Министерство образования Республики Беларусь

Учреждение образования
«Полоцкий государственный университет
имени Евфросинии Полоцкой»

В. Е. Питолин
С. Г. Сурто

АРХИТЕКТУРА ПЭВМ И ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия для специальностей 1-40 02 01 «Вычислительные машины, системы и сети», 1-36 04 02 «Промышленная электроника»

Новополоцк
Полоцкий государственный университет
имени Евфросинии Полоцкой
2022

УДК 004.2(075.8)
ББК 32.973-02я73
П35

Рекомендовано к изданию в качестве пособия Президиумом Совета
Учебно-методического объединения в области информатики и радиоэлектроники
(протокол № 4 от 07.04.2022)

РЕЦЕНЗЕНТЫ:

канд. техн. наук, доц., зав. каф. ЭВМ
Белорусского государственного университета информатики и радиоэлектроники
Б. В. НИКУЛЬШИН;
канд. техн. наук, доц., зав. каф. ЭВМ и систем
Учреждения образования «Брестский государственный технический университет»
С. С. ДЕРЕЧЕННИК

Питолин, В. Е.

П35 Архитектура ПЭВМ и вычислительных систем : пособие для студентов спец. 1-40 02 01 «Вычислительные машины системы и сети», 1-36 04 02 «Промышленная электроника» / В. Е. Питолин, С.Г. Сурто. – Новополоцк : Полоц., гос. ун-т им. Евфросинии Полоцкой, 2022. – 286 с.

ISBN 978-985-531-808-9.

Содержит изложение базовых вопросов организации вычислительных систем: принципов построения микропрограммной структуры, системы команд центрального процессора, иерархии памяти, схем различных устройств расширения архитектуры и их взаимодействия между собой и центральным процессором. Для закрепления изучаемого материала предусмотрено выполнение лабораторных работ, а также курсовой работы. Представлен перечень индивидуальных заданий и методика выполнения курсовой работы. Разработаны вопросы экзамена по дисциплине.

Предназначено для студентов специальностей 1-40 02 01 «Вычислительные машины системы и сети», 1-36 04 02 «Промышленная электроника».

УДК 004.2(075.8)
ББК 32.973-02я73

ISBN 978-985-531-808-9

ВВЕДЕНИЕ

Целью изучения дисциплины «Архитектура ПЭВМ и вычислительных систем» является освоении теоретических основ логической организации цифровых вычислительных машин и систем, определяющих процесс обработки данных, а также состав, назначение, принципы взаимодействия технических средств и программного обеспечения. Изучение дисциплины предусмотрено в рамках квалификационных требований, предъявляемых к инженеру-системотехнику по специальности 1-40 02 01 «Вычислительные машины, системы и сети», а также к инженеру-электронику по специальности 1-36 04 02 «Промышленная электроника».

Дисциплина «Архитектура ПЭВМ и вычислительных систем», согласно учебному плану упомянутых специальностей, изучается на 3-м курсе обучения в весеннем семестре и включает в себя 50/76 часов лекционных занятий, 16 часов лабораторных или практических занятий. Предусмотрено выполнение курсовой работы. Итоговая форма контроля – экзамен.

Лекции проводятся с использованием презентаций, но основная часть теоретического материала приводится в данном лекционном курсе. В лекциях будет также рассматриваться дополнительный материал в виде пояснений и практических примеров. Промежуточный контроль усвоения лекционной части дисциплины проводится в виде теста дважды в течение семестра на аттестационных неделях. Тест может проводиться в виде письменного опроса или компьютерного тестирования. Результаты тестирования оцениваются по 10-бальной шкале. Кроме тестового опроса в аттестацию входит контроль исполнения практических заданий и лабораторных работ. На лекциях, посвященных изучению архитектуры конкретных устройств из состава архитектуры ПЭВМ, могут проводиться управляемые самостоятельные занятия студентов, на которых для индивидуального или группового выполнения могут быть предложены практические задания с консультированием студентов по ходу выполнения и контролем результатов.

Целью практикума является освоение практических навыков программирования систем и устройств из состава архитектуры ПЭВМ с использованием программной модели ЭВМ и аналога языка ассемблера. Работы выполняются в компьютерном классе в соответствии с индивидуальным вариантом задания. Номер варианта индивидуального задания выдается студенту на первом занятии и сохраняется до конца семестра. Отчет по лабораторной работе представляется в электронном виде. Отчет должен содержать номер лабораторной работы, наименование работы, фамилию и инициалы студента, вариант задания, результаты по каждому пункту выполнения работы, выводы по работе. Защита лабораторных работ производится индивидуально исключительно на занятиях по лабораторному практикуму в соответствии с расписанием. Защита лабораторных работ принимается только в течении семестра до календарной даты начала сессии.

Студенты, не защитившие лабораторные работы, не допускаются к сессии согласно п. 20 «Правил аттестации…».

Изучение дисциплины «Архитектура ПЭВМ и вычислительных систем» реализует следующие виды практических занятий студентов:

– самостоятельное изучение и освоение программной модели ПЭВМ (упражнения на компьютерном тренажере);

– использование основной и дополнительной литературы при подготовке рефератов по дисциплине;

– самостоятельную подготовку к промежуточной и итоговой аттестациям.

Курсовая работа по дисциплине «Архитектура ПЭВМ и вычислительных систем» для студентов специальности 1-40 02 01 «Вычислительные машины, системы и сети» имеет специализацию разработки программ для обслуживания системных устройств, а специальности 1-36 04 02 «Промышленная электроника» – проектирования операционного блока и низкоуровневого программирования. Целью курсовой работы является практическое закрепление курсового материала и развитие навыков самостоятельной работы студентов с современными вычислительными системами.

Объем работы определяется индивидуальным заданием на курсовую работу. Конкретное содержание курсовой работы и основные требования по её исполнению изложены в методических указаниях по курсовому проектированию. В процессе выполнения курсовой работы предусмотрен поэтапный контроль согласно утвержденному графику. Защита курсовой работы проводится согласно установленному графиком сроку. Выполнение и успешная защита курсовой работы является обязательным условием допуска к сдаче экзамена по курсу. На защите разрешается пользоваться распечатками выданных электронных материалов по курсу и любой справочной литературой.

Итоговая диагностика компетенции осуществляется при защите курсовой работы и сдаче экзамена (или зачета) по изучаемой дисциплине.

Экзамен представляет собой письменно-устный ответ, на подготовку к которому отводится максимум один час. Экзаменационный билет включает в себя три теоретических вопроса по изученному курсу из числа вопросов, приведенных в настоящем пособии. Экзаменационная оценка выставляется на основании общего ответа и ответов на дополнительные вопросы.

Письменная подготовка к ответу выполняется на черновиках, которые выдаются преподавателем. После окончания ответа черновики подписываются студентом и возвращаются преподавателю.

При себе на экзамене следует иметь только ручку и зачетную книжку.

ЛЕКЦИОННЫЙ КУРС

Тема 1. Введение в архитектуру вычислительных машин и систем

1.1. Из истории развития ЭВМ

Выполнение вычислений – неотъемлемая операция в любой области человеческой деятельности, особенно в экономике. С древнейших времен человек пытался облегчить этот род деятельности, создавая различные механические приспособления. Для астрономических вычислений, например, были разработаны сложнейшие вычислительные приспособления, позволяющие с высокой точностью определять положения известных планет. Для инженерных расчетов была разработана т.н. «логарифмическая» линейка, позволяющая в одно действие производить любые математические операции над числовыми значениями.

Первой реальной вычислительной машиной, реализующей автоматическое выполнение последовательности математических операций обработки числовых значений и сохраняющей результат вычислений, можно считать разностную машину Ч. Беббеджа, изготовленную им в 1819 г. для расчета астрономических величин.

Прототип аналитической вычислительной машины Ч. Беббеджа показан на рисунке 1. Здесь можно найти все основные элементы современных вычислительных машин: арифметико-логическое устройство, память, устройство управления и программирования на перфокартах.

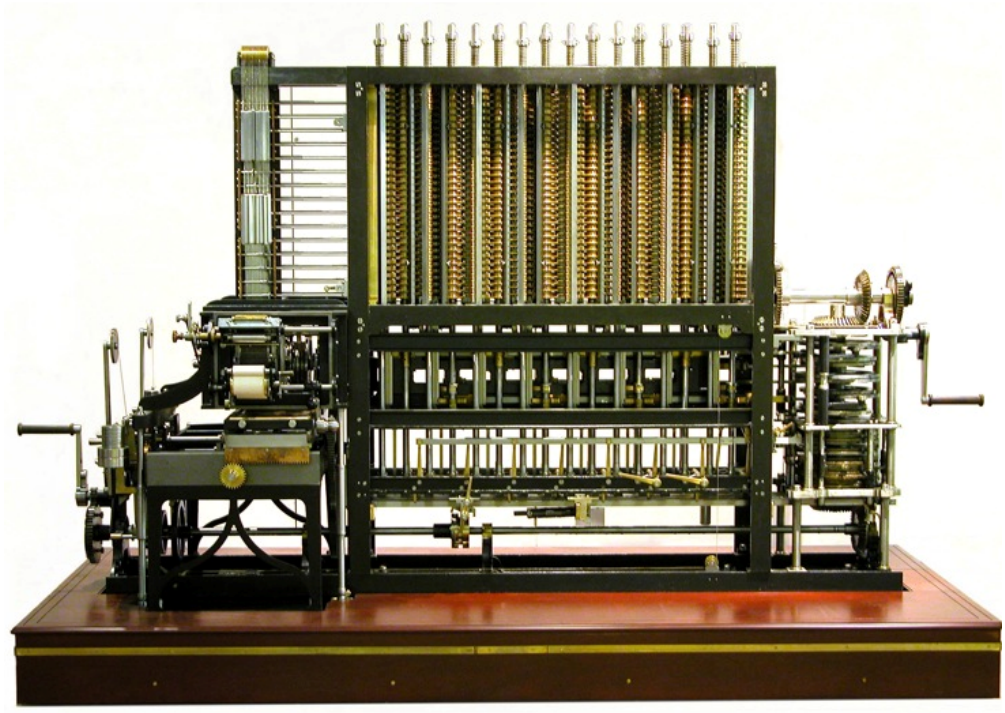


Рисунок 1. – Внешний вид аналитической машины Ч. Беббеджа 1819 г.

Однако в полной мере идеи унификации и автоматизации вычислительных операций удалось реализовать только после создания электронной элементной базы релейно-контактных схем и разработки основ двоичного счисления – булевой алгебры. Исходя из основных принципов реализации релейно-контактных схем электронно-вычислительных машин (ЭВМ) принято рассматривать несколько этапов их развития.

Этап первый – 1944–1960 гг. ЭВМ с элементной базой, построенной с использованием большого количества электронных ламп (рисунок 2).

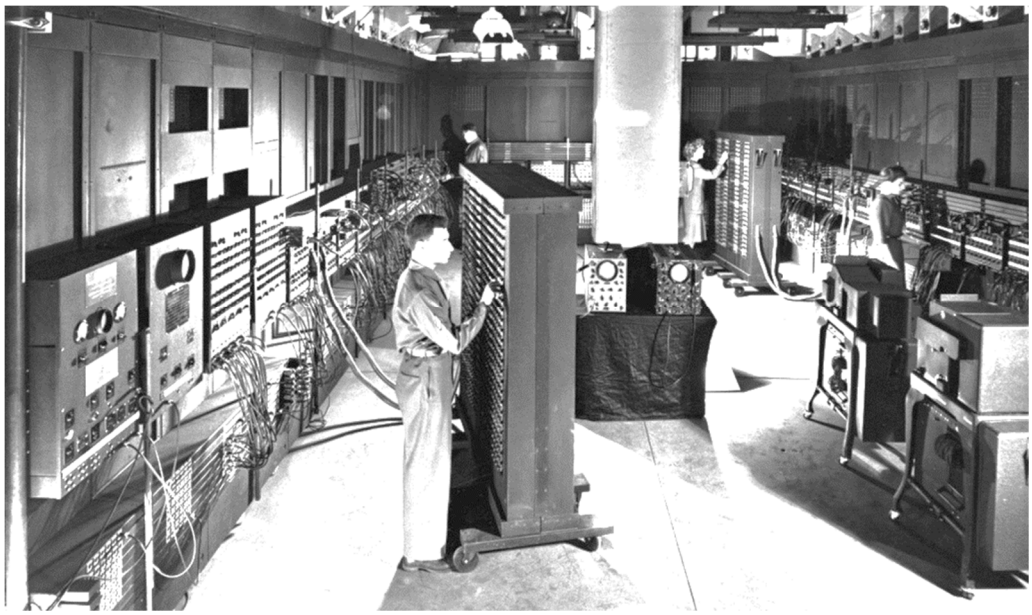


Рисунок 2. – Первая действующая ЭВМ ENIAC (США) 1945 г.

Этап второй – 1960–1970 гг. – электронные лампы заменяются полупроводниковыми приборами – транзисторами, память строится на магнитных схемах с ферритовыми сердечниками.

Этап третий – 1970–1980 гг. – появление и освоение производства интегральных схем, содержащих огромное количество транзисторов и конденсаторов, используемых в качестве ячеек памяти.

Этап четвертый – 1980 г. и по настоящее время – освоение массового производства гибридных схем, микроминиатюризация элементной базы ЭВМ, позволившая создать массовые вычислительные устройства.

1.2. Общие понятия и принципы построения архитектуры ЭВМ

Архитектура ЭВМ – это абстрактное представление ЭВМ (или персональной ЭВМ), которое отражает её структурную, схемотехническую и логическую организацию.

Информация – сведения о чем-либо, совокупность данных об определенном объекте, явлении, выраженные в определенном виде.

Аналоговый вид информации – сравнение с аналогом (по степени отклонения стрелки, яркости, цвету, вкусу). Этот вид информации является в большой степени субъективным и малопригодным для использования в ЭВМ. Иногда создаются отдельные конструкции аналоговых ЭВМ, предназначенные для узкоспециализированных целей моделирования некоторых физических явлений.

Дискретный вид информации – выраженный в виде цифровых значений. Этот вид информации более точен и объективен, позволяет передавать и обрабатывать информацию в отличие от аналогового вида двумя принципиально разными способами: последовательно и параллельно.

Цифровой автомат (ЦА) – всякая искусственная система обработки информации, представленной в дискретной форме (рисунок 3).

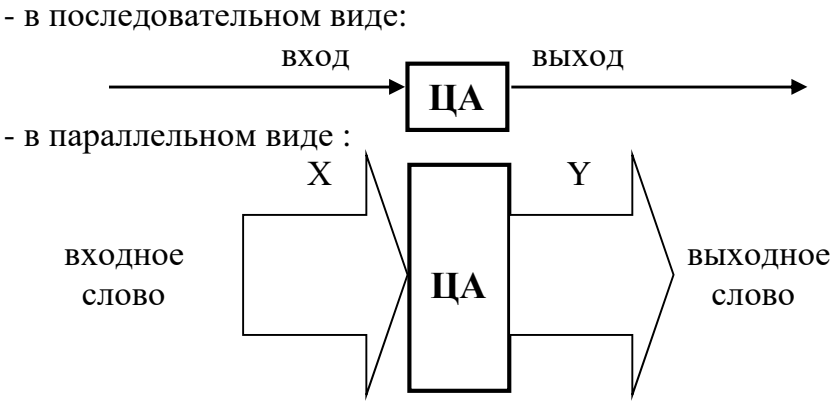


Рисунок 3. – Схема обработки информации цифровым автоматом

Работа ЦА заключается в преобразовании входного слова того или иного алфавита в выходное слово.

Алфавит – совокупность символов, из которых состоят слова информации.

Преобразование слов в ЦА происходит только в строго определенные моменты времени, которые называются ***тактовыми***.

Интервал времени между двумя тактовыми моментами называется ***тактом***, величина обратная такту – ***тактовой частотой***.

Длительность тактов может быть неодинаковой, но в общем случае они равновелики и определяются генератором синхронизации (ГС).

Скорость работы процессоров ЭВМ характеризуется значением тактовой частоты, которая может достигать нескольких десятков гигагерц.

Основой для создания любой программы и классической архитектуры ЭВМ являются принципы построения алгоритма, разработанные математиками ещё в IX – XII веках:

1. Алгоритм – это жесткая инструкция для исполнителя.

2. Отдельный алгоритм может составлять часть другого, более сложного алгоритма.

3. Любой алгоритм является дискретным, т.е. распадающимся на ряд последовательных предписаний.

4. Исполнитель вычислений не должен ошибаться.

5. Алгоритм универсален, т.к. работает с цифровой обезличенной информацией.

6. Алгоритм результативен: он всегда имеет начало и завершение.

Для создания проекта первой ЭВМ ENIAC были привлечены ученые двух крупнейших университетов США: Принстонского и Гарвардского. Профессор Принстонского университета Джон фон Нейман выполнил задание существенно быстрее, сформировав основные принципы, которые и были положены в основу создания первой ЭВМ:

1. ***Принцип кодирования информации.*** Информация кодируется в двоичной форме и распределяется на единицы (элементы) информации, называемые «словами».

2. ***Принцип использования информации.*** Разнотипные слова информации различаются по способу использования, размеру, но не по способу кодирования.

3. ***Принцип адресации.*** Слова информации размещаются в ячейках памяти ЭВМ и идентифицируются номерами ячеек (адресами).

4. ***Принцип построения программы.*** Алгоритм решаемой задачи представляется в форме последовательности управляющих слов – ***команд***, которые определяют наименование операции и слова информации, участвующие в операции. Алгоритм вычислейний, представленный в терминах машинных команд, называется ***программой***.

Официально считается, что первую программу, представленную в виде таблицы с полем данных (символов), полем результата и набором основных и служебных (системных) команд, написал Алан Тьюринг (1936 г.), её назвали «Машиной Тьюринга».

5. ***Принцип исполнения программы.*** Выполнение вычислений, предписанных алгоритмом, сводится к последовательному выполнению команд в порядке, однозначно определяемом программой. Первой выполняется команда, заданная пусковым адресом программы. Адрес следующей команды однозначно определяется в процессе выполнения текущей команды и может быть либо адресом следующей по порядку команды (естественный ход вычислений), либо адресом любой другой команды, адрес которой был определен в процессе выполнения предыдущей команды.

Процесс вычислений продолжается до тех пор, пока не будет выполнена команда, предписывающая прекращение вычислений.

Общее определение: ЭВМ – это программно управляемая инженерная система, предназначенная для восприятия, хранения, обработки и передачи информации (рисунок 4).

Определение Джона фон Неймана: ЭВМ – это универсальный цифровой автомат, работающий под управлением программы, хранящейся в его памяти.

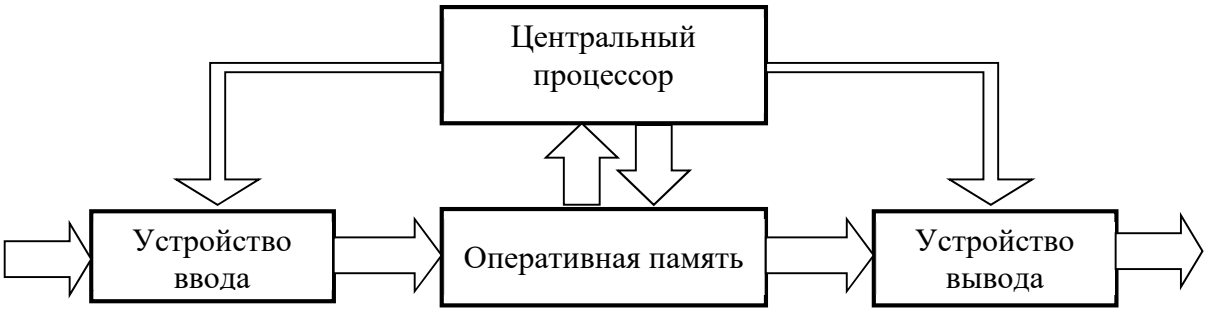


Рисунок 4. – Классическая схема базовой архитектуры ЭВМ

Устройство ввода – ввод информации с различных носителей.

Устройство вывода – вывод информации в форме, доступной человеку.

Оперативная память – содержит программу обработки, сохраняет исходные данные и записывает промежуточные результаты.

Центральный процессор – универсальный цифровой автомат для обработки информации и управления вводом-выводом данных, содержащий арифметико-логическое устройство и схемы управления.

Однако несмотря на простоту и очевидные достоинства классической архитектуры ЭВМ она обладает рядом недостатков:

– ЭВМ этой архитектуры работает только с цифровыми данными, поэтому для создания изображений, например, требуются сложные аппаратно-программные структуры векторной графики.

– Несоответствие машинных операций операторам языков высокого уровня, для исполнения которых иногда требуется несколько сотен машинных операций. Разрядность кодов машинных операций даже для последних моделей центральных процессоров остается на уровне первых ЭВМ, т.е. не превышает восьми бит.

– Устаревшая организация памяти, которая предполагает единое адресное пространство в отличие от Гарвардской архитектуры, в которой имеется адресное пространство хранения данных и отдельное адресное пространство хранения исполняемых программ.

– В классической архитектуре не предусмотрена параллельная работа нескольких процессоров при выполнении одной и той же расчетной задачи.

– Двоичное кодирование также требует дальнейшего усложнения.

В ЭВМ будущих поколений для использования «искусственного интеллекта» предполагается существенное усложнение классической архитектуры, которое постепенно осуществляется уже в настоящее время. Конечно, для использования уже созданного поколениями программистов контента потребуется создание соответствующих интеллектуальных трансляторов программного кода для его использования во вновь создаваемых архитектурах ЭВМ.

Тема 2. Кодирование информации в ЭВМ

2.1. Физический носитель информации в архитектуре ЭВМ

Наименьшая порция двоичной информации в ЭВМ называется «бит». Это понятие в 1948 г. предложил использовать американский ученый Клод Шеннон в своей работе по теории цифровых автоматов. Им же для обеспечения наилучшей помехоустойчивости была предложена схема представления «0» и «1» в различных электронных устройствах, в том числе ЭВМ, которая используется и в современных компьютерах (рисунок 5).

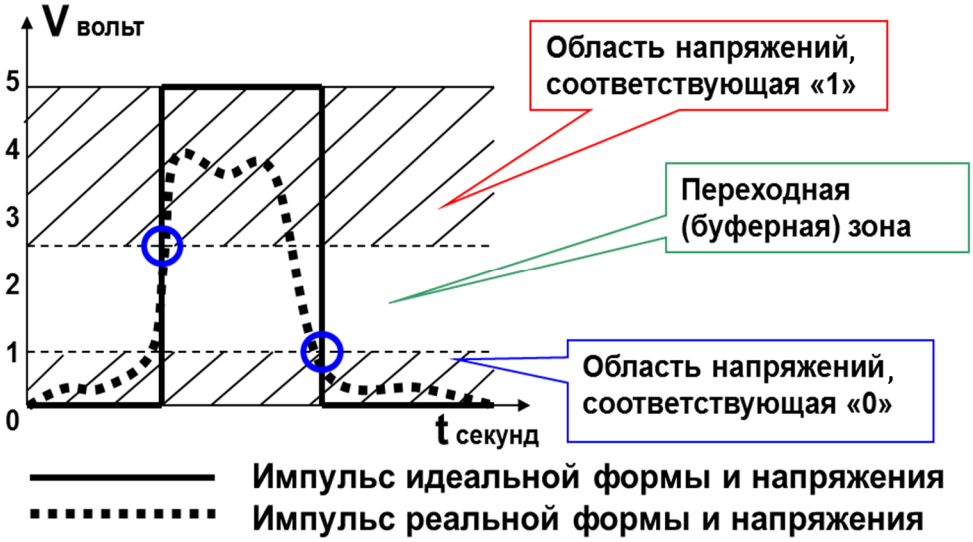


Рисунок 5. – VT-диаграмма носителя нуля и единицы в ЭВМ

Для отображения единицы («1») в электрической схеме ЭВМ используется напряжение, снимаемое с источника питания +5 В (вольт). Все провода этого напряжения от блока питания обычно маркируются красным цветом. Системные решения определяют реальные рабочие уровни напряжения «1», которые могут отличаться от исходного в меньшую сторону. При этом безразлично, что является причиной этого напряжения: заряд конденсатора, сопротивление p-n-перехода транзистора или полупроводникового слоя ячейки памяти. Длительность импульса задается генератором синхросигнала. Для обеспечения условий энергосбережения для некоторых устройств из состава архитектуры ЭВМ, например, ядра центрального процессора или оперативной памяти, информационное напряжение может быть снижено до 1,75 В и менее с пропорциональным уменьшением областей «0», «1» и буферной зоны. Реальное значение этого напряжения обычно указывается в маркировке центрального процессора или модулей оперативной памяти.

2.2. Двоичное кодирование целых чисел

Двоичное число, носителем которого является напряжение, называют двоичным ***разрядом*** или ***битом.*** Восемь разрядов, объединенных в единое целое, называется ***байтом*** и изображаются

1	1	0	0	0	1	0	0

Два байта составляют ***машинное слово***.

Машинное слово есть фрагмент данных фиксированного размера, обрабатываемый процессором как единое целое. Этот размер может отличаться для различных процессорных архитектур, но на процессорах архитектуры x86 «словом» традиционно называют 16 бит (2 байта), хотя эти процессоры могут одновременно обрабатывать и более крупные блоки данных. В синтаксисе языка ассемблера (машинно-ориентированный язык программирования низкого уровня) для всех процессоров архитектуры x86 для обозначения блока данных размером 2 байта принята аббревиатура **DW** (define word), что можно перевести как «определить слово».

Числа кодируются прямым двоичным кодом, т.е. простым переводом десятичного числа в двоичную форму. Простые числа без знака занимают все разряды байта (**от 0 до 255**).

Кодирование простых чисел со знаком предполагает кодирование и знака числа: знак числа 0 – «+» 1 – «-»

1	1	0	0	0	1	0	0

Пределы представления числа со знаком: **от –128 до +127**.

Но при таком представлении числа возникает проблема представления нуля. Для вычислительной машины существует два нуля: +0 и –0, так как 00000000 ≠ 10000000.

Существует решение этой проблемы: числа с ***кодом со сдвигом***. База сдвига кода определяется старшим разрядом байта, равным 10000000. В коде со сдвигом все числа положительные. Число в коде со сдвигом получается за счет дополнения базы сдвига реальным числом с учетом его знака. Например: 10000000 + 11001 = 10011001 – положительное число +25, 10000000 – – 11001 = 01100111 – отрицательное число –25. Пределы представления однобайтовых чисел в коде со сдвигом: **от –128 до +127**.

Кроме такого представления простых чисел со знаком для исключения операции вычисления знака числа может быть использован метод формирования ***дополнительного*** и ***обратного кодов***.

Обратный n-разрядный двоичный код положительного целого числа состоит из одноразрядного кода знака (двоичной цифры 0), за которым следует (n−1)-разрядное двоичное представление модуля числа. Обратный n-разрядный двоичный код отрицательного целого числа состоит из одноразрядного кода знака (двоичной цифры 1), за которым следует (n−1)-раз-

рядное двоичное число, представляющее собой инвертированное (n−1)-разрядное представление модуля числа. Для изменения знака числа необходимо проинвертировать все его разряды, включая знаковый.

Для преобразования отрицательного числа в положительное тоже применяется операция инвертирования. В качестве недостатка следует отметить, что в обратных двоичных кодах имеются два кода числа 0: «положительный нуль» 0000 0000 и «отрицательный нуль» 1111 1111. Это приводит к усложнению операции суммирования. Поэтому в дальнейшем перешли к дополнительным кодам записи знаковых целых чисел.

Дополнительный код – наиболее распространённый способ представления отрицательных целых чисел в современных компьютерах. Он позволяет заменить операцию вычитания на операцию сложения и сделать операции сложения и вычитания одинаковыми для знаковых и беззнаковых чисел. «Обратный код» называют «дополнением единиц», а «дополнительный код» называют «дополнением двоек.

Дополнительный код для отрицательного числа можно получить инвертированием его двоичного модуля и прибавлением к инверсии единицы, либо вычитанием числа из нуля. Например:

Исходное число	Прямой код	Смещенный код	Дополнительный код	Обратный код
+7	0,0111	1,0111	0,0111	0,0111
–7	1,0111	0,1001	1,1001	1,1000
–5	1,0101	0,1011	1,1011	1,1010
0	0,0000	1,0000	0,0000	0,0000

Увеличение пределов значимой части числа достигается за счет количества байтов в слове. Как правило, целые числа используются только для кодирования индексов или показателей степени.

При выборе целочисленного типа кодирования следует помнить, что, при выходе за пределы допустимого диапазона разрядности числа, соответствующие «лишние» старшие биты отбрасываются без диагностики переполнения, поэтому получившийся результат может быть искажен.

2.3. Двоичное кодирование вещественных чисел

Кодирование вещественных чисел (или так называемых чисел с плавающей точкой) в вычислительной технике осуществляется с использованием международного стандарта IEEE 754.

Число **R** формата представляется в виде произведения знака **s** мантиссы **m** и экспоненты $\mathbf{2^{p-d}}$:

$$R = s \cdot m \cdot 2^{p-d}.$$

Число **p** называется порядком или степенью вещественного числа. Оно является изменяемой частью вещественного числа.

Число **d**, называемое сдвигом порядка, постоянное для всех чисел заданного типа.

Таким образом побитовое представление числа real*4 будет выглядеть следующим образом. Первый бит – знак числа. Затем восемь битов – порядок (степень) числа, представленная в смещенном коде, где **d** – база смещения. Оставшиеся 23 бита занимает мантисса числа без первой единицы, которая вставляется по умолчанию.

Для определения порядка и мантиссы вещественное число необходимо привести к нормализованному виду, например, десятичное число –25 после приведения к нормализованному виду равно $-25 = -0{,}25 \times 10^2$.

Преобразуем его в двоичное число, получим: $0{,}11001 \times 2^{101}$.

Запишем его в стандарте IEEE 754 (рисунок 6.).

s	Порядок (8 бит)								Мантисса (23+1 бит)																							
1	1	0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рисунок 6. – Схема кодирования вещественного числа в ЭВМ

В связи с тем, что мантисса двоичного числа всегда начинается с единицы, её исключают из кода вещественного числа для экономии места.

Такая запись кода вещественного числа называется базовой. При нулевой степени вещественного числа существует особый порядок записи.

Базовая 4-хбайтовая запись кода вещественного числа позволяет кодировать вещественные числа в пределах **от 10^{-38} до 10^{39}** и размером мантиссы до 8-мизначных цифр.

Для длинных вещественных чисел предусмотрено два дополнительных формата real*8 и real*10, т.е. 8-ми и 10-тибайтовые форматы соответственно с 23-х и 53-хбитной мантиссой.

Возможны типичные ошибки при работе с кодами вещественных чисел. Так, сравнение двух вещественных чисел в формате с плавающей запятой на равенство или неравенство в отличие от точной арифметики, когда они должны быть равны, не приводит к желаемому результату. Например, при умножении чисел 2*2, записанных в формате чисел с плавающей запятой, обычно получаем 3,9999…, а не 4.

2.4. Кодирование символьной информации

Принцип кодирования символов заключается в присвоении порядковых номеров стандартным изображениям символов, используемых в алфавите.

Сами изображения символов представляют собой т.н. «текселы», т.е. элементарные изображения в виде матрицы 8×8 пикселов, объединенные в массив 8-байтовых элементов и хранящийся в ПЗУ BIOS по адресу F000:FA6E. Например, прописной латинский символ «a», стоящий под номером 97 таблицы ASCII имеет вид, показанный на рисунке 7.

0	0	0	0	0	0	0	0	00
0	0	0	0	0	0	0	0	00
0	0	0	0	0	0	0	0	00
0	0	1	1	1	0	0	0	38
0	0	0	0	1	1	0	0	0C
0	1	1	1	1	1	0	0	7C
1	1	0	0	1	1	0	0	CC
0	1	1	1	0	1	1	0	76

Рисунок 7. – Схема представления символьной информации в ПЭВМ

В массиве данных строку с адреса FFD75: 00 00 00 38 0C 7C CC 76 преобразуем к двоичному виду. При этом единицы будут означать горящие пикселы, а ноль – погашенные.

На первый (из 8) байт элемента тексела в ПЗУ BIOS указывает вектор прерывания Int1f, расположенный в таблице векторов по адресу 0000:007C. Вектор используется, как указатель на массив *расширения* таблицы ASCII для считывания 8-ми байт элемента изображения тексела (например, начиная с F000:FD75 для символа «а»).

К сожалению стандартная таблица ASCII IBM не содержит изображений символов кириллицы. Для этой цели принято изображение национальных шрифтов помещать в таблицу расширения ASCII, т.е. с номера 128 и далее, на котором в стандартной таблице расположены символы греческого алфавита. В MS DOS страница ASCII с кириллицей имеет номер 866.

Существует несколько стандартизованных таблиц изображений символов. Фирма Intel в базовых конфигурациях BIOS своих изделий, да и другие разработчики BIOS, используют таблицу ASCII.

Существуют и советские аналоги для ЭВМ отечественного изготовления: КОИ-7, КОИ-8 и т.д. (от слов «КОдирование Информации»).

Фирма MicroSoft для OS Windows с целью облегчения применения различных шрифтов и знаков использует собственную таблицу ANSI. Кириллический шрифт в этой таблице смещен по номерам знакомест по отношению к таблице ASCII, что вызывает появление символов греческого алфавита при загрузке таблицы ANSI для текстов, написанных в период действия таблицы ASCII. Эта проблема обычно легко решается, если при перезагрузке таблиц выполнять изменение и номеров символов кириллицы для ASCII по таблице ANSI.

Ниже показано смещение номеров символов кириллицы для различных таблиц символов:

ANSI (WINDOWS)	ASCII 866 (MS DOS)
$192 \le N_{(ANSI)} < 240$	$N_{(ASCII)} = N_{(ANSI)} - 64$
$240 \le N_{(ANSI)} < 256$	$N_{(ASCII)} = N_{(ANSI)} - 16$

$N_{(ANSI)} = 184$ $N_{(ASCII)} = 241$
$N_{(ANSI)} = 168$ $N_{(ASCII)} = 240$

Для различных офисных приложений, разработанных фирмой MicroSoft и другими (MS Word, MS Excel, AutoCAD, CorelDraw, PhotoShop и им подобных) сздано очень большое количество различных шрифтов для различных областей приложений. Все образцы написания символов этих шрифтов обычно находятся в каталоге C:\Windows\Fonts в папке с наименованием соответствующего шрифта. При открытии файла шрифта, выполненного в русифицированном варианте, можно видеть характерную фразу «Съешь еще этих мягких французских булок, да выпей чаю. 1234567890», написанную в нескольких размерах.

Обеспечена возможность и самостоятельной разработки как шрифтов, так и символов различного назначения. Основное условия применения шрифтов: обеспечение увязки номеров символов с соответствующими СКЭН-кодами клавиатуры и алгоритмом обработки этих СКЭН-кодов через загруженную BIOS- и OS-таблицу символов, что обеспечивается соответствующими редакторами шрифтов, например FontLabs, FontForge и другими.

Тема 3. Понятие о микропрограммном автомате (МПА)

3.1. Принцип микропрограммного управления

ЭВМ (или ПЭВМ, что принципиального значения не имеет) это сложная многоуровневая инженерная система, состоящая из большого количества самостоятельных устройств обработки информации. Прежде всего, это клавиатура, манипулятор мышь, дисплей, графический контроллер, контроллеры связи (USB, PS/2, мостовые контроллеры, контроллеры шин), контроллер памяти, а «на вершине пирамиды» центральный процессор, являющийся первым среди равных. Каждое из этих устройств можно рассматривать в виде самостоятельного цифрового автомата, который ведет обработку поступающей информации, преобразуя её из одного вида в другой, и передает далее по цепочке к центральному процессору.

Обработка информации цифровым автоматом не может осуществляться произвольно, она включает в себя выполнение множества операций: запись в буфер обмена, передача в сумматор, шифратор, мультиплексор или иное операционное устройство, преобразование, контроль завершения, генерацию сигнала готовности следующему устройству, ожидание сигналов подтверждения и т.д. Порядок функционирования этих устройств и их структура базируются на ***принципе микропрограммного управления***, который заключается в следующем:

1. Любая операция обработки информации, реализуемая устройством, рассматривается, как сложное действие, которое разделяется на последовательность элементарных действий над словами информации, называемых ***микрооперациями***.

2. Для управления порядком следования микроопераций используются ***логические условия***, которые определяются содержанием слов информации или готовностью устройств, участвующих в микрооперации.

3. Процесс выполнения микроопераций в устройстве описывается в форме алгоритма, представляемого в терминах микроопераций и логических условий и называемого ***микропрограммой***. Микропрограмма определяет порядок проверки логических условий и следования микроопераций, необходимый для получения требуемого результата.

4. Микропрограмма используется как форма представления функции устройства, на основе которой определяется структура устройства, набор сигналов управления отдельными элементами устройства и порядок работы устройства во времени (тактовая диаграмма).

3.2. Концепция операционного и управляющего автоматов

В любом таком устройстве обработки цифровой информации можно выделить два основных блока (автомата) – ***операционный***, который собственно и занят обработкой и преобразованием информации (изменение разрядности, распределение по блокам, запись в памяти, чтение из памяти, сложение слов, вычитание, передача из регистра в регистр, шифрование слов или дешифровка и т.д.) и ***управляющий***, который выполняет синхронизацию процесса, переключение регистров, соединение с внешней и внутренней шинами, переключение счетчиков и т.д.

Операционный блок (или операционный автомат) любого устройства из состава архитектуры ЭВМ состоит из буферных регистров, сумматоров, шифраторов, дешифраторов, мультиплексоров, демультиплексоров, счетчиков и других узлов, обеспечивающих прием из внешней среды, обработку и выдачу во внешнюю среду цифровой информации, а также выдачу в управляющий блок и внешнюю среду оповещающих сигналов о состоянии операционного блока.

Управляющий блок (или управляющий автомат) вырабатывает распределенную во времени последовательность управляющих сигналов, порождающих в операционном блоке нужную последовательность микроопераций. Совокупность управляющих сигналов зависит от кода операции и совокупности поступающих оповещающих сигналов.

Схемно для любого устройства из состава архитектуры ЭВМ операционный и управляющий блоки показаны на рисунке 8.

Операционный автомат служит для приема слов информации D_{IN} в различном формате, их хранения, преобразования и выдачи во внешнюю среду D_{OUT}. Этот процесс сопровождается выработкой множества информационных сигналов {u}, используемых в качестве логических условий для управляющего автомата.

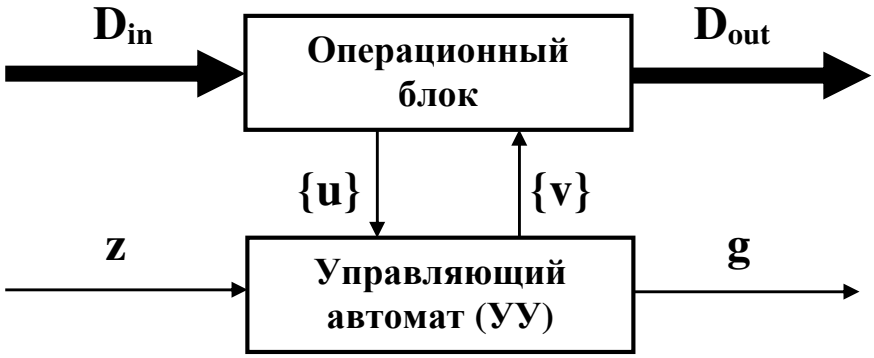


Рисунок 8. – Функциональная схема устройства обработки информации

Управляющий автомат генерирует последовательность управляющих сигналов {v}, распределенных во времени, обеспечивающих выполнение в операционном автомате заданной последовательности элементарных действий, которая реализуется алгоритмом выполняемой операции.

Обычно управляющие автоматы объединяют в единый блок, называемый ***устройством управления (УУ)***, а конкретный алгоритм (микропрограмма) активируется кодом операции, который дешифруется в конкретный сигнал Z, определяющий тип выполняемой операции.

После завершения операции управляющий автомат активирует сигнал g, отмечающий завершение операции и готовность выходных данных D_{OUT}.

3.3. Пример проектирования операционного автомата

В качестве исходных данных для разработки структуры операционного автомата любого устройства в ЭВМ являются:

– описание формата входных и выходных слов (множеств $\{D_{IN}\}$ и $\{D_{OUT}\}$);

– список множества операций из F, которые должны выполняться над словами;

– разработка общей структуры операционного автомата;

– построение тактовой диаграммы сигналов управления;

– разработка алгоритма управления операционным автоматом, который должен быть реализован устройством управления (управляющим автоматом).

В качестве примера операционного автомата рассмотрим упрощенную схему сумматора простых чисел.

Определение формата данных. Будем считать, что в операции сложения участвуют два 32-разрядных числа.

Список множества операций, выполняемых над словами информации в проектируемом устройстве. Слагаемые операнды поступают с системной шины памяти по адресам памяти, определенным схемой страничной адресации, которая в данной структуре операционного автомата не рассматривается, т.к. является функциональной частью контроллера памяти.

Операнды поступают поочередно через буферный регистр с контролем готовности контроллера памяти, для чего предусмотрены пустые циклы.

После срабатывания сумматора результат помещается в постоянно открытый аккумулятор, а из него – во внешний буфер обмена системной шины для размещения по адресу второго операнда.

Разработка структурной схемы операционного автомата. Структурная схема устройства показана на рисунке 9.

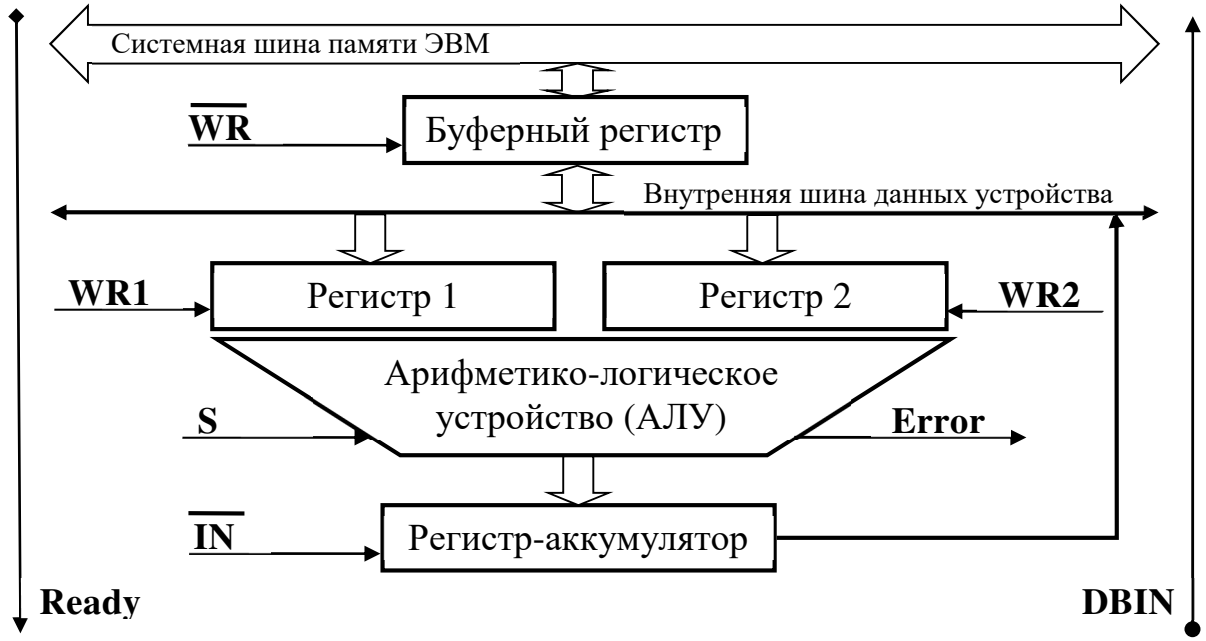


Рисунок 9. – Структурная схема операционного автомата устройства

Основным логическим условием состоявшейся операции сложения будет служить отсутствие сигнала переполнения разрядной сетки «Error». Признаком завершения – сигнал «DBIN» (открытие внешнего регистра на чтение).

Основные сигналы управления операционного автомата:

1. WR – открытие внешнего регистра системной шины на запись. Сигнал постоянно активен, а регистр постоянно открыт на шину данных.
2. WR1 – открытие первого буферного регистра на запись.
3. WR2 – открытие второго буферного регистра на запись.
4. S – сигнал активации сдвигового сумматора АЛУ.
5. IN – открытие регистра-аккумулятора на запись (сигнал постоянно активен, т.е. регистр постоянно открыт в сторону сумматора).
6. DBIN – открытие внешнего регистра на чтение данных в память.

Основные оповещающие сигналы.

1. Ready – сигнал готовности контроллера памяти к операции.

2. Error – сигнал, источником которого является единица в старшем (дополнительном) разряде сумматора (переполнения разрядной сетки).

