК – регистр адреса текущей микрокоманды длиной n бит; РгМК – регистр текущей микрокоманды длиной m бит, ОпЧМК и АЧМК –операционная (длиной m') и адресная (длиной m") части микрокоманды; ФАСлМК – формирователь адреса следующей микрокоманды;

ФормУС – формирователь управляющих сигналов $У_i$; ГТ– генератор тактов; РгСостВУ – регистр состояния внешнего устройства.

Код операции выполняемой машинной команды подается в РгАМК и определяет адрес в МПЗУ начала микропрограммы, реализующей эту машинную команду. На РгМК выбирается первая микрокоманда, операционная часть которой поступает на формирователь сигналов, управляющих выполнением микроопераций, входящих в состав этой МК, а адресная часть поступает на формирователь адреса следующей МК. Адрес следующей МК либо принудительно задается в коде текущей МК, либо формируется в зависимости от условий $\{Q_i\}$ завершения текущей МК. Сформированный адрес следующей МК поступает на РгАМК и из МПЗУ выбирается следующая МК. Этот процесс повторяется до завершения микропрограммы выполняемой машинной команды, после чего на РгКом выбирается следующая машинная команда, инициирующая выполнение микропрограммы этой команды.Основными достоинствами микропрограммного управления являются:

1. Регулярная структура устройства управления позволяет реализовать его на основе серийных БИС.

2. Большая гибкость в плане изменения или специализации набора команд ВМ путем замены или перезаписи микропрограммной памяти.

3. Возможность рационального распределения аппаратного и микропрограммного управления порождает семейство машин с одним набором команд, но разной производительностью.

4. Возможность эмуляции на микропрограммном уровне новых архитектур машин на имеющейся инструментальной машине.

5. Микропрограммная реализация системных программ (компиляторы, файловые системы) значительно повышает производительность ВМ и широко используется в суперкомпьютерах.

Основные недостатки микропрограммного управления:

1. Снижение производительности процессора из-за замедления выполнения каждой микрокоманды, связанного с обращением к МПЗУ.

2. Увеличение площади кристалла, необходимой для размещения устройства МПУ.

5.2.1. Способы кодирования микрокоманд

Обычно число различных микроопераций в несколько раз превышает число машинных команд. Кроме того, в составе микрокоманды требуется указывать прямой адрес следующей микрокоманды в МПЗУ. Это приводит к тому, что длина микрокоманды $L_{мк}$ может достигать больших размеров (до 200 бит). В то же время эффективно использовать все 200 бит микрокоманды невозможно. Поэтому возрастает роль выбираемого способа кодирования микрокоманд. Различают следующие основные способы.

1. Горизонтальное или унитарное кодирование.

Как видно из рис. 5.2, каждый бит ОпЧМК определяет выполнение только одной микрооперации (МОп). Это позволяет в одной МК задавать любые сочетания МОп и соответствующих им сигналов $У_i$, обеспечивая их параллельное выполнение. Кроме того, отсутствует необходимость в декодировании ОпЧМК. В то же время длина ОпЧМК определяется максимально возможным числом микроопераций:

$$L_{ОпЧМК} = (N_{МОп})_{max} \quad .$$

Это основной недостаток горизонтального кодирования, увеличивающий расход памяти в МПЗУ при низкой эффективности использования разрядов ОпЧМК в конкретной МК.

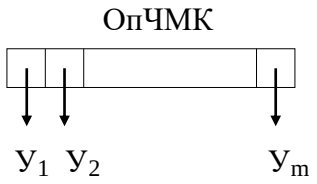


Рис. 5.2

Достоинствами горизонтального кодирования являются:

1) простота и высокая скорость формирования управляющих сигналов;

2) максимальный параллелизм выполнения МОп.

2. Вертикальное кодирование микрокоманд .

При вертикальном кодировании каждой микрооперации присваивается отдельная двоичная комбинация, например, соответствующая двоичному коду

ее номера в общем списке возможных МОп. Разрядность этого двоичного кода будет определять длину ОпЧМК в соответствии с выражением

$$L_{ОпЧМК} = Int\,[\log_2 (N_{МОп})_{max}] + 1,$$ где Int(X) – целая часть числа X .

При этом ОпЧМК будет иметь минимальную длину и это приведет к уменьшению затрат памяти на хранение микропрограмм. С другой стороны, для распознавания кода микрооперации необходим дешифратор МК, преобразующий код МОп в соответствующие УС и требующий на это некоторое время. Кроме того, в каждой МК можно задать только одну МОп, что исключает параллельное выполнение МОп, увеличивает длину микропрограммы и время ее реализации.

3. Смешанное кодирование микрокоманд.

В этом случае весь набор микроопераций делится на группы, связанные с управлением конкретными устройствами (АЛУ, память и т. п.), как показано на рис.5.3. В каждую группу включаются взаимно несовместные МОп, которые не встречаются вместе в одной МК. При этом УС, формируемые в одном и том же такте, оказываются в разных группах и могут выполняться параллельно.

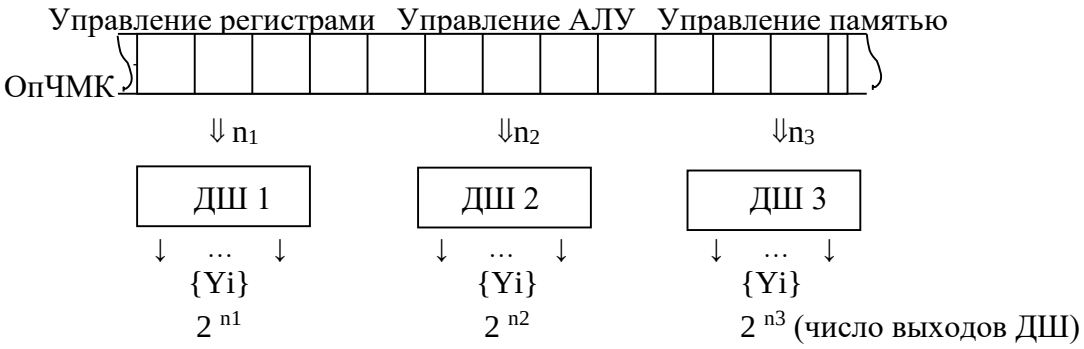


Рис. 5.3

Внутри каждой группы МОп кодируются вертикальным способом и двоичный код группы поступает на соответствующий дешифратор, где выполняется его декодирование. Такой способ смешанного кодирования называется горизонтально-вертикальным, так как выбор группы осуществляется по горизонтальному принципу, а внутри группы используется вертикальное кодирование. При использовании К независимых групп длина ОпЧМК определяется соотношением

$$L_{ОпЧМК} = \sum_{i=1}^{K} n_i \;, \quad где \;\; n_i = Int[Log_2(N_{МОп})_i] + 1 \quad .$$

Способ смешанного кодирования сочетает достоинства и снижает влияние недостатков чисто горизонтального (большая длина МК при низкой эффективности использования разрядов ОпЧМК) и чисто вертикального (увеличение времени выполнения микропрограммы) способов кодирования.

5.2.2. Способы адресации микрокоманд

В зависимости от способа указания адреса очередной МК различают устройства МПУ с принудительным и естественным порядком следования МК. В случае ***принудительной адресации МК*** адрес следующей МК указывется непосредственно в текущей МК. Этот адрес может задаваться безусловно или выбираться в зависимости от условия, определяемого по признакам (флажкам) выполнения текущей МК, что позволяет реализовать ветвления в микропрограммах. ***Естественный порядок следования МК*** предусматривает выборку очередной МК из ячейки памяти с адресом на 1 больше адреса текущей МК. В этом случае роль формирователя адреса следующей МК может выполнять обычный счетчик, необходимость в адресной части МК фактически отпадает и МК может содержать только операционную часть (операционная МК или ОМК). Но в этом случае в микропрограмме (МПМ) нельзя реализовать ветвления. Для обеспечения ветвлений в МПМ надо ввести дополнительный формат МК – управляющие (УМК), содержащие только поле логического условия (ЛУ) и поле адреса очередной МК, к которой осуществляется переход при выполнении этого условия. При невыполнении ЛУ адрес следующей МК равен текущему, увеличенному на 1. Безусловные переходы реализуются УМК с нулевым полем ЛУ. Для различения ОМК и УМК в обоих форматах надо применять однобитовое поле признака.

5.3. Компьютеры с сокращенным набором команд

Компьютеры с сокращенным набором команд (КСНК), более известные по английской аббревиатуре как RISC-компьютеры (RISC – Reduced Instruction Set Computer), воплощают направление развития архитектуры ВМ, связанное с возвращением к принципам аппаратного управления выполнением команд с целью повышения производительности.

Система команд первого и частично второго поколений машин содержали не более пятидесяти команд. Основная проблема, по которой набор команд не

расширялся, – это цена аппаратуры (и управляющая, и обрабатывающая части процессора реализовывались аппаратно), а также необходимость программирования в кодах (программист не мог запомнить большое количество команд). Период доминирования аппаратного управления (50-е – начало 60-х годов) можно назвать «эрой аппаратчиков».

С середины 60-х до 80-х годов доминирует микропрограммное управление выполнением команд, воплощающее «эру программистов», основным лозунгом которой было: «Больше команд хороших и разных!». Этот лозунг соответствовал основным требованиям к процессорам того времени:

1. Минимизация длины кода программы.

2. Упрощение реализации компиляторов за счет снижения семантического разрыва между ЯВУ и машинными командами.

Это вызвало рост набора команд компьютеров за счет увеличения их сложности и увеличения числа форматов от ~50 до ~300 команд (рекордсменом был компьютер фирмы DEC Vax11/780, у него было 303 команды). Компьютеры с большим набором команд и разнообразием их форматов получили название CISC-компьютеров (Complex Instruction Set Computer – машина со сложным набором команд). Для них характерно увеличение сложности и соответственно размеров микропрограммного устройства управления, которое интерпретирует выполнение этих команд. К тому времени благодаря технологическим достижениям тактовая частота процессоров стала достигать 100 Мгц и повышение производительности требовало размещения всех частей процессора на одном кристалле для сокращения длины соединений его элементов. В то же время микропрограммное управление из-за своей сложности стало занимать до шестидесяти процентов площади кристалла, что либо не допускало использования эффективных средств арифметической обработки данных, либо требовало размещения частей процессора на разных кристаллах. Все это приводило к существенному ограничению производительности, увеличивало сроки разработки и снижало выход годных кристаллов.

В 80-х годах рядом исследователей было замечено, что при выполнении большинства программ наиболее активно используется около 30 % сравнительно простых команд арифметики и управления. Постепенно стало формироваться направление развития архитектуры компьютеров, требующее чтобы система команд процессора содержала минимальный набор наиболее часто используемых и

наиболее простых команд (возврат к примерно 50 командам). Это направление получает название компьютеров с сокращенным набором команд или RISC-компьютеров и имеет лозунг: «Проще команды, выше скорость выполнения!».

В результате в конце 80-х гг. благодаря развитию технологии производства СБИС и их удешевлению, а также развитию методов и опыта разработки оптимизирующих компиляторов, постепенно сложились основные принципы (или законы) RISC-архитектур:

1. Основной набор команд не должен интерпретироваться микрокомандами, а должен выполняться аппаратным обеспечением.

2. Все команды должны иметь одинаковую длину и минимальное число форматов (обычно не более 2–3), это упрощает логику управления при выборе и при исполнении команды.

3. Любая команда основного набора должна выполняться за один машинный цикл, обратно пропорциональный тактовой частоте процессора (стандартом является команда сложения регистра с регистром, занимающая от 3–10 нс); это достигается одновременным (параллельным) выполнением максимально возможного числа команд путем конвейеризации или использования нескольких обрабатывающих узлов.

4. Обращение к памяти производится только по специально выделенным командам работы с памятью типа: Load – загрузка и Store – сохранение, а вся обработка данных должна вестись в регистровом формате; при этом количество регистров должно быть велико (более 100).

5. Система команд должна обеспечивать поддержку компиляции с конкретного языка программирования (компиляторы для RISC на порядок сложнее, чем компиляторы для CISC).

Немного истории: само название RISC появилось в середине восьмидесятых годах в университете Беркли, где под руководством Дэвида Паттерсона и Карло Секвина была создана машина RISC-1, а затем последовало создание RISC-2, позже принятой за основу машин семейства SPARC фирмы Sun. Почти одновременно в Стэнфордском университете был разработан процессор MIPS, положивший начало выпуску машин R4000 – R10000 фирмы MIPS.

Затем почти все ведущие производители ЭВМ стали разрабатывать и выпускать машины на основе RISC-архитектур:

1) Hewlett - Packard – PA7xxx – PA9xxx (PA – Precision Architecture)
2) DEC – Alpha 21xxx : 3) IBM + Apple – Power PC .

Формат команд и структура процессора RISC–1

Как и в большинстве RISC-процессоров команды являются трехадресными:

7	1	5	5	1	13
КОП	Усл	Dest	SRC1	IMM	SRC2
31 23	24	23 19	18 14	13	12 0

КОП – код операции; Усл – бит условия (для команд переходов);

Dest – номер регистра назначения (длина пять бит – $N_{РОН} = 32$);

SRC1 – номер регистра-источника 1;

SRC2 – регистр (IMM = 1) или непосредственное значение источника 2.

В процессоре RISC-1 используется два вида формата команды:

1. $R_d \Leftarrow R_{s1}$ oper S_2 – для выполнения операций обработки;

2. $R_d \Leftrightarrow M((R_{s1}) + S_2)$ – для выполнения обмена с памятью.

Структура процессора RISC -1 показана на рис. 5.4.

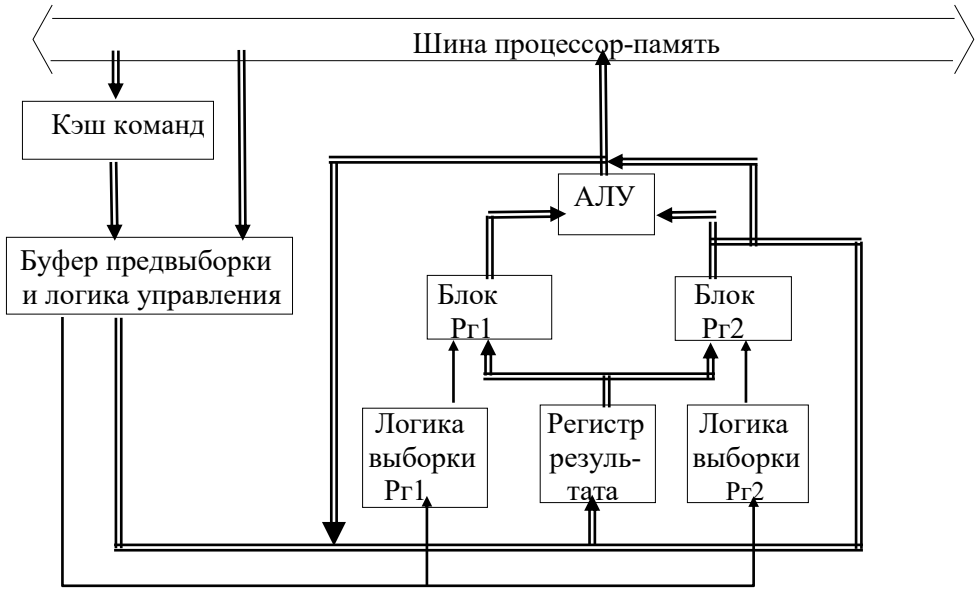


Рис. 5.4

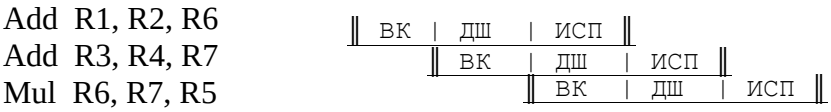
Регистровый формат команд обработки данных облегчает реализацию конвейерного исполнения команд, так как исключает конфликты при обращении к памяти.