Построение тактовой диаграммы сигналов управления. Тактовая диаграмма строится в координатах тактов, т.е. положительных фронтов синхросигнала генератора тактовых импульсов. Порядок использования синхросигнала (по фронту или уровню) определяется свойствами электронных модулей, установленных в операционном блоке. Если уровень сигнала значения не имеет, то он изображается в виде прямоугольника (рисунок 10).

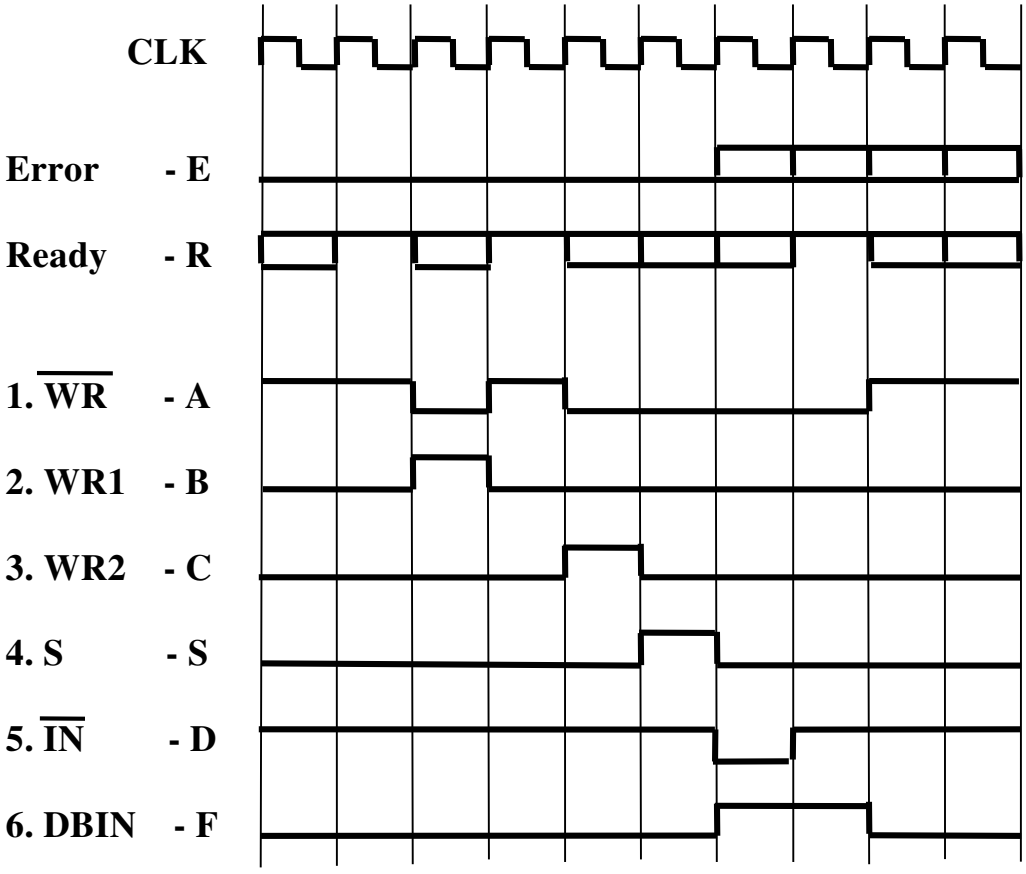


Рисунок 10. – Тактовая диаграмма сигналов управления

Разработка алгоритма управления операционным автоматом.

Алгоритм управления операцией сложения двух чисел разрабатывается в виде граф-схемы, показанной на рисунке 11.

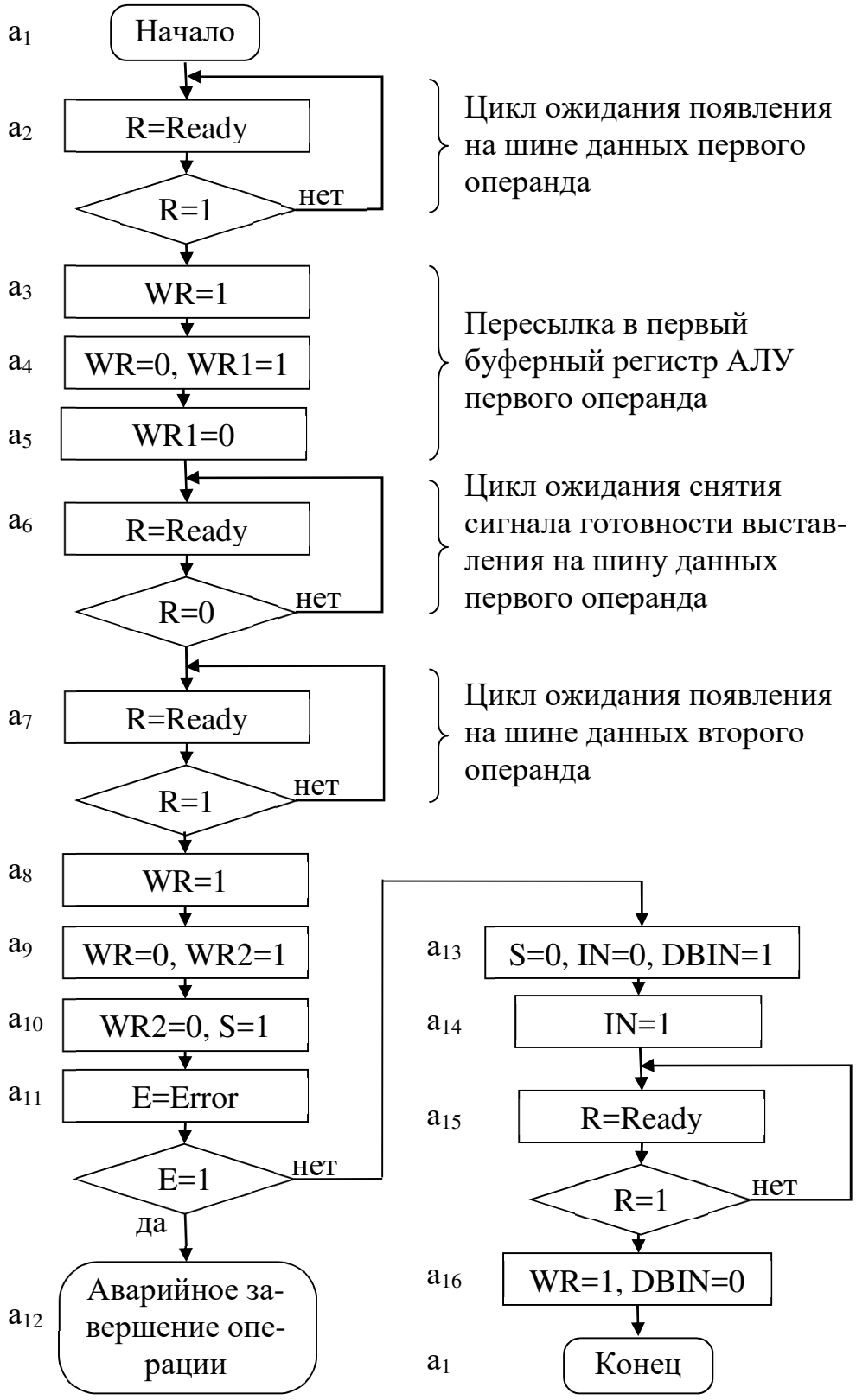


Рисунок 11. – Граф-схема алгоритма управления операционным автоматом (автомат Мура) при выполнении операции сложения двух простых чисел

3.4. Управляющий автомат (устройство управления)

Любая команда, операция или процедура, выполняемая в операционном блоке, описывается некоторой микропрограммой или комбинационной схемой и реализуется за несколько циклов, в каждом из которых выполняется одна или несколько микрокоманд. Управляющий автомат (или устройство) с микропрограммным управлением принято называть микропрограммным автоматом (МПА).

Проектирование МПА основано на схемной или микропрограммной реализации алгоритма управления, соответствующего автомату Мура, граф-схема которого, например, приведена на рисунке 11.

Различают два основных класса управляющих автоматов:

– управляющие автоматы с «жесткой» логикой (автомат Мура, автомат Мили или С-автомат);

– управляющие автоматы с программой в памяти (программируемой логикой выполнения операций).

Следует отметить различия автоматов Мура и Мили: для первого выход зависит только от текущего состояния, для автомата Мили выход зависит от текущего состояния, а также от текущего ввода.

3.4.1. Управляющий автомат с «жёсткой» логикой

Автоматы с «жесткой» логикой проектируются как обычные конечные структурные автоматы. Каждой операции, задаваемой кодом операции, строится конкретный набор комбинационных схем, которые в соответствующих тактах вырабатывают соответствующие сигналы управления. Комбинационные схемы могут содержать триггерные ячейки памяти, дешифраторы, мультиплексоры, сумматоры и т.д.

Чтобы получить более полное представление об МПА с «жесткой» логикой рассмотрим порядок их проектирования.

1. Выполняется разметка алгоритма микропрограммы (см. рисунок 11), каждому состоянию автомата, согласно алгоритму микропрограммы, ставится в соответствие вершина графа a_n. Начальную и конечную вершины графа сопоставляют с начальным состоянием автомата a_1.

2. Строится графо-аналитическая модель автомата, заданного размеченной микропрограммой. Все вершины этого графа соответствуют возможным состояниям автомата, показанным на тактовой диаграмме (см. рисунок 10). Соединим ориентированными ребрами те пары вершин графа, между которыми существуют переходы. При этом пометим ребра графа соответствующими условиями перехода. Построенный таким способом граф (рисунок 12) фактически задает алфавиты внутренних состояний управляющего автомата и входных символов, а также определяет функции переходов. Обозначения (a_1 – a_{16}) соответствуют обозначениям граф-схемы рисунка 11.

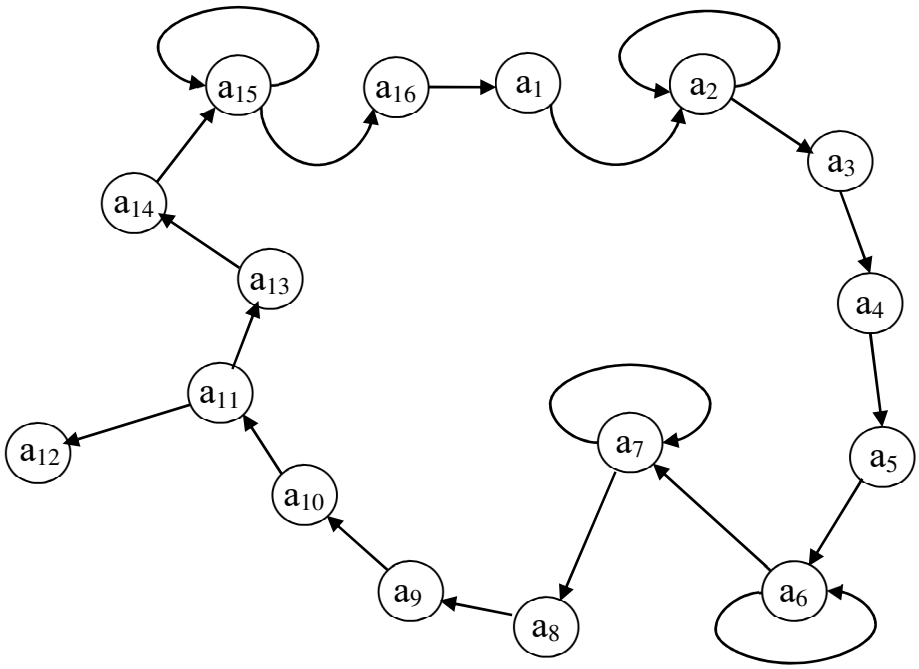


Рисунок 12. – Граф микропрограммного автомата Мура для управления операционным автоматом (блоком) сложения двух чисел

3. Выполняется кодирование входных и выходных символов. В данном случае входными символами будут состояния оповещающих сигналов Ready и Error, а также номер такта операции, а выходными символами – значения управляющих сигналов.

4. В процессе кодирования внутренних состояний автомата должны быть решены проблемы возможных перекрестных ссылок. Составляется таблица кодов состояния автомата.

5. Выбирается тип элементов памяти (триггеров) D- или T-триггеры, если они необходимы для реализации алгоритма управления.

6. Для управления триггерами строится автоматная таблица переходов. Строка таблицы соответствует одному переходу автомата. Таким образом, в таблице столько строк, сколько ребер у графа, включая ребра-петли. Разрядность функции возбуждения соответствует количеству триггеров, а значение исходного состояния – номеру вершины графа.

7. Выполняется синтез комбинационной схемы, реализующей функции переходов автомата.

8. Выполняется синтез комбинационной схемы, реализующий функцию выходов. Функция выходов автомата Мура, который не имеет внутренней памяти и потому зависит только от его внутреннего состояния, задана непосредственно на графе автомата. Т.е. выходами МПА являются требуемые значения управляющих сигналов в каждой микрооперации.

9. Решаются проблемы гонок (процесс распространения сигналов в различных цепях при существовании разбросов временных задержек этих цепей) и риска сбоев.

10 Проводится декомпозиция в виде сети связанных между собой компонентных автоматов.

11. Строится функциональная схема МПА.

Управляющий автомат с жесткой логикой реализуется аппаратными средствами, схемотехническими конструкциями.

Для формирования счетчика-дешифратора тактов в автомате с жесткой логикой используется сдвиговый регистр.

Обычно логические схемы МПА с «жесткой» логикой строятся на базе программируемых логических матриц (ПЛМ). Программирование ПЛМ осуществляется фотошаблонированием за счет выжигания ненужных логических связей. Функциональная схема сложного многопрограммного МПА с «жесткой» логикой показана на рисунке 13.

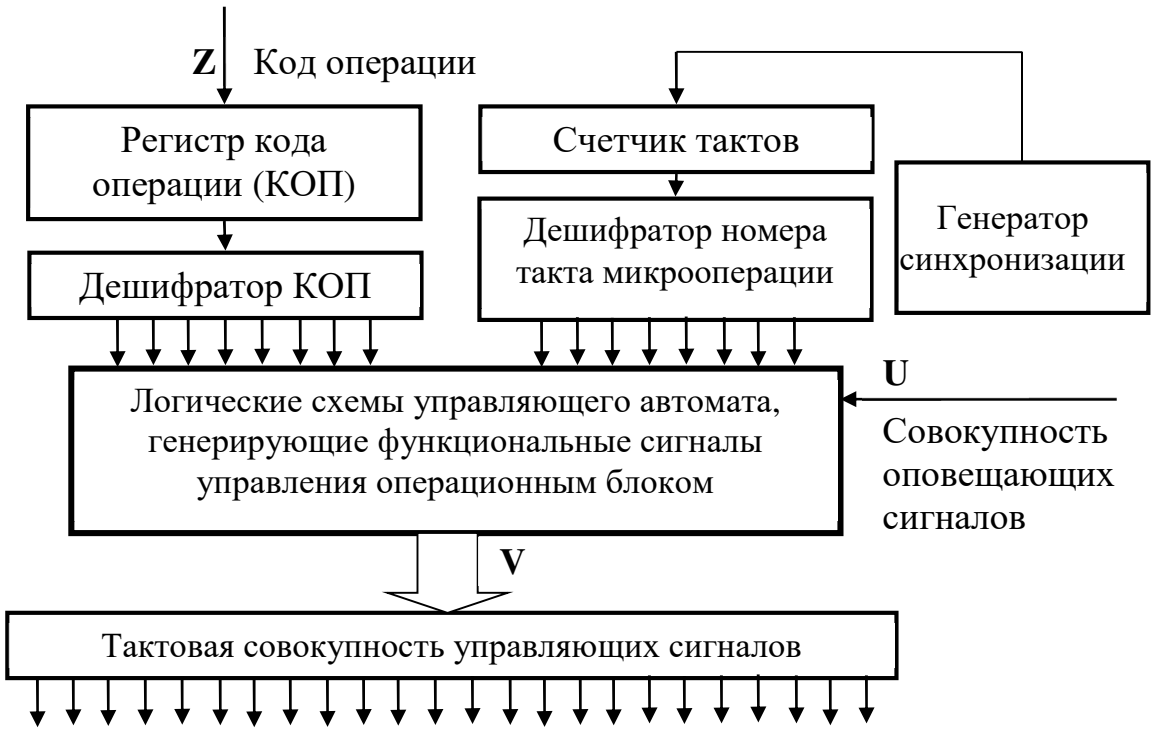


Рисунок 13. – Функциональная схема управляющего автомата с «жесткой» логикой

До 2010 г. МПА с «жесткой» логикой на базе комбинационных схем были очень распространены, т.к. являлись достаточно скоростными. Но изменение этих схем требует перепроектирования аппаратных средств реализации логики, и на определенном этапе эта сложность становилась непреодолимой. Кроме этого, логические схемы статичны и рассчитаны на ряд простых функциональных операций. Изменение состава микрокоманд и функциональных сигналов также требует перепроектирования схем «жесткой» логики управляющих автоматов.

Более глубокое изучение методов низкоуровневого программирования операционных автоматов с «жесткой» логикой (программируемых логических интегральных схем ПЛИС) будет осуществляться на курсе «Системотехника цифровых устройств», предусмотренном образовательным стандартом данных дисциплин.

3.4.2. Управляющий автомат с программируемой логикой

Функция любого управляющего автомата – генерирование последовательностей управляющих сигналов в соответствии с реализуемым алгоритмом обработки информации в операционном блоке и совокупности оповещающих сигналов. Если в запоминающем устройстве заранее разместить все необходимые для реализации заданного алгоритма микрокоманды, содержащие эти совокупности сигналов, а потом выбирать их из памяти в том порядке, который предусмотрен алгоритмом, с учетом сигналов оповещения, то получим управляющий автомат, структура которого является универсальной и практически не зависит от реализуемых алгоритмов, т.к. в основном определяется содержимым памяти. При изменениях реализуемого алгоритма в широких пределах структурная схема такого автомата изменяться не будет, достаточно лишь изменить содержание памяти (переписать микропрограмму).

Управляющий автомат, построенный по такому принципу, называется ***управляющим автоматом с программируемой логикой***. Функциональная схема приведена на рисунке 14.

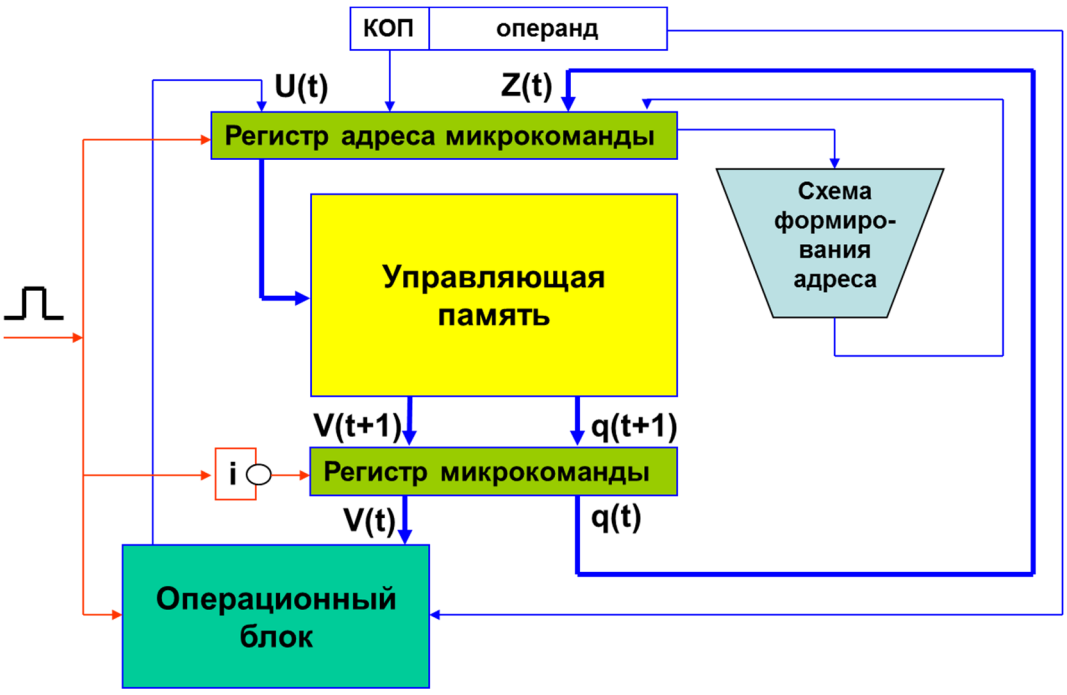


Рисунок 14. – Функциональная схема управляющего автомата с программируемой логикой

Управляющий автомат такого типа содержит всего несколько элементов: управляющую память, регистр выбранной микрокоманды и регистр формирования адреса следующей микрокоманды.

В управляющих автоматах с программируемой логикой коду операции (КОП) ставится в соответствие микропрограмма, т.е. совокупность хранимых в памяти слов – микрокоманд. Каждая микрокоманда состоит из двух полей. Первое поле содержит информацию в явной или зашифрованной форме о состоянии сигналов управления микрооперациями V(t), подлежащими выполнению в течении одного машинного такта, которые поступают в операционный блок. Второе поле – часть адреса следующей микрокоманды q(t), который дополняется значениями сигналов оповещения в регистре формирования адреса.

Совокупность микропрограмм хранится в специальном ПЗУ. Микропрограммы используются в явной форме, программируются в двоичных кодах и в таком виде заносятся в память. Воздействие сигналов V(t) на операционный блок синхронизируется сигналом генератора, открывающего регистр микрокоманды в соответствующие тактовые моменты.

3.4.2.1. Принцип принудительной адресации микрокоманд. Разрядность памяти ПЗУ для хранения микропрограмм является критичной, т.к. количество сигналов управления в сложных операционных блоках (например, центральном процессоре) может достигать ста и более, а логических условий, определенных сигналами оповещения, – до десяти и более.

Для обеспечения сокращения размера памяти за счет адресной части, которая может многократно дублироваться в различных микропрограммах, адрес микрокоманды формируется специальной комбинационной схемой формирования адреса (см. рисунок 14). Схема заполнения адресной части показана на рисунке 15.

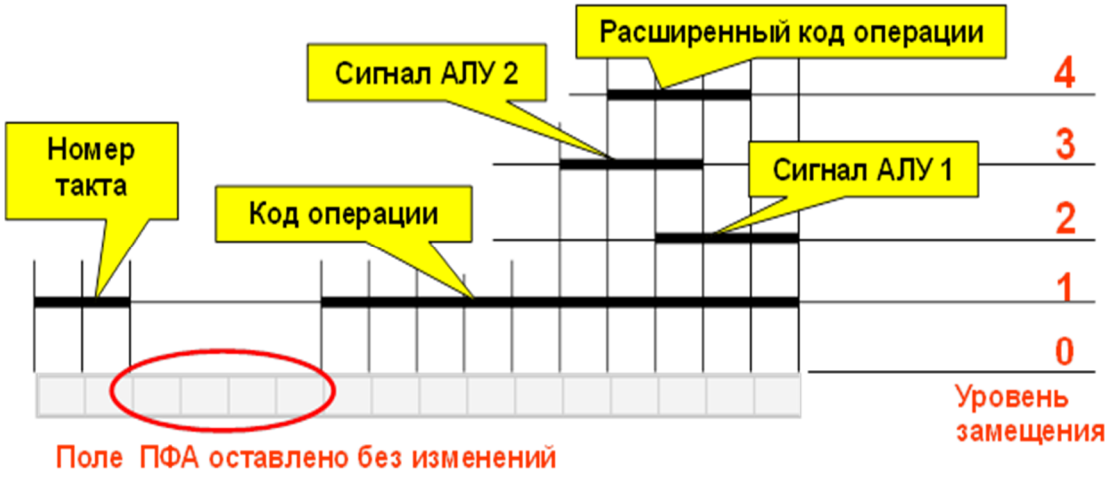


Рисунок 15. – Схема принудительного заполнения регистра адреса

Адрес очередной микрокоманды (микропрограмма обычно состоит из нескольких десятков микрокоманд) можно назначить без учета кода операции Z(t) и совокупности оповещающих сигналов U(t). Поле адреса разбивается по этому признаку на ряд групп, часть из которых заполняется принудительно (последовательно) схемой формирования адреса, а другие остаются без изменений, например:

– поле типа формирования адреса (ТФА), которое определяет набор специальных микропрограмм, например, микропрограмм без ветвления цепочек, т.е. с минимумом логических условий;

– поле формирования очередного адреса (ПФА), заполняемое данными из предыдущей микрокоманды.

Остальные поля заполняются с замещением имеющейся там информации по уровню актуальности. Например, код операции позволяет однозначно определить и вызвать на исполнение микропрограмму управления заданной операцией, а оповещающие сигналы АЛУ той или иной части содержимого регистра EFLAGS позволяют модифицировать процесс управления операцией в ходе её реализации в операционном блоке.

Такая схема формирования адресной части микрокоманд называется ***принудительной адресацией микрокоманд***. Ее основной недостаток заключается в появлении дополнительных тактов, затрачиваемых на формирование адреса.

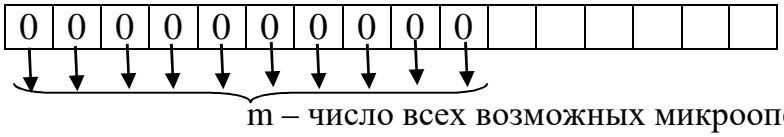
3.4.2.2. Принцип естественной адресации микрокоманд заключается в замене части поля адреса при выполнении цепочечных операций без ветвления значениями счетчика команд, используя естественный порядок следования микрокоманд. В этом случае в состав устройства формирования адреса включается регистр-счетчик адреса микрокоманд.

Схема формирования адреса при естественной адресации строится следующим образом: если заданное логическим признаком условие ветвления соблюдается, то выполняется микрокоманда по адресу, сформированному принудительно, в противном случае адрес следующей микрокоманды определяется по текущему значению счетчика, увеличенному на единицу.

3.4.3. Методы низкоуровневого микропрограммирования

Как уже отмечалось ранее, каждая микрокоманда состоит из двух типов полей: поля данных, описывающих состояние сигналов управления в текущем такте выполнения операций обработки информации в операционном блоке, и поля части адреса следующей микрокоманды, попадающей в поле ПФА регистра адреса. Существует несколько способов использования разрядов поля данных для формирования управляющих функциональных сигналов, направляемых в операционный блок: горизонтальное, вертикальное и смешанное микропрограммирование сигналов.

3.4.3.1. Горизонтальное микропрограммирование. Каждому разряду операционной части микрокоманды ставится в соответствие определенный управляющий функциональный сигнал, представленный в явной форме (0 или 1), вызывающий определенную микрооперацию в операционном блоке.



Достоинства метода: возможность в одном такте выполнить одновременно любой набор из m микроопераций, а также простота формирования управляющих сигналов, подаваемых в операционный блок непосредственно из регистра управляющего автомата.

Недостатки: требуется большая длина микрокоманды и, как следствие, число разрядов регистра, т.к. число сигналов может быть больше 100.

Такой метод микропрограммирования в настоящее время очень широко используется в мелкосерийном и любительском штучном производстве различных электронных изделий, содержащих микропрограммные автоматы, например, дверные звонки с набором мелодий, различные елочные гирлянды и украшения, электронные детские игрушки, некоторые медицинские приборы, измерительные инструменты, оснащенные индикаторами и т.д. Популяризации этого метода способствует дешевизна и доступность легко перезаписываемой флэш-памяти, которая используется для хранения микропрограмм.

3.4.3.2. Вертикальное микропрограммирование. При этом методе микропрограммирования функциональному сигналу ставится в соответствие не разряд, а двоичный код операционной части микрокоманды (есть даже код отсутствия микрооперации). Структура вертикальной микропрограммы показана на рисунке 16.

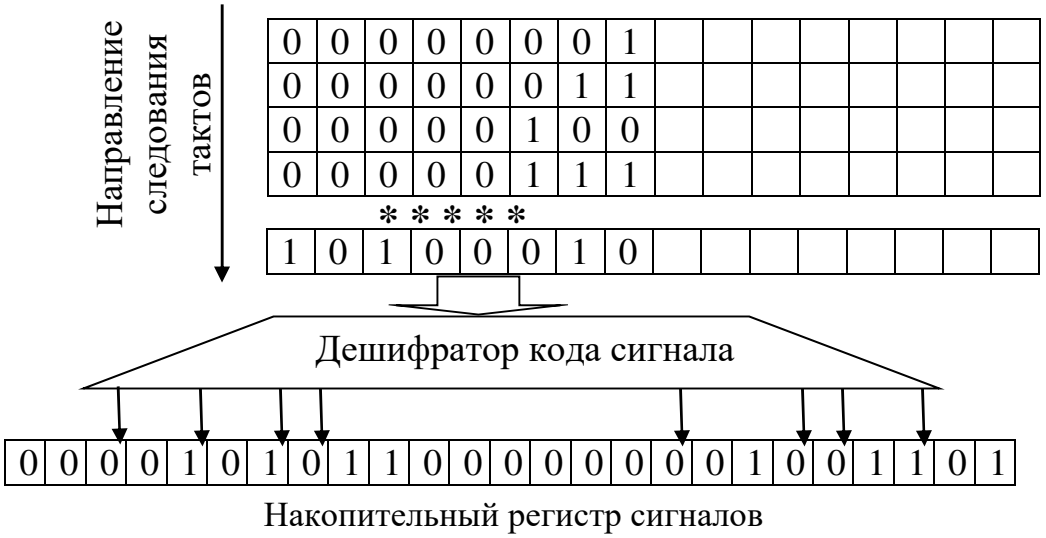


Рисунок 16. – Схема вертикального микропрограммирования

Для шифрования 100 сигналов управления операционным блоком по такому методу достаточно всего 8 разрядов поля данных в памяти:

$$n = \log_2(m+1) = \log_2(100+1) \approx 8,$$

где m – число функциональных сигналов в операционном блоке;

n – число разрядов операционной части микрокоманды.

Достоинства метода: малая длина микрокоманды (8 – 16 разрядов).

Недостатки:

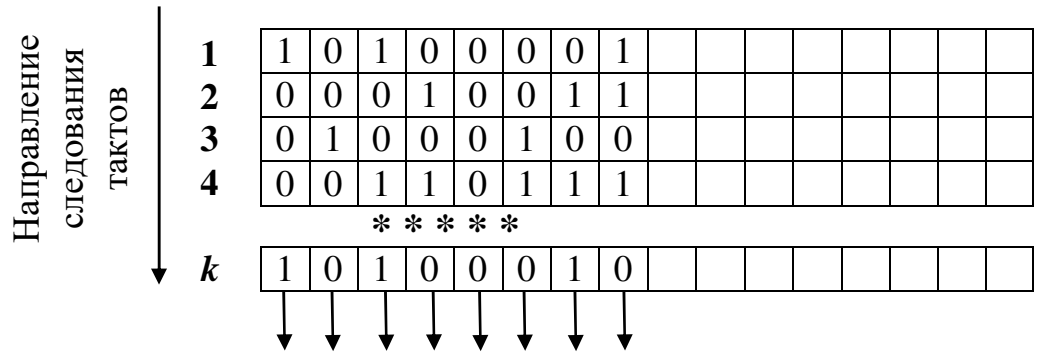
1. Сложность дешифрации кода сигнала микрооперации, требуются дополнительные схемные элементы: дешифратор, накопительный регистр, счетчик тактов.

2. Одной микрокоманде соответствует всего один функциональный сигнал, поэтому число микрокоманд в пакете должно соответствовать числу функциональных сигналов.

В чистом виде метод вертикального микропрограммирования в архитектуре ЭВМ не используется.

3.4.3.3. Вертикально-горизонтальное микропрограммирование сочетает достоинства горизонтального и вертикального методов.

Множество микроопераций разбивается на несколько подмножеств. Микрооперации внутри подмножеств кодируются горизонтально, т.е. представлены в явном виде. В каждое подмножество сводят типовые наборы сигналов управления микрооперациями, встречающиеся в одном такте. Подмножества строят равнообъемными, чтобы уравнять разрядность микрокоманд. Схема метода показана на рисунке 17.



***k* – количество подмножеств сигналов управления.**

Рис. 17. Схема вертикально-горизонтального микропрограммирования

Достоинства метода: малая длина микрокоманды (8 – 12 разрядов); сигналы представлены в явной форме и не требуют дешифрации.

Недостатки заключаются в следующем:

1. Вся совокупность микроопераций реализуется несколькими микрокомандами за несколько тактов, при этом необходимо соблюдать порядок следования микрокоманд в пакете.

2. Выделение сигналов в подмножества часто сопряжены с непреодолимыми трудностями, требующими системных решений. Разбиение сигналов на подмножества связано с понятием совместимости микроопераций в операционном блоке. Некоторые из используемых в микропрограмме операций могут выполняться параллельно во времени, а некоторые – только последовательно. Свойство совокупности микроопераций, гарантирующее возможность их одновременного исполнения, называется ***совместимостью***. Микрооперации, не обладающие указанным свойством, называются ***несовместимыми***. Очевидно, что в каждое подмножество можно включать только взаимно несовместимые операции.

3.4.3.4. Горизонтально-вертикальное микропрограммирование. В этом методе также, как и в вертикально-горизонтальном, все функциональные сигналы разбиваются на несколько подмножеств, которые располагаются горизонтально, т.е. в одной микрокоманде, и генерируются за один такт. Но внутри каждого подмножества выбирается только один сигнал и представляется в поле данных в виде кода, следовательно, генерироваться в каждом такте будет только один сигнал из подмножества. Поэтому в подмножества сводят только те сигналы, которые не могут появляться синхронно с другими сигналами подмножества.

Достоинство метода – микрокоманда выполняется за один такт.

Недостатки: сложность дешифрации кода сигнала микрооперации, требующая, как и при вертикальном микропрограммировании, дополнительных схемных решений, а также сложность сведения функциональных сигналов в подмножества.

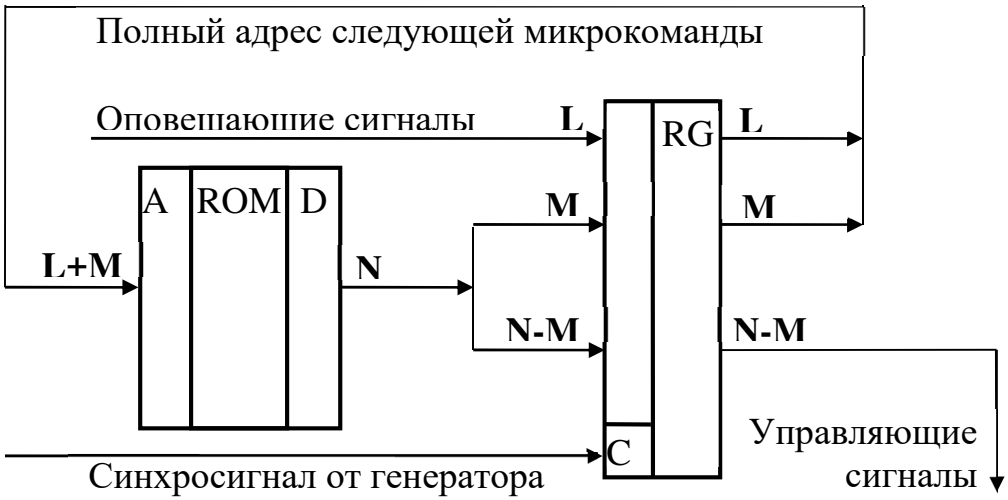
3.5. Проектирование и разработка микропрограммы МПА

Проектирование микропрограммного автомата суммирования двух чисел, функциональная схема операционного блока которого показана на рисунке 9, начнем с разработки структурной схемы управляющего автомата.

Принципиальная структурная схема устройства управления строится в соответствии с функциональной схемой управляющего автомата с программируемой логикой, представленной на рисунке 14. В состав схемы входит управляющая память и совмещенный регистр, который позволяет принимать данные текущей микрокоманды из памяти и одновременно формировать адрес следующей микрокоманды. Схема микропрограммного автомата синхронизируется с элементами операционного автомата с помощью генератора синхроимпульсов «Г». ПЗУ является энергонезависимой долговременной памятью и синхронизации не подлежит, т.к. выводит записанные в ней микрокоманды на вход регистра при подаче соответствующего адреса. Поэтому внешний регистр служит и для сопряжения ПЗУ управляющего автомата

с программируемой логикой с остальной схемой микропрограммного автомата для обеспечения возможности управления операционным блоком.

В качестве метода микропрограммирования воспользуемся наиболее простым горизонтальным с явным представлением в поле данных микрокоманды сигналов управления операционным автоматом. Принципиальная структурная схема управляющего автомата показана на рисунке 18.



N – исполняемая микрокоманда; M – данные поля микрокоманды, содержащие часть адреса следующей микрокоманды; N-M – данные поля микрокоманды, содержащие информацию о состоянии текущих сигналов управления операционным блоком МПА; L – сигналы оповещения о состоянии операционного блока; L+M – модифицированный адрес следующей микрокоманды.

Рисунок 18. – Принципиальная структурная схема управляющего автомата с программируемой логикой

Разработку микропрограммы для устройства управления МПА будем выполнять с использованием алгоритма управления операционным блоком, граф-схема которого представленного на рисунке 11.

В ответ на положительный уровень сигнала, определяемого кодом операции сложения двух чисел, управляющим автоматом вырабатывается последовательность выходных сигналов, которые в нужные моменты времени открывают или закрывают буферные регистры и активируют сумматор. Кроме этого, в определенные моменты времени управляющий автомат, формируя выходную последовательность, не должен реагировать на входной оповещающий сигнал, а после завершения операции должен ожидать следующего положительного уровня оповещающего сигнала, сообщающего о готовности контроллера памяти к приему или передаче информации.

Для упрощения задачи примем все тактовые интервалы равными 1 микросекунде. Поэтому частота тактового генератора должна быть 1 мГц.

Как следует из тактовой диаграммы сигналов управления, приведенной на рисунке 10, выходная последовательность состоит из девяти тактовых моментов, поэтому требуется, как минимум, 3 основных адресных разряда ПЗУ

(8 возможных состояний). Кроме этого, адресная часть при формировании полного адреса дополняется двумя входными сигналами, следовательно, количество адресных разрядов ПЗУ должно быть не менее 5, а количество разрядов регистра должно быть равно 9 (к 3 разрядам адреса добавляется 6 разрядов поля данных, определяющих состояние выходных сигналов).

С учетом изложенного реальная структурная схема управляющего микропрограммного автомата будет выглядеть следующим образом (рисунок 19).

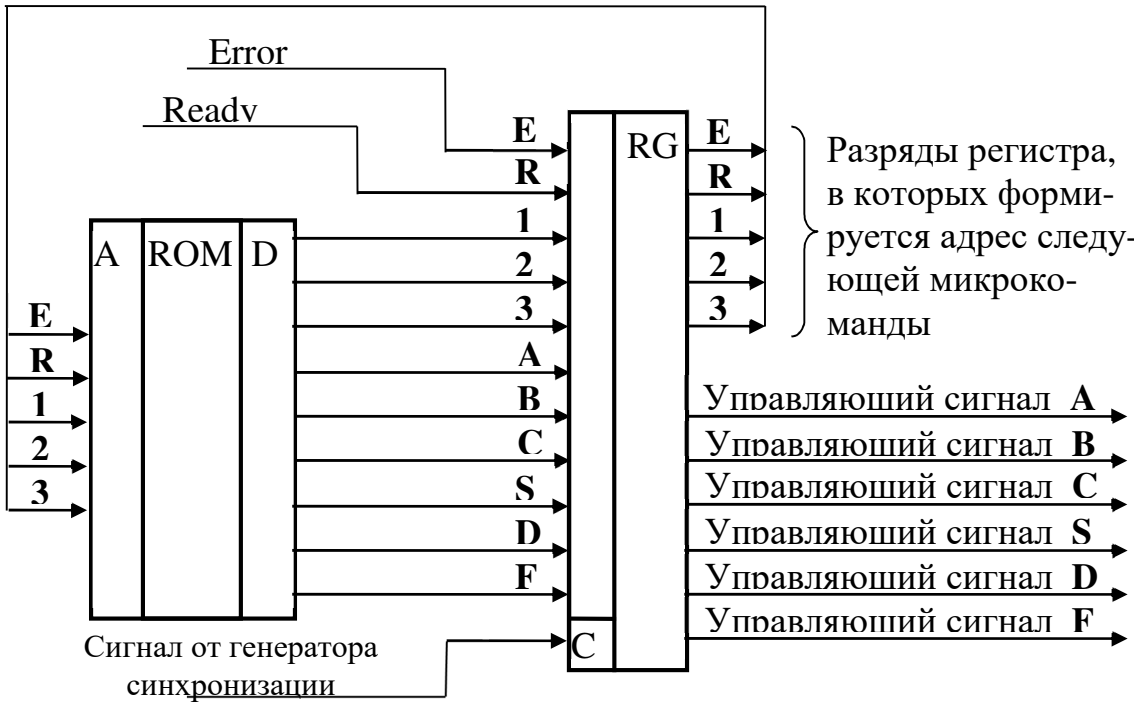


Рисунок 19. – Структурная схема устройства управления операционным блоком сложения двух чисел

В микропрограмме, записываемой в ПЗУ управляющего автомата, реализованы следующие режимы:

1. Последовательный перебор адресов при генерации выходной последовательности сигналов управления.

2. Остановка с ожиданием изменения состояния сигнала готовности.

3. Отключение реакции на входной сигнал (путем дублирования последовательного перебора в зоне адресов, соответствующей изменению входного сигнала).

Содержание микропрограммы представлено в таблице 1.

Адреса микрокоманд в примере сформированы в соответствии с алгоритмом исполнения микропрограммы. Порядок следования адресов микрокоманд в микропрограмме при подготовке её к записи в ПЗУ должен быть оптимизирован и представлен в порядке возрастания номеров. Повторение адресов является признаком ошибки написания микропрограммы и должно быть устранено. Отсутствующие (пропущенные) адреса замещаются нулевыми значениям микрокоманд.

Таблица 1. – Общий вид микропрограммы управляющего автомата с программируемой логикой для управления операцией сложения двух чисел в операционном блоке МПА

Адрес микрокоманды					Содержание микрокоманды									Комментарий
					Поле состояния сигнала						Поле адреса			
E	**R**	**1**	**2**	**3**	**A**	**B**	**C**	**S**	**D**	**E**	**1**	**2**	**3**	
0	0	0	0	0	1	0	0	0	1	0	0	0	0	Цикл ожидания сигнала готовности
0	1	0	0	0	0	1	0	0	1	0	0	0	1	Заполнение буфера 1
0	1	0	0	1	1	0	0	0	1	0	0	0	1	Цикл ожидания снятия сигнала Ready
0	0	0	0	1	1	0	0	0	1	0	0	1	0	Переход на ожидание сигнала Ready
0	0	0	1	0	1	0	0	0	1	0	0	1	0	Цикл ожидания сигнала готовности
0	1	0	1	0	0	0	1	0	1	0	0	1	1	Заполнение буфера 2
0	1	0	1	1	0	0	0	1	1	0	1	0	0	Активация сумматора и аккумулятора
0	1	1	0	0	0	0	0	0	0	1	1	0	1	Заполнение внешнего регистра буфера
0	0	0	1	1	0	0	0	1	1	0	1	0	0	Активация сумматора и аккумулятора
0	0	1	0	0	0	0	0	0	0	1	1	0	1	Заполнение внешнего регистра буфера
0	0	1	0	1	0	0	0	0	0	1	1	0	1	Цикл ожидания сигнала готовности
0	1	1	0	1	1	0	0	0	1	1	1	1	0	Запись аккумулятора в буфер
0	0	1	1	0	1	0	0	0	1	0	0	0	0	Переход на начальный адрес
0	1	1	1	0	1	0	0	0	1	0	1	1	0	Цикл ожидания сигнала готовности
1	0	0	0	0	1	0	0	0	1	0	0	0	0	Повтор операций заполнения буферов и активация сумматора при некритичной ошибке сумматора (остаточном активном сигнале Error)
1	1	0	0	0	0	1	0	0	1	0	0	0	1	
1	0	0	0	1	1	0	0	0	1	0	0	0	1	
1	1	0	0	1	1	0	0	0	1	0	0	1	0	
1	0	0	1	0	1	0	0	0	1	0	0	1	0	
1	1	0	1	0	0	0	1	0	1	0	0	1	1	
1	0	0	1	1	0	0	0	1	1	0	1	0	0	
1	1	0	1	1	0	0	0	1	1	0	1	0	0	
1	0	1	0	0	0	0	0	0	0	0	1	0	0	Обработка критичной ошибки исполнения Error
1	1	1	0	0	0	0	0	0	0	0	1	0	0	

Тема 4. Архитектура центрального процессора ПЭВМ

Центральный процессор (ЦП) занимает основное место в архитектуре любой вычислительной системы. Он предназначен для обработки информации любых форматов и назначения. Центральный процессор, являющийся по своей сути микропрограммным автоматом, имеет наиболее сложную структурную схему из всех устройств ПЭВМ. Она включает в свой состав большое количество программно доступных регистров, АЛУ, счетчиков, шифраторов, дешифраторов и др., т.е. имеет несколько операционных блоков, работающих под управлением единого устройства управления.