Пример. Пусть требуется вычислить выражение E = (A + B) x (C + D) .

1. Двухадресная обработка команд в CISC-компьютерах с использованием ячеек памяти приводит к большому числу пропусков тактов в конвейере (такты ожидания) из-за конфликтов за ресурс памяти:

ВК – выборка команды, ДШ – дешифрация, ВО – выборка операндов,
ИСП – исполнение операции, ЗР – запись результата, **…** – такт ожидания.

2. Трехадресная обработка команд в RISC-компьютерах с использованием регистров практически исключает конфликты приобращении к памяти:



5.3.1. Использование «Окон перекрытия регистров»

Основная проблема RISC-компьютеров: при большом количестве используемых регистров надо сохранять их в памяти при переключениях с одной процедуры на другую. Решение проблемы сохранения и восстановления набора регистров при вызове подпрограмм предлагается реализовать путем организации так называемых «Окон перекрытия регистров» – MORS – Multiple Overlapping Register Set (Rolodex в RISC–2) [11].

Сто тридцать восемь регистров в RISC-1 распределялись между 8 программами (по 32 регистра) как показано на рис. 5.5.

Передача параметров (до 6 штук) между процедурами реализуется без затрат времени просто за счет перекрытия логических регистров (физические регистры совпадают).

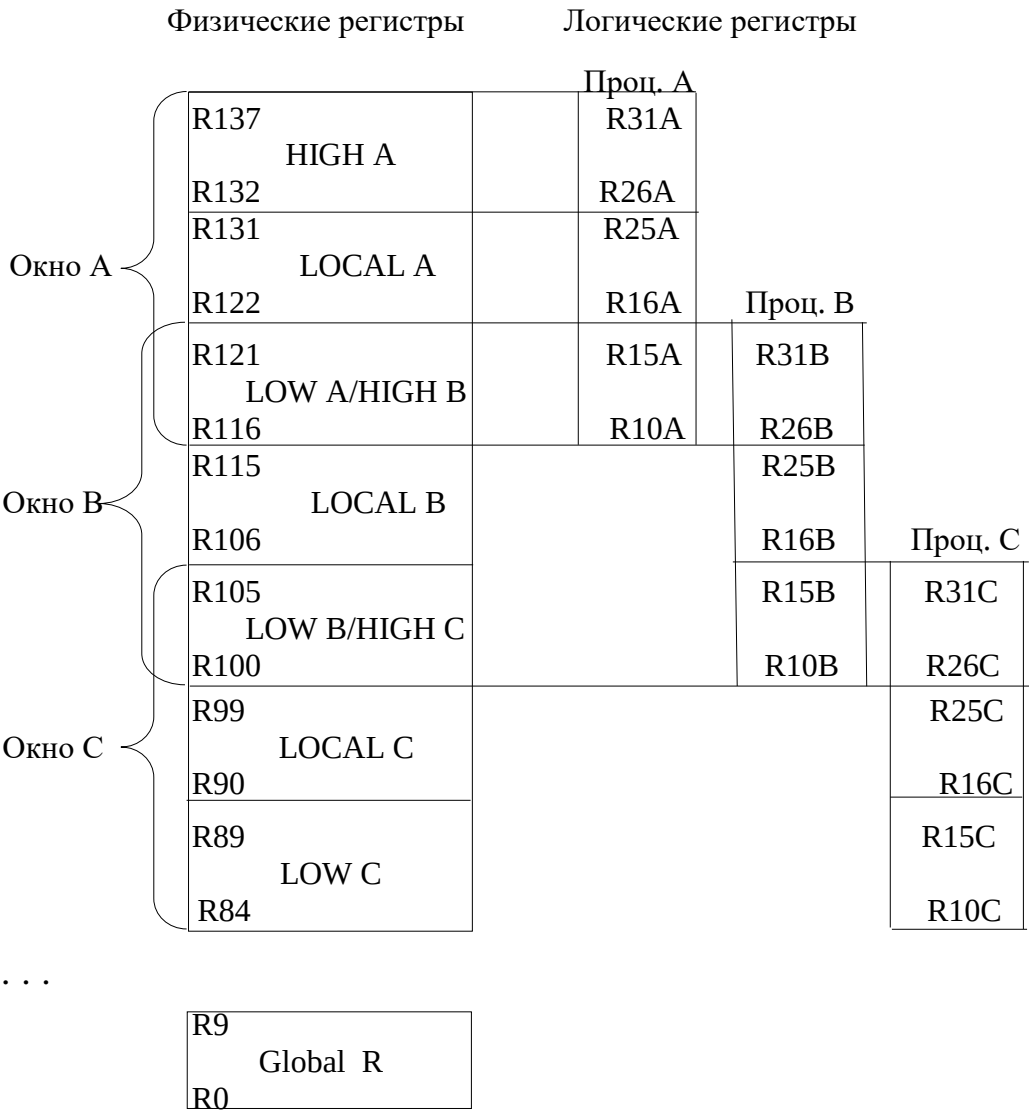


Рис. 5.5

Недостатки MORS:

1) ограниченное число параметров при передаче через регистры;

2) ограниченное число подпрограмм, которым могут предоставляться регистровые окна.

5.3.2. Арифметические особенности и производительность RISC-процессоров

В RISC-процессорах имеется возможность увеличения аппаратной поддержки арифметических операций благодаря уменьшению места на кристалле для

размещения управляющей части процессора. В частности, в арифметико-логическом устройстве процессора широко применяются схемные методы ускорения выполнением операций (например, одновременная обработка нескольких разрядов в одном такте, использование групповых переносов и т. д.).

Время выполнения программы упрощенно можно оценить по следующему выражению:

$$T_{программы} = N_{команды} * C_{такт/ком} * T_{такт},$$

где $T_{такт}$ – длительность такта, и для параметров CISC- и RISC-процессоров справедливы соотношения:

$$N_{комRISC} = (1.5 - 2.0) * N_{комCISC},$$

$$C_{такт/ком.CISC} = (5 - 10) * C_{такт/ком.RISC},$$

$$T_{тактCISC} = (3 - 4) * T_{тактRISC}.$$

Тогда для времени выполнения программ с большим числом арифметических операций будет справедливо соотношение

$$T_{прогр.CISC} = (10 - 20)T_{прогр.RISC}.$$

5.3.3. Достоинства и недостатки RISC- процессоров

Основные *достоинства RISC- процессоров:*

1. Повышение производительности обработки программ вычислительных задач.

2. Благодаря использованию простых команд и минимума их форматов сокращается время разработки RISC-процессора.

3. Улучшение технологичности RISC-процессоров благодаря большей свободе в размещении их элементов на кристалле интегральной схемы и повышение вероятности выхода годных схем.

Недостатки RISC- процессоров:

1. Нарушение основных принципов программирования:

   а) минимум длины исполняемого кода программы.

   б) снижение семантического разрыва между исходным описанием и машинным кодом.

2. Сложность построения компилятора, поскольку программа с языка высокого уровня должна транслироваться в микрокод с оптимизацией использования регистров.

3. Высокие требования к быстродействию памяти.

Кроме того, следует учитывать, что сейчас производительность вычислительной системы в большой степени определяется не процессором, а скоростью передачи данных по шинам и протоколами сетевого взаимодействия. Гораздо важнее требование совместимости с созданным ранее программным обеспечением, которому многие RISC-процессоры не удовлетворяют, что сдерживает их распространение.

Современные процессоры, как правило, реализуются по гибридному принципу: содержат ядро RISC, которое выполняет простые и самые распространенные команды за 1 такт, а сложные команды выпоняются как последовательность микрокоманд. Пользователь имеет дело с обычной системой команд CISC-архитектуры, а внутренняя реализация команд его не интересует.

5.4. Компьютеры с VLIW – архитектурой.

Логическим продолжением идеологии RISC, расширяющей её на архитектуры с несколькими вычислительными устройствами, является VLIW-архитектура, где аббревиатура **VLIW** (*very large instruction word*) означает «очень длинная машинная команда» и характеризуется тем, что одна инструкция процессора содержит несколько операций, которые должны выполняться параллельно различными функциональными устройствами (ФУ) процессора. При этом длина инструкции может достигать 128 или даже 256 бит.

Создание подобных систем основано на разработке специальных компиляторов, которые упаковывают несколько простых команд в «очень длинное командное слово» таким образом, чтобы в одной длинной команде можно было использовать все операционные блоки процессора. В этом случае командное слово соответствует набору ФУ. В суперскалярных процессорах также есть несколько вычислительных модулей, но задача распределения работы между ними решается аппаратно, что сильно усложняет устройство процессора. VLIW-архитектуру можно рассматривать как статическую суперскалярную, так как распараллеливание кода происходит на этапе компиляции, а не

динамически во время исполнения, т. е. в командах VLIW присутствует явный параллелизм.

Подход VLIW сильно упрощает архитектуру процессора, перекладывая задачу распределения вычислительных устройств на компилятор. В то же время код для VLIW обладает невысокой плотностью. Из-за большого количества пустых инструкций для простаивающих устройств программы для VLIW-процессоров могут быть гораздо длиннее, чем аналогичные программы для традиционных архитектур.

Одним из примеров VLIW-архитектуры может служить разработанная Intel совместно с HP концепция 64-разрядной архитектуры IA-64 (процессор Intel Itanium). Для обозначения концепции была принята аббревиатура EPIC (Explicitly Parallel Instruction Computing – вычисления с явным параллелизмом команд), но по сути она соответствует VLIW-архитектуре.

Процессор, разработанный по этой концепции, имеет особенности:

– явный параллелизм в машинном коде; поиск зависимостей между командами производит не процессор, а компилятор;

– большое количество регистров: 128 64-разрядных целочисленных, 128 80-разрядных регистров плавающей арифметики, 64 1-битных предикатных регистра;

– масштабируемость архитектуры до большого количества функциональных узлов;

– команды из разных ветвей условного ветвления снабжаются предикатными полями и запускаются на выполнение параллельно.

Формат команды IA-64 содержит код операции, три 7-разрядных адреса (операнды – только в регистрах) и 6-разрядное предикатное поле. Команды группируются компилятором в «связку» длиной в 128 разрядов. Связка содержит три команды и шаблон, указывающий зависимости между командами в связке, а также между командами разных связок. Одна связка из трех команд соответствует набору из трех ФУ процессора. Благодаря тому, что в шаблоне указана зависимость и между связками, процессору с N одинаковыми блоками из трех ФУ будет соответствовать командное слово из N связок. Этим обеспечивается масштабируемость IA-64.

Достоинства такой концепции:

– процессор не тратит время на анализ потока команд для возможности их параллельного выполнения – эту работу выполняет компилятор;

– вместо логики распараллеливания в процессоре можно разместить больше регистров и функциональных устройств.

Недостатки концепции:

– существенно увеличивается сложность компиляторов и отладки, так как отлаживать нужно оптимизированный параллельный код;

– компилятор производит статический анализ программы, но при изменении данных путь выполнения программы может измениться.

Архитектура VLIW выглядит непривычной для программиста. Из-за сложных внутренних зависимостей кода программирование вручную, на уровне машинных кодов для VLIW-процессоров, является достаточно сложным. Приходится полагаться на оптимизацию компилятора.

В качестве других примеров реализации VLIW-архитектуры можно привести процессоры TriMedia фирмы Philips, семейство DSP C6000 фирмы Texas Instruments, а также видеопроцессоры AMD/ATI Radeon, начиная с R600. Современные системы на кристалле Qualcomm Snapdragon фирмы Qualcomm, предназначенные для использования в качестве центрального процессора телефонов и планшетов, также содержат сопроцессоры с VLIW-архитектурой Hexagon (QDSP6). На них могут выполняться алгоритмы обработки звука и мультимедиа, а также часть цифровой обработки беспроводных сигналов. Ежетактно могут запускаться на исполнение наборы из 4 инструкций и поддерживается аппаратная многопоточность.

Среди отечественных разработок следует отметить многопроцессорный вычислительный комплекс (МВК) «Эльбрус-3М1» и микропроцессоры серии «Эльбрус 2000», которые являются VLIW-процессорами. МВК «Эльбрус-3М1» создан по заказу Министерства обороны РФ на основе VLIW-процессора «Эльбрус 2000» фирмы МЦСТ (Московский центр SPARC-технологий – российская частная компания, специализирующаяся на разработке универсальных микропроцессоров, является базовой организацией кафедры информатики и вычислительной техники МФТИ). В режиме двоичной компиляции МВК эмулирует систему команд IntelX86. МВК поставляется с операционными системами ОС Эльбрус (OSL) в варианте OSL-3M1 и защищённой ОС в варианте

МСВС-3М1, системой программирования с оптимизирующим компилятором, системой двоичной компиляции, средствами для обеспечения программной совместимости с многопроцессорными вычислительными комплексами (МВК) «Эльбрус-2». В тесте SPEC МВК «Эльбрус» с тактовой частотой 300 MHz в режиме совместимости с платформой x86 обогнал Pentium III 500 MHz. Теоретическая производительность двухпроцессорного МВК «Эльбрус-3М1», работающего на частоте 300 МГц, составляет 4,8 Гфлопс (64-bit double). Для сравнения: двухъядерный процессор Intel Core 2 Duo с 2,4 ГГц имеет 19,2 Гфлопс (64-bit double), двухъядерный Itanium 2 с 1,66 ГГц – 13,2 Гфлопс (64-bit double). ВК Эльбрус-3М1 обеспечивает работу в многопользовательском, многозадачном режиме вычислений в реальном масштабе времени.

6. ОРГАНИЗАЦИЯ ПАМЯТИ В КОМПЬЮТЕРЕ

6.1. Назначение и основные характеристики памяти

Память используется для хранения следующих объектов:

1) компьютерные программы;

2) данные (постоянные или переменные);

3) состояния всех устройств.

В памяти недопустима обработка данных и, следовательно, применимы всего две операции: выборка или чтение (информация не разрушается) и запись (предыдущая информация разрушается).

Память понимается как линейная последовательность ячеек, наделенных адресами, по которым осуществляется доступ к содержимому. Различают следующие единицы адресуемой информации:

1. МАЕП – минимально адресуемая единица памяти.

В зависимости от вида данных:

– 1 бит (флаги слова состояния процессора, внешних устройств);

– 1 байт (арифметические данные, команды).

2. Слово – наибольшая длина данного, выбираемого за одно обращение к памяти (16, 32, 64, 128 бит).

Основные характеристики памяти:

1. **Емкость** (обозначается С) с диапазоном от 1 байта (регистр памяти) до n*100 Гбайт или n*Тбайт (магнитные диски, оптические диски, флеш-память).

2. **Быстродействие** (обозначается Т) с диапазоном: n* 1нс (регистровая память) – n* 10 с (магнитная лента, оптический диск).

Обычно, чем больше емкость памяти, тем меньше ее быстродействие. Для преодоления противоречия между емкостью и быстродействием используется иерархическая организация памяти.