4.1. Типы архитектур центрального процессора ЭВМ

Понятие архитектуры центрального процессора включает в себя, прежде всего, его структурную схему, создаваемую для реализации различных принципов построения вычислительных операций в операционном блоке ЦП.

Конвейерная архитектура ЦП – различные узлы или блоки операционного автомата ЦП (дешифратор кода операции, блок формирования адреса операнда, АЛУ, аккумулятор) обычно используются последовательно в процессе исполнения одной машинной команды, а остальное время простаивают. Если процесс исполнения команды разбить на несколько последовательных циклов (каждый цикл выполняется в своем операционном блоке), то можно сформировать структурную схему ЦП таким образом, что она будет осуществлять параллельную обработку сразу нескольких команд в порядке их естественной адресации. При этом каждый блок будет загружен исполнением последовательных команд на различной стадии их готовности, как конвейер сборочного цеха. Это существенно (в 4–5 раз) увеличивает скорость исполнения программы.

Суперскалярная архитектура ЦП – структурная схема ЦП строится таким образом, чтобы выполнять машинную команду (а это несколько микрокоманд) за один такт. Для этой цели увеличивается количество однотипных операционных блоков, вводится внутренняя (увеличенная в несколько раз) тактовая частота синхронизации и дополнительный блок в устройство управления – планировщик загрузки операционных устройств.

CISC архитектура ЦП (Complex instruction set computer) – структурная схема такого процессора строится с минимальным количеством регистров (не более 20), но усложненным набором команд (длина машинной команды изменяется от одного до 15–20 байт, а количество команд – до нескольких сотен). Такая архитектура предполагает существенное усложнение микропрограммной и аппаратной части управляющего автомата.

RISC архитектура ЦП (Reduced instruction set computer) – архитектура строится на базе упрощенного набора команд (не более 100) фиксированной длины. Для её реализации требуется большое количество внутренних регистров (100–150) и отказ от методов косвенной адресации, что ведет к существенному усложнению аппаратной части операционного автомата.

MISC архитектура ЦП (Minimum instruction set computer) – это дальнейшее совершенствование RISC-архитектуры, где аппаратная часть операционного блока состоит из стековой структуры регистров, а количество команд сокращено до 20–30.

VLIW архитектура ЦП (Very long instruction word) или ***EPIC архитектура ЦП*** (Explicitly parallel instruction computing) – обеспечивает вычисления с явным распараллеливанием исполняемых команд в отличие от суперскалярной архитектуры ЦП. Процесс планировки заданий для различных операционных устройств является естественным, но размер кода операции увеличен в 2

раза, а, следовательно, количество активируемых микропрограмм и их сочетаний увеличено на несколько порядков. Код операции распараллеливания задается компилятором, который, в отличие от обычного компилятора, анализирует исходный текст программы с целью поиска и обработки возможного ветвления программы или вычисляемых логических условий.

Многоядерная архитектура ЦП – несколько центральных процессоров одной и той же архитектуры (фактических клонов) в одном корпусе, предназначенные для работы одной копии операционной системы на нескольких ядрах, представляют собой высокоинтегрированную реализацию мультипроцессорности. Такая архитектура позволяет на аппаратном уровне реализовать многозадачность современных операционных систем с распараллеливанием потоков информации в реальном масштабе времени.

Современные центральные процессоры наиболее массового выпуска (фирмы Intel) представляют собой гибридную архитектуру, наследующую черты нескольких из перечисленных выше.

Ядро центрального процессора построено на базе CISC-архитектуры, но имеет конвейерную схему обработки основного набора команд с 4- – 6-разрядным конвейером с некоторыми свойствами суперскалярности (например, увеличенной в два раза внутренней частотой исполнения микрокоманд). Отдельные боки, например, математический сопроцессор или устройства SMM-мультимедиа, имеют MISC-архитектуру с глубиной стека до 32-х 64-разрядных регистров.

Кроме этого, современные процессоры имеют многоядерную архитектуру с числом ядер от двух до восьми. И это не предел.

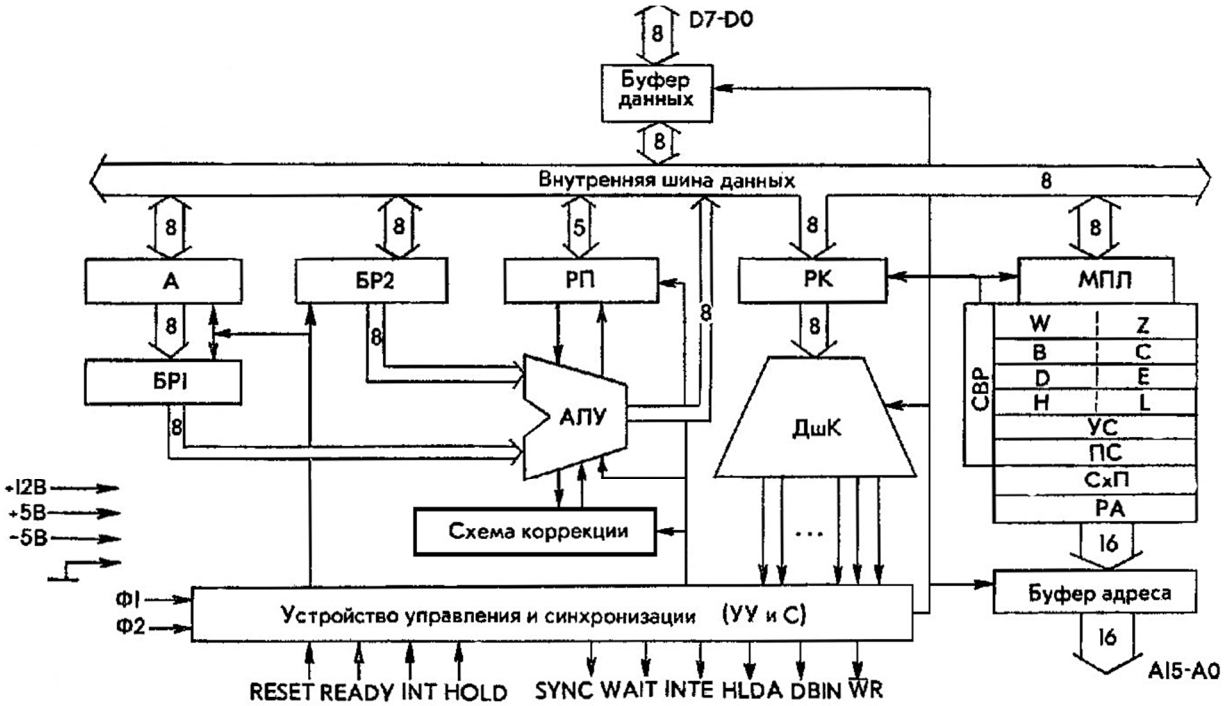
4.2. Базовая архитектура центрального процессора ПЭВМ

На примере схемы ядра 8-разрядного центрального процессора Intel i8080, разработанного фирмой для первых персональных ЭВМ, рассмотрим порядок построения базовой архитектуры микропроцессоров.

Функциональная схема микропроцессора представлена на рисунке 20.

Операционный блок центрального процессора включает в себя 20 регистров различного назначения, а также дешифратор кода операции, арифметико-логическое устройство и два мультиплексора, предназначенных для выполнения операций адресации регистров общего назначения. Генератор тактовых сигналов частотой 2,5 МГц является внешним и обслуживает все остальные устройства ЭВМ. Для обеспечения ускоренного выполнения некоторых микроопераций на вход управляющего автомата подаются два синхросигнала, сдвинутые по фазе один относительно другого на 120^{o}.

Схема процессора построена таким образом, что блок АЛУ, дешифратор кода операции и мультиплексорная схема выбора адресуемых регистров выделены в полностью самостоятельные операционные устройства. Такая архитектура позволяет разделить циклы выбора и обработки машинной команды, разбив процесс на несколько этапов, что обеспечивает возможность реализации конвейерной обработки нескольких машинных команд.



Условные обозначения на схеме:

А – регистр аккумулятор,
БР1, БР2 – буферные регистры (программно недоступны),
РП – регистр признаков (флагов) – слово состояния ЦП,
АЛУ – арифметическое логическое устройство – сумматор,
РК – регистр команд (регистр приема кода операции),
ДшК – дешифратор команд (дешифратор кода операции),
МПЛ – мультиплексор выбора регистров,
СВР – схема выбора и подключения регистров к шине данных,
W, Z, B, C, D, E, H, L – блок регистров общего назначения (аналогичны регистрам ah, al, dh, dl, ch, cl, bh, bl процессора i8080),
УС – 16-разрядный регистр – указатель стека (регистр sp),
ПС – программный счетчик (счетчик команд – регистр ip),
СхП – 16-разрядный регистр схемы формирования адреса,
РА – 16-разрядный регистр адреса команды или данных.

Внешние сигналы управляющего автомата процессора:

RESET – сигнал начальной установки состояния процессора,
READY – сигнал готовности памяти или порта ввода/вывода к обмену данными с центральным процессором,
INT – сигнал запроса на прерывание (от контроллера прерываний),
HOLD – сигнал запроса на захват шины (от контроллера прямого доступа к памяти – ПДП),
SYNC – сигнал сопровождения выдачи на шину данных слова состояния центрального процессора из регистра РП,
WAIT – сигнал ожидания готовности READY,
INTE – сигнал разрешения прерывания выполнения программы,
HLDA – сигнал подтверждения контроллеру ПДП на захват шины,
DBIN – сигнал подтверждения включения буфера данных на чтение,
WR – сигнал подтверждения включения буфера данных на запись,
Ф1, Ф2 – синхросигналы генератора с различным сдвигом фаз.

Рисунок 20. – Функциональная схема микропроцессора i8080

Имеются внутренние шины данных для обеспечения прямой связи между регистрами в процессе выполнения микроопераций. Управляющий автомат с программируемой логикой выполнен по совмещенной схеме горизонтально-вертикальной структуры размещения микрокоманд. Внешними шинами является 8-разрядная двунаправленная шина данных D7-D0 и однонаправленная 16-разрядная шина адреса A15-A0. Для связи с шинами служат соответствующие буферные регистры, активируемые согласно протоколу обмена данными.

4.3. Эволюция архитектуры процессоров ПЭВМ семейства x86

Центральные процессоры производства фирмы Intel (США) в настоящее время занимают основной сегмент рынка микропроцессорной техники ПЭВМ и являются наиболее доступными для пользователей вычислительной техники. Поэтому представляет определенный интерес рассмотреть эволюцию их архитектуры и направления дальнейшего совершенствования.

Следует четко представлять себе цель совершенствования архитектуры центрального процессора – увеличение производительности вычислительных операций, сокращение времени исполнения программ.

Уже первая реализация архитектуры процессора i8086, положившая начало созданию линейки процессоров семейства x86 (i80286, i80386, i80486, Pentium) содержала существенные доработки структурной схемы:

– расширена система команд (добавлены команды реализации косвенной и совмещенной адресации);

– доработана структура операционного блока процессора с добавлением дополнительных устройств (устройства обработки операндов и связи с магистралью) с целью реализации конвейерной архитектуры, которая позволила совместить во времени процессы обработки данных и передачи их по внешней шине;

– увеличена вдвое разрядность операндов за счет увеличения размера регистров общего назначения;

– введена возможность аппаратной обработки вещественных чисел с помощью внешних устройств, реализованная в последующих процессорах линейки x86, начиная с процессора i80286;

– введена сегментация памяти, что позволило на порядок увеличить объем оперативной памяти, доступной центральному процессору.

Дальнейшие этапы совершенствования архитектуры процессоров были направлены, прежде всего, на увеличение тактовой частоты. Тактовая частота современных процессоров на несколько порядков превышает частоту процессора i8080. Это стало возможно за счет уменьшения габаритных размеров элементов электронных схем. Существенно возросла разрядность обрабатываемых операндов, кстати, разрядность кода операции (1 байт) при этом не изменилась.

В составе операционного блока процессора появилось много новых устройств, предназначенных для работы с векторной (игровой) графикой. В архитектуру процессора ведены элементы суперскалярности и многоядерности для аппаратного обеспечения многозадачных режимов.

Более детально характеристики эволюции архитектуры центрального процессора можно видеть в таблице 2.

Таблица 2. – Эволюция архитектуры процессора x86 фирмы Intel.

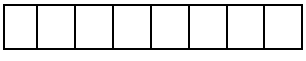
Тип	Год	Разрядность операндов (разряды)	Тактовая частота (МГц)	Оператив-ная память	Расширение операций
i8080	1976	8	4	640 КВ	Базовый набор команд
i8086	1979	16	10	1 МВ	Расширенный базовый набор
i80286	1982	16	1	16 МВ	Внешний сопроцессор
i80386	1985	32	33	4 ГВ	Внешний сопроцессор
i80486	1989	32	40	4 ГВ	+ Внутренний сопроцессор
Pentium	1993	32	100	4 ГВ	+ Суперскалярность
Pentium II	1997	32	400	4 ГВ	+ MMX
Pentium III	1999	32	900	4 ГВ	+ XMM (SSE)
Pentium IV	2000	32	1500	4 ГВ	+ SSE2,SSE3
Core 2 Duo	2008	64	3000	8 ГВ	+ SSE4
Core 2 Duo (Sandy Bridge)	2011	64	3000	16+ ГВ	+ AVX
Core I	2013	64	4000	16+ ГВ	+ AVX2
Core Xeon	2016	64	1500	16+ ГВ	+ AVX-512

Центральные процессоры аналогичных типов, совместимые по архитектуре и набору команд с x86, в разное время проектировались и выпускались фирмами IBM, Cyrix, VIA, AMD и другими. Кроме вышеуказанных типов ЦП различными фирмами, например, Texas Instruments, Motorola, RED, Omron, Honeywell и т.д. выпускаются ЦП для контроллеров и микроЭВМ технологического назначения. Архитектура и система команд этих процессоров обычно строится на принципах RISC-архитектуры и существенно отличается от ЦП типа x86 фирмы Intel.

4.4. Системная память центрального процессора ПЭВМ

Системная или регистровая память центрального процессора образует ядро операционного блока процессора.

Регистр – это ячейка памяти (обычно триггерного типа), объем которой составляет один или несколько байт (8, 16, 32, 64 или 128 двоичных разрядов), используемая для временного хранения обрабатываемых данных в ходе выполнения команд.

Схема 8-разрядного регистра – 

4.4.1. Основные программно-доступные регистры ПЭВМ

Обычно в составе операционного блока ЦП современной ПЭВМ доступными для программиста являются:

а) 16 (шестнадцать) пользовательских регистров, из них:

– восемь 32-битных регистров общего назначения (РОН): eax (ax, ah, al), ebx (bx, bh, bl), edx (dx, dh, dl), ecx (cx, ch, cl), ebp/bp, esi/si, edi/di, esp/sp;

– шесть 16-битных регистров сегментов: cs, ds, ss, es, fs, gs;

– два 32-битных регистра состояния и управления – флагов eflags/flags и счетчика (указателя) команд eip/ip;

б) 8 (восемь) 32-битных отладочных регистра: dr0 ... dr7;

в) 8 (восемь) 32-битных регистров тестирования: tr0 ... tr7;

г) 5 (пять) 32-битных управляющих регистра: cr0, cr1, cr2, cr3, cr4;

д) 4 (четыре) 48-битных дескрипторных регистра: gdtr, idtr, tr, ldtr;

е) 8 регистров данных математического сопроцессора: r0 … r7, имеющих стековую структуру хранения вещественных чисел;

ж) 8 (восемь) 64-битных регистра MMX (блок MMX): mmx0 ... mmx7, для размещения целочисленных переменных 2D- и 3D-графики;

з) 8/16 128-битных регистра XMM (блок XMM): xmm0 … xmm7 для размещения вещественных чисел 3D-графических изображений.

Доступ к регистрам первой группы, так называемым пользовательским регистрам, предусмотрен в базовых командах языка ассемблера. Доступ к остальным регистрам является более сложным и предусмотрен в командах языка ассемблера дополнительных групп. Например, к регистрам математического сопроцессора – из команд группы математического сопроцессора, начинающихся с буквы «f», а xmm или mmx – из группы команд обработки потоковых данных мультимедиа – SSE.

При этом некоторые группы регистров, имеющие различные обозначения, например, r0 и mmx0, на самом деле физически являются одними и теми же регистрами, только используемыми для различных целей. По этой причине существует запрет на использование в наборе команд матсопроцессора команд мультимедиа и наоборот, такие ситуации отслеживаются компиляторами и отмечаются как ошибки программы.

4.4.2. Программно-регистровая модель ПЭВМ

К изучению языка ассемблера ПЭВМ имеет смысл приступать только после выяснения того, какая часть компьютера оставлена видимой и доступной для программирования на этом языке.

Программно-регистровой моделью ПЭВМ принято называть совокупность аппаратных средств компьютера, доступных из программы. Другими словами, программно-регистровая модель (ПРМ) – это видение программистом аппаратных средств ПЭВМ.

Как было показано в предыдущем разделе, центральный процессор современной ЭВМ содержит 65 регистров в той или иной мере доступных для использования программистом. Данные регистры принято разделить на две большие группы: 1) 16 пользовательских регистров, на основе которых и строится ПРМ ЭВМ; 2) 49 системных регистров, доступ к которым затруднен или нежелателен по тем или иным причинам. В программах на языке ассемблера, использующих базовый набор команд, пользовательские регистры, особенно РОН, используются очень интенсивно.

Часто ПРМ ПЭВМ изображают в виде набора пользовательских регистров определенной разрядности и присвоенного буквенного обозначения, которое употребляется в командах языка ассемблера и распознается компилятором. Такое представление ПРМ ЭВМ, хоть и не отражает особенностей её архитектуры и фактически показывает только часть структурной схемы операционного блока ЦП, является достаточно удобным для написания программ верхнего уровня. На рисунке 21 показана такая программно-регистровая модель с учетом эволюции архитектуры процессоров x86.

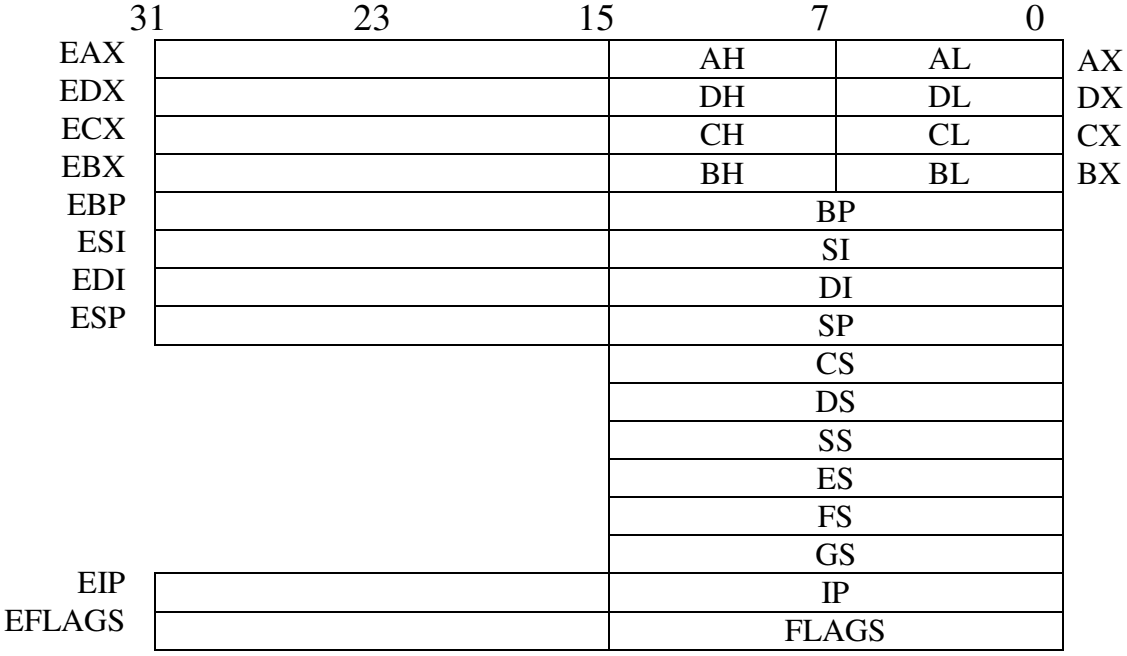


Рисунок 21. – Программно-регистровая модель ПЭВМ архитектуры x86

На рисунке 22 показана ПРМ ПЭВМ 64-разрядной архитектуры ЦП, которая незначительно отличается от схемы ПРМ x86. Использование обозначений регистров 64-разрядной ПРМ в командах языка ассемблера воспринимается компилятором, как признак 64-разрядной архитектуры ЭВМ, и, соответственно, перед исполнением такой программы ЦП должен быть переведен в режим <long mode> (подрежим <64-bit mode>) специальной программой-загрузчиком или загруженной 64-разрядной операционной системой.

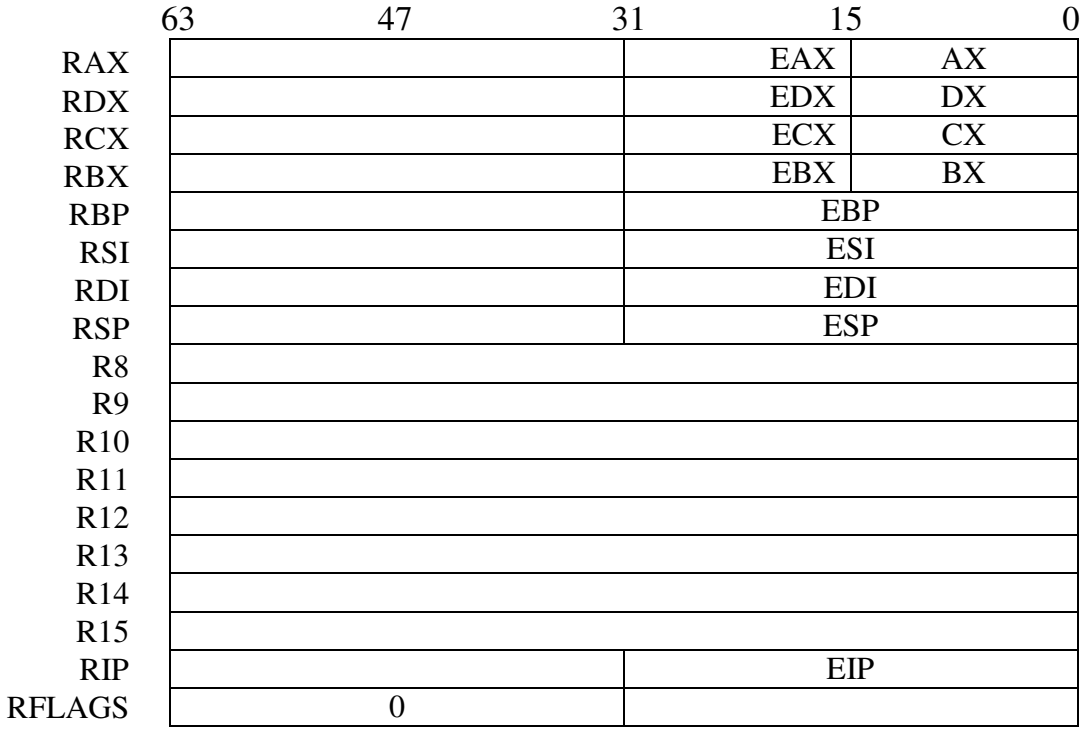


Рисунок 22. – Программно-регистровая модель ПЭВМ x86-64 с 64-разрядной архитектурой центрального процессора

Большинство пользовательских регистров имеют определенное функциональное назначение.

Регистры общего назначения (РОН):

eax (ax, ah, al) – аккумулятор для хранения результата операции;

ebx (bx, bh, bl) – базовый регистр для хранения базовой части адреса;

edx (dx, dh, dl) – регистр хранения промежуточного результата;

ecx (cx, ch, cl) – регистр-счетчик цикла (например, в команде Loop);

ebp/bp – регистр-указатель базовой части адреса стека (база стека);

esi/si – регистр индекса источника (постоянного смещения адреса);

edi/di – регистр индекса приемника (постоянного смещения адреса);

esp/sp – указатель адреса последней записи в стеке (вершина стека);

Все регистры общего назначения, кроме фиксированного назначения, могут использоваться для адресных вычислений и хранения результатов большинства арифметических и логических операций.

Сегментные регистры:

cs – регистр параграфа адреса исполняемой команды (сегмент кода);

ds – регистр параграфа адреса области стека (сегмент стека);

ss – регистр параграфа адреса области данных (сегмент данных);

es, fs, gs – дополнительные сегментные регистры, замещаемые по соответствующему префиксу.

Регистры состояния и управления:

eflags/flags – регистр флагов результата исполнения команды;

eip/ip – регистр адреса исполняемой команды.

4.4.3. Содержание регистра флагов ЦП ПЭВМ

Побитовое содержание регистра флагов (признаков) в табличной форме представлено на рисунке 23.

31	…	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	…	AC	VM	RF	0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

Рисунок 23. – Содержание регистра флагов ЦП архитектуры x86 – x86-64

Регистр флагов служит для хранения признаков завершения исполнения последней команды центральным процессором и выдается на шину данных для анализа системой контроля возможных ошибок, нештатных ситуаций, особых случаев и их обработки. Регистр флагов содержит **флаги состояния:**

NT – флаг вложенности задачи и TSS дает обратную связь;

IOPL – уровень привилегированности ввода-вывода;

OF – флаг арифметического переполнения АЛУ;

CF – флаг переноса/займа из старшего разряда в младший BCD;

PF – флаг четности числа единиц (паритета) в результате операции;

AF – вспомогательный флаг переноса/займа (из 3-го разряда);

ZF – флаг нуля (нулевой результат);

SF – флаг отрицательного знака результата.

Флаг управления:

DF – флаг направления модификации адресов в массиве (увеличение или уменьшение) при исполнении команд цепочечных вычислений.

Системные флаги:

TF – флаг трассировки (управляет пошаговым режимом работы ЦП);

IF – флаг прерывания (при IF = 0 внешние прерывания маскируются);

RF – флаг возобновления (маскирования ошибок отладки);

VM – флаг виртуального режима 8086 в защищенном режиме;

AC – флаг контроля выравнивания операндов при обращении к оперативной памяти (обращение к невыровненному по размеру машинного слова операнду вызывает исключительную ситуацию).

4.5. Командный цикл процессора

Командой называется элементарное действие, которое может выполнить центральный процессор без дальнейшей детализации мнемоники машинной команды. Действия по выбору из памяти и выполнению одной машинной команды процессором называются ***командным циклом*** процессора.

Основные этапы выполнения командного цикла процессора состоят из двух этапов: ***этапа выборки команды*** из оперативной памяти и ***этапа исполнения команды*** и размещения результата в оперативной памяти.

Этап выборки команды состоит из двух машинных циклов:

Цикл 1 – на шину адреса из счетчика команд выдается адрес очередной команды, поступающей на исполнение, а на шину данных – слово состояния процессора, сопровождаемые соответствующими сигналами SYNC

и WR. Если шина мультиплексируемая, соблюдается очередность её использования для адреса и слова состояния.

Цикл 2 – анализируется наличие сигнала READY. Если его нет, осуществляется переход на ожидание, если есть – чтение команды, сопровождаемое сигналом DBIN.

Этап исполнения команды состоит из пяти машинных циклов, не считая циклов ожидания готовности контроллера памяти:

Цикл 1 – код операции команды (8 битов) поступает на дешифратор, инициируется исполнение соответствующей микропрограммы обработки команды. Если команда содержит префиксы, то они также поочередно дешифруются с присвоением системным переменным соответствующих значений до поступления кода операции.

Цикл 2 – инициируется схема передачи из памяти или из соответствующего регистра первого операнда в буферный регистр БР1.

Цикл 3 – инициируется схема передачи из памяти или из соответствующего регистра второго операнда в буферный регистр БР2.

Цикл 4 – выполняется операция в АЛУ, результат в аккумуляторе.

Цикл 5 – результат передается в оперативную память по адресу нахождения первого операнда.

Если операция не требует обработки операндов, соответствующие циклы исключаются из процесса обработки команды в ЦП.

4.6. Система машинных команд процессора

Разнообразие типов данных, форм их представления и действий, которые необходимы для обработки входной информации и формирования массивов информации для внешних устройств отображения, характерное для CISC-архитектуры, порождает необходимость использования постоянно расширяемых наборов команд, называемых ***системой команд*** центрального процессора ЭВМ. Система команд должна обладать двумя свойствами – функциональной полнотой и эффективностью.

Функциональная полнота – достаточность системы команд для описания любого алгоритма обработки информации. Известно, что свойством функциональной полноты обладает система, состоящая всего из трех команд (присвоение 0, проверка на 0, присвоение 1), однако построение программ на её основе неэффективно из-за их большого объема.

Эффективность системы команд – степень соответствия системы команд назначению ПЭВМ, т.е. классу алгоритмов, для выполнения которых предназначена ПЭВМ. Необходимо отметить, что реализация развитой системы команд требует расширения операционной части процессора, увеличения числа и разрядности дополнительных групп регистров, что приводит к увеличению стоимости и энергоемкости процессоров.

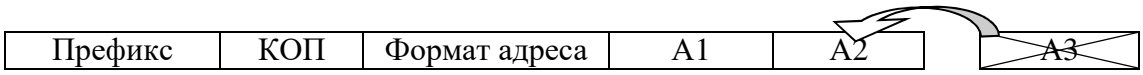
Система команд процессора характеризуется системой операций, реализуемых в операционном блоке процессора и выбираемых по коду операции (КОП), а также форматами машинных команд, определяемых способами адресации операндов.

Под форматом команды понимается её длина в байтах, количество полей, их размер и способ адресации операндов в оперативной или регистровой памяти ПЭВМ.

Состав команды включает в себя следующие виды информации:

1. Тип (код) операции, которую следует реализовать в операционном блоке для исполнения процессором данной команды.
2. Код способа адресации операндов.
3. Адрес памяти, откуда следует взять первый операнд (А1).
4. Адрес памяти, откуда следует взять второй операнд (А2).
5. Адрес памяти, куда следует поместить результат (А3).

Формат машинной команды в CISC-архитектуре обычно имеет вид:



Для сокращения размера команды в качестве А3 используется обычно последний вычисленный адрес, т.е. А2 или А1. Более того, при использовании регистра-аккумулятора в некоторых командах (например, пересылки данных) второй адрес также не требуется.

Существуют и безадресные (системные) команды, например, команды управления состоянием центрального процессора.

Дополнительное поле кода операции содержит указатели на тип формата операнда: байт, слово или двойное слово. Для уточнения формата операндов в отдельных командах может использоваться дополнительное поле префикса.

4.7. Способы адресации операндов в машинной команде ЦП

Способ адресации определяет, каким образом следует использовать информацию, размещенную в поле адреса команды. Существует пять основных способов адресации операндов в машинной команде, реализованных в CISC-архитектуре ЦП:

1. ***непосредственная*** (в поле адреса располагается не адрес операнда, а непосредственно сам операнд (константа));
2. ***прямая*** (в поле адреса располагается эффективный адрес операнда);
3. ***косвенная*** (в поле адреса располагается не адрес операнда, а адрес ячейки оперативной памяти, в которой хранится адрес операнда);
4. ***относительная*** (адрес операнда формируется, как сумма двух слагаемых: базы адреса, хранящейся в специальном регистре (например, BX) или в одном из регистров общего назначения и смещения адреса, извлекаемого из поля адреса в команде. Иногда к базе добавляется индекс, увеличивающий или уменьшающий свое значение на единицу после каждого обращения);

5. ***безадресная*** поля адреса операндов в команде отсутствуют, а сам адрес операнда или не имеет смысла, так как операнд не нужен, (что характерно для системных команд), или адрес его подразумевается по умолчанию (например, аккумулятор). Одной из разновидностей безадресного обращения является использование стековой памяти.

4.8. Классификация машинных команд по системам операций

Все операции, выполняемые в операционном блоке центрального процессора, по кодам операций машинных команд принято подразделять на пять основных классов:

Арифметико-логические и специальные – команды, при исполнении которых процессор выполняет преобразование информации (сложение, вычитание, умножение и т.д., а также различные виды преобразования форматов операндов).

Пересылки и загрузки – передача информации между регистрами центрального процессора и оперативной памятью или между областями адресуемой оперативной памяти.

Ввода/вывода – обеспечивают передачу информации между процессором и внешними устройствами через адресное пространство портовых регистров.

Передачи управления – команды, которые изменяют естественный порядок следования команд программы с изменением содержания счетчика команд.

Существует три разновидности команд передачи управления: внутрипрограммные переходы; вызовы подпрограмм; возвраты из подпрограмм.

Системные команды – выполняющие управление процессом обработки информации или состоянием внутренних ресурсов процессора.

Формально префиксы не являются командами, хотя и содержат коды операции, т.к. при их загрузке в дешифратор в операционном блоке ЦП никаких действий не производится, а всего лишь выполняется присвоение новых значений системным переменным. При этом слово состояния процессора не изменяется.

4.9. Схема построения машинных команд ЦП x86 – x86-64

Генерацию машинных команд (***ассемблирование***) выполняют компиляторы различных языков программирования или интерпретаторов виртуальных компьютерных платформ (например, байт-кода виртуальной машины Java или платформы .NET Framework).

В основу всех современных языков программирования и интерпретаторов различного назначения, созданных для центральных процессоров архитектуры x86 или x86-64, положен язык ассемблера, который дополняется и совершенствуется по мере развития системной базы ЦП. По этой причине декомпиляция (***дизассемблирование***) исполняемого кода (машинных команд) любой программы, выполняемая отладчиком, вне зависимости от

языка программирования, использованного программистом для её написания, создает текст программы в командах языка ассемблера.

Схема построения машинной команды показана на рисунке 24.

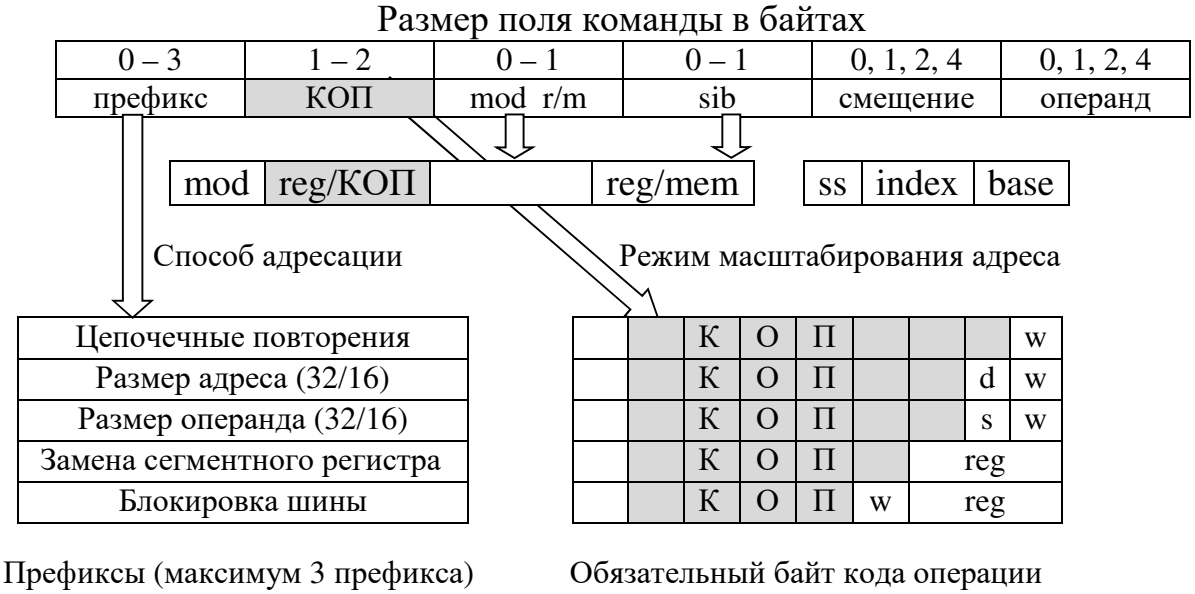


Рисунок 24. – Схема построения машинной команды ЦП x86 – x86-64

Байт <mod r/m> определяет способ адресации операнда в команде:
mod = 00 – в команде указан эффективный адрес операнда без смещения;
01 – имеется 1 (или 2 при w = 1) байт смещения адреса операнда;
10 – имеется 2 (или 4 при w = 1) байта смещения адреса операнда;
11 – операнд находится в регистре reg.

Этот байт также может содержать дополнительные три разряда кода операции (дополняющие КОП до 8 бит) или конкретный указатель на пользовательский регистр:

reg = 000 – это регистр AL или AX при значении w = 1;
001 – регистр CL или CX при значении w = 1;
010 – регистр DL или DX при значении w = 1;
011 – регистр BL или BX при значении w = 1;
100 – регистр AH или SP при значении w = 1;
101 – регистр CH или BP при значении w = 1;
110 – регистр DH или SI при значении w = 1;
111 – регистр BH или DI при значении w = 1.

SIB-байт используется только при формировании адреса в защищенном режиме адресации и содержит масштабный множитель индекса ss, равный 1, 2, 4 или 8 при значении ss = 00, 01,10 или 11.

Дополнительный байт кода операции появляется при обращении к дополнительным наборам команд языка ассемблера для обеспечения операций в математическом сопроцессоре (КОП = D8h – DFh) или операционных блоках SSE (КОП = FFh).

Из схемы, приведенной на рисунке 24 видно, что минимальная длина команды составляет 1 байт, а максимальная может составлять 15 байт.

При всей своей кажущейся сложности соответствия машинных кодов и команд ассемблера и, как следствие, системы преобразования программ, написанных на языках высокого уровня в исполняемые машинные коды, в архитектуре CISC, в частности, x86 – x86-64, существует строгий порядок преобразования таких программ в машинные коды.

На основании таблицы соответствия кодов машинных команд командам языка ассемблера (например, таблица 3) с учетом введенных обозначений строятся программы-трансляторы языков высокого уровня, которые могут генерировать исполняемые машинные коды.

Таблица 3. – Коды операций базовых команд ЦП архитектуры x86

	0	1	2	3	4	5	6	7
0	ADD						PUSH	POP
	Eb.Gb	Ev.Gv	Gb.Eb	Gv.Ev	AL.lb	eAX.lv	ES	ES
1	ADC						PUSH	POP
	Eb.Gb	Ev.Gv	Gb.Eb	Gv.Ev	AL.lb	eAX.lv	SS	SS
2	AND						SEG	DAA
	Eb.Gb	Ev.Gv	Gb.Eb	Gv.Ev	AL.lb	eAX.lv	=ES	
3	XOR						SEG	AAA
	Eb.Gb	Ev.Gv	Gb.Eb	Gv.Ev	AL.lb	eAX.lv	=SS	
4	INC регистр общего назначения							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
5	PUSH регистр общего назначения							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
6	PUSHA	POPA	BOUND	ARPL	SEG	SEG	Размер	Размер
		eCX	Gv,Ma	Ew,Rw	=FS	=GS	операнда	адреса
7	Близкий переход по условию (JB)							
	JO	JNO	JB	JNB	JZ	JNZ	JBE	JNBE
8	Непосред. Grpl		MOVB	Grpl	TEST		XCHG	
	Eb,lb	Eb,lb	AL,imm	Ev,lb	Eb,Gb	Ev,Gv	Eb,Gb	Ev,Gv
9	NOP	XCHG с регистром слова или двойного слова eAX						
		eCX	eDX	eBX	eSP	eBP	eSI	eDI
A	MOV				MOVSB	MOVSW/D	CMPSB	CMPSW/D
	AL,Ob	eAX,Ov	Ob,AL	Ov,eAX	Xb,Yb	Xv,Yv	Xb,Yb	Xv,Yv
B	MOV с непосредственным байтом в байтовом регистре							
	AL	CL	DL	BL	AH	CH	DH	BH
C	Сдвиг Grp2		ближний RET		LES	LDS	MOV	
	Eb,lb	Ev,lb	lw		Gv,Mp	Gv,Mp	Eb,lb	Ev,lb
D	Сдвиг Grp2				AAM	AAD		XLAT
E	LOOPNE	LOOPE	LOOP	JCXZ	IN		OUT	
	Jb	Jb	Jb	Jb	Al,lb	eAX,lb	lb,AL	lb,eAX
F	LOCK		REPNE	REP	HLT	CMS	Унарный Grp3	
				REPE			Eb	Ev

Окончание таблицы 3.

	8	9	A	B	C	D	E	F
0	OR						PUSH	переход 2-й байт
	Eb.Gb	Ev.Gv	Gb.Eb	Gv.Ev	AL.lb	eAX.lv	CS	
1	SBB						PUSH	POP
	Eb.Gb	Ev.Gv	Gb.Eb	Gv.Ev	AL.lb	eAX.lv	DS	SS
2	SUB						SEG	DAS
	Eb.Gb	Ev.Gv	Gb.Eb	Gv.Ev	AL.lb	eAX.lv	=CS	
3	CMP						SEG	AAC
	Eb.Gb	Ev.Gv	Gb.Eb	Gv.Ev	AL.lb	eAX.lv	=DS	
4	DEC регистр общего назначения							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
5	POP регистр общего назначения							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
6	PUSH	IMUL	PUSH	IMUL	INSB	INSW/D	OUTSB	OUTSW/D
	lv	GvEvlv	lb	GvEvlb	Yb,DX	Yv,DX	DX,Xb	DX,Xv
7	Близкий переход по условию (JB)							
	JS	JNS	JP	JNP	JL	JNL	JLE	JNLE
8	MOV					LEA	MOV	POP
	Eb.Gb	Ev.Gv	Gb.Eb	Gv.Ev	Ew,Sw	Gv,M	Sw,Ew	Ev
9	CBW	CWD	CALL	WAIT	PUSHHF	POPF	SAHF	LAHF
			Ap		Fv	Fv		
A	TEST		STOSB	STOSW/D	LODSB	LODSW/D	SCASB	SCASW/D
	AL,lb	eAX,lv	Yb,AL	Yv,eAX	AL,Xb	eAX,Xv	AL,Xb	eAX,Xv
B	MOV непосредственное слово или двойное слово в регистр							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
C	ENTER	LEAVE	Дальний RET		INT	INT	INTO	IRET
	lw,lb		lw		3	lb		
D	ESC (Выход на байт множества команд сопроцессора)							
E	CALL	JMP			IN		OUT	
	Jv	Jv	AP	Jb	AL,DX	eAX,DX	DX,AL	Dx,eAX
F	CLC	STC	CLI	STI	CLD	STD	INC/DEC Grp4	INC/DEC Grp5

Набор базовых команд языка ассемблера представляет собой группу машинных кодов от 00h до FFh. Код такой машинной команды, поступающий на дешифратор, вызывает на исполнение из ПЗУ управляющего автомата соответствующую микропрограмму, формирующую схему преобразования информации в операционном блоке и управляющую ходом операции. Каждому коду из этой группы ставится в соответствие только одна команда языка ассемблера. Кроме того, в зависимости от способа адресации операнда в этой команде или конкретных регистров общего назначения, используемых в ней, схема операционного блока будет также изменяться. Изменяется и порядок использования собранной схемы при обработке операндов. Поэтому некоторым командам ассемблера присваивается несколько кодов машинных команд.

Условные обозначения кодирования методов адресации:

A – прямая адресация: команда не имеет байта modR/M; адрес операнда содержится непосредственно в команде; регистр базы, регистр индекса, коэффициент масштабирования не используется.

C – поле reg байта modR/M указывает управляющий регистр.

D – поле reg байта modR/M указывает регистр отладки.

E – за кодом операции следует байт modR/M, описывающий операнд. Операндом может быть регистр общего назначения или адрес памяти. Если это адрес памяти, он вычисляется исходя из сегментного регистра и следующих величин: регистр базы, индексный регистр, масштабирование, смещение.

F – регистр флагов.

G – поле reg байта modR/M указывает регистр общего назначения.

I – непосредственные данные (константа). Значение операнда кодируется последовательностью байтов непосредственно в команде.

J – команда содержит относительное смещение, прибавляемое к регистру счетчика команд (EIP).

M – байт modR/M может ссылаться только на память.

O – команда не содержит байта modR/M; относительный адрес операнда кодируется как слово или двойное слово (в зависимости от атрибута размера адреса) непосредственно в команде. Регистр базы, регистр индекса, коэффициент масштабирования не используется.

R – поле mod байта modR/M может указывать только на регистр общего назначения.

S – поле reg байта modR/M указывает сегментный регистр.

T – поле reg байта modR/M указывает регистр.

X – адресация памяти при помощи пары регистров DS:SI; например, MOVS, COMPS, OUTS.