Основные параметры, характеризующие быстродействие памяти:

а) $t_{ВЫБОРКИ}$ – время от запуска памяти для считывания данного до его появления в буферном регистре (не включает задание и дешифрацию адреса);

б) $t_{ОБРАЩЕНИЯ}$ – среднее время, затраченное на обращение к памяти в двух последовательных циклах чтения и записи данных по разным адресам (включая время задания адреса и его дешифрации).

Как правило, выполняется соотношение $t_{ОБРАЩЕНИЯ} \geq 2t_{ВЫБОРКИ}$.

3. **Надежность** – зависит от возникновения сбоев при считывании или записи данных и обеспечивается с помощью средств контроля (обнаружения и исправления ошибок):

а) Parity control – контроль по четности, позволяет обнаружить одиночные ошибки (в одном бите);

б) ECC (error checking and correction control) – контроль с использованием корректирующих кодов, использует два дополнительных бита. Позволяет обнаружить двойную ошибку или скорректировать одиночную ошибку.

4. **Плотность записи** (бит /см^2) – зависит от типа среды хранения информации, наиболее высокая плотность у оптических накопителей.

5. **Стоимость хранения** одного бита – важна для пользователя с финансовой точки зрения.

6.2. Основные среды хранения информации

1. Магнитная среда.

Исторически самые первые запоминающие устройства использовали магнитную среду, где в качестве носителя информации использовались магнитные материалы. В настоящее время магнитные материалы применяются только в устройствах внешней памяти из-за низкого быстродействия. К ним относятся: накопители на жестких магнитных дисках (НЖМД), или винчестеры, в английской мнемонике – HDD (hard disc drive) и накопители на магнитных лентах.

2. Полупроводниковые среды:

Среда с накоплением зарядов (DRAM) – динамическая энергозависимая память с произвольным доступом.

В данном случае в качестве элемента памяти используются конденсатор и транзистор, позволяющие хранить один бит информации. В зависимости от вида материалов различают: биполярную полупроводниковую память (более быстрая) и память на МОП-структурах (металл – оксид – полупроводник), более медленную, но дешевую. Из-за разряда конденсатора информация хранится кракovременно и ее надо регенерировать каждые 5–15 мсек.

Память на активных элементах с усилительными свойствами (SRAM) – статическая энергозависимая память с произвольным доступом.

В качестве элементов памяти используются триггеры – электронные полупроводниковые схемы с двумя устойчивыми состояниями, а сами структуры хранения называются регистрами (самый быстрый вид памяти, но имеет малую емкость). При наличии питающего напряжения информация хранится сколь угодно долго.

Флеш-память (англ. flash memory)— разновидность полупроводниковой технологии электрически перепрограммируемой памяти (EEPROM – см.раздел 6.6).

Основным компонентом флеш-памяти является транзистор с плавающим затвором, который является разновидностью МОП-транзисторов. Его отличие состоит в том, что у него есть дополнительный затвор, который изолирован и хранимый в нём отрицательный заряд будет оставаться надолго. Флеш-память является энергонезависимой памятью, характеризующейся большой емкостью, дешевизной, механической прочностью и высокой скоростью работы. Серьёзными недостатками данной технологии является ограниченный ресурс носителей в смысле числа циклов перезаписи, а также чувствительность к электростатическому разряду.

Твердотельные накопители – SSD (solid state drive) - энергонезависимое немеханическое запоминающее устройство на основе микросхем памяти.

Твердотельные накопители (ТТН) являются устройствами, хранящими данные в микросхемах вместо вращающихся металлических дисков или магнитных

лент. По существу ТТН представляет собой очень вместительную флешку и является альтернативой НЖМД благодаря увеличению скорости работы в 100-1000 раз, отсутствию шума и повышению надежности хранения данных. Минусы ТТН в основном связаны с высокой стоимостью и уменьшением допустимого числа циклов перезаписи информации.

3. Оптические запоминающие устройства.

Для хранения информации в оптических ЗУ (оптических дисках) используется металлический отражающий слой (алюминий или другие металлы), покрываемый защитным акриловым слоем от контакта с внешней средой. Данные наносятся на поверхность диска в виде *впадин* (pit), чередующихся с плоскими участками – *площадками* (land). Впадины и площадки образуют непрерывную спиральную дорожку, а их чередование представляет цифровой сигнал в виде двоичных данных. В процессе считывания информации лазерный луч направляется на поверхность вращающегося диска и отражается, меняя свою интенсивность (логические 0 или 1) из-за участков с различными коэффициентами отражения. Затем отражённые световые импульсы преобразуются с помощью фотоэлементов в электрические импульсы. Оптические диски характеризуются высокой плотностью записи и малой ценой хранения одного бита информации.

6.3. Методы доступа к данным.

Запоминающие устройства (ЗУ) используют различные способы доступа к данным:

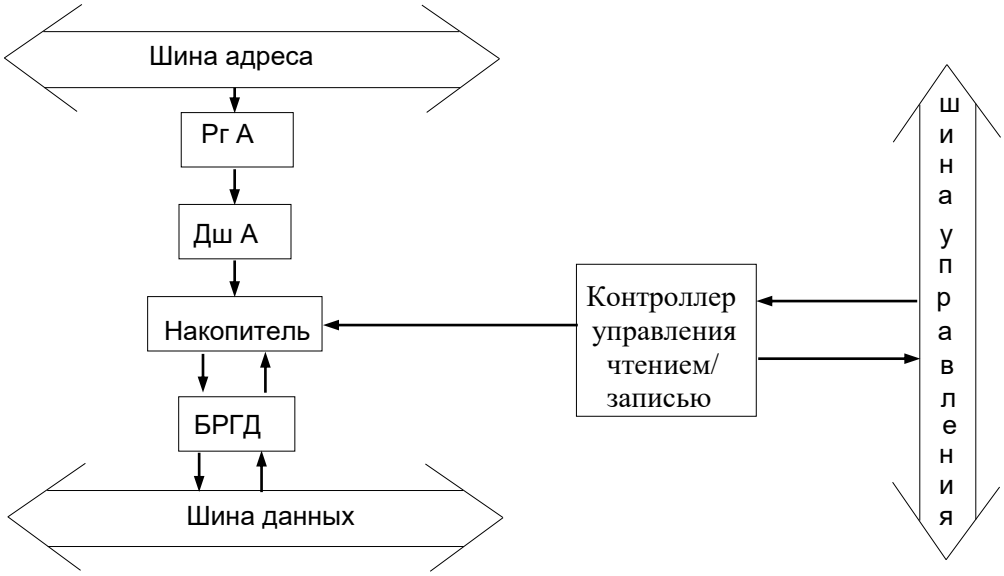
1. Последовательный доступ. Ориентирован на хранение последовательности блоков данных (записей), и для доступа к нужному элементу (байту или слову) надо прочитать все предшествующие ему данные. Время доступа зависит от положения записи среди других и положения элемента в записи. Примером может служить ЗУ на магнитной ленте.
2. Прямой доступ. Доступ к записи осуществляется адресно, а к элементу внутри записи – последовательно. Соответственно, время доступа – переменное в небольших пределах. Примером может служить ЗУ на магнитных дисках.
3. Произвольный доступ. Каждая ячейка памяти имеет свой физический адрес. Обращение к любой ячейке производится в произвольной очередности

и занимает одинаковое время. Примеры – основная память типа RAM (Random Access Memory) и постоянные ЗУ – память типа ROM (Read Only Memory).

4. Ассоциативный доступ (АЗУ) – доступ к ячейкам памяти осуществляется по их содержимому, а не по адресу путем параллельного сравнения содержимого всех ячеек с заданным образцом.

6.4. Память с произвольным доступом (ППД)

Память с произвольным доступом имеет структуру, показанную на рис. 6.1.