Y – адресация памяти при помощи пары регистров DS:DI; например, MOVS, COMPS, INS.

Обозначения кодирования типов операндов в команде:

a – два операнда длиной в слово в памяти или два операнда длиной в двойное слово в памяти, в зависимости от атрибута размера операнда (используется для команды BOUND).

b – байт (независимо от атрибута размера операнда).

c – байт или слово, в зависимости от атрибута размера операнда.

d – двойное слово (независимо от атрибута размера операнда).

p – 32- или 48-разрядный указатель, в зависимости от атрибута размера операнда.

s – 6-разрядный псевдо-дескриптор.

v – слово или двойное слово, в зависимости от атрибута размера операнда.

w – слово (независимо от атрибута размера операнда).

Коды регистров, используемых в команде. Когда в качестве операнда используется закодированный в команде регистр, он определяется по имени, например, AX, CL или ESI. Имя регистра определяет его размер: 32, 16 или 8 бит. В случае, если размер регистра определяется атрибутом размера операнда, используется запись такого формата EXX.

Например, запись EAX означает, что при атрибуте размера операнда равном 16, используется регистр AX, а при атрибуте размера операнда равном 32 – регистр EAX.

Для нахождения строки, содержащей нужную операцию, необходимо использовать значение старших четырех битов кода операции, а для нахождения столбца – значение младших четырех битов.

Если код операции равен 0Fh или попадает в диапазон D8h … DFh, необходимо обращаться к дополнительным таблицам кодов операций, созданным для реализации дополнительных операций по обработке чисел с плавающей запятой или потоковых данных систем мультимедиа, и использовать значение второго байта кода операции для нахождения соответствующей строки и столбца в этих таблицах.

Существует ряд дополнительных таблиц расширения кодов операций машинных команд, доступных через первый байт КОП:

1. Команды математического сопроцессора. Эти наборы команд доступны через инструкцию 11011xxx (или D8x) первого байта (см. основную таблицу кодов), которая называется ESC-командой. Наборы команд математического сопроцессора содержат 8 дополнительных таблиц команд, т.е. позволяет реализовать до 1280 дополнительных машинных команд для различных модификаций математического сопроцессора архитектуры x86.

В связи с тем, что в настоящее время реализовано только 63 машинные команды для 84-х команд языка ассемблера, этот набор команд обладает высокой избыточностью.

2. Команды модулей SSE-расширений архитектуры x86. Эти наборы команд сведены в отдельную таблицу команд, доступную через первый байт КОП 00001111 (или 0f), либо располагаются в таблицах команд математического сопроцессора и доступны через инструкцию 11011xxx (или D8x) первого байта (см. основную таблицу кодов).

3. Команды дополнительных модулей 3DNow!-расширений для процессоров AMD Athlon, совмещенных с архитектурой x86. Данные наборы имеют в своем составе 3 байта КОП. Причем первые два имеют конструкцию 00001111 00001111 (или 0f 0f), т.е. расположены в собственной таблице, спроектированной фирмой AMD для своих процессоров дополнительно к x86.

4. Команды многопроцессорных вычислительных систем. В настоящее время разработчиками архитектур процессоров ведется работа

по расширению наборов команд языка ассемблера и соответствующих машинных команд для мультипроцессорных систем, например, 3-х координатных (кубических) систем процессоров nCUBE 2.

Рассмотрим несколько примеров формирования кода машинной команды, соответствующих командам языка Ассемблера.

Команда ***MOV*** пересылки данных:

Из регистра 1 в регистр 2	1000100w	11 reg1 reg2		88
Из регистра 2 в регистр 1	1000101w	11 reg2 reg1		8B
Регистр – память	1000100w	mod reg r/m		88
Константа – регистр	1100011w	11 000 reg	непосредственные данные	C6
Константа – память	1100011w	mod 000 r/m	непосредственные данные	C6
Сумматор – память	1010001w	полное смещение (4 байта)		A3

Команда ***SAHF*** загрузка регистра флагов flags в регистр ***AH***:

10011110 9E.

Команда ***CBW*** преобразования байта в слово (2 байта):

10011000 98.

Команда ***CWDE*** преобразования слова в двойное слово (4 байта):

10011001 99.

4.10. Системы команд языка ассемблера для архитектур x86

Каждая модель (или семейство) процессоров различной архитектуры имеет свой набор или систему команд, инициирующих соответствующие микропрограммы в управляющем автомате, которые, в свою очередь, обеспечивают работу операционного блока, и соответствующий ему язык ассемблера. Наиболее популярные синтаксисы языков ассемблера это Intel-синтаксис (для архитектуры x86) и AT&T-синтаксис для процессоров коммуникационного оборудования этой фирмы.

Существуют компьютеры со встроенной системой трансляции, реализующие в качестве машинного язык программирования высокого уровня (например, Форт, Лисп, Эль-76). Фактически, в таких компьютерах они выполняют роль языков ассемблера.

4.10.1. Классификация базовых команд ассемблера x86

Как уже отмечалось, базовые команды языка ассемблера, машинные коды операций которых входят в основную таблицу кодов (см. таблицу 3), можно классифицировать согласно положениям раздела 4.8. следующим образом (рисунок 25).

Рисунок 25. – Классификация базовых команд ассемблера x86

4.10.2. Классификация дополнительных команд языка ассемблера математического сопроцессора x86

Схема центрального процессора архитектуры x86 содержит обычный сумматор, предназначенный для обработки простых чисел со знаком.

При выполнении операций с вещественными числами на дешифратор КОП ядра процессора поступает соответствующая команда с первым байтом кода операции 00011011. После этого управление дальнейшими операциями вычислений передается математическому сопроцессору. Все обрабатываемые операнды передаются в буферные регистры математического сопроцессора, представляющего собой самостоятельный операционный блок цифрового автомата. Здесь над ними производятся все необходимые для вычислений математические операции. Результат возвращается в аккумулятор центрального процессора.

Если математический сопроцессор отсутствует, то исполнение математических операций над вещественными числами осуществляется соответствующими программами BIOS, эмулирующими работу математического сопроцессора. Однако, при этом, вещественное число преобразуется в совокупность простых чисел (мантиссу и степень), с которыми работает АЛУ центрального процессора, затем производится обратное преобразование. Программное эмулирование процесса обработки вещественных чисел ядром центрального процессора приводило к существенному (на 2 порядка и более) замедлению процесса вычислений.

Математический сопроцессор появился в архитектуре x86 как дополнительный блок, обслуживаемый центральным процессором. Затем он вошел в состав архитектуры центрального процессора, как отдельный операционный блок, выполняющий несколько функций:

– простых операций с вещественными числами;

– арифметических действий с вещественными числами;
– математических вычислений трансцендентных функций;
– обработки потоковых данных формата MMX.

Принципиальная схема математического сопроцессора представляет собой простую стековую структуру программно-доступных регистров с физическим обозначением r0 – r7 и логическим st0 – st7 (рисунок 26). Логическое обозначение стековых регистров предусмотрено, в частности, для замены реальной кольцевой структуры их функционирования на стековую, что удобно для программиста, поскольку реальная структура стека может отличаться от его физической реализации.

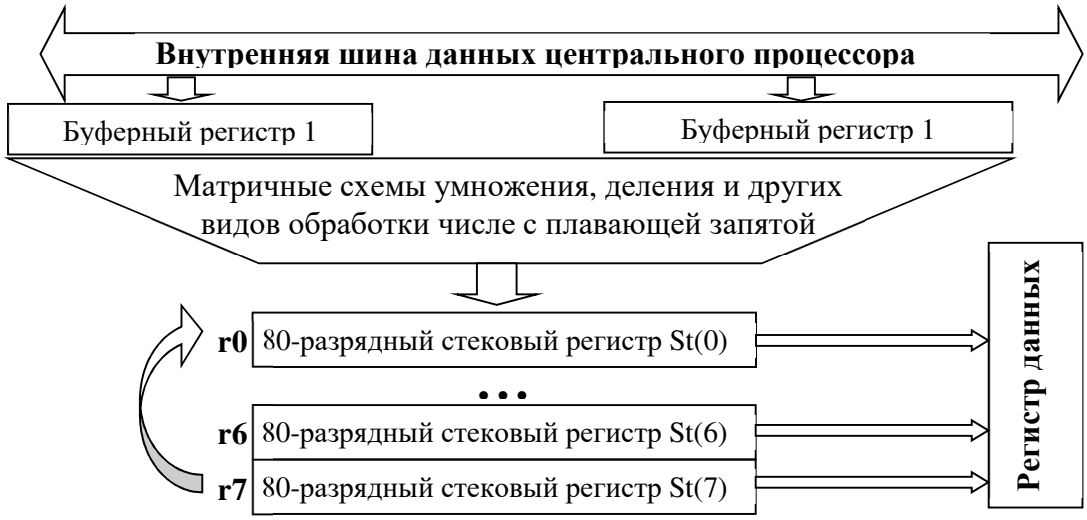


Рисунок 26. – Функциональная структура математического сопроцессора

Команды языка ассемблера для математического сопроцессора архитектуры x86 отличаются наличием первой буквы «F». Схема классификации команд представлена на рисунке 27.

4.10.3. Классификация команд языка ассемблера для различных расширений архитектуры ЦП x86

Дальнейшие расширения набора команд языка ассемблера для архитектуры x86 можно видеть на рисунках 28 – 30.

Наиболее подробное описание всех SSE-команд ассемблера, разработанных фирмой Intel для 64-разрадной архитектуры центрального процессора можно найти в справочнике «Intel 64 and IA-32 Architectures Software Developer Manuals», который можно найти по адресу: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html. Некоторые фирмы-производители клонов процессоров x86 также пытались совершенствовать архитектуру ЦП, вводя дополнительные операционные блоки и системы команд для них (см. рисунок 30).

Конечно, такие наборы команд были несовместимы с ЦП основной линейки x86 – x86-64, как несовместимо и использование команд старших моделей ЦП для предыдущих моделей.

Рисунок 27. – Классификация команд языка ассемблера для математического сопроцессора архитектуры x86

Рисунок 28. – Классификация MMX-команд языка ассемблера x86

Рисунок 29. – Классификация SSE-команд языка ассемблера x86-64

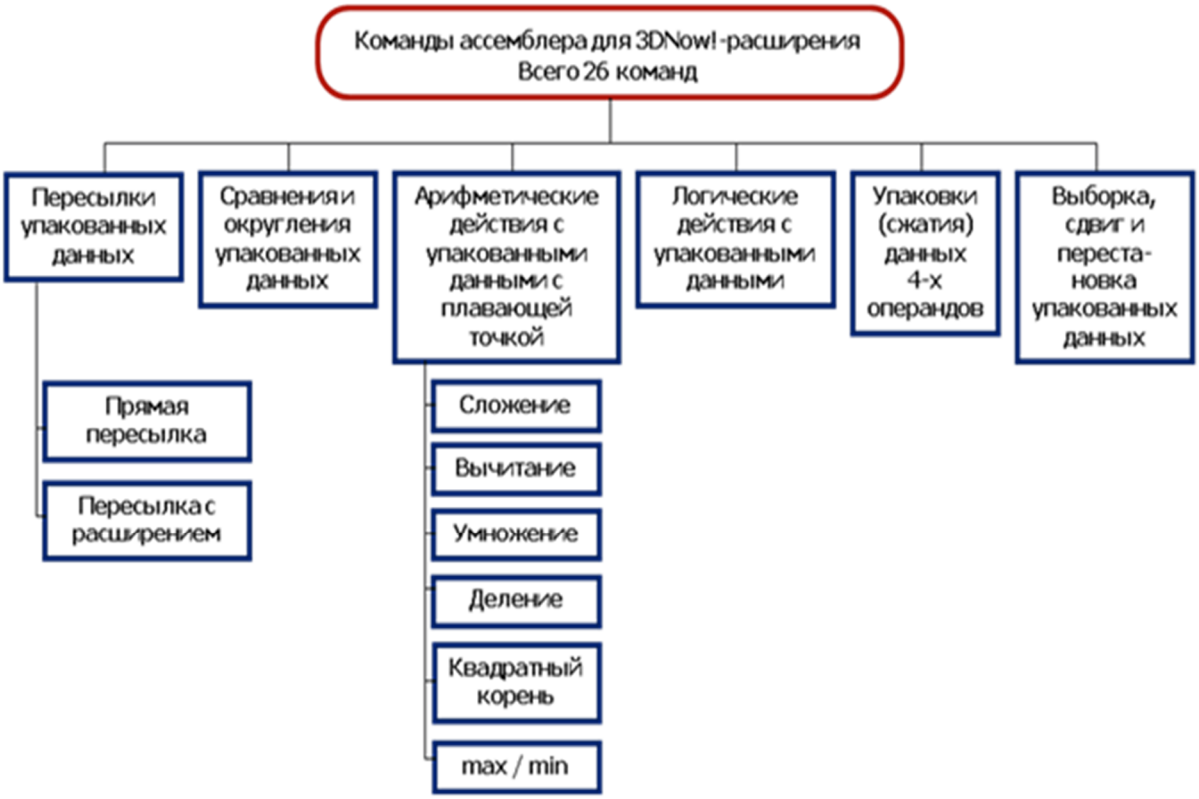


Рисунок 30. – Классификация частных расширений языка ассемблера x86

4.10.4. Основные программы для работы с машинным кодом

Для работы с исполняемым машинным кодом (при отсутствии исходного кода программ) созданы и широко используются программы-дизассемблеры. Это иногда необходимо для анализа сбойных участков программ, пошаговой отладки, поиска и уничтожения вирусов. Наиболее часто используются следующие программы-отладчики:

1. Microsoft Debugger (www.microsoft.com/whdc/devtools/ddk). Этот отладчик обычно входит в состав WDK (Windows Driver Kit – бывший Driver Development Kit или DDK), а также в комплект Debugging Tools. Оба варианта отладчика бесплатны, но WDK намного больше по объему и требует предварительной регистрации для получения Windows Live ID, в то время как Debugging Tools раздается без регистрации вместе с SDK, в которую входит документация, заголовочные файлы, библиотеки и несколько примеров написания плагинов.

2. OllyDbg (www.ollydbg.de). Считается лучшим отладчиком 32-х битных приложений. Имеет встроенный анализатор машинного кода, который распознает и визуально обозначает процедуры, циклы, константы и строки, а также внедренные в код обращения к функциям API NT и их параметры.

3. Immunity Debugger (www.immunitysec.com/products-immdbg.shtml). Отладчик является расширением предыдущей программы, выполненной фирмой Immunity под требования специалистов по безопасности, содержит встроенный яык интерпретации Питон, на котором написана прилагаемая библиотека плагинов.

Тема 5. Организация оперативной памяти ПЭВМ

5.1. Общие понятия

Организация памяти любой ЭВМ представляет собой аппаратный механизм, позволяющий операционной системе создавать для выполняющихся программ упрощенную среду. Например, при одновременном выполнении нескольких программ каждой из них должно быть дано независимое адресное пространство. При разделении всеми этими программами одного и того же адресного пространства каждая из них должна была бы выполнять сложные и занимающие много процессорного времени проверки, чтобы избежать влияния на другие программы.

Организация памяти состоит из сегментации и подкачки страниц.

Сегментация памяти служит для того, чтобы дать каждой программе несколько независимых, защищенных адресных пространств.

Подкачка страниц используется для поддержки среды, в которой большие адресные пространства моделируются на базе небольшой области оперативной памяти и некоторой дисковой памяти.

Разработчики систем могут использовать как оба этих механизма, так и любой из них. При одновременном выполнении нескольких программ для защиты программ от влияния на них других программ можно использовать любой механизм. Сегментация позволяет иметь полностью неструктурированную и простую память, подобную модели памяти 8-битового процессора, либо высоко структурированную память, с использованием трансляции адресов и защиты.

Средства организации памяти работают с единицами, которые называются сегментами. Каждый ***сегмент*** представляет собой независимое, защищенное адресное пространство. В режиме реальных адресов размер сегмента фиксированный и составляет 64 килобайта. В защищенном режиме размер сегмента определен соответствующим дескриптором и максимально составляет 1 мегабайт. Доступ к сегментам управляется данными, в которых описаны их размер, уровень привилегированности, который нужен для доступа к ним, типы ссылок к памяти, применимые к этому сегменту (выборка команды, помещение или извлечение из стека, операция чтения, операция записи и т.д.), а также его присутствие в памяти.

В случае простой архитектуры памяти все адреса относятся к одному и тому же адресному пространству. Это модель памяти была разработана для 8-битовых процессоров, таких как 8080, где логический и физический адреса полностью совпадали. Процессор с архитектурой x86 тоже может быть использован подобным же образом посредством отображения всех сегментов в одном и том же адресном пространстве и запрещения подкачки страниц памяти. Такая модель памяти называется «виртуальный режим 8086» и может понадобиться при адаптации старых разработок для 32-разрядного центрального процессора.

В процессорах архитектуры x86 предусматривается разделение пространств памяти и ввода-вывода. Пространство памяти (Memory Space) предназначено для хранения кодов инструкций и данных (в отличие от Гарвардовской архитектуры, где и эти пространства разделены на пространство памяти кодов и пространство памяти данных).

Для доступа к памяти имеется несколько способов адресации.

Память для центрального процессора x86 представляется в виде линейной последовательности байт. Память для 32-разрядных процессоров x86 подразделяется на байты (8 бит), слова (16 бит), двойные слова (32 бит) и учетверенные слова (64 бит). Поэтому для адресации памяти используется коэффициент масштабирования ss = 1, 2, 4 и 8. Слово (word) записывается в двух смежных байтах, начиная с младшего. Адресом слова является адрес его младшего байта (Low byte). Следующий байт (адрес на единицу больше) содержит старший (High) байт слова.

Все пространство памяти разбивается на ***параграфы*** – области из 16 смежных байт, начиная с нулевого адреса. Выравнивание по границе параграфа означает, что четыре младших бита адреса – нулевые.

Более крупными единицами организации памяти являются страницы и сегменты. Память может логически организовываться в виде одного или множества сегментов переменной длины (в реальном режиме – фиксированной). Кроме сегментации, в защищенном режиме возможно разбиение (Paging) логической памяти на ***страницы*** размером 4 Кбайт, каждая из которых может отображаться на любую область физической памяти. Начиная с 5-го поколения процессоров x86, появилась возможность увеличения размера страницы до 4 Мбайт, а сегмента – до 4 Гбайт.

Сегментация и разбиение на страницы могут применяться в любых сочетаниях.

Сегментация является средством организации логической памяти на прикладном уровне. Разбиение на страницы применяется на системном уровне для управления физической памятью. Сегменты и страницы могут выгружаться из физической оперативной памяти на диск и по мере необходимости подкачиваться с него обратно в физическую память. Таким образом реализуется ***виртуальная память***.

Применительно к оперативной памяти различают три адресных пространства: ***логическое, линейное*** и ***физическое*** адресные пространства.

Основным режимом работы 32-разрядных процессоров считается защищенный режим, в котором работают все механизмы преобразования адресных пространств.

5.2. Концепция многоуровневой памяти ПЭВМ

Память определяет эффективность работы ПЭВМ.

Обычно рассматривается три характеристики памяти: ***объем, быстродействие, стоимость***. Эти требования являются взаимно противоречивыми, поэтому невозможно реализовать один тип памяти, который бы отвечал всем этим требованиям. В современных ПЭВМ реализована многоуровневая схема памяти, обеспечивающая оптимальные характеристики для эффективной работы. Функциональная схема многоуровневой памяти представлена на рисунке 31.

Основной адресуемой памятью в этой схеме безусловно является оперативное запоминающее устройство (ОЗУ). Объем адресуемой памяти ОЗУ современных ПЭВМ может достигать 4 терабайта, однако эта память является энергозависимой и требует постоянной регенерации (перезаписи данных). Естественно, что при выключении электроснабжения ПЭВМ все данные в ОЗУ обнуляются.

Память внешних запоминающих устройств (ВЗУ) для центрального процессора адресно недоступна. По этой причине в архитектуру ПЭВМ введена система подкачки страниц, которая позволяет переносить и обновлять содержимое внешних накопителей в оперативную память, в сегменты программ и данных действующих программ.

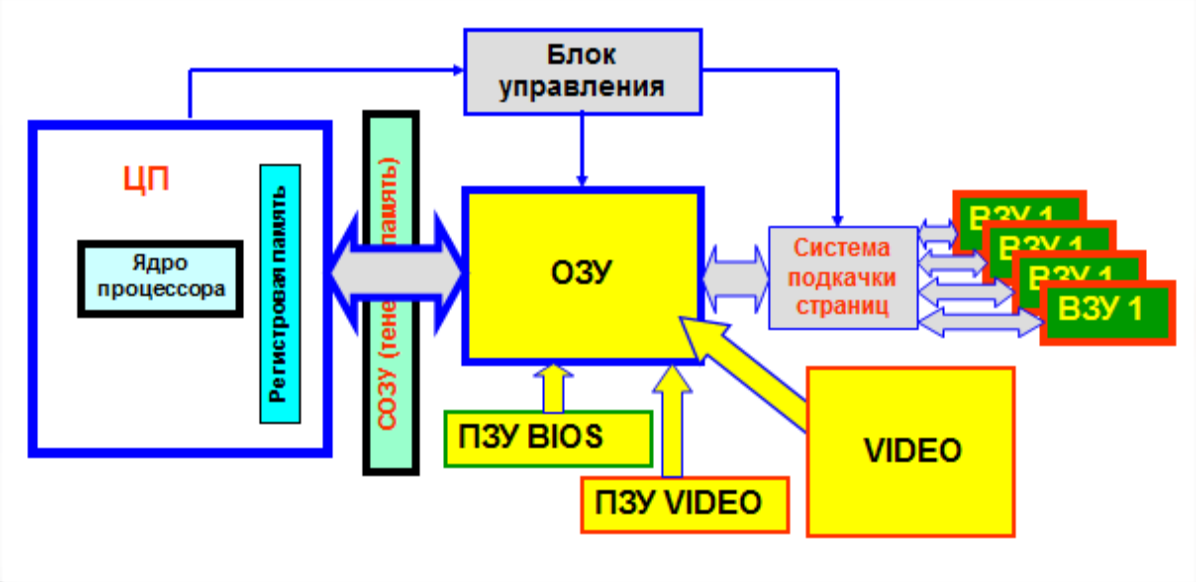


Рисунок 31. – Функциональная схема многоуровневой памяти ПЭВМ

Память видеоадаптера, которая может иметь свой достаточно большой объем, встраивается в адресное пространство центрального процессора и становится доступной для прямого обращения к ней. Для этой цели скорость обращения к видеопамяти выполняют по тактовой частоте, сравнимой с тактовой частотой ОЗУ.

Память энергонезависимых постоянных запоминающих устройств (ПЗУ) ввода-вывода BIOS системной платы и видеоадаптера обычно на порядок медленнее. По этой причине их содержание при включении ПЭВМ переписывается в ОЗУ.

Для ускорения работы оперативной памяти за счет сокращения числа обращений к ней вводится промежуточное сверхоперативное запоминающее устройство (СОЗУ или КЭШ от английского слова casher – прятать). Это скрытая память, тактовая частота которой сравнима с тактовой частотой центрального процессора. Прямая адресация этой памяти со стороны центрального процессора отсутствует, но реализован механизм т.н. ассоциативного доступа. Работает этот механизм следующим образом: при адресации ЦП в ОЗУ считывается не одна команда, а целый блок команд, следующих одна за другой (обычно несколько килобайт), который затем записывается в СОЗУ, а при следующем обращении к памяти (выборке следующей команды) анализируется содержание СОЗУ и, при совпадении критериев поиска, следующая команда считывается уже отсюда, что существенно сокращает время исполнения программы.

Стоимость СОЗУ (как и быстродействие) на порядок больше стоимости (и быстродействия) ОЗУ, поэтому их суммарная стоимость определяется оптимизацией объема, необходимого для ускорения процесса вычислений. Иногда принимается компромиссное решение о введении нескольких уровней КЭШ различной скорости и объема.

Следует отметить, что и ОЗУ и СОЗУ являются операционными блоками самостоятельных цифровых автоматов, оперирующих процессами приема-передачи и чтения-записи информации с обеспечением протоколов обмена. Для управления этими процессами в их состав входят устройства управления со схемной («жесткой») логикой.

5.3. Режимы адресации оперативной памяти ЦП

Процессор ПЭВМ может работать в одном из семи режимов адресации оперативной памяти и переключаться между ними достаточно быстро как в ту, так и в другую сторону:

1. Real Address Mode (Real Mode) – режим реальной адресации, полностью совместим с 8086 – наследуется от первой ПЭВМ архитектуры x86, а именно i8086. Формируемый линейный адрес в этом режиме является одновременно и физическим адресом доступной оперативной памяти. Возможна адресация до 1 Мбайта физической памяти. Режим реальной адресации оставлен в составе режимов, доступных современному ЦП, для обеспечения преемственности наработанных программных продуктов.

2. Protected Virtual Address Mode (Protected Mode) – защищенный режим виртуальной адресации. В этом режиме процессор позволяет адресовать до 4 Гбайт физической памяти, через которые при использовании механизма страничной адресации могут отображаться до 64 Тбайт виртуальной памяти каждой задачи. Это основной режим работы современного центрального процессора.

3. Virtual 8086 Mode – режим виртуального процессора 8086. Этот режим является особым состоянием задачи защищенного режима, в котором процессор функционирует как 8086. На одном процессоре в таком режиме может параллельно исполняться несколько задач с изолированными друг от друга ресурсами. При этом использование физического адресного пространства памяти управляется механизмами сегментации и трансляции страниц. Попытки выполнения недопустимых команд, выхода за рамки отведенного пространства памяти и разрешенной области ввода-вывода контролируются системой защиты.

4. Big Real Mode (Unreal Mode) – «нереальный» режим, который поддерживают все 32-разрядные процессоры, позволяет адресоваться ко всему 4-гигабайтному пространству памяти. Инструкции исполняются так же, как и в реальном режиме, но с помощью дополнительных сегментных регистров FS и GS программы получают непосредственный доступ к данным во всей физической памяти.

Этот режим появился случайно из-за отсутствия системы ограничений, вступающих в действие немедленно после возврата из защищенного режима, и действовал до первого системного обращения к защищенной памяти (поэтому и называется «нереальным»). Нереальный режим использовался некоторое время разработчиками игровых программ для расширения

адресуемой оперативной памяти и, как следствие, существенному ускорению формирования игровой (векторной) графики.

5. System Management Mode (SMM) – особый режим системного управления, при котором процессор выходит в иное, изолированное от остальных режимов, пространство памяти. Иное название этого режима – «спящий режим» ПЭВМ.

Спящий режим широко используется для целей энергосбережения, когда обращение к нему превышает определенный временной интервал или по прямому сигналу «гибернации». При этом вся информация из оперативной памяти записывается на жесткий диск. Диск останавливается, монитор «гаснет», ЦП переводится под управление миниоперационной системы SMM, находящейся в изолированной области памяти с ограниченным доступом. Иногда этот режим используется в служебных и отладочных целях.

На рисунке 32 показана схема переключений ЦП между режимами, которые могут осуществляются программно, опциями OS или с помощью специальных клавиш-переключателей «спящего» режима.

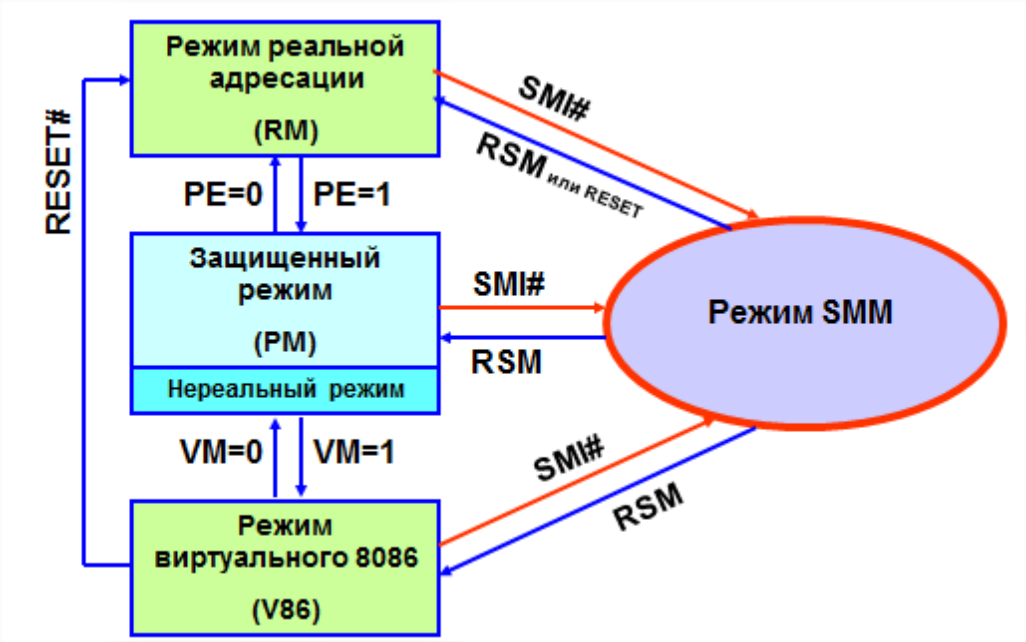


Рисунок 32. – Схема переходов ЦП x86 между режимами адресации

6. 64-bit mode – 64-битный режим – это режим полной поддержки 64-битной виртуальной адресации и 64-битных расширений регистров. В этом режиме используется только плоская модель памяти (общий сегмент для кода, данных и стека). По умолчанию разрядность адреса составляет 64 бита, а операндов (для большинства инструкций) – 32 бита, однако префиксом (REX) можно заказать 64-битные операнды. Имеется новый способ адресации данных – относительно указателя инструкций. Режим предназначен для использования 64-битными ОС при запуске 64-битных приложений: он включается операционной системой для сегмента кода конкретной задачи.

7. Compatibility mode – режим совместимости позволяет 64-битным ОС работать с 32- и 16-битными приложениями. Для приложений процессор выглядит как обычный 32-битный со всеми атрибутами защищенного режима, сегментацией и страничной трансляцией. 64-битные свойства используются только операционной системой, что отражается в процедурах трансляции адресов, обработки исключений и прерываний. Режим включается операционной системой для сегмента кода конкретной задачи.

5.4. Схема сегментного распределения оперативной памяти в режиме прямой адресации

Адресное пространство оперативной памяти в архитектуре i8086 (или виртуальной 8086) определяется разрядностью шины адреса и составляет 2^{20} байта = 1 Мбайт. И в этом адресном пространстве ЦП доступны лишь 4 сегмента памяти, два из которых (DS и ES) предназначены для размещения данных, сегмент CS предназначен для размещения исполняемой программы, а сегмент SS – для организации стека.

Размеры сегментов статические и определяются разрядностью логических адресов команд, данных и стека. Логические адреса вершины стека и команд хранятся в 16-разрядных регистрах SP и IP соответственно (рисунок 33), а логический адрес данных вычисляется в ЦП на втором этапе цикла одним из многочисленных методов адресации и также составляет 16 разрядов. Таким образом, размер каждого сегмента составляет 2^{16} байт = 64 Кбайта. Положение сегмента в адресном пространстве ОЗУ – его начальный адрес, определяется параграфом, хранящимся в 16-разрядном сегментном регистре.

Формирование физического адреса показано на схеме (см. рисунок 33), где видно, что граница сегмента в адресном пространстве может быть установлена не произвольно, а так, чтобы начальный адрес был кратен 16.

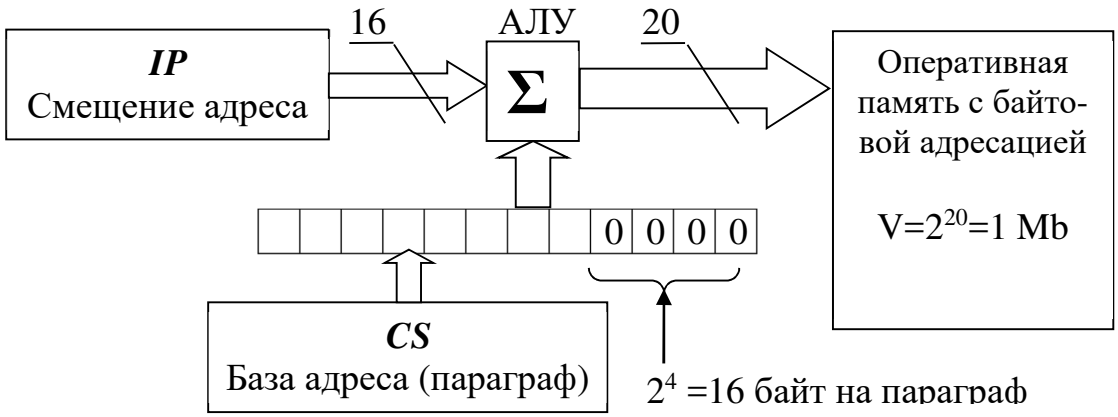


Рисунок 33. – Схема сегментного формирования физического адреса

Кроме пары CS : IP, для адресации данных и стека можно назначать любые другие регистры, используя префиксы.

Границы сегментов (параграф) могут быть выбраны таким образом, что сегменты будут изолированы друг от друга, а могут и пересекаться или вообще накладываться один на другой. Например, если указать условие: CS = DS = ES = SS = 0 (вариант организации адресного пространства ОЗУ архитектуры i8080), то все сегменты будут начинаться с нулевого адреса и совпадать друг с другом.

Недостатки сегментной адресации:

1. Размещение сегментов в памяти произвольное и на парактике осуществляется от свободного параграфа.

2. Сегменты могут перекрывать друг друга.

3. Защита памяти от несанкционированного доступа выполнена только программными средствами.

4. При случайном обращении к «чужой» памяти происходит зависание программы.

Для устранения этих недостатков в современных ПЭВМ используется система защищенной адресации, а сегментная адресация применяется только в виртуальных задачах для обеспечения преемственности программного обеспечения.

5.5. Распределение памяти в режиме исполнения приложений

Логическая структура распределения оперативной памяти в режиме прямой адресации по адресам от 00000 до A0000 (640 Kb), принятая в x86 (виртуальная машина i8086), представлена на схеме (рисунок 34).

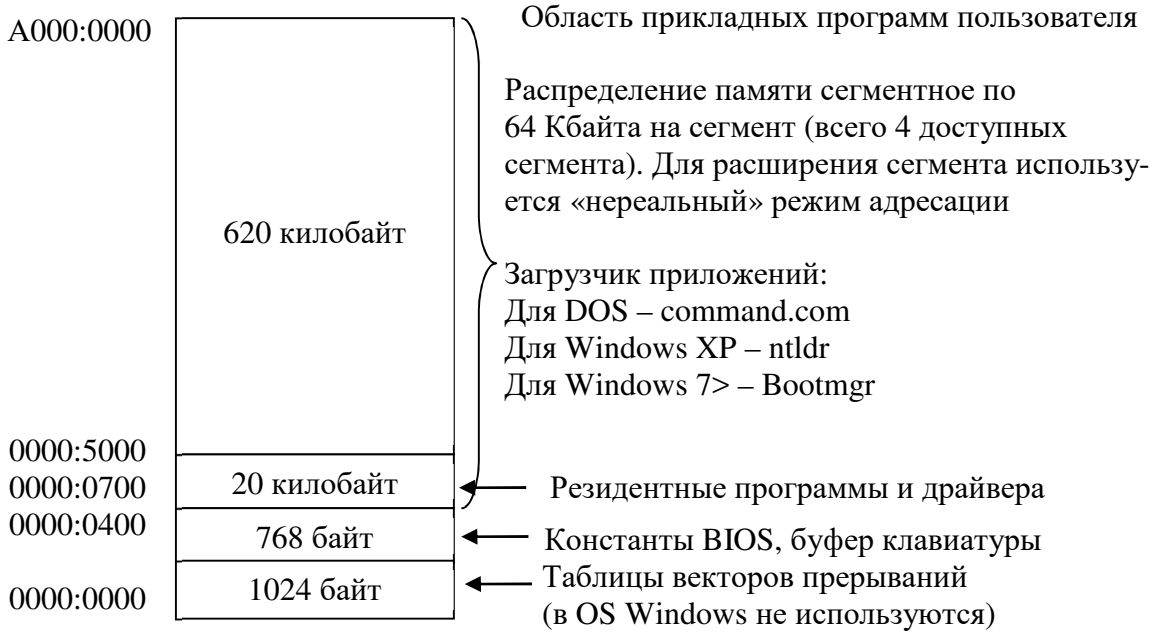


Рисунок 34. – Схема распределения пользовательской области оперативной памяти в режиме прямой адресации

Такая структура распределения памяти была разработана для режима прямой адресации в архитектуре x86 и применяется при выполнении большинства прикладных задач пользователя для обеспечения преемственности программного обеспечения, разработанного для x86. Логическая структура системной области памяти по адресам от A0000 до FFFFF (384 Kb), используемая в прикладных задачах, показана на рисунке 35.

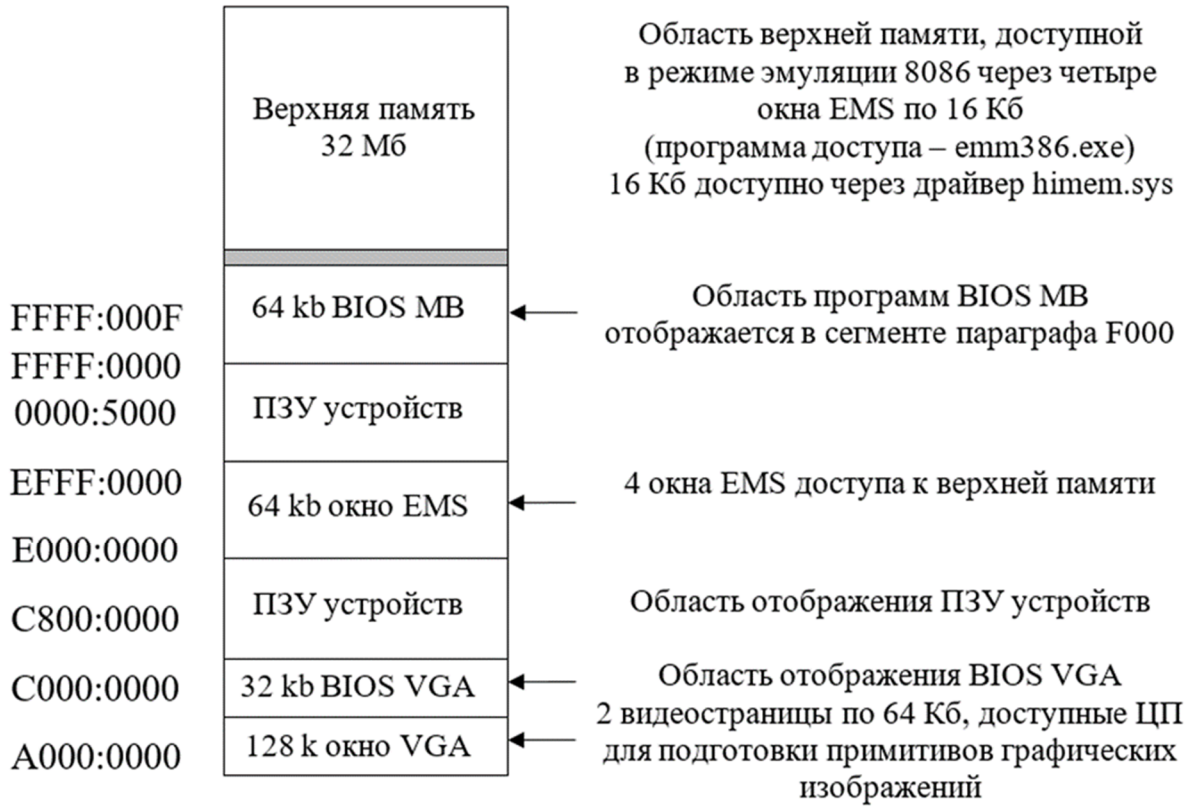


Рисунок 35. – Схема распределения системной области оперативной памяти в режиме прямой адресации (виртуальной 8086)

5.6. Схема формирования физического адреса в x86

Как уже было отмечено в пункте 4.7. при рассмотрении структуры машинной команды x86, существует несколько способов адресации операндов, вызываемых в операционный блок ЦП на втором этапе обработки машинной команды. Рассмотрим их более детально, чтобы представить схему формирования физического адреса оперативной памяти, используемую в современных процессорах архитектуры x86.

В данном разделе не рассматриваются конкретные команды языка ассемблера и форматы кодов машинных команд. Представлены только возможные методы и схемы формирования адресов оперативной памяти, используемые в архитектурах x86 – x86-64.

1. ***Неявная адресация.*** Операнд задается неявно на микропрограммном уровне. Такие команды не содержат операндов.

2. ***Непосредственная адресация.*** Операнд находится в команде и является частью её кода. Непосредственный операнд (константа) может быть только вторым операндом (источником).

3. ***Регистровая адресация.*** Операнд находится в одном из регистров РОН.

4. ***Адресация I/O.*** Операндом является содержимое порта ввода-вывода.

5. ***Стековая адресация.*** Операнд находится в стеке.

6. ***Прямая адресация.*** Эффективный адрес находится в команде. При этом возможны варианты:

6.1. ***Абсолютная прямая адресация.*** Эффективный адрес берется из поля смещения в команде.

6.2. ***Относительная прямая адресация.*** К полю смещения команды добавляется значение регистра – указателя команд IP/EIP.

7. ***Косвенная адресация.*** Операнд находится в оперативной памяти ПЭВМ или является результатом исполнения предыдущей команды. Возможны следующие варианты формирования косвенного адреса:

– ***Косвенная базовая адресация.*** Эффективный адрес вычисляется и размещается в одном из РОН.

– ***Косвенная базовая адресация со смещением.*** К базовому адресу добавляется значение вычисленного смещения, помещенного в другом РОН.

– ***Косвенная индексная адресация.*** Адрес в индексном регистре.

– ***Косвенная базовая индексная адресация.*** Эффективный адрес формируется как сумма базового РОН и индексного регистра.

– ***Косвенная базовая индексная адресация со смещением.*** Эффективный адрес формируется как сумма базового, индексного и РОН, в котором помещено вычисляемое смещение адреса.

Схема формирования физического (конечного) адреса оперативной памяти в режиме реальной адресации и защищенном режиме представлена на рисунке 36.

На схеме показаны также этапы формирования логического (вычисляемого в ядре ЦП), эффективного (с использованием АЛУ), линейного (с учетом используемой сегментной структуры оперативной памяти) и, наконец, физического адреса (в рамках доступной страницы памяти работающему приложению), который формируется с учетом страничного разбиения всей оперативной памяти ПЭВМ. Также отражены все возможные способы формирования адреса, как в режиме прямой адресации, так и в режиме защищенных адресов.

Сегментная и селекторная модели адресации памяти являются взаимнопереключаемыми при переходе из режима в режим. Заполнение дескрипторных регистров (регистров-указателей) и соответствующих им таблиц распределения памяти выполняется загружаемой операционной системой.

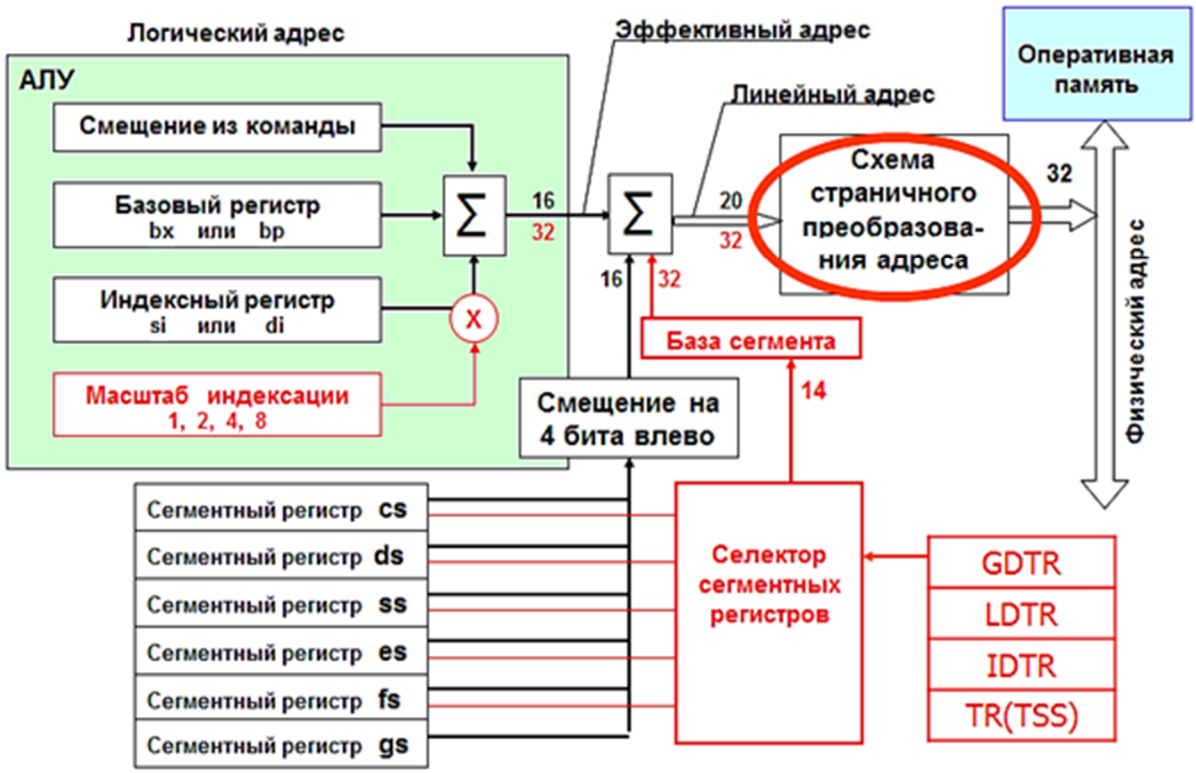


Рисунок 36. – Общая схема формирования адреса для ЦП x86

В режиме DOS (режим реальной 8086) дескрипторные регистры и таблицы заполнены нулями и используется только первый мегабайт установленной оперативной памяти. В режиме виртуальной 8086 используется схема сегментной адресации адресного пространства страницы оперативной памяти объемом в 1 мегабайт, выделенной операционной системой под прикладную задачу.