РгА – регистр адреса, ДшА – дешифратор адреса, БРгД – буферный регистр данных

Рис. 6.1

Накопитель ППД организован в виде матрицы ячеек, состоящих из запоминающего элемента (ЗЭ), хранящего 1 бит информации и имеющего свой адрес, определяемый адресом строки (RAS – Row Address) и адресом столбца (CAS – Column Address). Они хранятся в регистрах адреса строки и адреса столбца, связанных со своими дешифраторами (Дш), к выходам которых подсоединены ЗЭ матрицы, расположенные на пересечении соответсвующих строки и столбца. Совокупность ЗЭ, логических схем выбора строк и столбцов и логики интерфейса образуют интегральную микросхему (ИМС) памяти.

В случае реализации ЗЭ на основе конденсатора и запирающего ключа ППД называется динамической (DRAM – Dynamic RAM) и требует регенерации, а при

реализации ЗЭ на основе триггера ППД называется статической (SRAM – Static RAM) и сохраняет информацию, пока подается питание. Динамическая ППД более дешевая и медленная и имеет большую емкость, а статическая ППД более быстрая, дорогая, но ввиду малой емкости применяется только как быстрая буферная память (память типа Кэш).

6.4.1. Режимы работы DRAM

Для повышения быстродействия обращения к памяти применяются различные режимы работы ИМС памяти, позволяющие: исключить повторные указания адреса строки, выполнять передачу данных по обоим фронтам ипульса синхронизации и выполнять групповую (пакетную) передачу данных, требующую задания адреса столбца только для 1-го данного пакета. Примерами ускоренных режимов обращения к памяти могут служить:

1. Режим быстрого страничного доступа FPM (Fast Page Mode) – режим, при котором при многократном последовательном обращении к одной и той же строке номер строки задается только первый раз, а кроме того адрес столбца меняется по заднему фронту сигнала RAS. Память такого типа была популярна в первой половине 1990-х годов и работала на частотах до двадцати пяти мегагерц.

2. EDO (Extended Data Out – расширенный выход данных) – режим при котором адресация нового столбца осуществляется до завершения предыдущего, производительность повышается почти вдвое. Наличие регистра-защёлки данных обеспечивает конвейеризацию работы и повышение производительности при чтении. (системная шина от пятидесяти до шестидесяти мегагерц, время полного доступа— 60 и 50 нс).

3. SDRAM (Synchronous Dynamic RAM – синхронная память, создана для замены EDO DRAM в связи с повышением рабочих частот системных шин. Особенностями этого типа памяти стали использование тактового генератора для синхронизации всех сигналов и использование конвейерной обработки пакетов из четырех 32-битных или 64-битных слов, Память надёжно работает на частотах системной шины 100 МГц и выше.

4. DDR SDRAM (Double data rate SDRAM) – основана на SDRAM и отличается удвоенной скоростью передачи данных или пропускной способностью. Рабочие частоты памяти типа DDR SDRAM – 100, 133, 166 и 200 МГц, время полного доступа – 30 и 22,5 нс. Так как данные передаются по 2 бита на один

синхроимпульс, как по фронту, так и по спаду тактового импульса, то эффективная частота передачи данных лежит в пределах от 200 до 400 МГц.

5. RDRAM (Rambus Direct RAM) – режим, сочетающий DDR с асинхронным блочно-ориентированным протоколом передачи, при котором данные между контроллером и памятью передаются пакетами. Работает на частотах 400, 600 и 800 МГц с временем полного доступа до 30 нс и временем рабочего цикла до 2,5 нс. Первоначально стоила очень дорого, из-за чего производители компьютеров предпочитали менее производительную и более дешёвую DDR SDRAM.

6. DDR2 SDRAM – это эволюционное обновление DDR SDRAM, выпущенное в 2004 г. Благодаря удвоению скорости передачи данных (обработка двух команд чтения и двух команд записи за такт), DDR2 SDRAM работает на более высоких тактовых частотах. Стандартные модули памяти DDR работают с частотой 200 МГц, а стандартные модули памяти DDR2 работают с частотой 600 МГц.

7. DDR3 SDRAM – основана на DDR2 SDRAM и отличается увеличением частоты передачи данных по шине памяти и пониженным энергопотреблением. Память такого типа обеспечивает большую пропускную способность по сравнению с ранее существовавшими типами памяти. Работает на частотах до 800 МГц.

8. DDR4 SDRAM – тип DRAM, производство которого началось со второго квартала 2014 года, отличается повышенными частотными характеристиками и пониженным напряжением питания. Основное отличие DDR4 от предыдущего стандарта DDR3 заключается в удвоенном до 16 числе банков (в двух группах банков, что позволило увеличить скорость передачи). Пропускная способность памяти DDR4 в перспективе может достигать 25,6 ГБ/с (при повышении максимальной эффективной частоты до 3200 МГц). Надёжность работы DDR4 повышена за счёт введения механизма контроля чётности на шинах адреса и команд. Изначально в стандарте DDR4 был определён диапазон частот от 1600 до 2400 МГц с возможностью увеличения до 3200 МГц.

6.4.2. Конструктивные исполнения DRAM

Память типа DRAM конструктивно выполняют и в виде отдельных микросхем и в виде модулей памяти (типов SIMM, DIMM, RIMM).

SIMM (*single in-line memory module*) — модули памяти, представляющие собой длинные прямоугольные платы с рядом контактных площадок вдоль одной

из сторон платы. Наиболее распространены 30- и 72-контактные модули SIMM, используемые для типов памяти FPM и EDO.

DIMM (*dual in-line memory module*) – модули памяти, представляющие собой длинные прямоугольные платы с рядами контактных площадок вдоль обеих сторон платы. Микросхемы памяти на них могут быть размещены как с одной, так и с обеих сторон платы. Предназначен для памяти типа SDRAM.

RIMM (*rambus in-line memory module*) – модули памяти, применяемые парами. Выпускаются с памятью типа RDRAM, со 168 или 184 контактами. Из-за особенностей конструкции должны устанавливаться на материнские платы только в парах, в противном случае в пустые разъёмы должны устанавливаться специальные модули-заглушки.

6.5. Блочная организация основной памяти.

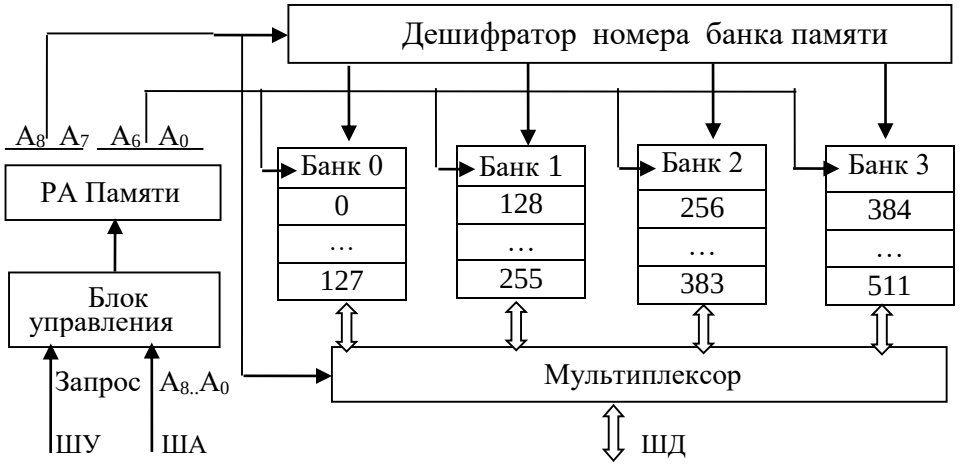
Емкость основной памяти в современных компьютерах слишком велика, чтобы ее можно было реализовать на одной ИМС. Кроме того, разрядность ячеек ИМС памяти, как правило, меньше разрядности слов в ВМ. Поэтому для получения основной памяти требуемой разрядности несколько ИМС объединяют в модуль памяти. Затем несколько модулей объединяют в банк памяти, имеющий требуемую разрядность, но еще недостаточную емкость. Наконец, несколько банков памяти объединяют для получения основной памяти требуемой емкости. Объединение банков производится по блочному принципу. При этом для памяти, состоящей из B банков, адрес A ячейки памяти преобразуется в пару (b, w), где b – номер банка, а w – адрес ячейки внутри банка. Распределение разрядов адреса A между частями b и w может выполняться следующим образом:

1) старшие разряды адреса A определяют номер банка b, а младшие – адрес w ячейки в банке (такая организация называется чисто блочной);

2) номер банка определяется по правилу $b = A \bmod B$, а адрес ячейки $w = A \operatorname{div} B$; такая блочная организация называется циклической, и для нее младшие разряды адреса A определяют номер банка b, а старшие – адрес w ячейки в банке.