5.7. Защищенный режим адресации оперативной памяти ЦП

Защита необходима при работе в мультизадачном режиме. Средства защиты могут быть использованы для предотвращения взаимного влияния или пересечения адресных пространств одновременно выполняемых задач. Например, может выполняться защита от обращения одной задачи к командам или данным другой задачи. При разработке конкретной программы механизм защиты поможет дать более четкую картину программных ошибок. Когда программа выполняет неожиданную ссылку к недоступной в данный момент области памяти, механизм защиты может блокировать данное событие и сообщить о нем в процессе исполнения программы. При сбое в программе (обычно это деление на ноль) действие его распространится только на ограниченный домен памяти. Операционная система при этом также защищена от разрушения, что позволяет сгенерировать и зарегистрировать

диагностическое сообщение об ошибке исполняемого приложения и сделать попытку автоматического восстановления исполняемой программы.

Защита может применяться к сегментам и страницам памяти.

Два младших бита регистра селектора адреса процессора (рисунок 38) определяют уровень привилегированности текущей выполняемой программы (этот уровень называется ***текущим уровнем привилегий***, или CPL). Во время формирования адреса ЦП в защищенном режиме системой страничной адресации выполняется проверка CPL для доступа к сегменту или странице, содержащей этот адрес.

Для выключения механизма защиты специального управляющего регистра или бита режима не предусмотрено, эффект защиты включается автоматически при переключении режима адресации оперативной памяти присвоением «1» биту 0 регистра cr0.

Системные регистры, используемые в защищенном режиме:

Регистры управления:

Cr0 – регистр переключения режимов адресации. Побитовое содержание представлено в табличной форме на рисунке 37.

31	30	…	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PG	CD	…	AM															TS	EM	MP	PE

Бит 0 – PE (Protect Enable) = 0 реальная адресация; = 1 защищенный режим.
Бит 1 – MP (Math Present) = 1 наличие матсопроцессора.
Бит 3 – TS (Task Switched) = 1 переключение задач.
Бит 18 – AM (Alignment Mask) = 1 устанавливает контроль выравнивания байтов в операндах (аналогичен байту AC Eflags).
Бит 30 – CD (Cache Disable) = 1 запрещение КЭШ-памяти.
Бит 31 – PG (PaGing) = 1 разрешение страничного распределения и преобразования адресного пространства ЦП.

Рисунок 37. – Содержание регистра cr0

Cr1 – зарезервирован и не используется.

Cr2 – регистр страничной организации памяти (свопинга) (хранит 32-битный линейный адрес, по которому был получен последний отказ страницы памяти).

Cr3 – регистр каталога страниц первого уровня (содержит 20-битный физический базовый адрес каталога страниц текущей задачи).

Cr4 – регистр (присутствует с x86 пятого поколения) разрешения новых архитектурных расширений центрального процессора, например, увеличения размера страницы с 4 Кбайт до 4 Мбайт, поддержка расширенной физической адресации до 36 бит (вместо 32) и др.

Системные регистры-указатели таблиц распределения памяти:

GDTR (Global Descriptor Table Register) – регистр глобальных дескрипторов.

IDTR (Interrupt Descriptor Table Register) – регистр таблицы дескрипторов прерываний.

TR (Task Register) – регистр селектора текущей задачи в таблице GDT, описывающий текущий сегмент состояния задачи TSS (Task Segment Status).

LDTR (Local Descriptor Table Register) – регистр локальных дескрипторов задачи.

Регистры отладки:

dr0, ***dr1***, ***dr2***, ***dr3*** – линейные адреса точек останова текущей исполняемой задачи.

dr6 и ***dr7*** – регистры состояния и управления процессом отладки.

5.8. Схема сегментного распределения памяти в режиме защищенных адресов

В соответствии со схемой формирования адреса в защищенном режиме (см. рисунок 38) сегмент, в котором находится искомый адрес памяти (точнее его граница) определяется индексом, входящим в состав селектора, содержащемся в сегментном регистре.

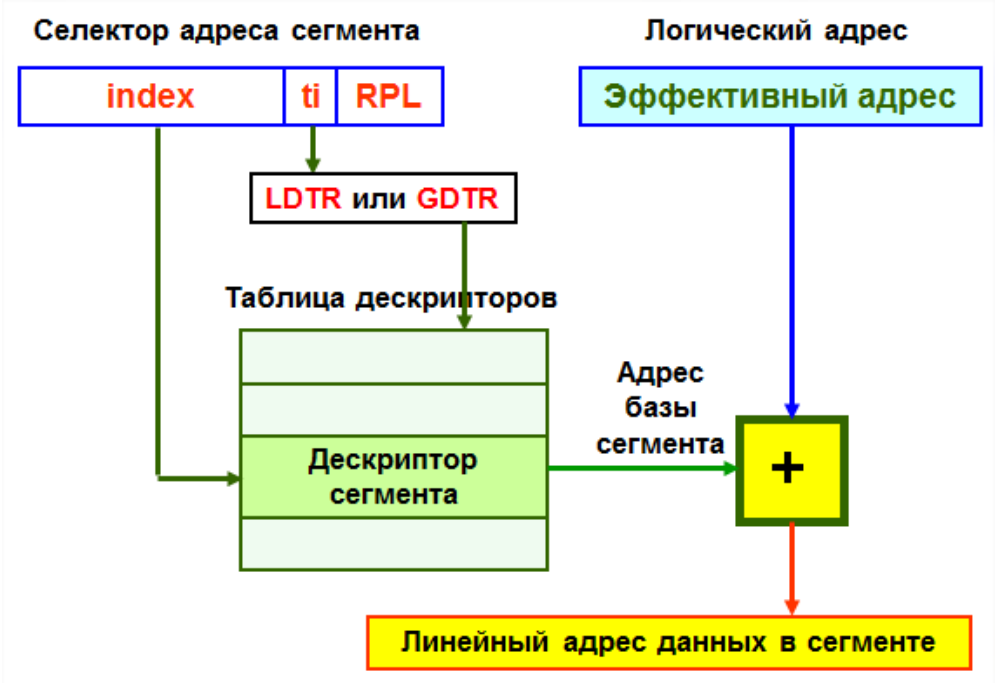


Рисунок 38. – Схема сегментной адресации в защищенном режиме x86

Базовый адрес сегмента берется из дескриптора сегмента – структуры данных в памяти, которая содержит информацию о размере и расположении

данного сегмента в памяти, а также информацию о требуемом уровне привилегий для доступа к нему. Дескриптор сегмента выбирается по индексу в одной из двух таблиц, таблицы глобальных дескрипторов (GDT) или таблицы локальных дескрипторов (LDT), определяемых битом TI селектора. Базовый адрес этих таблиц, к которому добавляется индекс в качестве смещения адреса, находится в соответствующем дескрипторном регистре GDTR или LDTR. Логический адрес транслируется в линейный адрес сложением смещения и найденного базового адреса сегмента. Схема формирования линейного адреса в защищенном режиме показана на рисунке 38.

Для всех программ (в том числе и операционной системы) существует единая (глобальная) таблица GDT, а также по одной локальной таблице LDT для каждой отдельной, выполняемой в текущий момент, программы. Каждый логический адрес связан со своим сегментом (даже, если система отображает все сегменты в одном и том же линейном адресном пространстве). Программа может иметь тысячи сегментов, но для немедленного использования доступны только шесть из них. В архитектуре x86 имеется только шесть сегментов, селекторы которых загружены в соответствующие регистры процессора.

Селектор сегмента содержит информацию, используемую для транслирования логического адреса в соответствующий линейный адрес. Для каждого вида ссылки к памяти (пространству кода, пространству стека и пространствам данных) в процессоре существуют отдельные сегментные регистры. Они содержат селекторы текущих используемых сегментов.

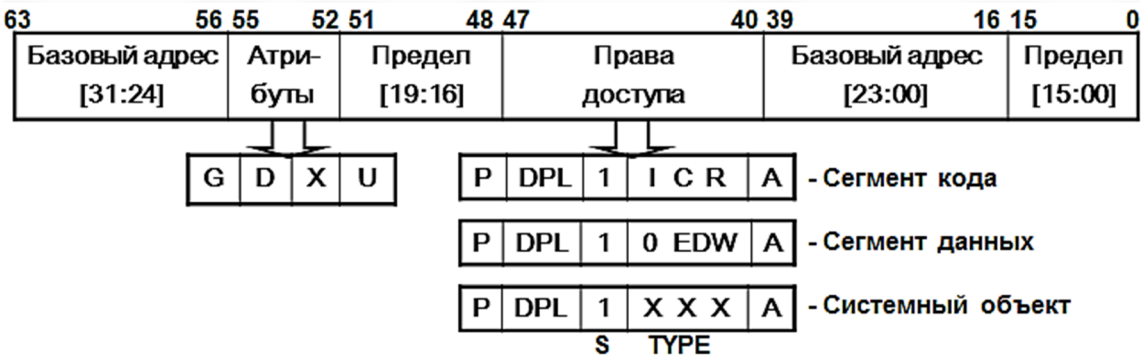
Доступ к другим сегментам требует загрузки сегментного регистра с помощью разновидности команды MOV. Одновременно может быть доступно до четырех сегментов данных, что в сумме с остальными составляет шесть сегментных регистров.

Бит ti селектора адреса (бит 2), равный нулю, определяет глобальную GDT, а единица – локальную LDT таблицу дескрипторов, базовый адрес которых содержится в соответствующих регистрах GDTR или LDTR. Два бита RPL (биты 0 и 1) определяют уровень привилегий задачи.

Содержание дескриптора, используемого для описания сегментов кода и данных прикладных программ, представлено на рисунке 39.

Кроме указанной информации дескриптор содержит конкретный базовый адрес расположения границы сегмента на текущей странице и его объем в байтах. Далее к этому базовому адресу добавляется смещение в виде вычисленного эффективного адреса и с этого места начинается выборка команд или данных.

Такая достаточно сложная процедура нахождения информации, существенно отличающаяся от схемы прямой адресации, создана в рамках архитектуры x86 для обеспечения более устойчивой работы программного обеспечения (операционных систем и пользовательских программ).



G – бит грануляции = 0 граница от 1 байта до 4 Мбайт, = 1 от 1 Кбайта до 4 Гбайт;
D – бит размера операндов по умолчанию. Распознается только в дескрипторах кодового сегмента: = 0 – 16-битовый размер; = 1 – 32-битовый размер;
X – зарезервировано Intel (не подлежит использованию);
U – доступно только для системного программного обеспечения;
P – присутствие сегмента (наличие указанной области памяти);
DPL – уровень привилегированности данного дескриптора;
S – бит типа дескриптора: = 0 – системный; = 1 – прикладной;
TYPE – тип сегмента (сегмент кода, сегмент данных, системный объект, определяет тип доступа и направление наращивания от базового адреса сегмента);
A – бит обращения (определяет наличие обращения к сегменту).

Рисунок 39. – Содержание дескриптора сегмента

5.9. Организация страничной памяти

Наряду с сегментной организацией оперативной памяти в архитектуре x86 возможна дополнительная ***страничная организация памяти*** (см. рисунок 36). Механизм страничной организации памяти включается/выключается программным путем при установке/сбросе флага PG регистра cr0. При этом все линейное адресное пространство оперативной памяти делится на разделы, число которых может достигать 1024. Каждый раздел в свою очередь может содержать до 1024 страниц. Размер каждой страницы фиксированный (в отличие от размера сегмента) и составляет 4 Кбайта. Кроме того, начальные адреса страниц жестко фиксированы в физическом адресном пространстве: границы страниц совпадают с с границами 4-килобайтных блоков.

Такая организация оперативной памяти является очень удобной при формировании обмена информацией между ОЗУ и внешними запоминающими устройствами на жестких дисках или flesh-накопителях, т.к. свопинг (подкачка информации) в ОЗУ осуществляется обычно небольшими блоками, имеющими фиксированный размер, равный или кратный размеру страницы.

При страничной адресации линейный адрес можно рассматривать, как интерпретацию физического адреса извлекаемой информации, содержащего три поля: номер раздела, номер страницы и смещение на странице.

Начальные адреса страниц текущего раздела (вместе с атрибутами) хранятся в ***страничной таблице PTE***. Поскольку в области памяти задачи (сегменте задачи) может быть несколько разделов, а, следовательно, столько же страничных таблиц, то начальные адреса всех страничных таблиц одного сегмента хранятся в специальной таблице – ***каталоге раздела PDE***. 20-битный физический базовый адрес каталога раздела текущей задачи хранится в регистре ***cr3***.

Линейный 32-разрядный адрес, полученный в схеме формирования, показанной на рисунке 36, является исходной информацией для формирования 32-х разрядного физического адреса ОЗУ с помощью каталога разделов и страничной таблицы (рисунок 40).

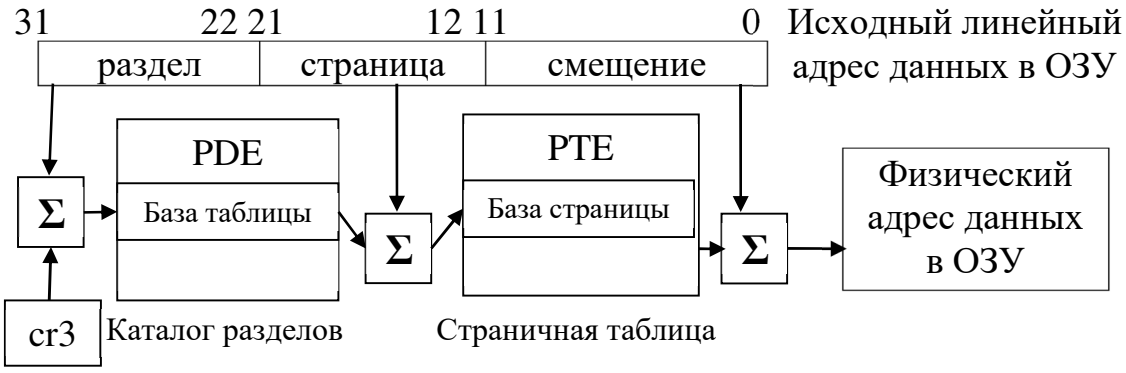


Рисунок 40. – Схема страничного преобразования линейного адреса

Старшие 10 разрядов линейного адреса можно рассматривать как смещение по отношению к базовому адресу каталога разделов, хранящемуся в регистре cr3. Их сумма определит номер строки каталога разделов, содержащей базовый адрес требуемой страничной таблицы. Следующие 10 разрядов (с 12 по 21) в сумме с базовым адресом страничной таблицы дадут линейный адрес строки страничной таблицы, содержащей базовый адрес искомой страницы. Сложив извлеченный базовый адрес искомой страницы, содержащей требуемую информацию, с младшими 12 разрядами исходного линейного адреса получим нужный физический адрес информации в ОЗУ.

Таким образом, при включении системы страничной организации ОЗУ контроллер оперативной памяти MMU (Memory Management Unit) в защищенном режиме работы ЦП, наряду с обязательной сегментацией памяти, реализует еще один уровень формирования физического адреса по исходному вычисляемому линейному адресу.

5.10. Уровни привилегий адресуемого пространства памяти и понятие защиты памяти

Четырехуровневая иерархическая система привилегий предназначена для управления использованием привилегированных инструкций и доступам к сегментам памяти. Уровни привилегий нумеруются от 0 до 3.

Нулевой уровень соответствует максимальным возможностям доступа к памяти (PL = 00). Третий уровень имеет самые ограниченные права и отводится прикладным задачам (PL = 11). Схема распределения всей доступной центральному процессору адресуемой памяти («куча» памяти – англ. ***heap memory***), выделенной операционной системой для размещения динамических структур данных (страниц и сегментов), показана на рисунке 41.

Рисунок 41. – Схема распределения памяти по уровням привилегий

Вся доступная память разделена на 4 области по уровням привилегий, доступ к которым осуществляется механизмом защиты OS. Защита оперативной памяти предназначена для предотвращения несанкционированного доступа к ней и выполнения критических инструкций, например, команд HLT (останов процессора), команд ввода-вывода, команд управления флагом разрешения прерываний и команд, влияющих на изменение размера и места расположения сегментов кода и данных.

Механизм защиты вводит следующие ограничения:

1. ***Ограничение использования сегментов***. В качестве сегментов могут использоваться только описанные в соответствующих дескрипторах таблиц GDT и LDT области оперативной памяти.

2. ***Ограничение доступа к сегментам*** через правила сравнения и непревышения привилегий.

3. ***Ограничение набора инструкций***: исключение использования некоторых системных команд в пользовательских приложениях.

4. ***Ограничение возможности межсегментных вызовов*** и передачи управления без использования механизма переключения задач.

5. ***Проверка механизма страничной адресации***.

5.11. Схемы физической организации оперативной памяти

5.11.1. Адресная память (память произвольного доступа)

Эта память носит название RAM (***Random Access Memory***) и является памятью, все ячейки которой имеют определенные адреса, что обеспечивает соблюдение одного из основных принципов построения архитектуры ЭВМ – ***принципа адресации***. Функциональная схема такой памяти является достаточно простой (рисунок 42).

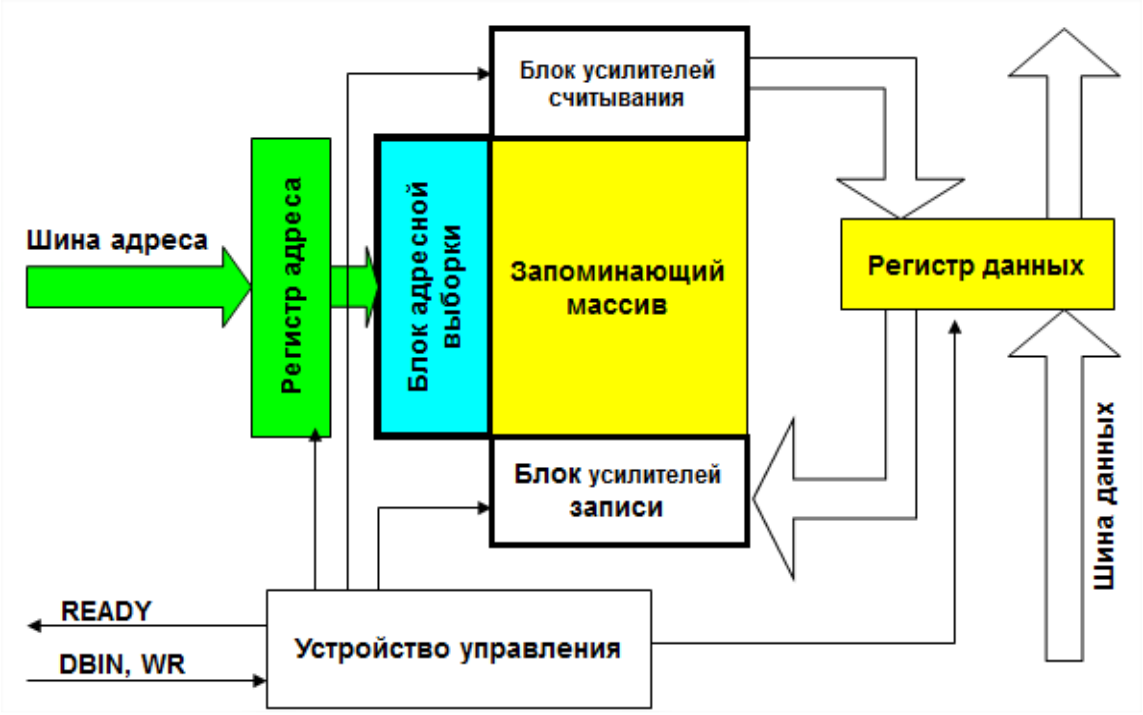


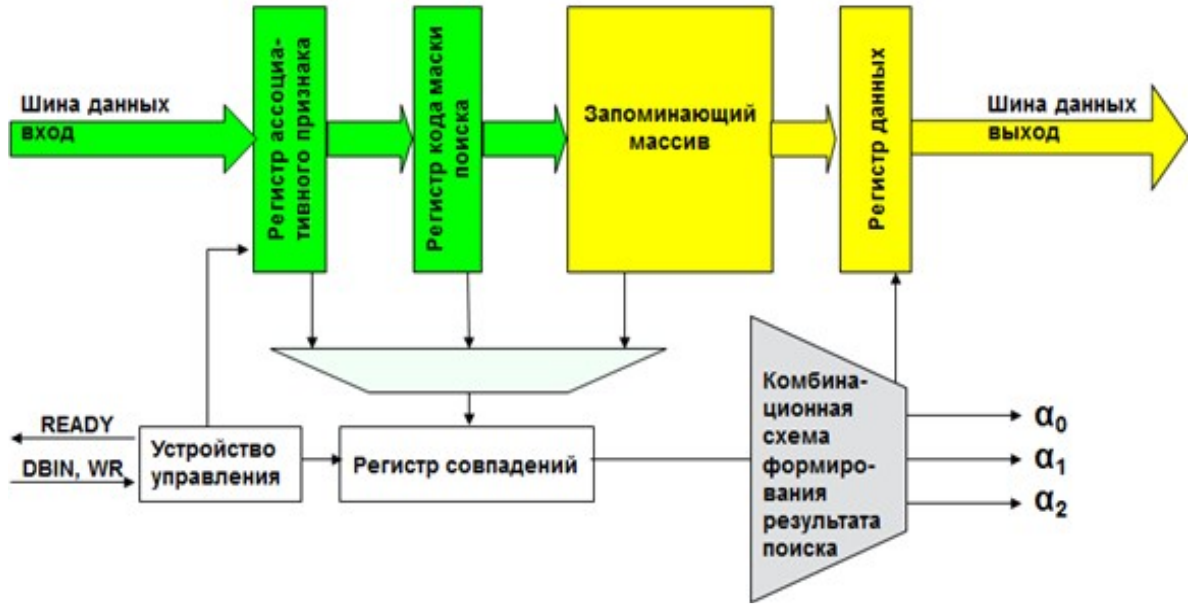
Рисунок 42. – Функциональная схема памяти произвольного доступа

Оперативная память является вполне самостоятельным устройством в составе архитектуры ЭВМ, которое поддерживает асинхронный обмен информацией с центральным процессором. Операционный автомат этого устройства состоит из запоминающего массива, усилителей чтения/записи и буферных регистров. Управляющий автомат построен на базе схемной логики и обеспечивает управление операциями чтения/записи с поддержанием протоколов обмена данными по системной шине, как с центральным процессором, так и с другими устройствами из состава архитектуры ЭВМ, например, с контроллером прямого доступа к памяти.

На шину адреса поступает адрес выборки, сопровождаемый сигналами чтения или записи. ЦП приступает к ожиданию сигнала готовности контроллера памяти. Процесс выборки завершается записью или считыванием соответствующего байта или блока данных, сопровождаемый сигналами запроса со стороны центрального процессора и сигналом готовности контроллера оперативной памяти.

5.11.2. Ассоциативная память (сверхоперативная память)

Ассоциативная память в составе архитектуры ЭВМ выполняет вспомогательные функции временного хранения блоков информации, предоставляемой центральному процессору (рисунок 43).



α_0 = 1 – считывание данных запрещено (данных в памяти не найдено);
α_1 = 1 – считывается найденное слово, являющееся единственным;
α_2 = 1 – считывается слово из ячейки, имеющей наименьший возраст среди найденных по результатам поиска (наименьшее время хранения).

Рисунок 43. – Функциональная схема памяти с ассоциативным доступом

Область ассоциативной памяти обычно располагают в статической сверхоперативной памяти. Обладая на порядок более высоким быстродействием, она работает на тактовой частоте ЦП. По этой причине обращение к этой памяти существенно ускоряет работу программ.

Процесс выборки данных из ассоциативной памяти осуществляется не по адресу, а по содержанию информации (ассоциативному признаку или по отдельным разрядам этого признака, например, младшей частью адреса операнда). Если процесс выборки информации из ассоциативной памяти заканчивается безрезультатно, то обращение следует к контроллеру оперативной памяти. Но при этом из оперативной памяти извлекается не одна требуемая команда или операнд, а целый блок, который затем записывается в ассоциативную память в предположении, что запросы информации со стороны ЦП будут следовать в естественном порядке чередования адресов (в порядке убывания или возрастания).

Запись результатов вычислений в эту память не производится.

5.11.3. Стековая память

Стековая память является безадресной, но имеет одномерную структуру. Адрес базы стека и вершины стека известен и находится в регистрах SS:BP и SS:SP соответственно (рисунок 44).

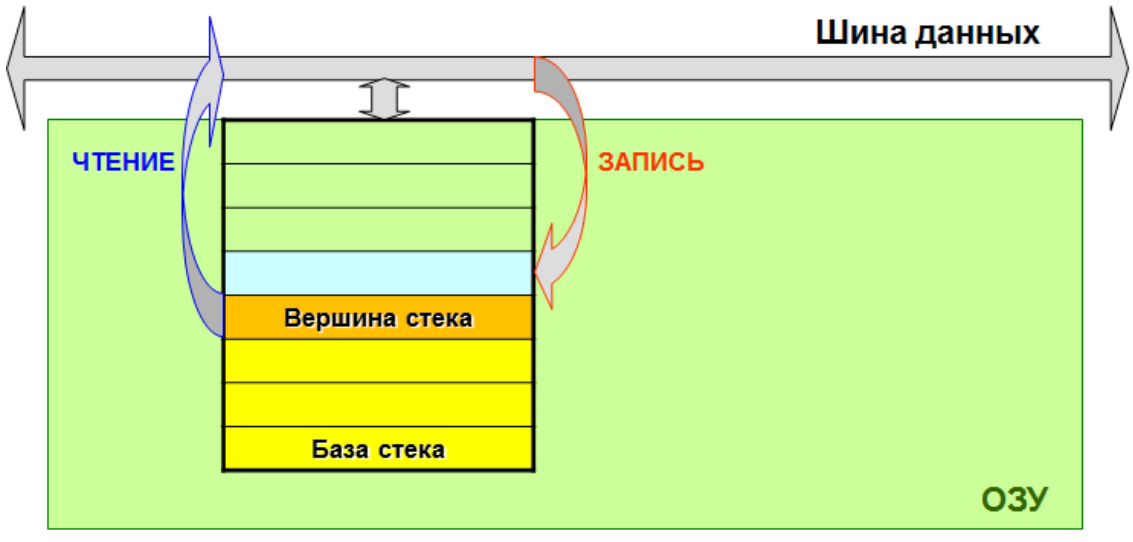


Рисунок 44. – Функциональная схема стековой памяти

Принцип действия стековой памяти обозначается акронимом FIFO, что означает First In, First Out — «первым пришел, первым ушел». Такой способ организации манипулирования данными в стековой памяти позволяет отказаться от очереди и обслуживать строго вложенные структуры данных, например, вложенные прерывания.

Запись и считывание возможны только из строго определенных ячеек памяти: чтение осуществляется из вершины стека (последней занятой ячейки стека), при этом значение смещения адреса SP уменьшается на единицу, запись осуществляется в следующую свободную ячейку стека перед ее вершиной и адрес этой ячейки памяти заносится в указатель вершины стека.

Стековая память обычно организуется в сегменте стека оперативной памяти, но возможны варианты организации стека на базе регистровой памяти (например, в схеме математического сопроцессора на рисунке 26).

5.12. Типы исполнения памяти (классификация памяти)

Все типы исполнения памяти, используемые в архитектуре x86 для организации ОЗУ, можно представить в виде схемы (рисунок 45).

Всю память, используемую в ПЭВМ, можно разделить на три группы:

1. Динамическая память DRAM (Dynamic Random Access Memory) – экономичный вид памяти. Для хранения разряда (бита) используется схема, состоящая из одного конденсатора и одного транзистора (в некоторых вариантах два конденсатора).

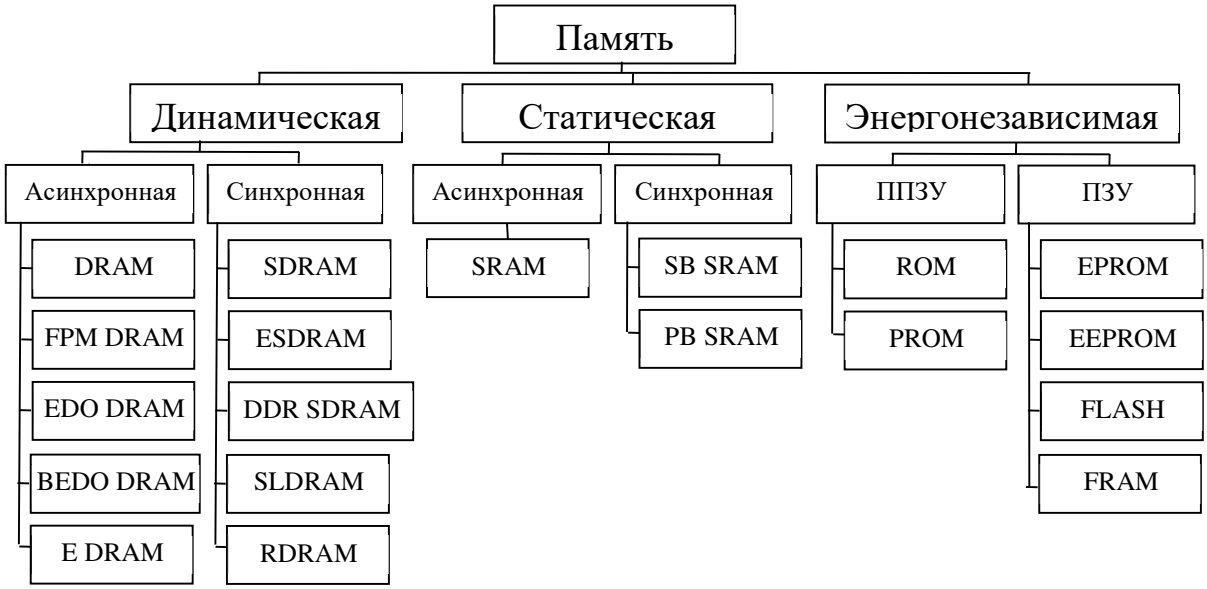


Рисунок 45. – Схема типов исполнения памяти

Такой вид памяти дешевле статической (один конденсатор и один транзистор на 1 бит информации в отличие от четырех транзисторов триггера), и занимает меньшую площадь на кристалле (на месте одного триггера статической памяти можно разместить несколько конденсаторов и транзисторов для хранения нескольких бит).

Но DRAM имеет ряд недостатков:

– Скорость работы медленнее, чем у статической памяти SRAM, поскольку, если в SRAM изменение управляющего напряжения на входе триггера сразу изменяет его состояние, то для того, чтобы изменить состояние конденсатора, его нужно зарядить или разрядить. Перезаряд конденсатора осуществляется на порядок медленнее, чем переключение триггера, даже если ёмкость конденсатора очень мала.

– Конденсатор со временем разряжается. Причем разряжается тем быстрее, чем меньше его электрическая ёмкость и больше ток утечки. Именно из-за того, что заряд конденсатора динамически уменьшается во времени, память на конденсаторах и получила свое название DRAM – динамическая память. Поэтому, чтобы не потерять содержимое памяти, заряд конденсаторов необходимо периодически восстанавливать («регенерировать») через определенное время, называемое циклом регенерации (обычно 2 мс). Для регенерации памяти в современных микросхемах достаточно выполнить циклограмму «чтения» по всем строкам запоминающей матрицы. Процедуру регенерации выполняет центральный процессор или контроллер памяти. Так как для регенерации памяти периодически приостанавливается обращение к памяти, это, в свою очередь, приводит к снижению скорости обмена.

2. Статическая память SRAM (Static Random Access Memory) – имеет существенно более сложную аппаратную реализацию (обычно транзисторно-транзисторную логику), строящуюся на базе отдельных триггеров,

и, поэтому, не требующую постоянного возобновления содержащейся в ней информации. Такой тип памяти используется обычно для формирования КЭШ различного уровня с ассоциативной организацией доступа к хранящейся информации, а также массивов буферов портовой (ввода/вывода) и регистровой памяти ЦП.

3. Энергонезависимая память – используется в ПЭВМ для хранения информации без подачи напряжения на элементы-носители информации. Обычно это магнитные или оптические накопители информации. Здесь следует отметить, что считывание/запись информации на таких типах памяти также невозможна без включения электропитания, обеспечивающего работу управляющих автоматов.

Группы динамической и статической памяти подразделяются на подгруппы асинхронной и синхронной памяти.

Синхронная память – устройства управления такой памяти не имеют в своем составе механизмов синхронизации (контроля готовности) с внешней средой исполнения программ (контроллерами шин), т.к. рассчитаны на работу с определенной тактовой частотой внешнего генератора синхронизации, указанной в маркировке памяти.

Такая память работает существенно быстрее асинхронной этого же типа за счет исключения циклов контроля готовности и подтверждения запросов на чтение/запись. Но при изменении тактовой частоты генератора синхронизации свыше допустимой величины (обычно не более 5%), что вполне допустимо в архитектуре современных ПЭВМ, она работать не будет из-за потери синхронизации.

Асинхронная память – этот тип памяти по схеме операционного блока может быть идентичен соответствующей синхронной памяти, но в состав аппаратной реализации её управляющего автомата включена схема синхронизации циклов чтения/записи, что позволяет использовать эту память при больших отклонениях частоты синхронизации от расчетной.

Группа энергонезависимой памяти является асинхронной. Для синхронизации работы в архитектуре ПЭВМ в ее состав введен буферный регистр. Тактовая частота такой памяти обычно на порядок меньше тактовой частоты ОЗУ. В свою очередь группа разбита на две подгруппы:

1) ***программируемое постоянное запоминающее устройство*** (***ППЗУ*** или ***ROM*** – read only memory) – этот тип энергонезависимой памяти заполняется при изготовлении или с использованием специальных устройств – программаторов ППЗУ и перезаписи информации не допускает;

2) ***постоянное запоминающее устройство*** (***ПЗУ*** или ***RW –*** read/write memory) – этот тип памяти разрешает многократную перезапись информации, хотя количество циклов перезаписи может быть ограничено. Например, прожигаемые микросхемы PROM допускают только 3–5 циклов перезаписи, а EEPROM (Electrically Erasable Programmable Read-Only Memory) – до миллиона циклов.

Дальнейшее разбиение типов памяти в подгруппах обусловлено технологиями их изготовления и используемыми физическими принципами представления информации на носителе:

– ***DDR SDRAM*** (Double Data Rate Synchronous Dynamic Random Access Memory) – синхронная динамическая память с произвольным доступом и удвоенной скоростью передачи данных). Удвоенная скорость передачи данных достигается за счет выполнения обращения к памяти не по уровню, а по фронту, т.е. два раза за время одного цикла.

– ***EDO DRAM*** (Extended data out dynamic Random Access Memory) – динамическая память с увеличенным временем доступности данных.

– ***SB SRAM*** (Sync Burst Static Random Access Memory) – синхронная статическая память, позволяющая вести пакетную операцию обмена, свойственную работе КЭШ-памяти. В структуру её операционного блока введен двухбитный счетчик адреса, а управляющий автомат генерирует сигналы для синхронизации с системной шиной и сигналы пакетной обработки. Время доступа – 8,5; 10 и 13,5 нс на частотах 66, 60 и 50 МГц.

– ***PB SRAM*** (Pipelined Burst Static Random Access Memory) – пакетно-конвейерная синхронная память. Конвейером является дополнительный внутренний регистр данных. Интерфейс PB SRAM аналогичен интерфейсу SB SRAM, но есть задержка из-за синхронизирующего перехода.

Современные технологии позволяют разрабатывать и изготавливать достаточно большой спектр типов памяти, используемых в вычислительной технике.

5.13. Схема доступа к ячейкам памяти ОЗУ

Современные технологии изготовления памяти предполагают её многослойную структуру. При этом весь адресуемый битовый массив памяти формируется в одной плоскости с координатами «строка-столбец», а количество слоев определяет разрядность памяти. Количество строк и столбцов может быть различным и не совпадает с исходным адресом информации в памяти. Для преобразования адреса в значения параметра «строка-столбец» используется схема дешифрации адреса.

Устройство управления асинхронной памяти может иметь свой генератор синхронизации. Функциональная схема организации памяти произвольного доступа, реализуемая на современном уровне, показана на рисунке 46.

Доступ к оперативной памяти осуществляется по системной шине памяти и шине адреса. Разрядность шины адреса составляет 32 бита, а разрядность шины данных определяется разрядностью памяти, и в современных ЭВМ достигает 256 разрядов и более. Порядок доступа к оперативной памяти определен протоколом работы шины и описывается временными диаграммами циклов работы шины.

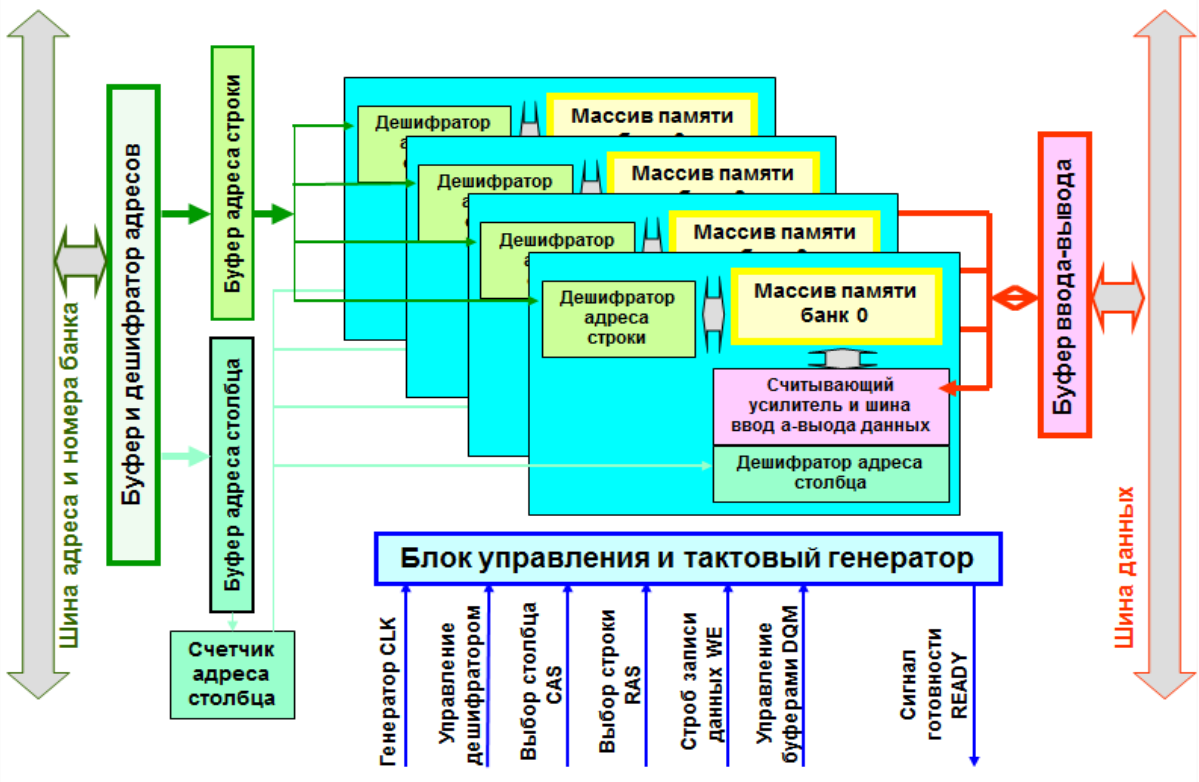
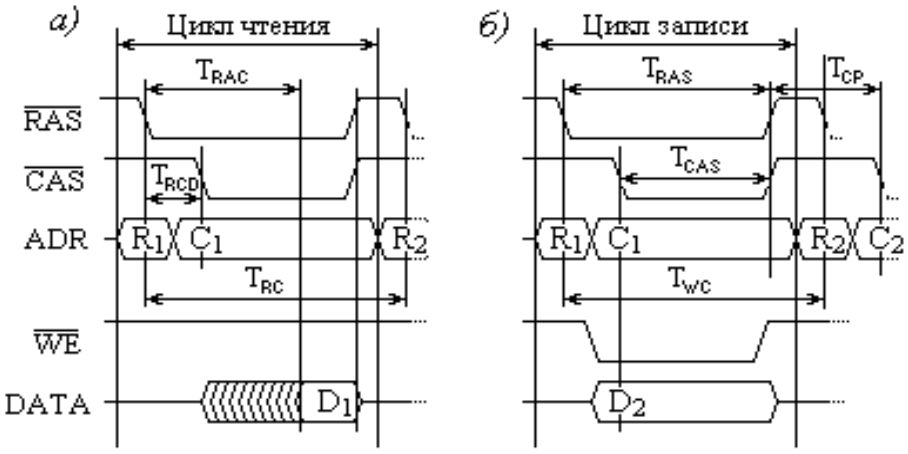


Рисунок 46. – Функциональная схема памяти произвольного доступа

Протоколы работы шины и временные диаграммы не являются стандартными и могут произвольно изменяться разработчиками памяти по согласованию с фирмами-производителями процессоров, т.к. это не отражается на составе команд языка ассемблера и, следовательно, на реализованном программном обеспечении ЭВМ.

Для получения представления о работе системной шины памяти ЦП в режимах чтения и записи ОЗУ синхронного типа рассмотрим временные диаграммы этих циклов (рисунок 47).



***а)* – чтение; *б)* – запись.**

Рисунок 47. – Временные диаграммы простых циклов синхронной динамической памяти

Можно видеть, что на диаграммах отсутствуют сигналы контроля готовности памяти и сигналы уведомления завершения операций. Режим чтения начинается сразу после выставления на шину дешифрованного адреса столбца, а длительность процесса чтения и записи определяется временем активации сигналов RAS и CAS, имеющих инверсную логику. Режим записи отличается от режима чтения только активацией сигнала сопровождения записи – WE, также имеющего инверсную логику.

В некоторых конструкциях контроллеров памяти в архитектуру ЭВМ введена возможность коррекции длительности сигналов RAS и CAS из программы-редактора констант BOIS (SETAP) для обеспечения лучшей совместимости ОЗУ различных производителей по частоте системной шины материнской платы.

Режимы пакетного доступа к ОЗУ, реализованный в архитектурах современных ПЭВМ для обеспечения работы КЭШ, незначительно отличается от рассмотренного выше. В отличие от одиночного обращения, временная диаграмма которого представлена на рисунке 47, этот режим имеет возможность чтения сразу нескольких блоков памяти для последующего размещения их в КЭШ (рисунок 48).

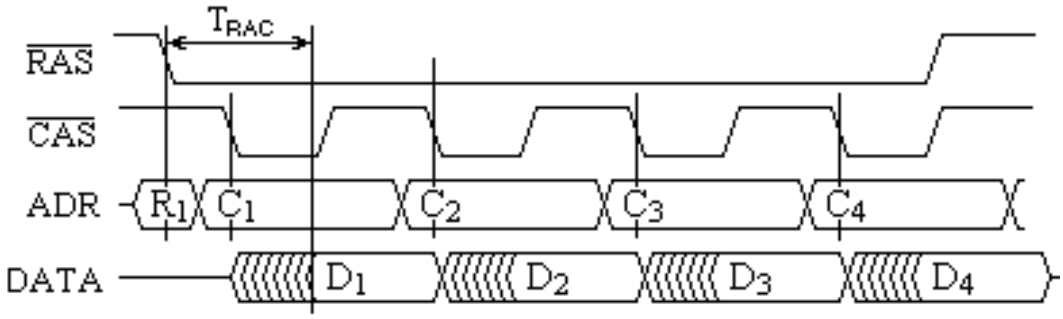


Рисунок 48. – Временная диаграмма чтения ОЗУ при пакетном доступе

На рисунке 48 представлены временные диаграммы циклов чтения синхронной динамической памяти произвольного доступа при пакетной обработке EDO SDRAM.

Можно видеть, что объем пакета, в данном случае, определяется количеством считываемых столбцов. Это соответствует естественной адресации операндов в ОЗУ для применяемой производителем памяти схемы дешифрации адреса.

Следует отметить, что режим пакетной записи в ОЗУ результатов вычислений в ЦП не предусмотрен, т.к. согласно этапам обработки команд в ЦП размещение результата вычислений осуществляется в одном из циклов по последнему вычисленному адресу операнда и не предусматривает формирования пакета.

Тема 6. Шинная архитектура ПЭВМ

6.1. Общие понятия и определения

Шина (магистраль или информационный канал) – это совокупность проводников (физических линий) данных, адреса и линий передачи сигналов управления, синхронизации и электропитания, а также протоколов обмена данными между устройствами, подключенными к шине.

Состав шины включает в себя:

– ***количество линий данных*** (определяет разрядность данных);

– количество линий адреса (определяет предельный объем адресуемого пространства оперативной памяти, доступный ЦП);

– ***частота шины*** (определяет её пропускную способность);

– количество линий сигналов управления, оповещения и синхронизации (формируют протокол обмена данными по шине).

Протокол работы шины – распределенная во времени строгая последовательность появления и исчезновения сигналов и данных на шине при осуществлении циклов обмена информацией.

Контроллер шины (арбитр шины) является устройством, связывающим центральный процессор с периферийными устройствами. Контроллер выполняет селекцию (арбитраж) информационного канала, т.е. обеспечивает однозначность выполнения процессов взаимодействия сопрягаемых элементов вычислительных систем посредством приоритетного разрешения внутриорганизационных конфликтов. Наличие конфликтов при доступе к информационному каналу является следствием взаимодействия параллельных процессов, протекающих в системе обработки информации.

При взаимодействии устройств, функционирующих параллельно во времени с общим информационным каналом, возможны конфликты двух уровней: при доступе устройства к информационному каналу интерфейса, а также при связи по шине одного устройства с другим. Первый уровень определяется занятостью информационного канала, и конфликт разрешается функцией селекции, второй – занятостью устройства, к которому происходит обращение, и разрешается при координации взаимодействия.

Задатчиками (хозяевами) шины могут выступать центральный процессор (обычная ситуация), контроллер ПДП, контроллер регенерации оперативной памяти и некоторые устройства расширения.

В каждом цикле обмена информацией по шине задатчиком всегда является только одно устройство. Например, контроллер ПДП захватывая магистраль, запрещает связь центрального процессора с шиной данных на время прямой передачи информации между устройством ввода/вывода и оперативной памятью. Контроллер регенерации оперативной памяти периодически становится задатчиком системной шины для проведения циклов регенерации динамической памяти через заданные интервалы времени.

Шинный интерфейс – это совокупность аппаратных программно поддерживаемых средств, обеспечивающих совместимость всех устройств ЭВМ (процессора, памяти, устройств ввода-вывода), подключаемых к данной шине и объединяемых в единую вычислительную систему.

6.2. Схема формирования шинного интерфейса ЭВМ

Любое внешнее устройство, предназначенное для установки в разъёмы расширений шины, должно содержать, как минимум, 2 регистра: регистр данных (буфер данных) и регистр признаков (сигналов запроса-готовности), который связан с линией передачи сигналов. Внешние устройства работают, обычно, существенно медленнее центрального процессора, т.е. асинхронно по отношению к нему, поэтому имеют в своем составе собственный генератор синхронизации. Готовность к приёму/передаче данных должна подтверждаться соответствующими сигналами готовности. ЦП или вспомогательные схемы контроллера шины производят циклический опрос регистров состояния внешних устройств и, при приеме сигнала готовности, обеспечивают подключение к шине через дешифратор номера этого устройства. Схема типового шинного интерфейса представлена на рисунке 49.

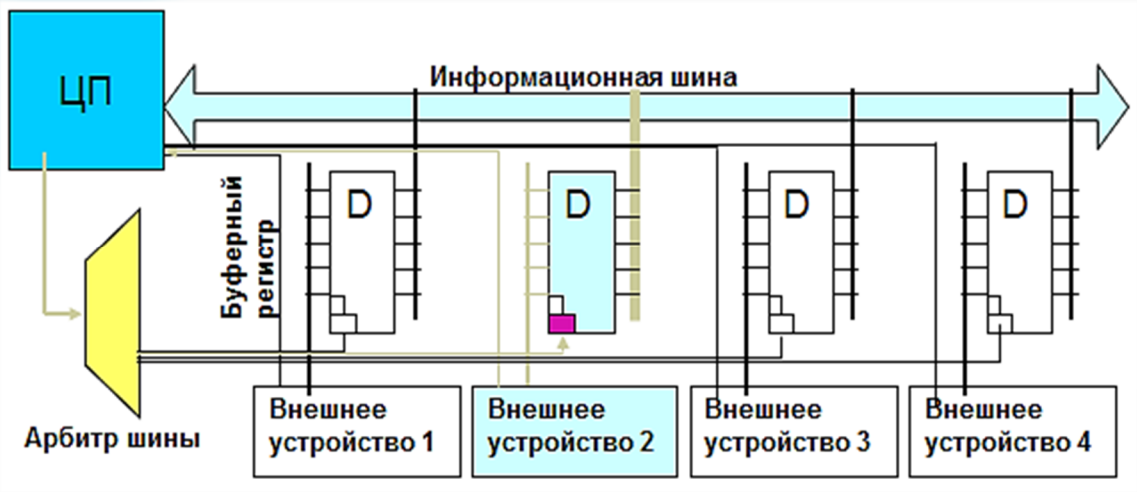


Рисунок 49. – Схема шинного интерфейса ЭВМ

Шинный интерфейс предусматривает не только электрическое (гальваническое) соединение устройств с ЦП, но и накладывает ряд дополнительных условий для обеспечения процесса передачи информации по шине. Эти условия определяются протоколом работы шины и представляются в виде временных диаграмм, таких, например, как для обеспечения работы оперативной памяти (см. рисунки 47 и 48.).

На рисунке 50 для примера приведена временная диаграмма протокола передачи данных по заданному адресу памяти по шине ISA.

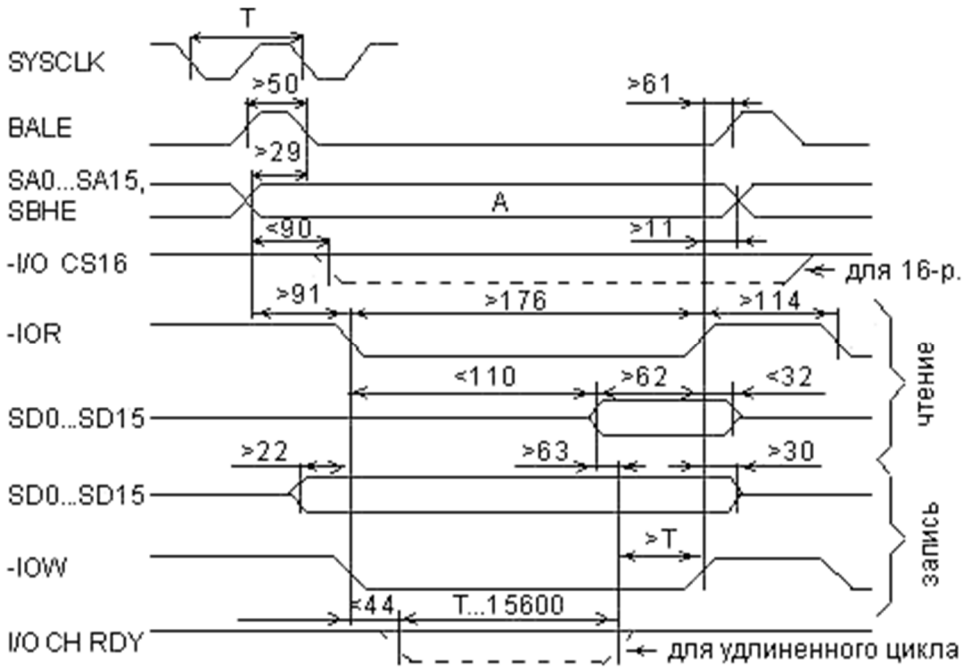


Рисунок 50. – Временная диаграмма приема/передачи данных по шине ISA

Здесь можно видеть, насколько сложен протокол передачи данных по шине. Протоколы передачи информации по шинам, использующимся в современных ЭВМ, существенно сложнее. 16-разрядная шина ISA была введена в архитектуру x86 с момента создания этой архитектуры и является наследуемой, т.е. обладающей преемственностью своих основных свойств в процессе совершенствования архитектуры x86.

6.3. Мостовой контроллер

Современное состояние архитектуры x86-64 характеризуется наличием в своем составе достаточно большого количества шин различной разрядности и частоты. Это, в свою очередь, требует введения в схему архитектуры дополнительных устройств, предназначенных для обеспечения связи между ними (не только гальванической, но и по протоколам передачи информации). Задачу передачи информации между различными шинами решают мостовые контроллеры. Упрощенная функциональная схема мостового контроллера, предназначенного для связи шин с различной разрядностью данных и мультиплексированной шины, представлена на рисунке 51.

Цель работы мостового контроллера заключается не только в согласовании протоколов и частоты работы шин, но и в требуемом изменении формата данных. Так, например, для изменения формата данных 16-разрядной шины ISA на 32-разрядный формат шины PCI необходимо принять сначала младшее слово и поместить его во внутренний регистр, а затем старшее слово. После этого выполняется заполнение буферного регистра 32-разрядной шины и передача этих двух слов одновременно в виде 32-разрядного операнда.

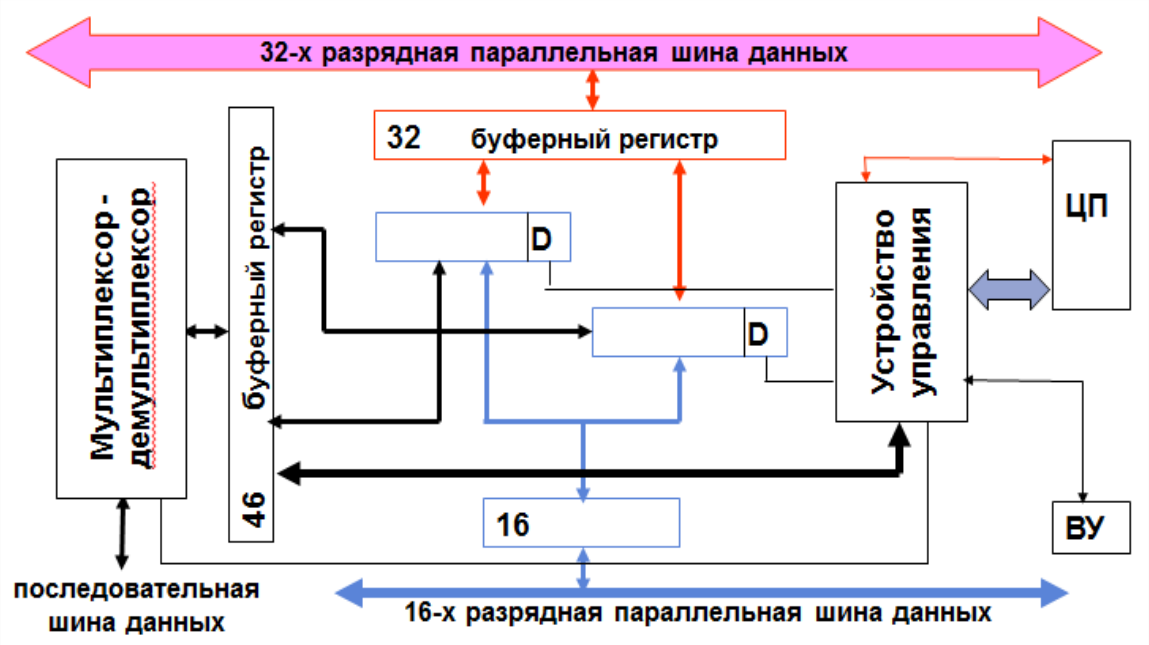


Рисунок 51. – Функциональная схема мостового контроллера

Преобразование 32-разрядного операнда в два 16-разрядных осуществляется в обратной последовательности.

Преобразование последовательного кода, передаваемого по последовательной шине (например, шине USB), осуществляется с использованием мультиплексора. При этом часть параллельного кода интерпретируется как передаваемый операнд, а часть – как набор управляющих сигналов, сопровождающих процесс передачи информации согласно протоколу работы шины и передаваемых в устройство управления мостового контроллера.

6.4. Схема наследуемой шинной архитектуры x86

Схема шинной архитектуры ПЭВМ на базе процессора i8086 была сформирована с использованием шины ISA, к которой были подключены все внутренние и внешние устройства ПЭВМ, за исключением контроллера памяти, который уже тогда имел свою системную шину для связи с ЦП и контроллером прямого доступа к памяти (рисунок 52). Такая шинная архитектура ПЭВМ использовалась для процессоров производства фирмы Intel i8086, i80286, i80386 и их клонов. Основным её преимуществом была простота и открытость платформы x86 для всех разработчиков расширений вычислительной системы подобного типа.

На рисунок 53 показана одна из последних реализаций такой архитектуры, построенная на базе центрального процессора i80386.

Дальнейшее совершенствование, как центральных процессоров, так и шин сопровождалось существенным усложнением структурной схемы ПЭВМ.

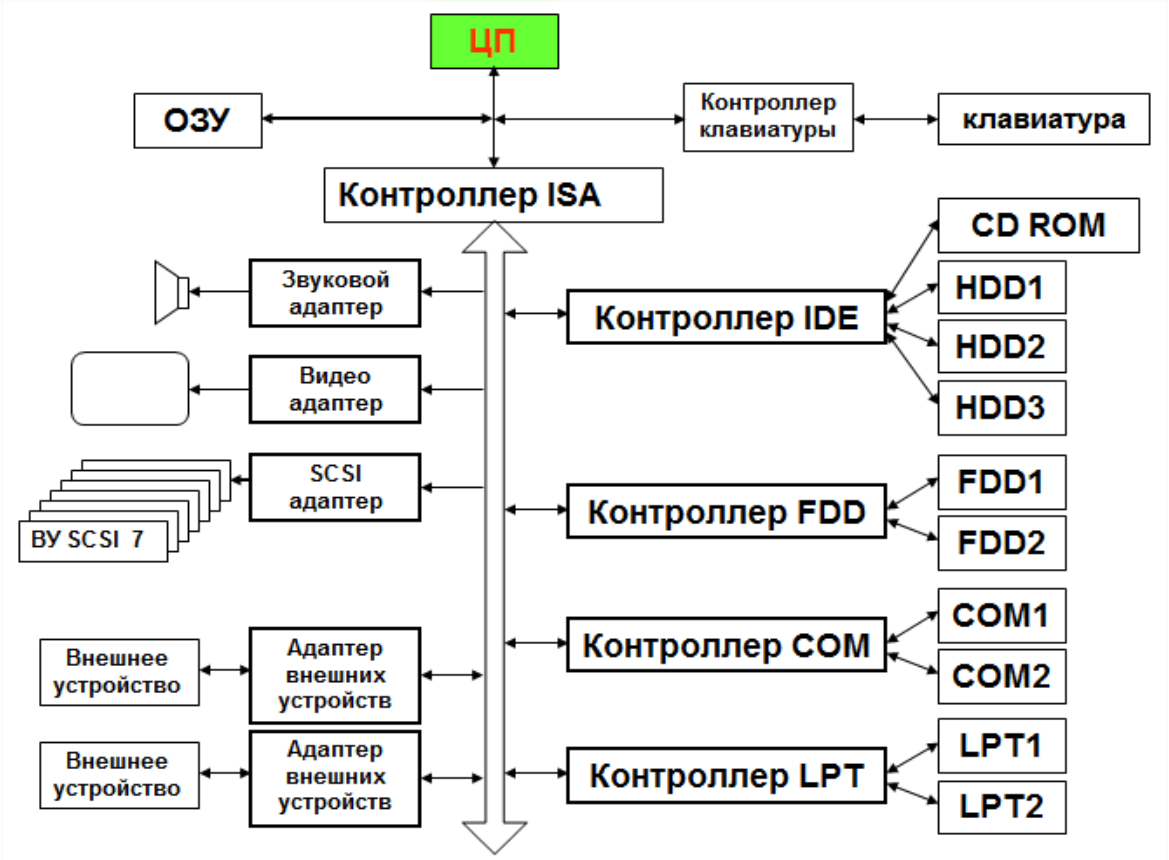


Рисунок 52. – Схема наследуемой шинной архитектуры ПЭВМ на базе центрального процессора x86

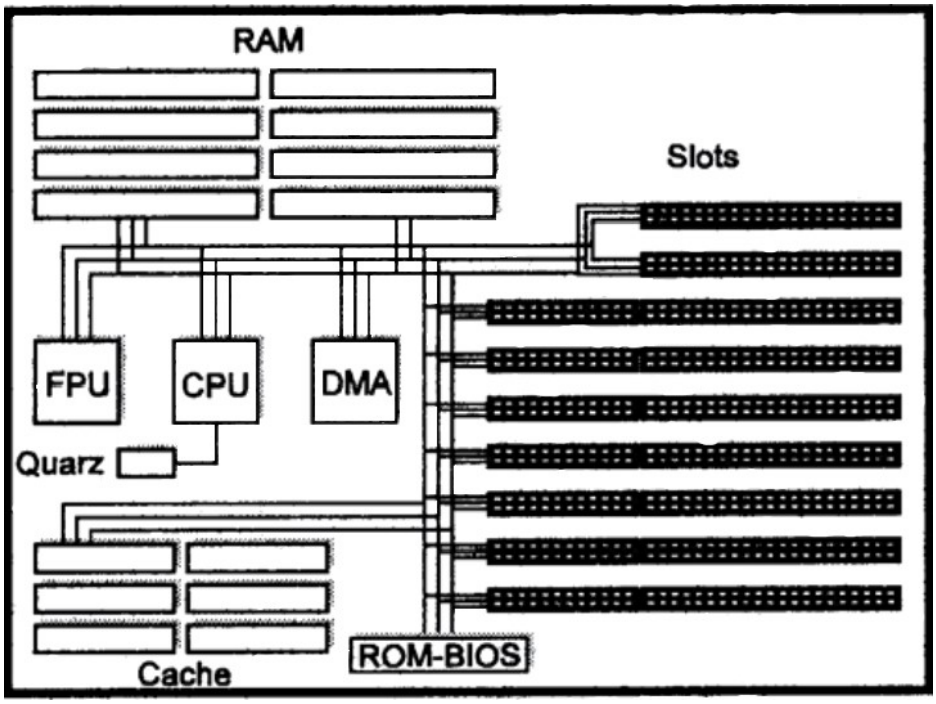


Рисунок 53. – Схема и внешний вид системной платы с наследуемой шинной архитектурой ISA центрального процессора i80386 и математического сопроцессора i80387 (отсутствует в слоте). Начало. Продолжение – С. 88

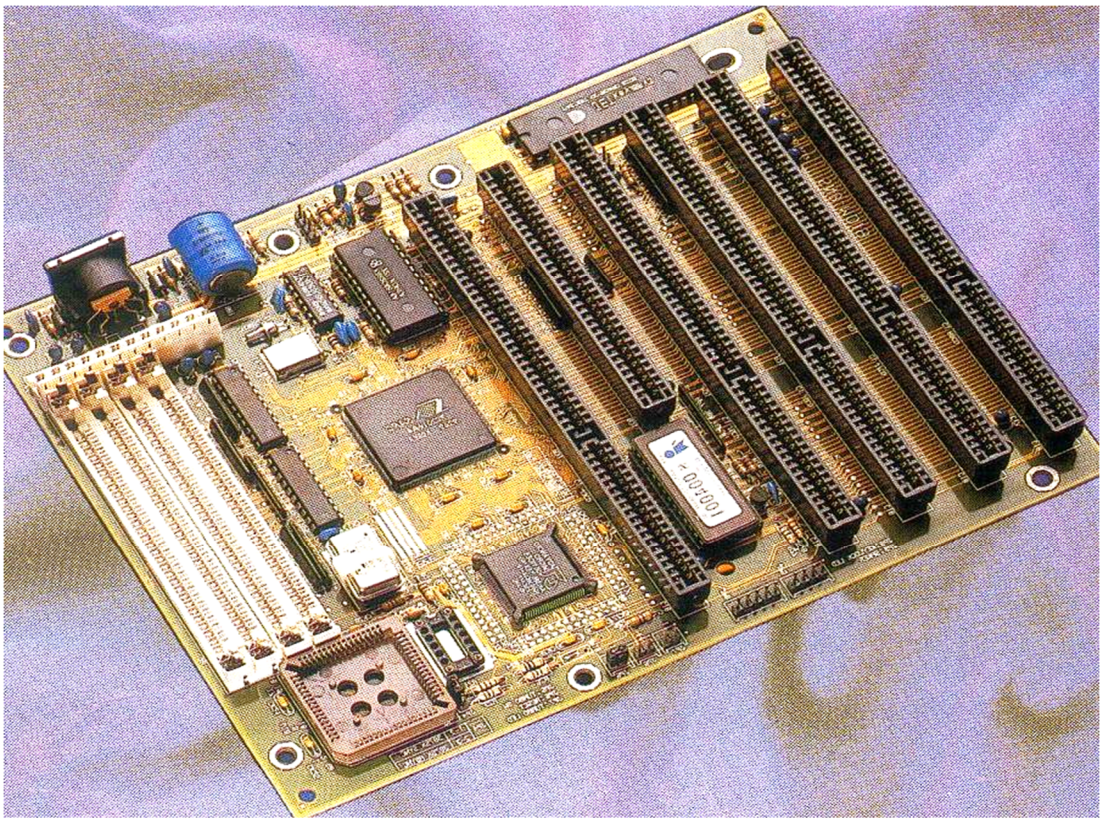


Рисунок 53. – Окончание

В таблице 4 показана динамика изменения характеристик основных системных шин, используемых в архитектуре x86.

Таблица 4. – Технические характеристики основных системных шин

Тип шины	Разрядность, бит	Тактовая частота, мГц	Пропускная способность, мБ/с
ISA	16	8,33	16,6
PCI	32	33	133,3
AGP	32	66	266,6
PCI 2.x	64	66	533,3
AGP x4	32	66 x4	1066,6
PCI-x1.0	64	100	1024
PCI Express x4	mult	1000	1600
PCI Express x16	mult	1000	до 8000

Здесь можно видеть существенное изменение разрядности и пропускной способности системных шин. Разрядность современных шин соответствует архитектуре x86-64. Современная шинная архитектура, функциональная схема которой представлена на рисунке 54, все более приближается к радиальной схеме подключения устройств расширения.

Интерфейс шины PCI сохраняется для обеспечения возможности установки устройств, спроектированных под эту архитектуру.

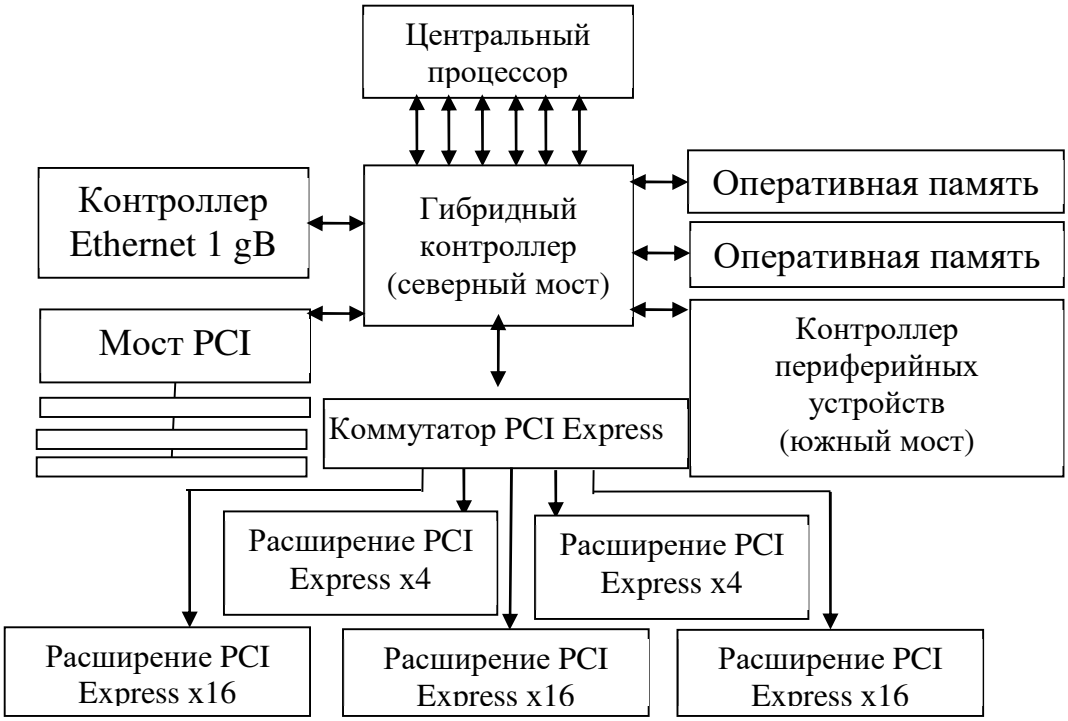


Рисунок 54. – Функциональная схема современной шинной архитектуры

Многоразрядность центрального процессора не повлияла на схему шинной архитектуры. Обращение центрального процессора к оперативной памяти осуществляется по специализированной многоразрядной системной шине параллельного типа (FSB/QPB у фирмы Intel и выделенная системная шина памяти у процессоров AMD). Обращение к памяти выполняется через общий контроллер памяти поочередно каждым ядром процессора на равноправной основе, а сам объем оперативной памяти разбит на два банка для обеспечения технологии суперскалярности.

Следует отметить, что при использовании двух центральных процессоров, что практикуется на материнских платах ЭВМ-серверов локальных вычислительных сетей, каждый процессор обеспечивается своим собственным объемом оперативной памяти, отдельным контроллером памяти и системной шиной.

Отказ от наследуемой шинной архитектуры позволил производителям системных плат разместить на той же площади гораздо больше системных устройств, повысив скорость обработки информации на несколько порядков. Последующее совершенствование шинной архитектуры, вероятно, заставит полностью отказаться от использования параллельных шин типа PCI, IDE (PATA), SCSI в связи с их громоздкостью аппаратного обеспечения и сложными протоколами обмена данными, заменив их последовательными шинами. Реализуемые при этом промежуточными контроллерами схемы соединений типа «точка–точка» позволяют лучше обеспечить целостность данных, т.к. уже не потребуются многочисленные преобразования их форматов, а также сокращается время передачи.

Тема 7. Подсистема прерываний ПЭВМ

7.1. Основные задачи прерывания исполнения программ

Цель прерывания программы – остановка текущей программы в процессе её исполнения для запуска другой программы, либо обработки какого-то события или ошибки исполнения текущей программы.

Основные задачи обработки прерывания:

1. Регистрация всех запросов на прерывание текущей программы.
2. Определения готовности и приоритетов устройств, требующих прерывания текущей программы.
3. Формирование адреса точки входа в программу-обработчик прерывания и инициализация процесса обработки прерывания с сохранением состояния текущей (прерванной) программы.
4. Восстановление исходного состояния прерванной программы и продолжение её выполнения после завершения обработки прерывания.

7.2. Механизм прерываний исполнения программ

В общих чертах механизм прерывания выполняемой программы с целью обработки прерывания достаточно прост (рисунок 55).

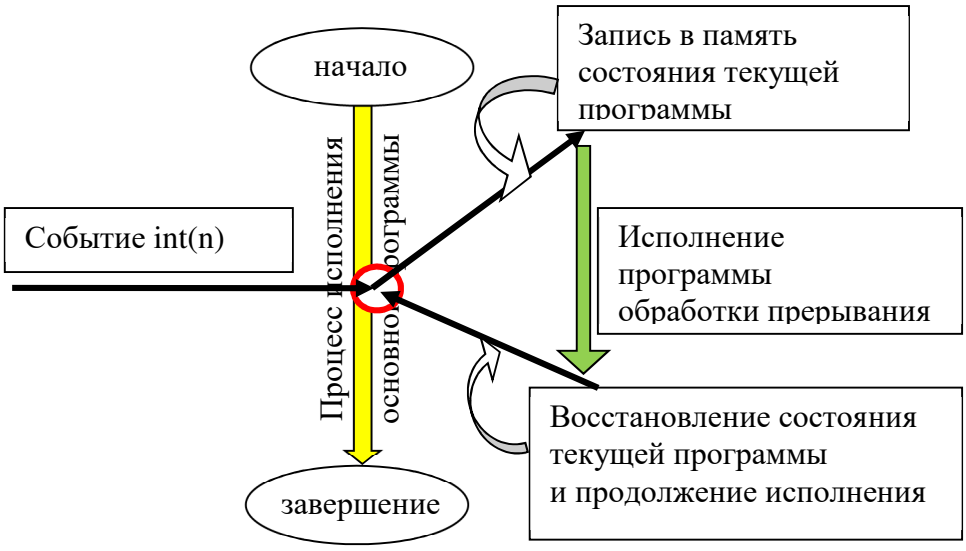


Рисунок 55. – Схема механизма прерывания исполнения программы

В архитектуре x86 при реализации прерывания программы была предусмотрена следующая последовательность действий:

1. Сигнал IRQn поступал от устройства (например, контроллера клавиатуры при нажатии клавиши), требующего прерывания программы и обработки его состояния. Сигнал поступал на вход шифратора контроллера прерываний.

2. Полученный шифр представлял собой младшую часть адреса указателя точки входа в программу обработки прерывания. Одновременно контроллер передавал ЦП сигнал запроса на прерывание INT.

3. ЦП по сигналу приостанавливал выполнение текущей программы, записывал в СТЭК два слова «CS:IP» и «FLAGS», определяющие точку останова программы (сегменты программы остаются неизменными), и генерировал сигнал INTE (разрешение обработки прерывания).

4. Контроллер прерывания по этому сигналу выставлял на шину адреса сформированный адрес 00:IP вектора прерывания, содержащего линейный адрес точки входа в программу-обработчик прерывания (в нашем примере – программу BIOS «Xlat»).

5. Выполнялась программа-обработчик прерывания (в нашем примере на экране монитора появляется изображение символа нажатой клавиши).

6. ЦП считывал из СТЭКа значения CS:IP, FLAGS и продолжал выполнение прерванной программы.

На начальных этапах развития архитектуры x86 было предусмотрено только два источника прерываний: аппаратные (или радиальные, т.к. имели звездообразную топологию с единым центром – контроллером прерываний) и программные (векторные), которые фактически были обычными процедурами и обращались напрямую к центральному процессору, используя систему векторов.

7.3. Типы прерываний в реальном режиме адресации

Существует два типа прерываний:

1. ***Аппаратные (радиальные) прерывания*** инициируются аппаратурой периферии либо с системной платы или платы расширения. Они могут быть вызваны сигналом микросхемы таймера, сигналом от принтера, нажатием клавиши на клавиатуре и множеством других причин.

Аппаратные прерывания не связываются с работой программного обеспечения, но обязательно имеют соответствующие программы-обработчики, которые инициируются векторами прерываний, связанными с аппаратными прерываниями через контроллер прерываний.

2. ***Программные (векторные) прерывания*** на самом деле ничего не прерывают. Это обычные процедуры или функции, которые вызываются прикладными программами для выполнения рутинной работы, такой как прием нажатия клавиши на клавиатуре или вывод на экран. Эти программы находятся в составе программ операционной или базовой систем ввода/вывода, и механизм прерываний дает возможность обратиться к ним.

Программные прерывания могут вызываться друг из друга. При этом не возникает конфликтов, так как каждая подпрограмма обработки преры-

вания сохраняет значения всех используемых ею регистров и затем восстанавливает их при выходе, тем самым, не оставляя следов того, что она занимала центральный процессор.

Адреса точек входа в программы прерываний называют векторами. Каждый вектор имеет длину 4 байта. В первом слове хранится значение IP, а во втором – CS. Совокупность векторов составляет таблицу векторов.

Источниками сигналов в системе радиальных прерываний в порядке их приоритета являются:

IRQ 0 – системный интервальный таймер } основной
IRQ 1 – клавиатура } контроллер
IRQ 2 – канал ввода/вывода 2-го каскада } прерываний

IRQ 8 – часы реального времени CMOS } каскадируемый контроллер прерываний
IRQ 9 – свободный, PnP
IRQ 10 – свободный, PnP
IRQ 11 – свободный, PnP
IRQ 12 – мышь PS/2
IRQ 13 – математический сопроцессор
IRQ 14 – контроллер ATA
IRQ 15 – свободный, PnP

IRQ 3 – COM1 (COM2) } основной контроллер прерываний
IRQ 4 – COM2 (COM1)
IRQ 5 – свободный, PnP
IRQ 6 – контроллер FDD
IRQ 7 – LPT1

Линии, связанные со стандартными источниками прерываний (показаны жирным шрифтом), не могут быть перепрограммированы или переназначены в системе PnP, т.к. это может привести к системным сбоям в работе ПЭВМ.

7.4. Программируемый контроллер прерываний i8259A

В первых персональных компьютерах фирмы IBM применялась только одна микросхема программируемого контроллера прерываний 8259 (советский аналог К1810ВН59). Она имела восемь входных линий запросов прерываний (IRQ0 – IRQ7) и была рассчитана на работу как с процессором 8080, так и 8086 фирмы Intel. Выбор конкретного типа процессора выполнялся при инициализации контроллера.

Создавая ПЭВМ IBM PC/AT, фирма IBM решила удвоить количество линий запросов прерываний и применила два контроллера типа 8259, соединив их каскадно. Ведущий контроллер прерываний обслуживал запросы прерываний по линиям IRQ0 – IRQ7, ведомый – по линиям IRQ8 – IRQ15. Выходная линия запроса прерывания ведомого контроллера поступала на одну

из входных линий (IRQ2) ведущего; для подключения внешних устройств использовались все восемь входных линий ведомого контроллера и семь (IRQ0, IRQ1 и IRQ3–IRQ7) ведущего, что в сумме составляет 15 линий.

Такая схема обработки прерывания стала стандартной для архитектуры x86. Однако она имеет ряд недостатков, важнейшими из которых являются нехватка линий запросов прерываний и неспособность работать в многопроцессорной системе. Этот недостаток стал особенно актуальным с появлением многоядерных процессоров.

Перечисленные проблемы были решены в усовершенствованном программируемом контроллере прерываний APIC (Advanced Programmable Interrupt Controller), который теоретически может поддерживать 256 прерываний (реально только 24) и входит составной частью в каждое ядро центрального процессора. В целях совместимости традиционный контроллер прерываний по-прежнему поддерживается и используется старыми операционными системами, не умеющими работать с APIC.

Следует заметить, что современные варианты контроллеров прерываний, входящие, как правило, в микросхему южного моста чипсета системной платы, поддерживают лишь часть возможных режимов исходной пары контроллеров 8259. Так, по очевидным причинам, выброшена возможность работы с микропроцессором 8080.

В последующих разделах будут описаны только те возможности контроллера прерываний, которые присутствуют в его современных реализациях.

Общая функциональная схема контроллера прерываний i8259A представлена на рисунке 56.

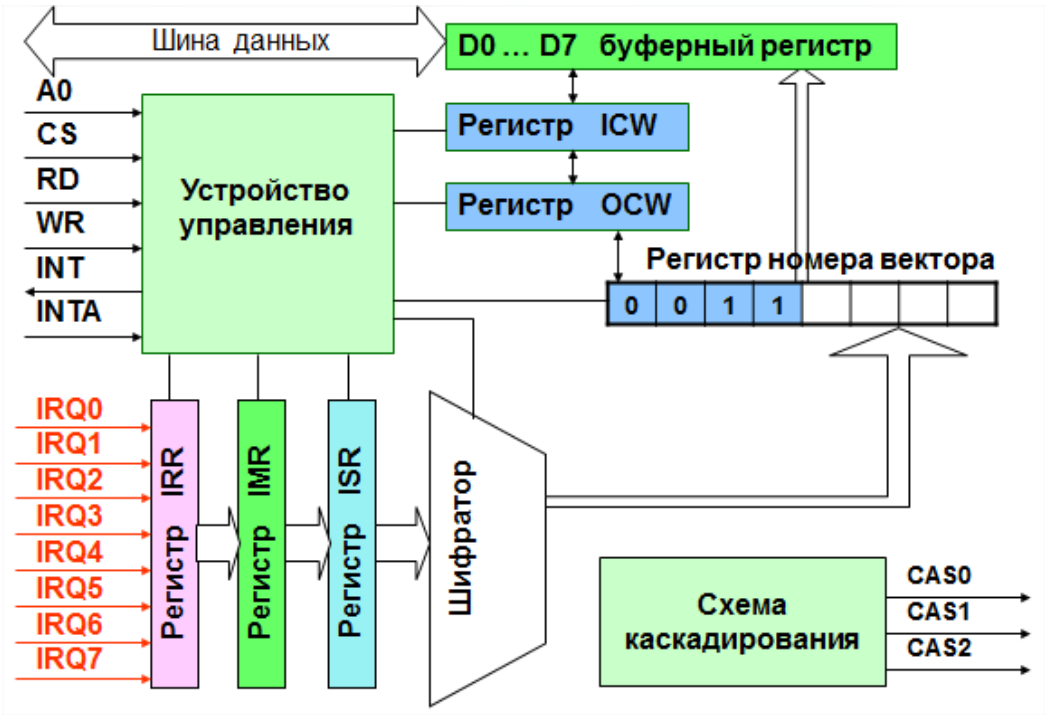


Рисунок 56. – Функциональная схема контроллера прерываний

Запросы от источников прерываний поступают по восьми линиям IRQ, причем в зависимости от установленного режима запросом может считаться либо наличие на линии IRQ высокого уровня, либо положительный перепад сигнала на этой линии (т.е. переход с низкого уровня на высокий).

Обнаруженные запросы запоминаются в регистре запросов прерываний IRR (Interrupt Request Register). Каждый бит этого регистра соответствует одной линии IRQ. Бит IRR равен 1, если был получен запрос от соответствующей линии, и 0, если запроса по этой линии не поступало. Отдельные линии IRQ могут быть замаскированы с помощью регистра маскирования прерываний IMR (Interrupt Mask Register). Установка какого-либо бита этого регистра маскирует запросы по соответствующей линии IRQ: они принимаются и фиксируются в IRR обычным образом, но их обработка не выполняется до тех пор, пока соответствующий бит в IMR не будет сброшен.

Третьим и последним входным регистром контроллера является регистр обслуживаемых прерываний ISR (Interrupt Service Register). Когда контроллер прерываний обрабатывает прерывание и выдает процессору его вектор, соответствующий ему бит устанавливается в ISR, а в IRR – сбрасывается. Установка некоторого разряда ISR в общем случае блокирует обработку новых прерываний, поступающих по той же линии IRQ, а также всех прерываний с меньшим приоритетом. Сброс разрядов ISR выполняется по командам процессора, которыми он уведомляет контроллер прерываний, что обработка того или иного прерывания завершена.

Таким образом, IRR показывает какие прерывания ожидают обработки, а ISR – какие обрабатываются в настоящий момент.

Кроме этих регистров в схеме операционного блока контроллера имеются регистры инициализации ICW и операционного управления OCW, а также регистр формирования номера вектора прерывания. Для связи ЦП с контроллером используются соответствующие портовые регистры 20h/21h (A0h/A1h для ведомого контроллера). В качестве устройства управления используется цифровой автомат с жесткой (схемной) логикой.

Основные функции контроллера:

1. Фиксация запросов на прерывания от восьми внешних источников.
2. Программное маскирование поступающих запросов.
3. Присвоение фиксированных или циклически изменяемых приоритетов входам контроллера, на которые поступают запросы.
4. Определение адреса и инициация вызова процедуры обработки поступившего аппаратного прерывания.

7.5. Порядок программирования контроллера прерываний

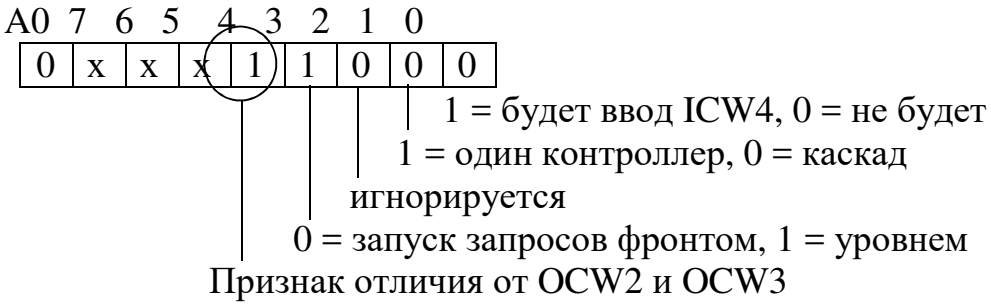
ПКП может находиться в двух основных состояниях: настройки и обслуживания запросов на прерывания. В состоянии настройки контроллер принимает управляющие слова инициализации (Initialization Command Words, ICW), в состоянии обслуживания – операционные управляющие слова (Operation Control Words, OCW).

Для ввода информации в ПКП используются 2 порта ввода-вывода: порт с четным адресом (20h/A0h) и порт с нечетным адресом (21h/A1h). Через эти порты могут быть переданы 4 слова инициализации (ICW1 – ICW4), задающие режим работы ПКП, и 3 операционных управляющих слова (слова рабочих приказов OCW1 – OCW3).

В порт с четным адресом выводятся ICW1, OCW2 и OCW3, а в порт с нечетным адресом – ICW2, ICW3, ICW4 и OCW1.

Формат слов инициализации:

Формат ICW1 – начальное слово в порт с четным адресом.



Формат ICW2 – указание величины смещения.

A0	7	6	5	4	3	2	1	0
1	0	0	0	0	1	0	0	0

Величина смещения номера вектора от номера аппаратного прерывания – 8 (для ведущего контроллера), 112 (для ведомого контроллера).

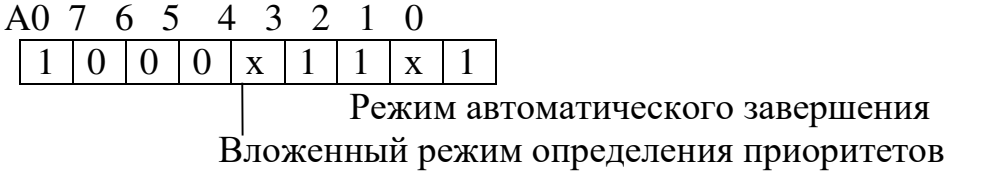
Формат ICW3 – слово признака каскадирования.

A0	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	0	0

Для ведущего – к какому входу IRQ подключен ведомый контроллер.

Для ведомого – уровень приоритетов относительно ведущего.

Формат ICW4 – слово завершения инициализации.



В процессе работы ПКП может быть перепрограммирован:

– допускается маскировать и демаскировать аппаратные прерывания;

– допускается изменять приоритеты уровней;

– можно давать команду принудительного завершения обработки аппаратного прерывания;

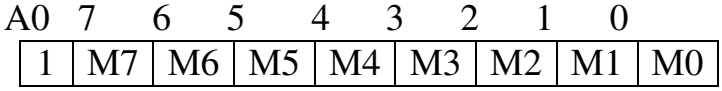
– допускается устанавливать/сбрасывать режим специальной маски;

– также предусмотрен перевод контроллера в режим опроса и считывания регистров ISR и IRR.

Для перепрограммирования контроллера в процессе работы используется одно из слов рабочих приказов OCW1 – OCW3.

Форматы слов рабочих приказов:

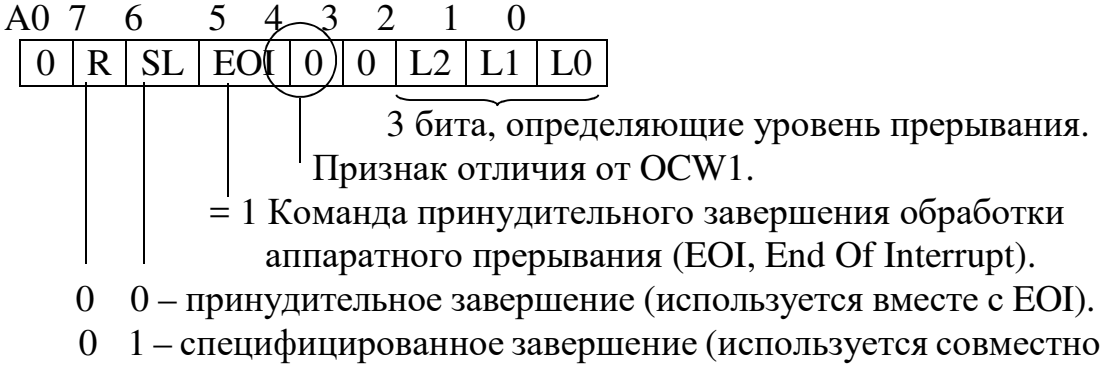
Формат OCW1 – слово маскирования прерываний.