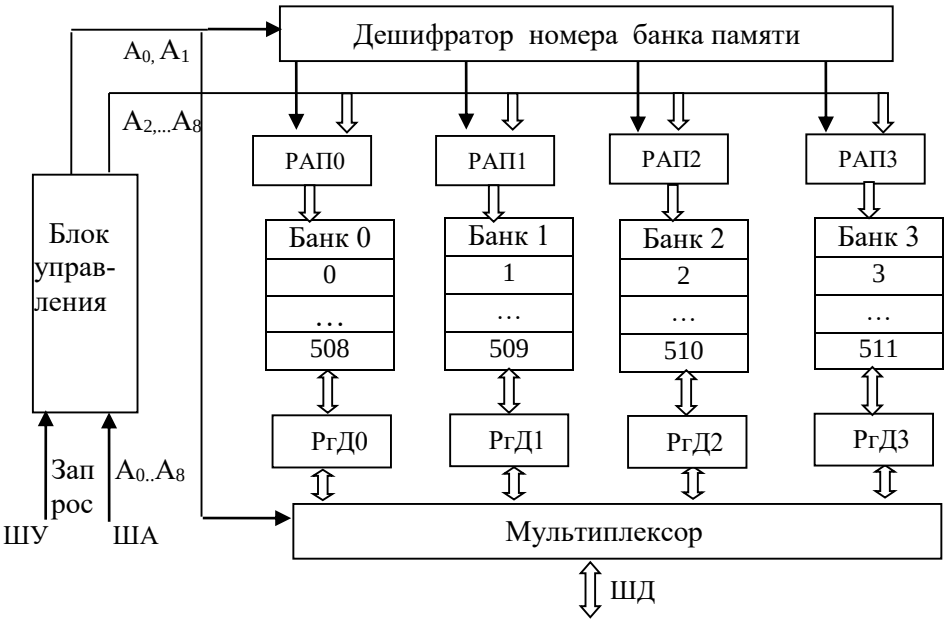
Примеры структур блочной организации ОП для памяти емкостью 512 слов, состоящей из четырех банков, показаны для чисто блочной схемы на рис. 6.2, а для циклической блочной схемы – на рис. 6.3.

В функциональном отношении чисто блочная память может рассматриваться как единое ЗУ, емкость которого равна суммарной емкости банков, а быстродействие соответствует быстродействию одного банка.



A = (b , w), b [7-8] – адрес банка, w [0-6] – адрес слова в банке

Рис. 6.2. Чисто блочная организация ОП



A = (w, b), b = [0-1] – адрес банка, w = [2-8] – адрес слова в банке

Рис.6.3. Циклическая блочная организация ОП (память с чередованием адресов)

Циклическая блочная организация использует принцип расслоения памяти или чередование адресов (interleaving) для параллельного доступа к смежным

ячейкам памяти, размещаемым в различных банках. Поскольку в каждом такте на шине адреса задается адрес только одной ячейки, квазипараллельное обращение производится со сдвигом на 1 такт (рис. 6.4). Адрес ячейки запоминается в индивидуальном регистре адреса (РАП) и дальнейшие операции по доступу к ячейкам каждого банка протекают независимо. Среднее время доступа к ОП может сокращаться почти в В раз.

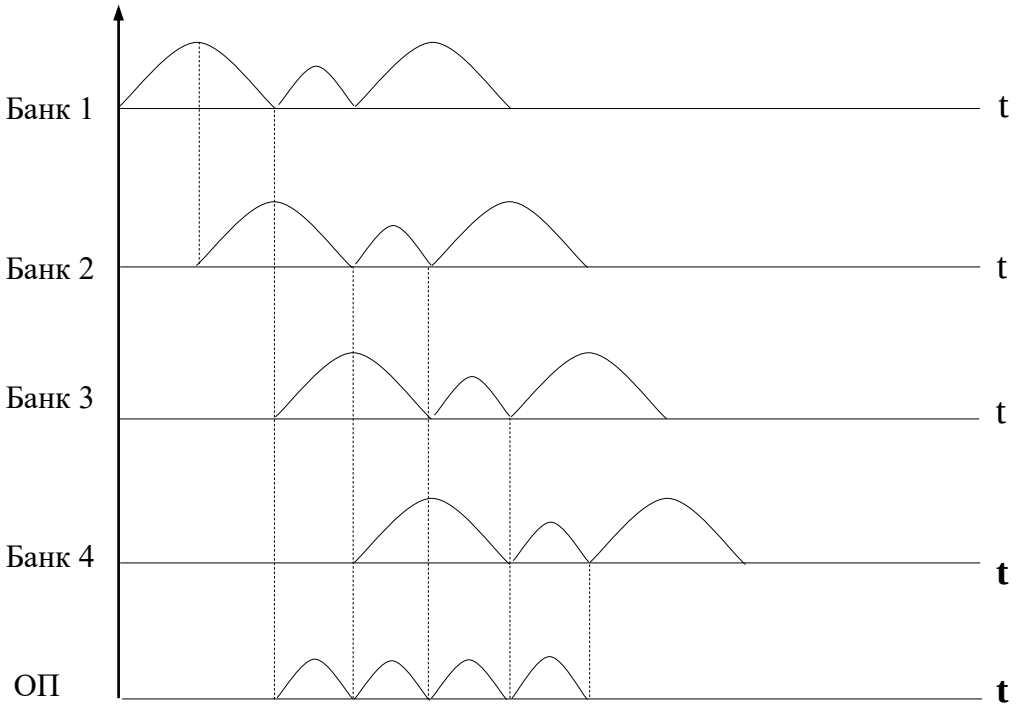


Рис. 6.4

В функциональном отношении память с циклической блочной организацией может рассматриваться как единое ЗУ, емкость которого равна суммарной емкости банков, а быстродействие соответствует быстродействию одного банка, помноженному на число банков.

6.6. Постоянные запоминающие устройства (ПЗУ - ROM)

ПЗУ – энергонезависимая память, используемая для хранения фиксированных программ, микропрограмм управления аппаратурой компьютера и подключенных к нему устройств, а также массивов неизменяемых данных. В частности, базовая система ввода-вывода (BIOS) компьютера – основная программа, обеспечивающая настройку взаимодействия процессора, материнской платы,

оперативной памяти и других узлов, хранится в специальной микросхеме ПЗУ, называемой ROM BIOS.

По технологии изготовления и способу использования различают следующие виды ПЗУ:

1. МПЗУ (масочное ПЗУ) – устройство, в котором запись информации осуществляется фирмой-изготовителем путем выжигания связей (участков) в процессе изготовления модуля памяти.

2. ППЗУ (программируемое ПЗУ (PROM)) – устройство, которое поставляется пользователю в исходном виде, и он сам прошивает необходимые связи для записи требуемой информации с помощью специального прибора – программатора. После такой процедуры ППЗУ не может больше перепрошиваться.

3. СППЗУ(стираемое ППЗУ) – EPROM (Erasable Programmable Read Only Memory) – ПЗУ, допускающее запись (программирование) информации с помощью программатора, возможность ее стирания с помощью ультрафиолетового света в течение длительного времени (более 10 минут) и затем перезаписи новой информации.