= 1 одного из битов M0 – M7 означает, что прерывания соответствующего уровня IRQ_n маскируются и не будут обрабатываться контроллером прерываний.

Четвертый бит в OCW1 не является признаком четности порта, т.к. вводится в порт с нечетным номером (используются все биты байта).

Формат OCW2 – порядок завершения прерывания.



Процедура обработки аппаратного прерывания должна перед своим завершением очистить свой бит в ISR выводом команды завершения обработки прерывания.

Существует два варианта команды EOI: обычный и специфицированный. Обычный EOI очищает бит в ISR, соответствующий прерыванию с максимальным приоритетом. Специфицированный EOI (R = 0, SL = 1, EOI = 1, L0 – L2 равно номеру уровня прерывания) очищает в ISR бит, соответствующий прерыванию с номером, указанным в L0 – L2 независимо от его приоритета.

Формат OCW3 – изменение режима или разрешение опроса.

Единичное значение бита P (бит опроса, Polling Bit) переводит контроллер в режим опроса. Если после этого считать данные из порта с четным адресом, получим байт следующего содержания:

I	0	0	0	0	L2	L1	L0

Если здесь бит I = 1, значит имеются запросы на аппаратные прерывания и тогда L0 – L2 – это номер уровня с наивысшим приоритетом, по которому имеется запрос. Если P = 0, можно считать информацию из ISR или IRR. Для этого необходимо дать команду чтения ISR или IRR и затем считать значение из порта контроллера с нечетным адресом (21h или A1h).

Система приказов BIOS для инициализации контроллера:

Код приказа	Вид приказа и уровень контроллера-приемника	№ порта
00010001	ICW1 в контроллер 1 – будет указан тип процессора	020h
00001000	ICW2 в контроллер 1 – величина смещения на 8h	021h
00000100	ICW3 в контроллер 1 – определяет ведущий контр-р	021h
00000001	ICW4 в контроллер 1 – архитектуры процессора - x86	021h
00010001	ICW1 в контроллер 2 – будет указан тип архитектуры	0A1h
01110000	ICW2 в контроллер 2 – величина смещения вектора на 70h	0A1h
00000010	ICW3 в контроллер 2 – указывает на ведомого в каскаде	0A1h
00000001	ICW4 в контроллер 2 – тип архитектуры процессора - x86	0A1h
10111000	В контроллер 1 – маска прерывания OCW1	021h
10111101	В контроллер 2 – маска прерывания OCW2	0A1h

Программирование контроллера прерываний имеет смысл производить только для режима прямой адресации (режим 8086), т.к. в защищенном режиме этот контроллер выполняет вспомогательные функции для обеспечения преемственности программного обеспечения прикладных задач, а таблица векторов прерываний не используется.

7.6. Режимы использования контроллера прерываний

7.6.1. Режимы формирования приоритетов

1. ***Режим фиксированных приоритетов.*** Запросы прерываний имеют жесткие приоритеты от 0 до 7 (0 – высший) и обрабатываются в соответствии с приоритетами. При этом прерывание с меньшим приоритетом никогда не будет обработано, если в процессе обработки прерываний с более высокими приоритетами постоянно возникают запросы на эти прерывания.

2. ***Автоматический сдвиг приоритетов.*** В этом режиме дается возможность обработать прерывания всех уровней без их дискриминации. Например, после обработки прерывания уровня 4 ему автоматически присваивается низший приоритет, при этом приоритеты для всех остальных уровней циклически сдвигаются.

3. ***Программно-управляемый сдвиг приоритетов.*** Программист может сам передать команду циклического сдвига приоритетов в контроллере, задав соответствующее управляющее слово. В команде задается номер уровня, которому требуется присвоить максимальный приоритет. После выполнения такой команды устройство работает так же, как и в режиме фиксированных приоритетов, с учетом их сдвига. При этом приоритеты сдвигаются циклически, таким образом, что если максимальный приоритет был назначен уровню 3, то уровень 2 получит минимальный и будет обрабатываться последним.

4. ***Режим специальной маски.*** Данный режим позволяет отменить приоритетное упорядочение обработки запросов и обрабатывать их по мере поступления. После отмены режима специальной маски предшествующий порядок приоритетов уровней сохраняется.

7.6.2 Режимы завершения прерываний

1. ***Режим автоматического завершения обработки прерывания (AEOI).*** В обычном режиме работы процедура обработки аппаратного прерывания должна перед своим завершением очистить свой бит в регистре ISR специальной командой, иначе новые прерывания не будут обрабатываться контроллером. В режиме AEOI нужный бит в ISR автоматически сбрасывается в тот момент, когда начинается обработка прерывания нужной процедурой обработки и от неё не требуется издавать команду завершения обработки прерывания (EOI).

Сложность работы в данном режиме обусловливается тем, что все процедуры обработки аппаратных прерываний должны быть повторно входными, т.к. за время их работы могут повторно возникнуть прерывания того же уровня.

2. ***Режим опроса (Polling Mode).*** В этом режиме аппаратные прерывания не происходят автоматически. Появление запросов на прерывание должно определяться считыванием IRR. Данный режим позволяет также получить от ПКП информацию о наличии запросов на прерывания и, если запросы имеются, номер уровня с максимальным приоритетом, по которому есть запрос.

7.7. Обработка прерываний в защищенном режиме

7.7.1. Исключения и прерывания

С разработкой нового контроллера прерываний APIC, входящего в состав операционного блока каждого ядра центрального процессора, появился новый класс источников прерываний программ – исключений, а также дополнительный класс немаскируемых прерываний с номерами IRQ 16 и более (до 255).

Исключения (особые ситуации) генерируются ядром центрального процессора как следствие исполнения очередной команды и представляют собой принудительную передачу управления задаче или процедуре обработки особых случаев состояния процессора или аппаратной части ПЭВМ, появляющихся в результате завершения этой команды.

Исключения обслуживают условия, обнаруживаемые процессором во время выполнения команд, например, деление на 0 или обращение к памяти с нарушением привилегий, обращение к несуществующей странице памяти и т.д. Существует два источника исключений:

– исключения, обнаруживаемые процессором, классифицируются как сбой, ловушка или аварийное завершение;

– программируемые исключения или остановы ЦП при отладке программы.

Команды INT0, INT3, INTn и BOUND могут служить программными переключателями исключений. Эти команды часто называют «программными прерываниями», но они обрабатываются процессором как программируемые исключения.

Основные типы исключений:

– ***Сбой (ошибка)*** – состояние исключения, при возникновении которого в стек записываются значения регистров cs:ip, указывающие на команду, вызвавшую данное прерывание. Это позволяет системе контроля ошибок, получив доступ к сегменту кода, исправить ошибочную команду в обработчике прерывания и, вернув управление программе, фактически осуществить её рестарт (в режиме прямой адресации при возникновении прерывания в стеке всегда запоминается адрес команды, следующей за той, которая вызвала прерывание).

– ***Ловушка*** – прерывание или исключение, при возникновении которого в стек записываются значения регистров cs:ip, указывающие на команду, следующую за командой, вызвавшей данное прерывание.

Здесь так же, как и в случае ошибок возможен рестарт программы. Для этого необходимо лишь исправить в обработчике прерывания соответствующие код или данные, послужившие источником ошибки. После этого перед возвратом управления нужно скорректировать значение ip в стеке на длину команды, вызвавшей данное прерывание.

Механизм ловушек похож на механизм прерываний в реальном режиме, за исключением случая, когда прерывание типа ловушки возникло в команде передачи управления jmp. Тогда содержимое пары cs:ip в стеке будет отражать результат этого перехода, то есть соответствовать команде назначения.

– ***Аварийное завершение*** – прерывание, при котором информация о месте его возникновения недоступна или неполна и поэтому рестарт команды практически невозможен, если только данная ситуация не была запланирована заранее созданием точки останова программы в режиме отладки.

Центральный процессор должен однозначно определять, какие прерывания являются ошибками, ловушками и авариями, а также возможное наложение аппаратных прерываний на прерывания защищенного режима. Для этой цели в пересекающихся прерываниях дополнительно генерируется и записывается в стек так называемый код ошибки.

Аппаратные (радиальные) прерывания, как и в режиме 8086, происходят в произвольные моменты времени выполнения программы в ответ на сигналы аппаратного обеспечения. Аппаратные прерывания используются для обработки событий, внешних по отношению к процессору, таких как запросы на обслуживание периферийных устройств, например, нажатия клавиши на клавиатуре или появления сигнала часов CMOS.

Существует два источника аппаратных прерываний:

– Маскируемые прерывания, получаемые на входе внутреннего контроллера прерываний APIC каждого ядра центрального процессора одновременно от стандартного программируемого контроллера прерываний i8259.

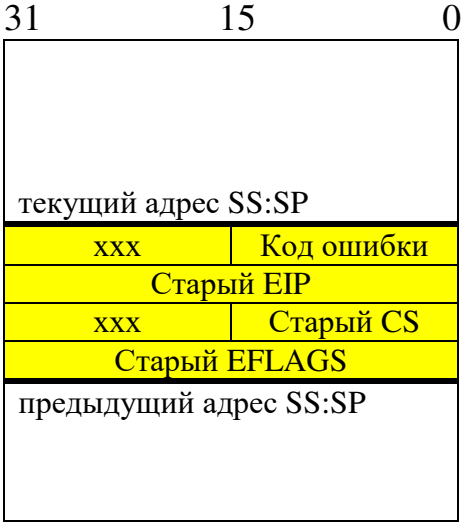
Но маскируемые прерывания не обрабатываются до тех пор, пока не будет установлен флаг разрешения прерываний (IF) в регистре состояния центрального процессора EFLAGS.

– Немаскируемые аппаратные прерывания, получаемые на вход внутреннего контроллера прерываний APIC каждого ядра процессора с номерами IRQ16 и более. Центральный процессор не содержит механизма отключения таких аппаратных прерываний и поэтому они обрабатываются всегда.

7.7.2 Схема обработки прерывания в защищенном режиме

При выполнении прерывания текущей программы вне зависимости от источника (исключение или прерывание) текущая точка остановки программы, как и в режиме прямой адресации, сохраняется в стеке.

Содержание информации, сохраняемой в стеке в защищенном режиме, дополняется словом, содержащим возможный код ошибки:



Содержание битов кода ошибки:

Бит 0 =1 – источник аппаратное прерывание,
= 0 – источник исключение.

Бит 1 =1 – в поле index № дескриптора в IDT,
= 0 – в поле index № дескриптора в GDT или LDT.

Бит 2 = 1 – № дескриптора в поле index сответствует таблице LDT,
= 0 – № дескриптора в поле index сответствует таблице GDT.

Биты 3…15 – index дескриптора в одной из таблиц IDT, LDT или GDT в соответствии с битами 1 и 2 или 000 для прерываний 8 и 17.

Для корректного возврата из обработчика прерывания с кодом ошибки необходимо применять в нем команду POP перед командой IRET для очистки стека от кода ошибки.

Схема вызова программы-обработчика прерывания в защищенном режиме представлена на рисунке 57.

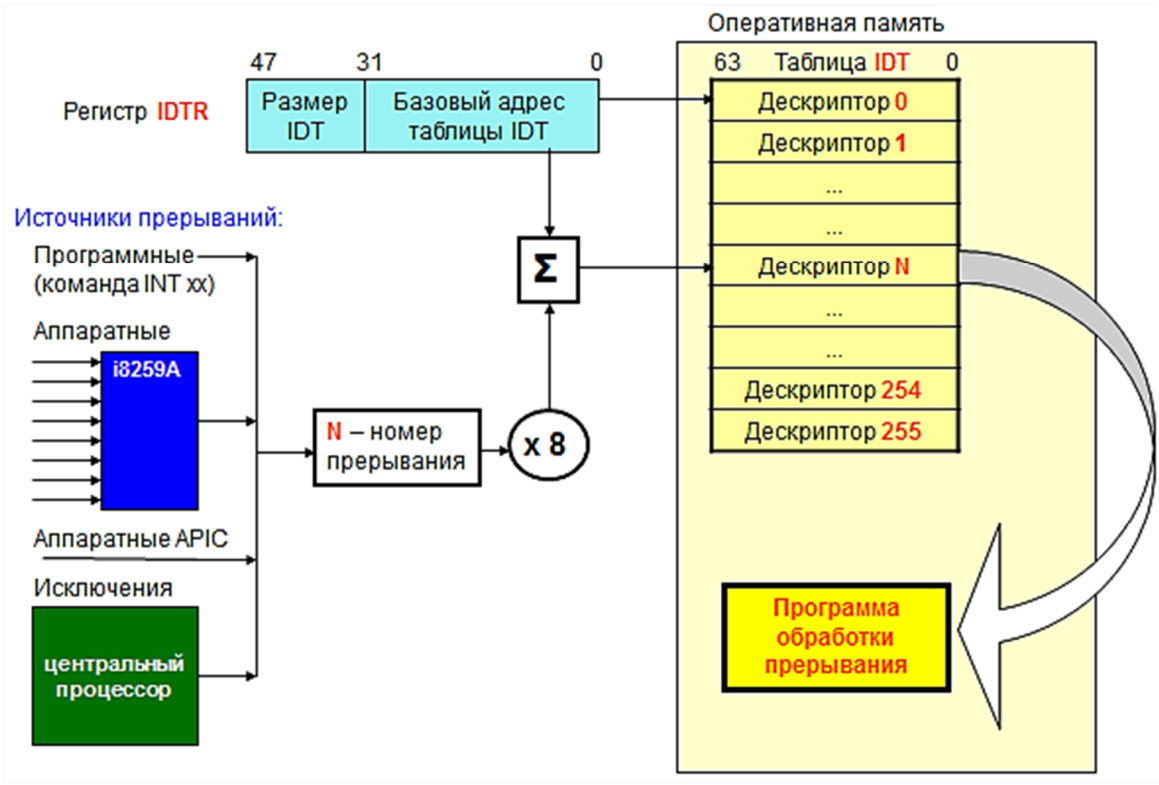


Рисунок 57. – Функциональная схема вызова программы-обработчика прерывания в защищенном режиме

Центральный процессор всегда определяет местоположение таблицы IDT по полю базового адреса в регистре IDTR, независимо от режима.

Для обеспечения преемственности ПЭВМ архитектуры x86 в реальном режиме 8086 поля базового адреса регистра IDTR равны нулю и процессор, используя тот же механизм обработки прерывания, обращается уже к таблице векторов прерываний, расположенной в области оперативной памяти 00000 – 00400. В защищенном режиме (в том числе и в режиме виртуальной 8086) обращение к этой таблице теряет смысл, т.к. она будет содержать фиктивные адреса программ-обработчиков прерываний.

Дескрипторы в таблице IDT строго упорядочены в соответствии с номерами прерываний. Размерность таблицы IDT – не более 256 элементов размером по 8 байт, по числу возможных источников прерываний.

Допускается неполное заполнение таблицы IDT (в отдельных случаях описываются не все 256 дескрипторов этой таблицы), для неиспользуемых номеров прерываний формируются нулевые шлюзы-заглушки. Если этого не сделать, то при незапланированном прерывании или с номером, превышающим пределы IDT для данной задачи, будет возникать ситуация исключения – сбой общей защиты (с номером 13 (0Dh)).

Формат дескрипторов в таблице IDT представлен на схеме (рисунок 58).

Дескрипторы в таблице прерываний IDT называются шлюзами (в отличие от дескрипторов в таблицах GDT и LDT). Шлюз предназначен для указания точки входа в программу обработки прерывания.

Шлюз ловушки (запрещает обработку аппаратных прерываний):

63 47				**type**	36		31 15	0
Offset_2	p	dpl	0	**1111**	000	Не используется	indicator	Offset_1

Шлюз прерывания:

63 47				**type**	36		31 15	0
Offset_2	p	dpl	0	**1110**	000	Не используется	indicator	Offset_1

Шлюз задачи (указывает через поле indicator на сегмент загруженной задачи TSS, которая является обработчиком прерывания):

63 47				**type**	36		31 15	0
Не используется	p	dpl	0	**0101**	000	Не используется	indicator	Не используется

p – бит присутствия соответствующей программы обработки;

dpl – минимальный уровень привилегий задачи, которая может передать управление обработчику прерываний через данный шлюз (обычно 10 или 01);

indicator – селектор, указывающий на дескриптор в LDT или GDT;

Offset_1 и Offset_2 – смещение и сегмент точки входа в программу обработчик (первое и второе слово соответственно).

Рисунок 58. – Формат дескрипторов в таблице IDT

В дескрипторной таблице прерываний IDT могут содержаться шлюзы трех типов: шлюз ловушки, шлюз прерывания и шлюз задачи, что объясняется наличием различных источников прерываний (см. рисунок 58) и особенностями передачи управления в программы их обработки.

Центральный процессор отличает шлюзы по значению в поле типа type (биты 40 – 43).

Тема 8. Подсистема прямого доступа к памяти

8.1. Схема и назначение прямого доступа к памяти

Прямой доступ к памяти (DMA, Direct Memory Access) обеспечивает высокоскоростной обмен данными между устройствами ввода-вывода и ОЗУ без использования центрального процессора, что позволяет освободить процессор для выполнения вычислений параллельно с обменом и независимо от него. Наиболее часто возможности DMA используются при работе с дисковыми накопителями, однако реализовано использование DMA и с адаптерами накопителей других типов, а также рядом других устройств, например, звуковыми адаптерами и мультимедиа контроллерами.

Ощутимые преимущества дает использование DMA в процессе обмена данными с устройствами, принимающими или передающими данные достаточно большими порциями и с высокой скоростью.

В ПЭВМ функции DMA выполняет контроллер прямого доступа к памяти (ПДП), построенный на базе микросхемы i8237A или ее аналогов i8237A-4 и i8237A-5, работающих с тактовой частотой 4 и 5 мГц соответственно (рисунок 59).

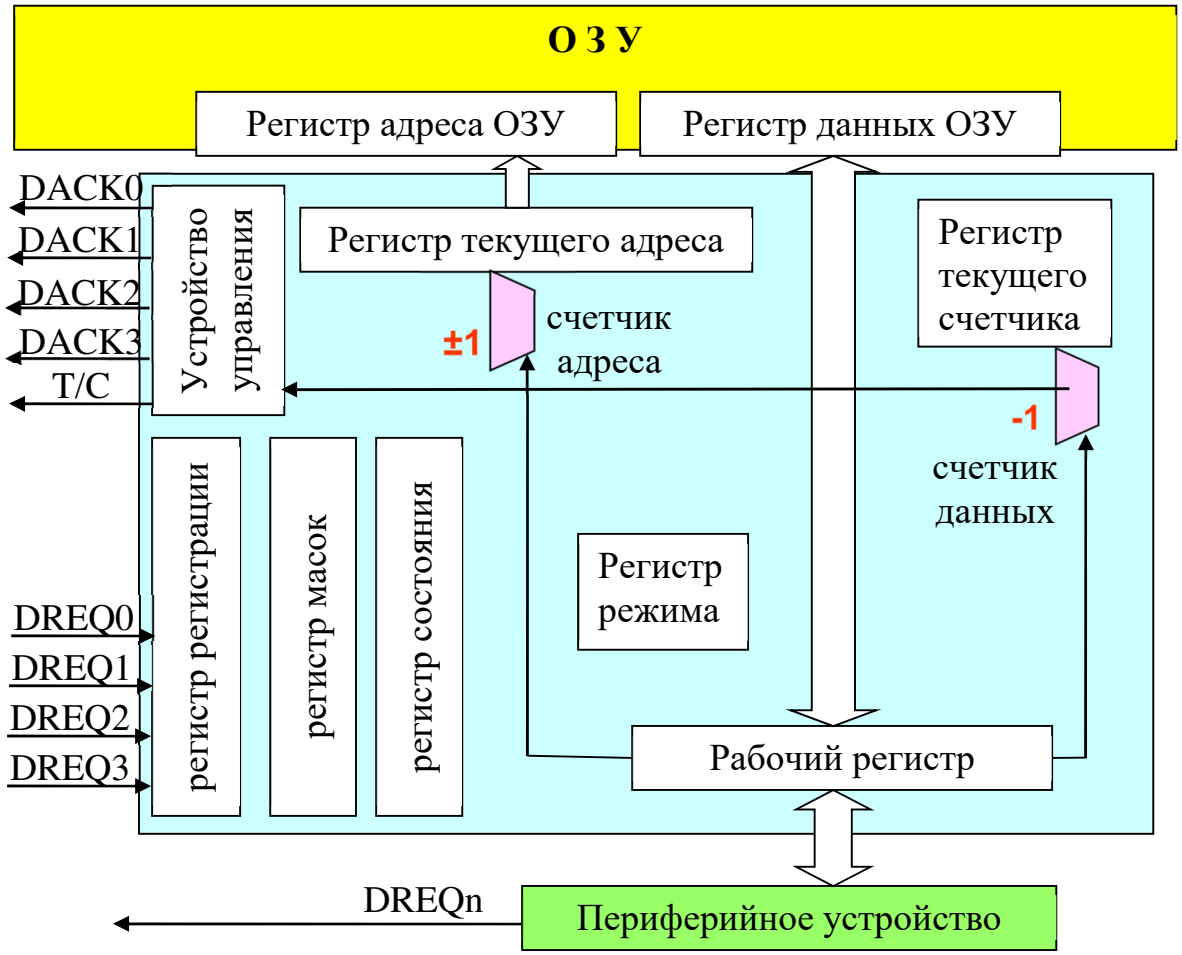


Рисунок 59. – Функциональная схема контроллера прямого доступа к оперативной памяти

Контроллер имеет четыре 8- или 16-разрядных независимых канала, каждый из которых может обслуживать одно периферийное устройство.

В современных схемах аппаратной части ПЭВМ частота контроллера ПДП соответствует частоте системной шины для обеспечения синхронной передачи данных в режиме «память – память».

Следует также учитывать, что многие современные устройства не поддерживают режим ПДП из-за сложности его инициализации и малой разрядности, предпочитая режим PIO (процессорный ввод/вывод), тем более, что современные центральные процессоры содержат до 6 ядер и более. Одно из них можно задействовать в режиме PIO без потери времени на исполнение программ. В этом легко убедиться, если открыть диспетчер устройств ПЭВМ на закладке «Прямой доступ к памяти (DMA)».

Основные задачи контроллера прямого доступа к памяти.

1. Управление инициализируемой периферийным устройством или центральным процессором передачей данных между ОЗУ и периферийным устройством.

2. Задание размера блока данных, который подлежит передаче, и область памяти, используемой при передаче.

3. Формирование адресов ячеек ОЗУ, участвующих в передаче.

4. Подсчет объема данных, передаваемых между периферийным устройством и ОЗУ, определение момента завершения операции ПДП.

Операционный блок контроллера содержит ***регистры***:

– *запросов* – регистрация поступающих запросов на ПДП;
– *масок* – регистр маскирования каналов ПДП;
– *состояния* – статусный регистр каналов;
– *режима* – определяет режим работы канала ПДП;
– *рабочий* – буферный регистр данных;
– *текущего адреса* – регистр изменяемого адреса;
– *текущего счетчика* – регистр изменяемого счетчика;
– *начального адреса* – для каждого канала (нет на схеме);
– *начального счетчика* – для каждого канала (нет на схеме);
– *рабочий регистр адреса* – для каждого канала (нет на схеме);
– *рабочий регистр счетчика* – для каждого канала (нет на схеме).

8.2. Порядок работы контроллера прямого доступа к памяти

Контроллер ПДП обычно имеет наивысший приоритет в занятии шины, и управление ОЗУ переходит к контроллеру ПДП сразу после его инициализации. Работа контроллера осуществляется в следующем порядке:

1. При инициализации ПДП от центрального процессора в текущий счетчик данных заносится размер подлежащих передаче блока данных (число байт или слов).

2. В регистр текущего адреса заносится начальный адрес области памяти ОЗУ, используемой при передаче.

3. Центральный процессор освобождает шину данных и передает управление передачей данных контроллеру прямого доступа до завершения передачи (в соответствии с требованиями режима передачи данных).

4. В процессе передачи каждого байта (или слова) содержимое регистра текущего адреса уменьшается или увеличивается на единицу и через масштабный коэффициент формируется адрес очередной ячейки памяти.

5. Одновременно уменьшается на единицу содержимое текущего счетчика данных.

6. Обнуление текущего счетчика данных указывает на завершение операции по прямой передаче данных между периферийным устройством и ОЗУ.

7. Устройство управления формирует сигнал центральному процессору о завершении передачи данных и освобождении шины.

8.3. Режимы работы контроллера ПДП

В стандартном контроллере ПДП предусмотрено 4 режима работы:

1. Режим одиночной передачи (Signle Transfer Mode). После каждого цикла передачи контроллер освобождает шину процессору, но сразу же начинает проверку сигналов запроса и, как только обнаруживает активный сигнал запроса, инициирует следующий цикл передачи.

2. Режим блочной передачи (Block Transfer Mode). В этом режиме наличие сигнала запроса требуется только до момента выдачи контроллером сигнала «Подтверждение запроса на ПДП» (DACK), после чего шина не освобождается вплоть до завершения передачи блока.

3. Режим передачи по требованию (Demand Transfer Mode). Данный режим является промежуточным между двумя первыми: передача идет непрерывно до тех пор, пока активен сигнал запроса, состояние которого проверяется после каждого цикла передачи. Как только устройство не может продолжить передачу, сигнал запроса сбрасывается им и контроллер приостанавливает работу.

Этот режим применяется для обмена с медленными устройствами, не позволяющими по своим временным характеристикам работать с ПДП в режиме блочной передачи.

4. Каскадный режим (Cascade Mode). Режим позволяет включить в подсистему ПДП более одного контроллера в тех случаях, когда недостаточно четырех каналов ПДП. В этом режиме один из каналов ведущего контроллера используется для каскадирования с контроллером второго уровня. Для работы в каскаде сигнал HRQ ведомого контроллера подается на вход DREG ведущего, а сигнал DACK ведущего подается на вход HDLA («Потверждение захвата шины») ведомого.

8.4. Типы передач данных контроллером ПДП i8237A

Контроллер ПДП i8237A реализует следующие типы передачи данных:

– ***Передача «память – память» (Memory-to-memory DMA).*** Используется для передачи блока данных из одного места памяти в другое. Исходный адрес определяется в регистрах нулевого канала, выходной – в регистрах первого канала. Передача происходит с использованием рабочего регистра в качестве промежуточного звена для хранения информации.

– ***Автоинициализация (автозагрузка, Autoinitialization).*** После завершения обычной передачи использованный канал ПДП маскируется и должен быть перепрограммирован для дальнейшей работы с ним. При автоинициализации маскировка канала после окончания передачи не происходит, а регистры текущего адреса и счетчик циклов автоматически загружаются из соответствующих регистров с начальными значениями.

– ***Режим фиксированных приоритетов.*** В этом режиме канал 0 всегда имеет максимальный приоритет, а канал 3 – минимальный и любая передача по каналу с более высоким приоритетом будет выполняться раньше, чем по каналу с более низким приоритетом.

– ***Циклический сдвиг приоритетов.*** Каждому каналу, по которому прошла передача, автоматически присваивается низший приоритет, после чего право на передачу получает канал с наивысшим приоритетом.

– ***Сжатие времени передачи (Compressed transfer timing).*** При синхронной передаче ПДП может сократить время выполнения такта передачи на 2 цикла за счет тактов ожидания, входящих в каждый цикл.

8.5. Программирование контроллера ПДП i8237A

В работе ПДП различаются 2 главных цикла: цикл ожидания (Idle cycle) и активный цикл (Active cycle). Каждый цикл подразделяется на ряд состояний, занимающих по времени один период часов (тактовый интервал).

Из цикла ожидания контроллер может быть переведен в состояние программирования (Program Condition) путем подачи на вход RESET сигнала высокого уровня, длительностью не менее 300 нс и следующей за ним подачи сигнала низкого уровня (уровня 0) на вывод CS (Chip Select). В состоянии программирования контроллер будет находится до тех пор, пока на выводе CS сохранится сигнал низкого уровня.

В процессе программирования контроллеру задаются:

– начальный адрес памяти для обмена;

– уменьшенное на единицу число передаваемых байтов;

– направление обмена, а также устанавливаются требуемые режимы работы (разрешить или запретить циклическое изменение приоритетов, автоинициализацию, задать направление изменения адреса при обмене и т.д.).

Загрузка 16-разрядных регистров контроллера осуществляется через 8-разрядные порты ввода-вывода. Перед загрузкой первого (младшего) байта должен быть сброшен (очищен) триггер-защелка (триггер первый/последний, First/Last flip-flop), который изменяет свое состояние после вывода в порт первого байта и таким образом дает возможность следующей командой вывода в тот же порт загрузить старший байт соответствующего регистра. Всего для программирования ведущего контроллера ПДП используется 21 восьмиразрядный портовый регистр и 23 – для ведомого.

Тема 9. Системный интервальный таймер ПЭВМ

9.1. Назначение и схема системного интервального таймера

Системный интервальный таймер (СИТ) является одним из основных устройств в составе архитектурной схемы ЭВМ. Он служит для задания временных интервалов и формирования сигналов с различными временными параметрами. В ПЭВМ применяется программируемый таймер i8254, разработанный фирмой Intel (об этом свидетельствует буква «i» в обозначении микросхемы таймера). В схемах современных ПЭВМ системный интервальный таймер входит в состав гибридного контроллера (северного моста), как обязательный элемент архитектуры ЭВМ. Схема интервального таймера показана на рисунке 60.

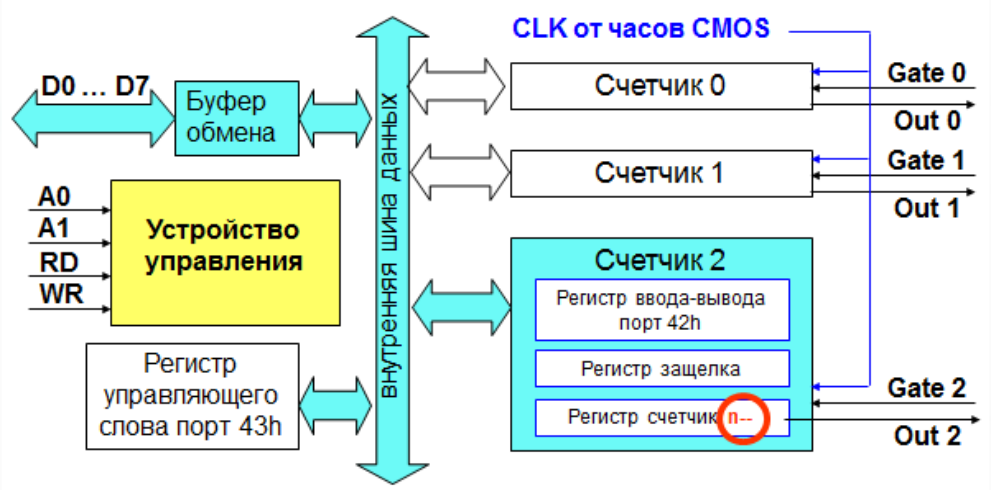


Рисунок 60. – Схема системного интервального таймера i8254

В состав операционного блока таймера входят:

– буфер шины данных;

– портовый регистр;

– три независимых канала, каждый из которых содержит регистры режима, внутренний буфер и 16-разрядный счетчик.

Управляющий автомат таймера выполнен на базе схемной жесткой логики и содержит три блока микропрограмм управления для каждого отдельного канала. Синхронизация устройства осуществляется сигналом CLK от генератора часов реального времени.

9.2. Режимы использования каналов СИТ

Управляющему автомату через регистр управления (порт 43h) с использование сигналов A0, A1 и WR для каждого канала можно задавать шесть режимов функционирования, управляемых сигналом Gate n и генерирующих сигнал Out n.

Режим 0 Прерывание терминального счета. После записи управляющего слова в регистр режима канала на выходе OUT устанавливается напряжение низкого уровня. Загрузка счетчика не изменяет это состояние. Затем начинается работа счетчика в режиме вычитания единицы в каждом цикле (декремент). В момент, когда счетчик обнулится, на выходе OUT устанавливается напряжение высокого уровня и сохраняется до загрузки счетчика новым значением. Временная диаграмма сигналов канала счетчика показана на рисунке 61.

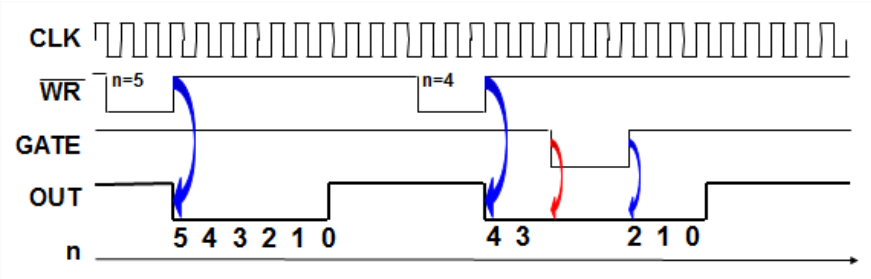


Рисунок 61. – Диаграмма сигналов счетчика СИТ на режиме 0

Счет возможен только при сигнале высокого уровня на входе GATE. Низкий уровень этого сигнала или ниспадающий фронт запрещают счет.

Перезагрузка счетчика во время счета приводит к остановке счета младшим байтом, загрузка старшего байта запускает новый цикл счета.

Минимально допустимое значение счетчика *n* равно 2.

Режим 1 Ждущий мультивибратор. На выходе OUT формируется отрицательный импульс длительностью $t = nT$, где n – число, загруженное в счетчик, T – период тактовых импульсов.

Низкий уровень на выходе OUT устанавливается со следующего такта после подачи на вход GATE сигнала высокого уровня. Временная диаграмма сигналов канала счетчика на этом режиме показана на рисунке 62.

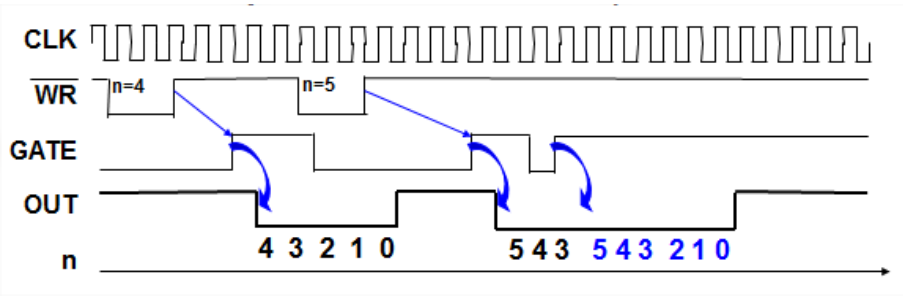


Рисунок 62. – Диаграмм сигналов счетчика СИТ на режиме 1

Загрузка в счетчик нового числа не влияет на длительность текущего импульса, но учитывается при следующем запуске счетчика.

Перезапуск счетчика производится нарастающим фронтом сигнала на входе GATE (без перезагрузки счетчика).

Минимально допустимое значение счетчика *n* равно 1.

Режим 2 Генератор частоты (периодический). Каждый раз после достижения счетчиком нуля на выходе OUT появляется отрицательный импульс с длительностью один такт. Перезагрузка значения *n* счетчика вступает в силу только после перезапуска счетчика.

При исчезновении сигнала высокого уровня на входе GATE прекращается счет и на выход OUT подается напряжение высокого уровня.

Временная диаграмма сигналов канала счетчика на этом режиме показана на рисунке 63.

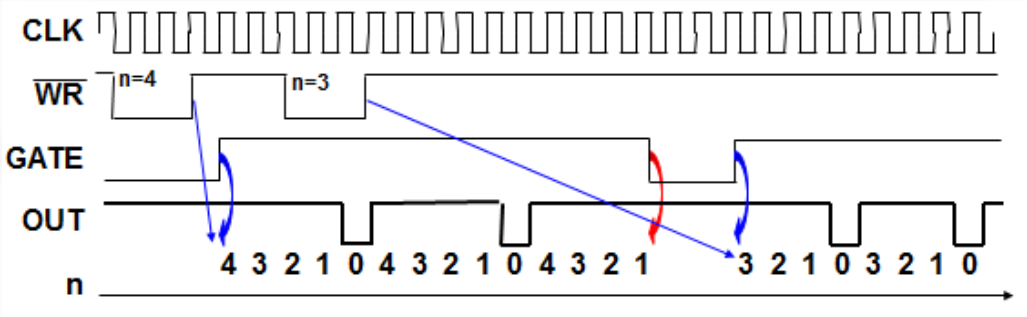


Рисунок 63. – Диаграмм сигналов счетчика СИТ на режиме 2

Перезапуск счетчика происходит при наличии на входе GATE сигнала высокого уровня.

Минимально допустимое значение счетчика *n* равно 4.

Режим 3 Генератор меандра (периодический). Аналогичен режиму 2, но выходной сигнал имеет форму меандра, т.е. прямоугольной формы, где положительный уровень выходного сигнала занимает первый полупериод, а отрицательный – второй полупериод (рисунок 64).

Если *n* (начальное значение счетчика) четно, то длительность положительного и отрицательного полупериодов равна $t_{1,2} = nT/2$, а если *n* нечетно – то $t_1 = (n + 1)T/2$ и $t_2 = (n – 1)T/2$, соответственно.

Низкий уровень сигнала на входе GATE запрещает счет, на выходе OUT устанавливается сигнал высокого уровня. Высокий уровень GATE разрешает счет, а нарастание его запускает счетчик начального состояния.

Значение счетчика *n* – не менее 4.

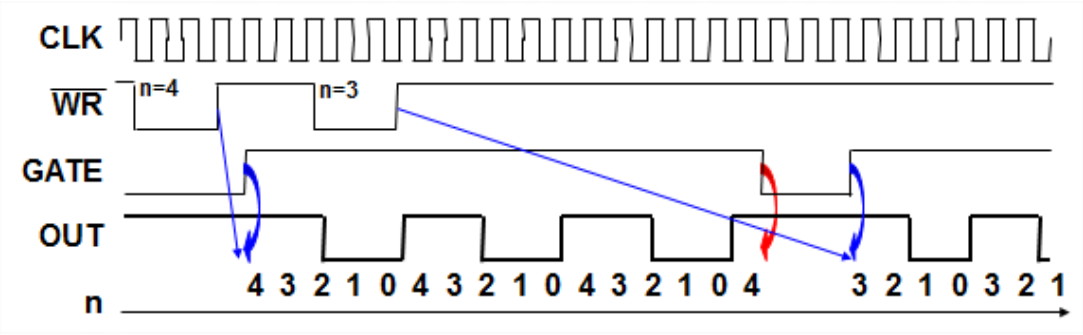


Рисунок 64. – Диаграмм сигналов счетчика СИТ на режиме 3

Режим 4 Счетчик событий. По окончании отсчета числа, загруженного в счетчик, на выходе OUT формируется отрицательный импульс длительностью один такт. Запись в счетчик во время счета младшего байта не влияет на текущий счет, а запись старшего байта перезапускает счетчик.

Низкий уровень входа GATE запрещает счет, высокий – разрешает.

Минимально допустимое значение счетчика *n* равно 1.

Временная диаграмма сигналов канала счетчика на этом режиме показана на рисунке 65.

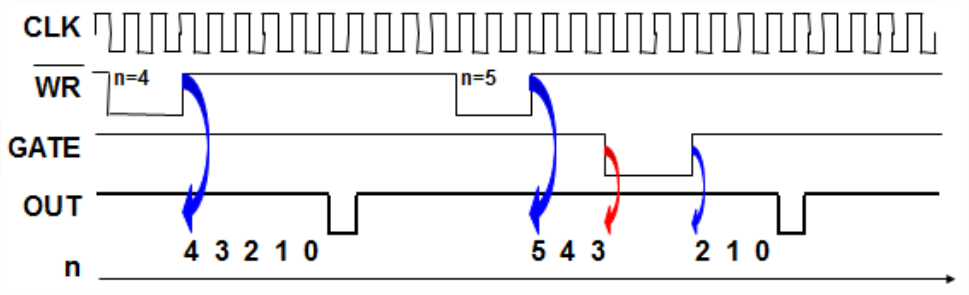


Рисунок 65. – Диаграмм сигналов счетчика СИТ на режиме 4

Режим 5 Счетчик событий с автозагрузкой. Отличие от режима 4 состоит в том, что каждое нарастание сигнала на входе GATE перезапускает счетчик. Перезагрузка счетчика не влияет на текущий цикл, однако следующий цикл определяется вновь занесенным числом *n*.

Временная диаграмма сигналов канала счетчика на этом режиме показана на рисунке 66.

Минимально допустимое значение счетчика *n* равно 1.

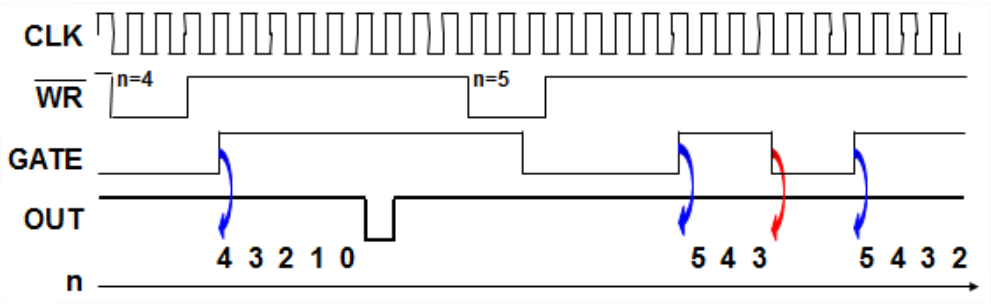


Рисунок 66. – Диаграмм сигналов счетчика СИТ на режиме 5

9.3. Основное назначение каналов СИТ в ПЭВМ

В ПЭВМ каналы таймера имеют следующее назначение:

Канал 0 – системные часы (IRQ0), режим 3, счетчик *n* = 0 (65536).

Тактовая частота каждого канала равна 1,193181 мГц, т.е. каждый такт имеет длительность 0,84 мкс. Вход GATE канала 0 всегда имеет высокий уровень, поэтому счет на этом канале разрешен всегда.

При начальной загрузке BIOS инициализирует этот канал для работы в режиме 3 со счетчиком 0 (т.е. 65536 декрементов на цикл счета). Поэтому частота системных часов равна 1193181 / 65536 = 18,2 Гц, а сигнал на выходе канала 0 – Out 0, воспринимаемый контроллером прерываний как сигнал прерывания IRQ0, инициирует вектор прерывания Int8 18,2 раза в секунду (т.е. каждые 55 мс).

Схема формирования системных часов в ПЭВМ представлена на рисунке 67.

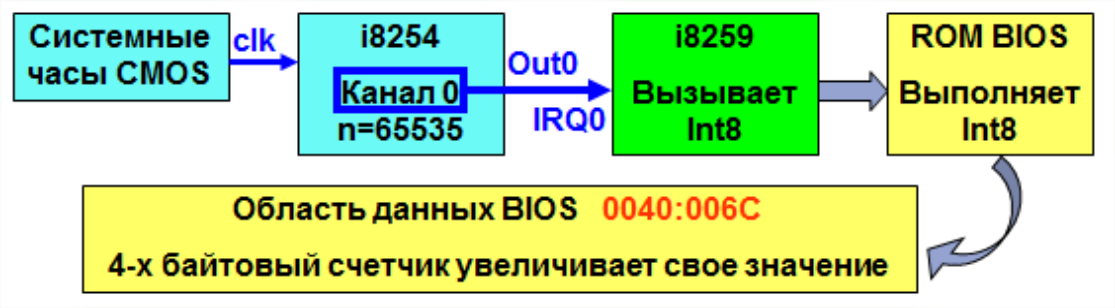


Рисунок 67. – Схема формирования системных часов в ПЭВМ

Канал 1 – Регенерация памяти, режим 2, значение счетчика n = 18.

Вход GATE канала 1 также, как и канала 0 всегда имеет высокий уровень, поэтому счет на этом канале также разрешен всегда. Выход счетчика связан со входом DRQ0 контроллера прямого доступа к памяти на захват канала 0 в режиме блочного приема-передачи данных из памяти в память.

Сигнал DRQ0 имеет приоритет над другими запросами прямого доступа к памяти.

Сигнал GATE представляет собой подтверждение контроллера прямого доступа к памяти на захват канала 0 – DACK0.

Канал 1 работает в режиме 2 со счетчиком 18, поэтому регенерация памяти происходит каждые 18 мкс или 66288 раз в секунду.

ВАЖНО! ***В архитектурах современных ПЭВМ функции регенерации оперативной памяти выполняет контроллер памяти в каждом цикле обращения к памяти, т.к. частота системной шины приближена к частоте работы центрального процессора (обычно равна одной четвертой этой частоты).***

Канал 2 – генератор звука системного динамика или другое назначение пользователя (например, генерация псевдослучайных чисел), режим 3, значение счетчика n задается пользователем при системном программировании. Функциональная схема использования канала 2 для генерации звука системным динамиком показана на рисунке 68.