4. ЕСППЗУ(электрически стираемое перепрограммируемое ПЗУ) – EEPROM (Electrically Erasable Programmable Read Only Memory) – ПЗУ, память которого может стираться и заполняться новыми данными несколько десятков тысяч раз, используется в твердотельных накопителях, одной из разновидностей EEPROM является флеш-память.

6.7. Ассоциативные запоминающие устройства (АЗУ)

В отличие от других видов памяти доступ к информации в АЗУ осуществляется не по адресу размещения данного, а по содержимому – значению самого данного или его части. При этом в АЗУ хранимая информация сравнивается с некоторым образцом и проверяется их соответствие. Образец, по которому производится поиск информации, называется ассоциативным признаком или ключем поиска. На рис. 6.5 показан вариант построения АЗУ, включающий:

1) память данных X для хранения N m-разрядных слов; несколько младших разрядов содержат служебную информацию;
2) регистр ассоциативного признака (разрядностью $k \leq m$) для создания образца поиска;

3) регистр маски, позволяющий исключить из признака поиска определенные биты;
4) схемы совпадения, реализующие параллельное сравнение каждого бита всех слов памяти данных с соответствующим битом признака поиска и выработки сигналов совпадения;
5) регистр совпадений, каждый бит которого соответствует одной из ячеек памяти и в него заносится 1, если все разряды этой ячейки совпали с одноименными разрядами признака поиска;
6) комбинационная схема, формирующая сигналы, характеризующие результаты поиска.

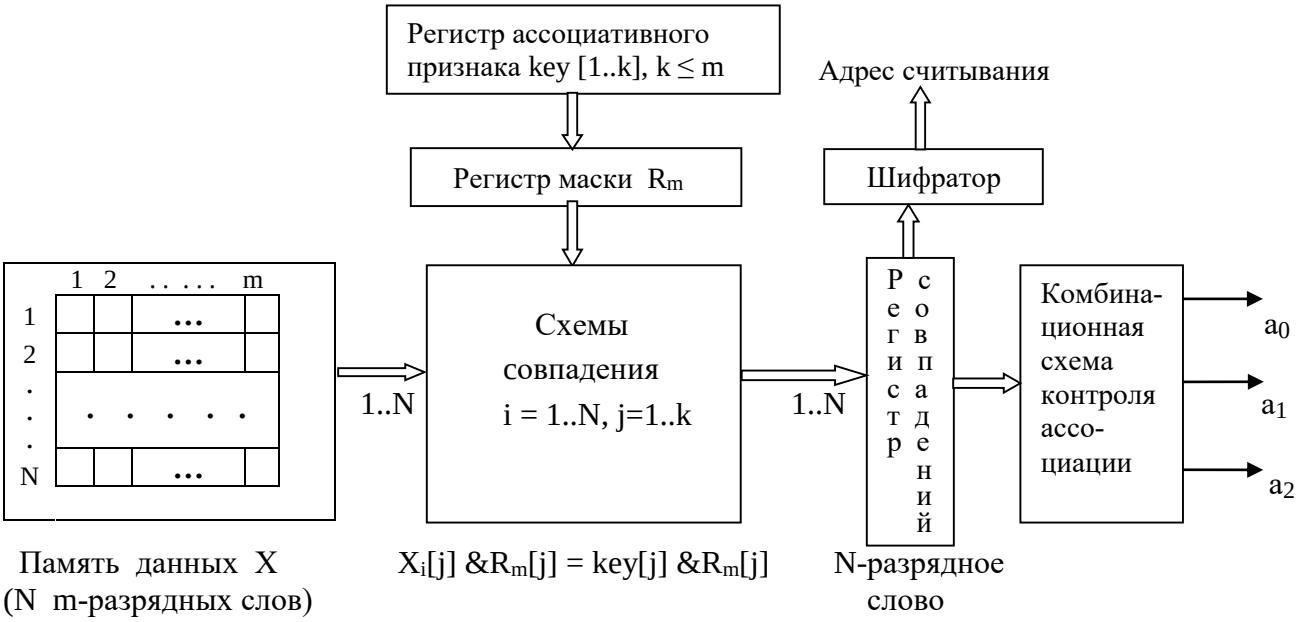


Рис.6.5. Структура ассоциативного ЗУ

Перед чтением данных из памяти предварительно все биты регистра совпадений устанавливаются в 1. Затем последовательно выбирается каждый (j-й) бит признака и параллельно сравнивается с j-ым битом всех слов памяти данных. Если для некоторого (i-го) слова данных биты не совпали, то i-ый разряд регистра совпадений сбрасывается в 0. После использования всех битов признака поиска производится анализ регистра совпадений. Поскольку результаты поиска могут быть неоднозначными, комбинационная схема контроля ассоциации по содержимому регистра совпадений формирует признаки a_i, показывающие, что:

1) a_0 – слово не найдено (все биты регистра совпадений равны 0);
2) a_1 – найдено только 1 слово (лишь 1 бит регистра совпадений равен 1);
3) a_2 – найдено несколько слов (несколько битов регистра совпадений равны 1).

При a_0=1, считывание отменяется, при a_1=1 считывается слово, на которое указывает 1 в регистре совпадений, наконец, при a_2 = 1 сбрасывается самая старшая 1 в регистре совпадений и извлекается соответствующее ей слово памяти данных; повторяя последнюю операцию, можно считать все слова, совпавшие с признаком поиска.

Запись данного в АЗУ выполняется в ячейку памяти данных, которая либо является пустой, либо дольше всех не использовалась (определяется путем считывания слов памяти данных, при котором не маскируются только служебные разряды).

Основными преимуществами АЗУ является высокое быстродействие за счет параллельного поиска и то, что время поиска зависит только от числа разрядов в признаке поиска и скорости опроса разрядов и не зависит от числа ячеек N памяти данных. С другой стороны, большие затраты оборудования на реализацию параллельного сравнения данных с признаком поиска позволяют использовать АЗУ только как буферную память небольшой емкости (чаще всего как кэш 1-го уровня).

СПИСОК ЛИТЕРАТУРЫ

1. ARM (архитектура) http://ru.wikipedia.org/wiki/ARM .
2. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. СПб.: Питер, 2014.
3. Орлов С.А., Цилькер Б.Я. Организация ЭВМ и систем. 2-е изд. СПб.: Питер, 2011.
4. Дэвид М. Харрис и Сара Л. Харрис. Цифровая схемотехника и архитектура компьютера, 2-е изд., пер. с англ., Изд-во Morgan Kaufman, 2013.
5. Миллер Р. Теория переключательных схем, т.1: Комбинационные схемы, пер. с англ., М.: «Наука», 1970.
6. Миллер Р. Теория переключательных схем, т.2: Последовательностные схемы и машины, пер. с англ., М.: «Наука», 1971.

7. VHDL IEEE Standard (IEEE STD 1076) доступен по URL : ieeexplore.ieee.org.
8. Weste N., and Harris D., CMOS VLSI Design, 4th ed., Addison-Wesley, 2011.
9. Зубков С.В. Assembler. Для DOS, Windows и Unix. М.: ДМК, 1999.
10. Королев Л.Н. Микропроцессоры, микро- и мини-ЭВМ. М.: Изд-во МГУ, 1988.
11. Фрир Дж. Построение вычислительных систем на базе перспективных микропроцессоров / пер. с англ. М.: Мир, 1990.

Владимир Андреевич Кирьянчиков

**Организация ЭВМ и систем.
Архитектура компьютеров. Организация
процессора и основной памяти**

Учебное пособие

Редактор М.Б. Шишкова

Подписано в печать Форма 60 x 84 1/16
Бумага офсетная. Печать цифровая. Печ.л. 7.0.
Гарнитура «Times New Roman». Тираж 63 экз. Заказ

Издательство СПбГЭТУ «ЛЭТИ»
197376, С-Петербург, ул. Проф. Попова,5