При генерации звука значение счетчика n канала 2 вычисляется по формуле n = 1193181 / f, где f – требуемая частота тона звука в герцах.

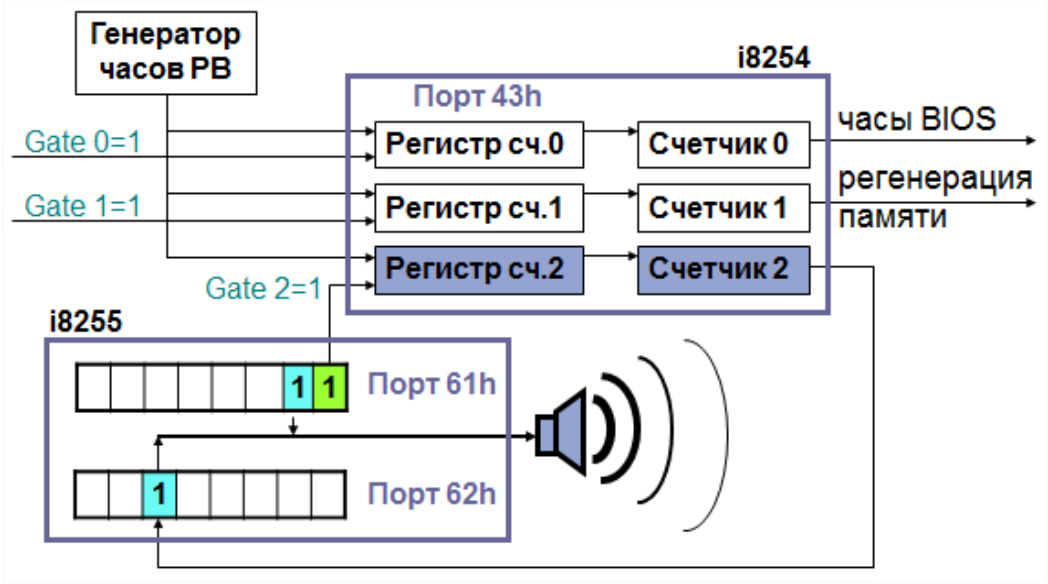


Рисунок 68. – Схема генерации звука системным динамиком в ПЭВМ

9.4. Порядок программирования СИТ

Программирование канала осуществляется путем ввода управляющих слов в регистр режима каналов 43h и начального значения в его счетчики через соответствующие порты каналов.

Системный таймер имеет следующие программно доступные регистры:

Адрес порта	Операция	Назначение
0040h	запись	Загрузка счетчика канала 0
	чтение	Чтение счетчика канала 0
0041h	запись	Загрузка счетчика канала 1
	чтение	Чтение счетчика канала 1
0042h	запись	Загрузка счетчика канала 2
	чтение	Чтение счетчика канала 2
0043h	запись	Запись управляющего слова в регистр режима канала

Назначение битов управляющего слова:

Бит 0 – режим кодирования значения *n*:

0 – двоичный код;

1 – двоично-десятичный код (формат BCD).

Биты 1–3 – режим работы канала:

000 – режим 0;

001 – режим 1;

X10 – режим 2;

X11 – режим 3;

100 – режим 4;

101 – режим 5.

Биты 4–5 – вид загрузки и чтения счетчика:

00 – «защелкивание» (биты 0–3 безразличны);

01 – только младший байт;

10 – только старший байт;

11 – младший байт, затем старший.

Биты 6–7 – номер канала:

00 – канал 0;

01 – канал 1;

10 – канал 2;

11 – запрещенная комбинация.

Существует два способа чтения текущего значения счетчика канала:

1) чтение с остановом счетчика. Для обеспечения стабильных показаний необходимо приостановить работу канала либо подачей сигнала низкого уровня на вход GATE (кроме режима 1), либо блокированием тактовых импульсов;

2) чтение «на лету». Для считывания счетчика без остановки процесса счета используется посылка в порт 43h управляющего слова в режиме «защелкивания». Это управляющее слово фиксирует текущее значение счетчика в буфере, а затем считывается младший байт, потом старший.

Тема 10. Часы реального времени ПЭВМ и память CMOS

10.1. Схема и назначение часов реального времени

Часы реального времени и память CMOS – самостоятельное энергонезависимое устройство (имеющее собственную батарейку напряжением 3–5 вольт с длительностью сохранения работоспособности до 10 лет) на системной плате ПЭВМ. Функциональная схема часов реального времени и памяти CMOS для ПЭВМ (рисунок 69) была впервые разработана фирмой «Motorola» (микросхема MC146818), в последующем была переработана фирмой «DALLAS» (микросхема № 1287) и введена в состав южного моста.

Часы реального времени являются неотъемлемым устройством любой вычислительной системы. Они предназначены для организации учета хронометрических данных (текущее время, дата, день недели и др.) и используются для различных времязадающих пользовательских приложений.

Кварцевый генератор часов реального времени обеспечивает синхросигнал с частотой 1,193181 мГц. Использование высокоточного генератора синхросигнала позволяло использовать ПЭВМ в качестве коммерческих систем контроля трафика, счетчиков энергоресурсов различного назначения, а также в системах автоматического управления технологическими процессами.

В настоящее время использование часов CMOS для этих целей ограничено в связи со снижением уровня требований к частоте генератора. Сама

частота тоже может быть изменена производителем системной платы произвольным образом. Для обеспечения времязадающих функций в высокоточных системах автоматизированного управления и счетчиках в настоящее время используются специальные блоки единого времени с синхронизацией показаний по каналам Интернет.

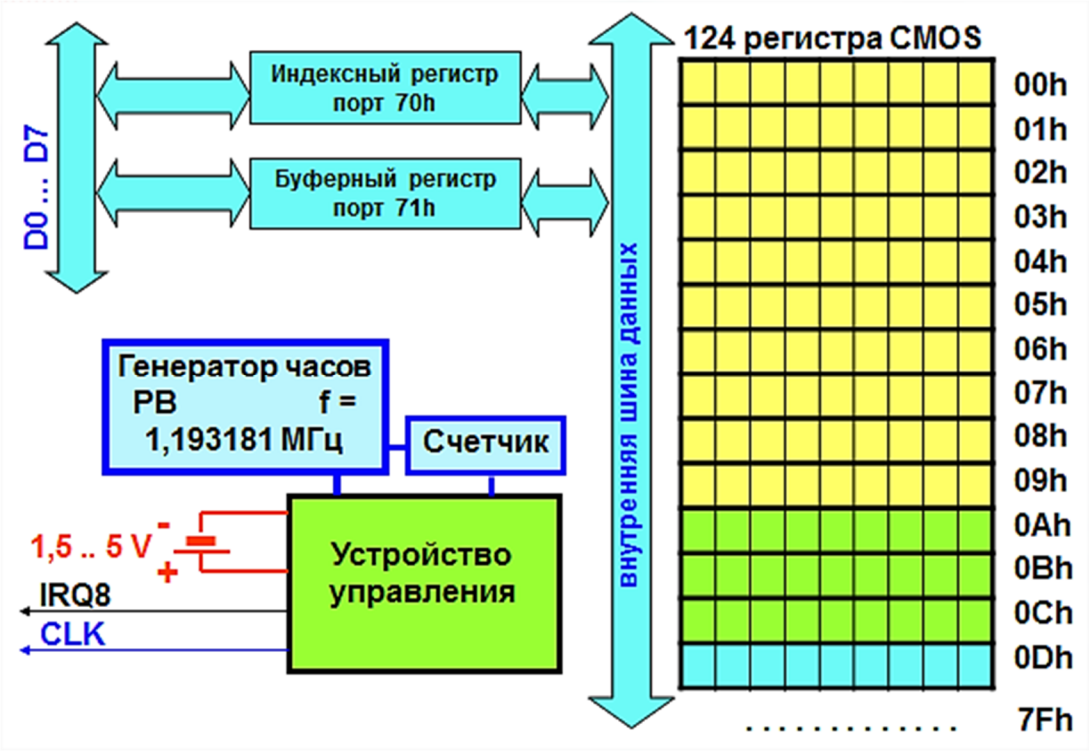


Рисунок 69. – Функциональная схема CMOS часов реального времени

Регистровая память CMOS содержит значения основных хронометрических переменных и характеристик режимов работы часов реального времени, а также многочисленные константы базовой системы ввода/вывода материнской платы (BIOS).

ВАЖНО! Значения основных хронометрических переменных и констант доступны для просмотра и частичного редактирования через редактор SETUP, входящий в миниоперационную систему BIOS. Вызов редактора осуществляется нажатием соответствующих клавиш (обычно это клавиши <Del>, <F2>, <Ctrl>+<alt>+<Del> или иные сочетания, выбранные разработчиками BIOS) в процессе выполнения операций контроля и начальной установки устройств ПЭВМ до начала загрузки операционной системы.

Микросхема BIOS, в которой находится миниоперационная система BIOS с редактором SETUP и набором тестовых программ, является энергонезависимой (ППЗУ в старых моделях, либо ПЗУ в современных). В современных системных платах допускается перезапись ПЗУ BIOS на более современное программное обеспечение.

Содержание ***основных регистров часов*** реального времени в CMOS:

– 00h секунды;
– 01h секундная тренога (будильник);
– 02h минуты;
– 03h минутная тренога (будильник);
– 04h часы;
– 05h часовая тревога (будильник);
– 06h день недели;
– 07h день месяца;
– 08h месяц;
– 09h год;
– 0Ah регистр статуса часов реального времени:
бит 7 = 1 – идет модификация времени (чтение часов запрещено);
– 0Bh регистр статуса часов реального времени:
бит 6 = 1 – разрешено периодическое прерывание;
бит 5 = 1 – разрешено прерывание треноги;
бит 4 = 1 – разрешено прерывание конца модификации;
бит 1 = 1 – часы считаются до 24;
= 0 – часы считаются до 12;
бит 0 = 1 – разрешено запоминание времени суток;
– 0Ch регистр статуса выполнения прерывания IRQ8;
– 0Dh регистр статуса состояния батарейки и памяти.

Содержание ***регистров SETUP*** в CMOS:

– 0Eh состояние констант POST после загрузки BIOS;
– 0Fh состояние констант POST после выключения ПЭВМ;
– 10h типы дисководов для гибких дисков;
– 11h резерв;
– 12h типы жестких дисков;
– 13h резерв;
– 14h константы SETUP;
– 15h – 16h размер основной памяти в килобайтах;
– 17h – 18h размер дополнительной памяти в килобайтах;
– 19h константы первого жесткого диска;
– 1Ah константы второго жесткого диска;
– 1Bh – 2Dh резерв;
– 2Eh – 2Fh контрольная сумма регистров CMOS;
– 30h – 31h расширение размера дополнительной памяти;
– 32h значение века в формате BCD;
– 33h свойства размера памяти (< 1 Мб или > 1 Мб);
– 34h – 3Fh другие константы SETUP и BIOS;
– 40h – 7Fh недокументированный резерв различных производителей системных плат и BIOS.

10.2. Использование времязадающих функций часов CMOS

Часы реального времени вызывают аппаратное (радиальное) прерывание IRQ8 и соответствующее векторное прерывание 70h. Программа пользователя может установить вектор этого прерывания на любую процедуру, которую требуется выполнить в определенное время. Для этого разработано векторное прерывание 1Ah.

Прерывание IRQ8 может появляться по одной из трех причин:

1. Периодическое прерывание происходит через определенные интервалы времени обновления счетчика часов реального времени. Их периодичность приближенно равна одной миллисекунде.

2. Прерывание тревоги происходит, когда значение трех регистров тревоги совпадает со значениями соответствующих временных регистров. Тревога устанавливается как смещение относительно текущего момента времени. Максимальная величина смещения установки тревоги равна 23:59:59.

Вектор прерывания 4AH может указывать на пользовательскую процедуру обработки тревоги.

3. Прерывание конца модификации происходит после каждого обновления значений регистров часов PB CMOS – 04h.

Примечание: номер причины вызова прерывания указывается в регистре 0Ch: бит 6 = 1 – периодическое; бит 5 = 1 – прерывание тревоги, бит 4 = = 1 – обновление часов.

Особенности использования прерывания 1Ah.

Функции 0 и 1: Прерывания 1Ah читают и устанавливают счетчик времени суток.

Функция 2: Чтение времени из часов реального времени AH = 02h.

При возврате: **CH** = часы в BCD; **CL** = минуты в BCD; **DH** = секунды в BCD.

Функция 3: Установка времени часов реального времени AH = 03h.

При входе: **CH** = часы в BCD; **CL** = минуты в BCD; **DH** = секунды в BCD; **DL** = 1 – переход на летнее время.

Функция 4: Чтение даты из часов реального времени AH = 04h.

При возврате: **CH** = век в BCD (19 или 20); **CL** = год в BCD (с 1980), **DH** = месяц в BCD, **DL** = день месяца в BCD.

Функция 5: Установка даты часов реального времени AH = 05h.

При входе: **CH** = век в BCD (19 или 20); **CL** = год в BCD (с 1980); **DH** = месяц в BCD; **DL** = день месяца в BCD.

Функция 6: Установка тревоги для часов AH = 06h.

При входе: **CH** = часы смещения в BCD; **CL** = минуты смещения в BCD; **DH** = секунды смещения в BCD.

Функция 7: Сброс тревоги AH = 07h.

ВАЖНО! Использование прерываний DOS и BIOS, а также портовых адресов разрешено только в режиме 8086 (режим прямой адресации) или при использовании специальных программ-эмуляторов этого режима («DOS-BOX»). В защищенных режимах любое прямое обращение пользовательского приложения к функциям прерывания или адресам портов рассматривается операционной системой как запрещенное, вызывающее состояние исключения без потери работоспособности OS. Это сделано разработчиками OS для повышения устойчивости операционных систем на бытовом уровне.

Следует помнить, что многозадачная операционная система Windows разработана, прежде всего, для непрофессиональных пользователей и не является системой реального времени, поэтому любой виртуальный таймер в Windows не может гарантировать какой бы то ни было фактической точности отсчета временного интервала. В любой момент времени система может прервать выполнение приложения, чтобы дать возможность работать другому приложению (простой, вызванный прерыванием, может длиться до 30 мс). В то же время, мультимедийный (программный) таймер Windows обеспечивает вполне приемлемую фактическую точность отсчета временных интервалов для многих прикладных задач непрофессионального пользователя.

Время в современных OS Windows – это количество миллисекунд, прошедших с момента старта операционной системы. Этот формат времени поддерживается для обратной совместимости DOS-программ. Время Windows хранится в виде 32-разрядного целого числа без знака, которое сбрасывается в нулевое значение после того, как Windows проработает примерно 49,7 дня без выключения ПЭВМ.

Операционная система управляет временем Windows через прерывания системного таймера, добавляя к текущему значению времени Windows приращение, равное периоду работы системного таймера. Кроме того, система периодически синхронизирует время Windows с показаниями часов реального времени, то есть с системным временем ПЭВМ.

Системный таймер Windows – это программное устройство, находящееся под управлением операционной системы (в отличие от аппаратного таймера часов CMOS, с которым работали программы под управлением MS-DOS).

10.3. Особенности использования формата BCD

Формат упакованного двоично-десятичного кода (BCD) очень часто используется в вычислительных системах для хранения времязадающих констант. Поскольку показания часов реального времени никогда не состоят более, чем из двух десятичных цифр, значения времени очень удобно выдавать в форме BCD, т.е. когда байт делится на две половины и каждая десятичная цифра представляется всего четырьмя битами. Такой формат позволяет легко переводить десятичные числа в номера соответствующих символов ASCII или ANSI (рисунок 70).

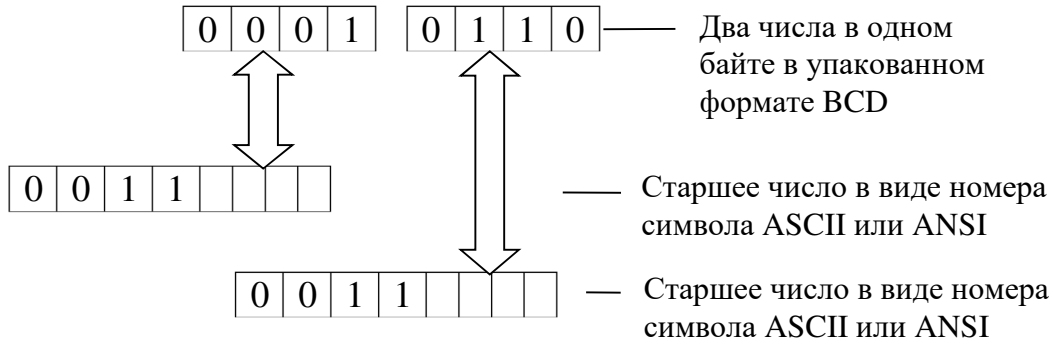


Рисунок 70. – Схема преобразования упакованного двоично-десятичного числа в номер соответствующего символа таблицы ASCII

Для такого преобразования необходимо поместить соответствующую половину байта в младший конец регистра и добавить 48 (двоичное 110000) для получения номера кода ASCII или ANSI, соответствующего графическому изображению данного числа.

Тема 11. Система управления энергопитанием ACPI

11.1. Общие понятия и задачи системы ACPI

Дальнейшее развитие системы BIOS заключается в создании и внедрении системы управления энергопитанием компонентов ЭВМ.

ACPI (Advanced Configuration and Power Interface) – усовершенствованный интерфейс управления конфигурацией и энергопитанием – открытый промышленный стандарт, впервые выпущенный в декабре 1996 г. Интерфейс разработан совместно компаниями HP, Intel, Microsoft, Phoenix и Toshiba и др. Он определяет общий интерфейс для обнаружения аппаратного обеспечения, управления питанием и конфигурацией материнской платы и подключенных к ней устройств. Спецификация 2.0 была представлена в сентябре 2000 г. Она распространяется на более широкий спектр компьютеров, включая корпоративные серверы, настольные системы и ноутбуки. Кроме того, в ACPI 2.0 добавлена поддержка 64-разрядных микропроцессоров для серверов, поддержка различных типов памяти, устройств PCI и PCI-X. Версия спецификации 3.0 была выпущена 10 октября 2006 г.

На настоящий момент последней версией спецификации ACPI является версия 5.0, выпущенная 6 декабря 2011 г.

Задача ACPI – обеспечить взаимодействие между операционной системой, аппаратным обеспечением и BIOS материнской платы с целью максимально возможного снижения энергопотребления при начальном конфигурировании устройств перед загрузкой OS.

Различные версии ACPI поддерживаются многими операционными системами, в том числе всеми версиями Microsoft Windows, начиная с Windows 98, системами GNU/Linux, FreeBSD, OpenBSD, NetBSD и eComStation.

Наиболее известной и используемой частью стандарта ACPI является управление питанием, входящее в состав отдельной таблицы SETUP BIOS любого производителя.

ACPI передает управление питанием операционной системе (OS). Такая модель выгодно отличается от существовавшей до этого модели APM (Advanced Power Manager), в которой за управление питанием отвечала только BIOS материнской платы, а возможности ОС в этом отношении были ограничены. В современной модели ACPI BIOS предоставляет операционной системе методы для прямого детализированного управления аппаратным обеспечением. Таким образом, ОС получает практически полный контроль над энергопотреблением.

Другая важная часть спецификации ACPI – это предоставление на серверах и настольных компьютерах возможностей по переводу в состояние чрезвычайно низкого энергопотребления, в котором питание подается лишь на оперативную память (а возможно, и она находится без питания), но при этом прерывания некоторых устройств (часы реального времени, клавиатура, модем и т.д.) могут достаточно быстро перевести систему из такого состояния в нормальный рабочий режим (то есть «пробудить» систему).

Помимо требований к программному интерфейсу ACPI также требует специальной поддержки от аппаратного обеспечения. Таким образом, поддержку ACPI должны иметь ОС, чипсет материнской платы и даже центральный процессор. Интерфейс ACPI организуется путём размещения в определённой области оперативной памяти нескольких таблиц, содержащих описание аппаратных ресурсов и программных методов управления ими. Каждый тип таблицы имеет определённый формат, описанный в спецификации.

Таблицы, содержащие методы управления устройствами и обработчики событий ACPI, содержат коды обработчиков на языке AML (ACPI Machine Language) – машинно независимый набор инструкций, представленный в компактной форме. Операционная система, поддерживающая ACPI, содержит интерпретатор AML, который транслирует инструкции AML в инструкции центрального процессора, выполняя таким образом методы или обработчики событий. Некоторые из этих таблиц полностью или частично хранят статические данные, которые при перезапуске операционной системы не изменяются. Статические данные, как правило, создаются производителем материнской платы или BIOS и описываются на специальном языке ASL (ACPI Source Language), а затем компилируются в коды на языке AML.

Динамические данные, которые зависят, например, от установок BIOS и комплектации материнской платы, формируются BIOS на этапе загрузки системы до передачи управления ОС.

Роль ОС в интерфейсе ACPI заключается в том, что она переводит различные компоненты аппаратного обеспечения из одного состояния (например, нормальный режим работы) в другое (например, режим пониженного энергопотребления). Переход из одного состояния в другое происходит, как правило, по событию. Например, падение температуры на ядре процессора является событием, по которому ОС может вызвать метод уменьшения скорости вращения вентилятора.

Другой пример: пользователь дал явное указание перехода системы в спящее состояние с сохранением оперативной памяти на диск, а через некоторое время администратор сети произвёл активацию системы с помощью функции Wake-on-LAN.

11.2. Основные состояния системы ACPI

1. Глобальные состояния:

G0 (S0) (Working) – нормальная работа.

G1 (Suspend, Sleeping, Sleeping Legacy) – машина выключена, однако текущий системный контекст (system context) сохранён, работа может быть продолжена без перезагрузки.

Для каждого устройства определяется «степень потери информации» в процессе засыпания, а также где информация должна быть сохранена и откуда будет прочитана при пробуждении, время на пробуждение из одного состояния до другого (например, от сна до рабочего состояния).

G2 (S5) (soft-off) – мягкое (программное) выключение: система полностью остановлена, но под напряжением, готова включиться в любой момент. Системный контекст при этом утерян.

G3 (mechanical off) – механическое выключение системы; блок питания ATX отключен.

2. Состояния сна системы.

S1 – состояние, при котором все процессорные кэши сброшены и процессоры прекратили выполнение инструкций, однако питание процессоров и оперативной памяти поддерживается; устройства, которые не обозначили, что они должны оставаться включенными, могут быть отключены;

S2 – более глубокое состояние сна (центральный процессор отключен).

S3 («Suspend to RAM» (STR) в BIOS, ***«Ждущий режим» («Standby»)*** – в этом состоянии на оперативную память (ОЗУ) продолжает подаваться питание, и она остаётся практически единственным компонентом, потребляющим энергию. Так как состояние операционной системы, всех приложений и открытых документов хранится в оперативной памяти, пользователь может возобновить работу точно на том месте, где он её оставил.

S4 («Спящий режим» (Hibernation) в Windows, ***«Safe Sleep»*** в Mac OS X, также известен как ***«Suspend to disk»)*** – в этом состоянии всё содержимое оперативной памяти сохраняется в энергонезависимой памяти, такой

как жёсткий диск: состояние операционной системы, всех приложений, открытых документов и т.д. Это означает, что после возвращения из S4, пользователь может возобновить работу с места, где она была прекращена, аналогично режиму S3.

Различие между S4 и S3 в том, что перебои с питанием компьютера в S3 приведут к потере всех данных в оперативной памяти, включая все несохранённые документы, в то время как компьютер в S4 этому не подвержен.

3. Состояния центрального процессора

Выделяют четыре состояния функционирования процессора:

C0 – оперативный (рабочий) режим.

C1 (известно как Halt) – состояние, в котором процессор не исполняет инструкции, но может незамедлительно вернуться в рабочее состояние. Некоторые процессоры, например, Pentium 4, также поддерживают состояние Enhanced C1 (C1E) для более низкого энергопотребления.

C2 (известно как Stop-Clock) – состояние, в котором процессор обнаруживается приложениями, но для перехода в рабочий режим требуется время.

C3 (известно как Sleep) – состояние, в котором процессор отключает собственный кэш, но готов к переходу в другие состояния.

4. Состояния устройств.

Выделяют четыре состояния функционирования других устройств (монитор, модем, шины, сетевые карты, видеокарта, диски, флоппи и т.д.)

D0 – полностью оперативное состояние, устройство включено.

D1 и ***D2*** – промежуточные состояния, активность определяется самим устройством.

D3 – устройство выключено.

5. Состояния производительности центрального процессора.

Пока процессор или устройство функционирует (D0 и C0, соответственно), он может находиться в одном или нескольких состояниях производительности. Эти состояния зависят от конкретной реализации процессора. Так, P0 – всегда наивысший уровень производительности; с P1 до Pn последовательное снижение уровня производительности, до предела реализации, где n не превышает 16.

P-состояния также известны как SpeedStep в процессорах Intel, как PowerNow! или Cool'n'Quiet в процессорах AMD, или PowerSaver в процессорах VIA.

P0 – максимальная производительность и частота.

P1 – меньше чем P0, напряжение/частота урезаны.

P2 – меньше чем P1, напряжение/частота урезаны.

...

Pn – меньше чем P(n – 1), напряжение/частота урезаны.

11.3. Порядок использования интерфейса ACPI

Операционная система или программа, желающая воспользоваться интерфейсом ACPI, сначала должна выполнить его детектирование и определить расположение таблиц обработчиков событий и статических данных.

Для этого, в области адресов выполняемого блока BIOS (E0000h – FFFFFh) необходимо найти структуру RSDP (Root System Description Pointer).

Структура имеет размер 36 байт, может быть расположена в произвольном месте внутри указанного 128-Кбайтного блока и обнаруживается по последовательности символов «RSD PTR». Если структура не обнаружена или её контрольная сумма неверна, детектирующая программа делает вывод, что ACPI не поддерживается. Если структура обнаружена и имеет правильную контрольную сумму, из нее считывается 32-битный адрес таблицы RSDT (Root System Description Table), содержащей каталог таблиц ACPI.

В реализациях ACPI версии 2.0 и выше, доступен также 64-битный адрес таблицы XSDT (Extended System Description Table). Функции этой таблицы такие же, как у таблицы RSDP, но за счет использования 64-битных указателей таблица XSDT, а также таблицы, на которые она ссылается, могут быть расположены выше 4 Гб.

Разработчиками ACPI предусмотрена возможность для дизассемблирования структур таблиц (например, дизассемблер фирмы Phoenix Technologies – AD.EXE (ACPI Dump).

Перечень основных таблиц ACPI:

Таблица FADT (Fixed ACPI Description Table) содержит набор параметров, описывающих платформу и чипсет.

Структура FACS (Firmware ACPI Control Structure) используется для управления переходом в спящий режим и выходом из него. Она содержит вектор для передачи управления на процедуру, запускаемую при выходе из спящего режима.

Таблица DSDT (Differentiated System Description Table) является самой сложной. Она содержит описание методов выполнения типовых системных операций (например, считывание температуры процессора или изменение номера линии запроса на прерывание, используемой заданным устройством), информацию об устройствах, шинах, всех системных объектах и методах взаимодействия с ними.

Таблицы SSDT (Secondary System Description Table) используются как модули расширения таблицы DSDT. В системе может присутствовать несколько таблиц SSDT.

Таблица MADT (Multiple APIC Description Table) используется для информирования операционной системы о присутствующих логических и физических процессорах и контроллерах прерываний I/O APIC (Input-Output Advanced Programmable Interrupt Controller).

Первые 36 байт каждой таблицы образуют заголовок, назначение полей которого одинаково для всех таблиц. Далее следует основной блок – это информация, структура которой определяется типом таблицы.

Для обеспечения стандарта ACPI разработчиками центральных процессоров (Intel, AVI, AMD и др.) в ядро процессора был введен расширенный контроллер прерываний APIC. Этот контроллер, наряду с уже известными задачами, перечисленными в разделе 10, выполняет обработку исключений и особых случаев, связанных с изменением режимов работы отдельных устройств из состава архитектуры ПЭВМ, определенных условиями энергосбережения. С этой целью контроллер APIC имеет свыше 100 каналов дополнительных аппаратных прерываний, которые используются для обработки событий изменения состояний системы.

Тема 12. Контроллер клавиатуры ПЭВМ

12.1. Схема контроллера клавиатуры

Алфавитно-цифровая клавиатура в настоящее время является основным инструментом для ввода текстовой информации в ЭВМ. Рабочее поле клавиатуры содержит 101 клавишу и состоит из нескольких блоков:

- блок алфавитных клавиш (34 клавиши);
- блок основных цифровых клавиш (10 клавиш);
- блок дополнительных цифровых клавиш (10 клавиш);
- блок функциональных клавиш (12 клавиш);
- блок клавиш стрелочных указателей и листания (10 клавиш);
- блок клавиш арифметических операций (6 клавиш);
- клавиши различных переключателей (19 клавиш).

Существуют мультимедийные клавиатуры с дополнительными наборами клавиш, предназначенных для работы с различными мультимедиа- и Интернет-приложениями.

Контроллер клавиатуры предназначен для обеспечения основного интерфейса взаимодействия пользователя с вычислительной системой. На рисунке 71 представлена функциональная схема контроллера клавиатуры.

Операционный блок контроллера состоит из контактного поля клавиш. Контакт между проводниками обеспечивается при нажатии соответствующих клавиш. Все проводники подключены к схеме шифратора, на выходе из которого формируется код сканирования или СКЭН-код. Для исключения влияния дребезга, вызванного неплотным контактом проводников в процессе создания контакта, вводится дополнительное время ожидания контакта или частота сканирования уменьшается до 800 – 400 Гц.

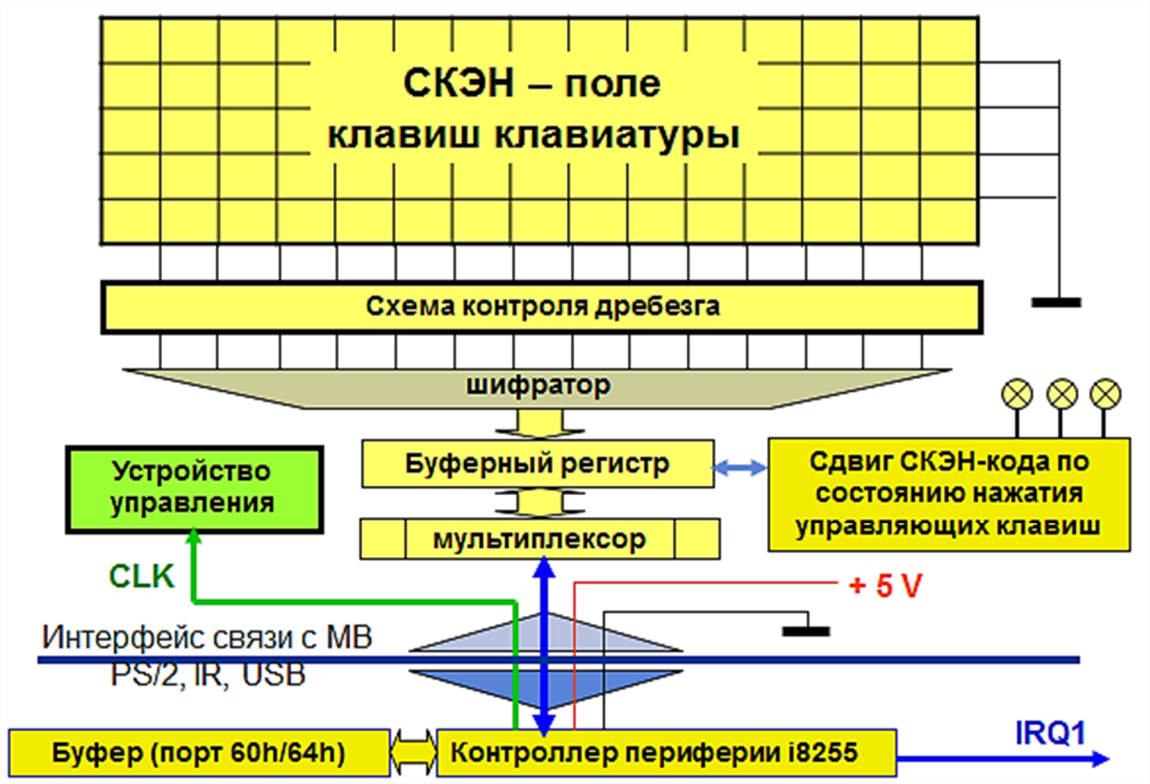


Рисунок 71. – Функциональная схема контроллера клавиатуры ПЭВМ

СКЭН-код может корректироваться системой сдвига при активации клавиш-переключателей регистра <Shift> или <Caps Lock>, либо клавиши замещения блока правой цифровой клавиатуры <Num Lock> блоком стрелок указателей и листания. Откорректированный СКЭН-код передается в порт 64h контроллера периферии, который располагается на системной плате ПЭВМ в составе южного моста. Передача СКЭН-кода сопровождается передачей в тот же порт байта-признака нажатия (00h) или отпускания (F0h) клавиши.

Устройство управления реализовано с использованием цифрового автомата с жесткой логикой.

Электропитание и сигнал синхронизации поступают от внешних источников, располагающихся также на системной плате.

Интерфейсы связи контроллера с ПЭВМ могут быть обычными (PS/2 или USB), либо беспроводными (реализованными на базе интерфейсов «Infra Red» или «Wi-Fi»).

12.2. Схема обработки прерывания от клавиатуры

Любое нажатие клавиши на клавиатуре вызывает аппаратное прерывание IRQ1. В соответствии с этим прерыванием реализуется схема обработки СКЭН-кода нажатой клавиши одной из процедур BIOS – «xlat», которая замещает СКЭН-код номером символа из таблицы ASCII (если была нажата алфавитно-цифровая клавиша) и заносит полученный результат

в буфер клавиатуры, расположенный в области оперативной памяти исполняемого приложения. Затем символ модифицируется с использованием загруженной таблицы гарнитуры шрифтов (художественно усовершенствованных пиксельных образов шрифтов различного размера) и выводится в формируемую видеостраницу, а затем – на экран монитора вместо указателя в окне активного браузера.

Обработка нажатия функциональных клавиш или клавиш, которые не имеют соответствующих графических образов (например, клавиш-переключателей) также выполняется процедурой «xlat», но заполнение буфера производится иначе. После этого производятся соответствующие переключения или вызываются на исполнение программы, с которыми они ассоциированы. Схематично процедура обработки представлена на рисунке 72.

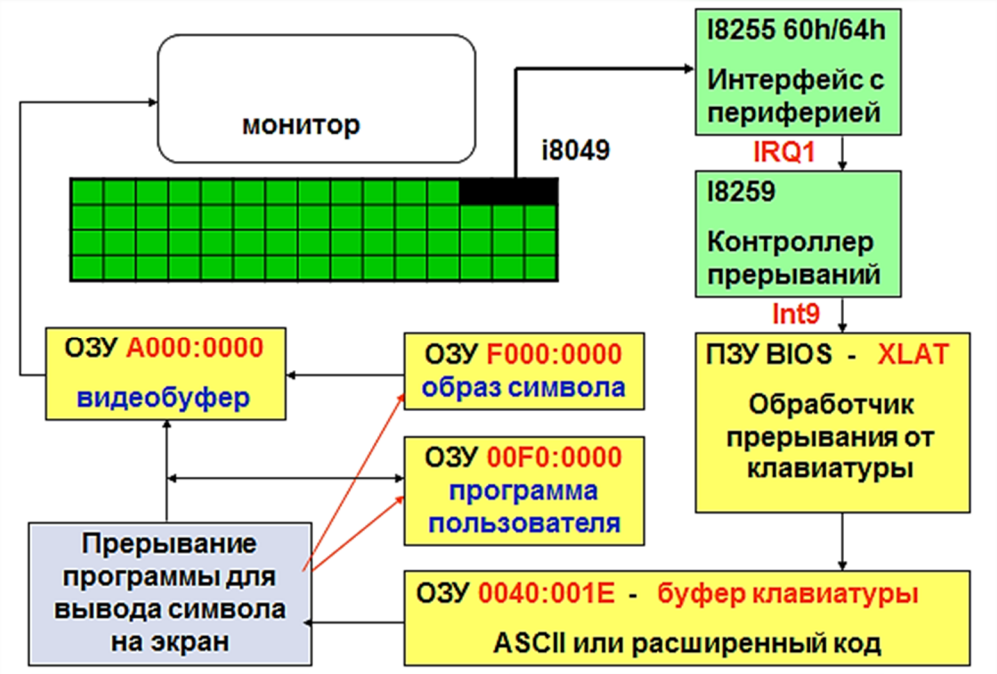


Рисунок 72. – Схема обработки нажатия клавиши клавиатуры в ПЭВМ

12.3. Схема организации буфера клавиатуры

Буфер клавиатуры располагается по строго определенным адресам сегмента прикладной задачи в области констант BIOS, строится по принципу «Fi-Fo (first in – first out)», т.е. первый пришел – первый и ушел. Буфер содержит до 15 2-байтовых ячеек памяти, содержание которых определяется тремя вариантами:

	старший байт	младший байт
Вариант 1	0	Расширенный СКЭН-код

Это, обычно, клавиши управления и их комбинации, СКЭН-коды, которых больше 128, являются расширением основной таблицы кодов.

	старший байт	младший байт
Вариант 2	Номер ASCII	СКЭН-код

Буквенно-цифровые клавиши и их комбинации с клавишами изменения регистра <Shift> и <Caps Lock>.

	старший байт	младший байт
Вариант 3	Номер ASCII	0

Это результат т.н. Alt-ввода информации, т.е. ввода номеров символов из таблицы ASCII на правом блоке цифровых клавиш при постоянно нажатой клавише <Alt>. Выбранный символ появляется на экране монитора при отпускании клавиши <Alt>.

Таблица ASCII имеет 255 графических символов (исключение составляет номер 0 – zero), но из них на экран монитора можно вывести только буквенно-цифровые при нажатии соответствующих клавиш клавиатуры. Alt-ввод позволяет выводить любые графические образы из этой таблицы, при наборе их номера. Такой метод вывода символов на экран используется, например, для построения таблиц в текстовом режиме, границы которых отображаются символами псевдографики.

Схема буфера клавиатуры приведена на рисунке 73.

Рисунок 73. – Схема буфера клавиатуры в оперативной памяти ЭВМ

Кроме ячеек памяти, содержащих СКЭН-коды нажатых клавиш, буфер содержит две ячейки указателя хвоста и головы, которые определяют первую свободную (голова) и последнюю занятую (хвост). Если значение байтов хвоста и головы совпадают, это означает, что буфер пуст.

Кроме этих ячеек памяти буфер содержит две служебные ячейки превышения границы, которые содержат постоянно записанные туда значения кодов нажатой клавиши <Enter> и возврата каретки (перевод строки). Эти значения служат для зацикливания буфера, превращения его в кольцевую схему заполнения. При этом попытка достижения головой буфера своего хвоста генерирует звуковой сигнал, а ввод в буфер новых кодов прекращается.

Следует особо отметить содержание констант статуса клавиш-переключателей клавиатуры в ОЗУ. Они занимают два байта:

Первый байт 0040:0017 значение бита = 1 означает

бит 7 – <insert>
6 – <caps lock>
5 – <num lock> } включено
4 – <scroll lock>
3 – <alt>
2 – <ctrl>
1 – <левый schift>
0 – <правый schift> } клавиша нажата

Второй байт 0040:0018

бит 7 – <insert>
6 – <caps lock>
5 – <num lock>
4 – <scroll lock> } клавиша нажата
3 – <ctrl + num lock> включено
2
1 } не используется
0

Изменение статуса некоторых клавиш сопровождается загоранием или погасанием соответствующего индикатора, причем, если изменить состояние статуса программно в байте статуса, то одновременно с записью байта изменится и состояние индикатора.

Для работы с буфером клавиатуры используются функции прерывания BIOS 21h, для работы с контроллером – 16h.

Тема 13. Архитектура дисковых подсистем

13.1. Основные термины и определения

Диск – пластинка круглой формы, выполненная из алюминиевого сплава (жесткий диск) или полимерного материала (гибкий диск), имеющая покрытие с одной или с обеих сторон из намагничивающихся материалов на основе железа и редкоземельных элементов. Жесткие диски в одном устройстве могут объединяться в пакет, состоящий из двух и более дисков.

Дорожка – данные на диске, расположенные по концентрическим окружностям. Дорожки нумеруются от 0-й, расположенной на периферии диска, к центральному отверстию диска.

Цилиндр – воображаемая поверхность, объединяющая дорожки с одним и тем же номером, расположенные на различных сторонах различных дисков. Нумерация цилиндров соответствует нумерации дорожек. Общее количество цилиндров накопителя обозначается, как *C*.

Сектор – каждая дорожка, размещенная на диске, делится на сектора одинакового углового размера. Количество секторов на дорожках *S* одинаково и не зависит от номера дорожки. Сектора имеют сквозную нумерацию для всех дорожек одной стороны диска, начинающуюся с 0-й на нулевой дорожке. Для операционной системы все сектора всех дисков накопителя объединяются в общую систему нумерации секторов.

Объем сектора – количество информации, помещающейся в одном секторе. Стандартная величина $\boldsymbol{V_s = 512}$ ***байт***, но накопители информации допускают кратное изменение объема, как в сторону увеличения, так и в сторону уменьшения.

Головка – магнитная головка для записи и считывания информации с одной стороны диска. Общее количество головок накопителя обозначается, как *H*, и нумеруется от единицы для первой стороны первого диска в пакете. Номер головки соответствует номеру стороны диска.

Объем диска – максимальное количество информации, которое можно записать на дисковый накопитель. Определяется, как $V_d = C \cdot H \cdot S \cdot V_s$ байт.

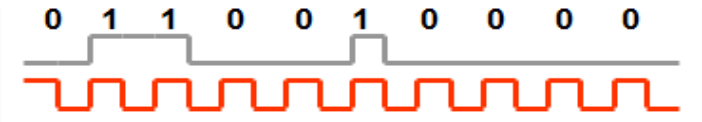
Виртуальная нумерация – для сокращения знакомест, отводимых BIOS под константы *C* и *S*, выполняется увеличение числа головок *H* до 32 с пропорциональным уменьшением числа цилиндров *C* или секторов *S* без изменения объема диска.

Кластер – условное объединение нескольких подряд расположенных секторов в более крупные адресуемые области памяти.

Кластеры имеют сквозную нумерацию для одного логического раздела диска и используются в таблицах расположения файлов (FAT). В кластеры обычно объединяют 8 секторов, и исходя из этой величины определя-

ется число секторов, занимаемое FAT-таблицей – для FAT-16(DOS), например, SFAT = S_d / NSFAT×VFAT/V_s с округлением в большую сторону. Здесь S_d – число секторов, составляющих логический диск, NSFAT – число секторов в кластере (4 или 8), VFAT – размер элемента в FAT-таблице (для FAT-16, например, VFAT = 2 байта).

Методы кодирования информации на магнитных дисках. Для кодирования информации на магнитных дисках используется метод частотной модуляции (ЧМ). Смысл метода заключается в следующем: на поверхности диска выполняется раздельная запись частотно модулированного кода и синхросигнала:

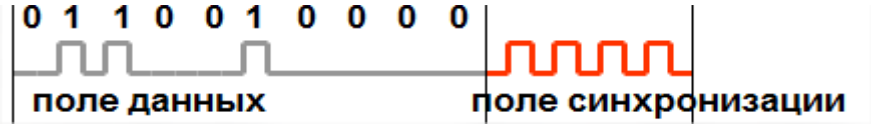


– цифровой код сигнала;
– запись цифрового кода;
– запись синхросигнала.

Такая схема записи позволяла использовать НЖМД (накопитель на жестком магнитном диске) с различными скоростями вращения жесткого диска, но занимала слишком много места и требовала одновременной записи сразу двух сигналов.

Для устранения этих недостатков была применена совмещенная запись частотно модулированного кода и синхросигнала:

Для большей экономии места этот метод был усовершенствован за счет вынесения записи сигнала синхронизации в отдельное поле сектора и стал называться модифицированным частотно модулированным (МЧМ):



При такой системе кодирования информации на жестком диске поле синхронизации в составе сектора занимает 12 байт, затем идет метка данных 4 байта, сами данные 512 байт, контрольная сумма 1 байт и разделитель секторов 22 байта.

Кодирование информации на НЖМД усовершенствованным методом МЧМ за счет повышения плотности, применения различных методов кодирования данных и уменьшения ширины дорожек и расстояний между ними, а также сокращения длины участка синхронизации обозначается RLL (run-length limited) и его модификации – ERLL, ARLL и т.д. Усовершенствованный модифицированный метод частотной модуляции в различных исполнениях (RLL, ERLL, ARLL и т.д.) широко используется изготовителями жестких дисков.

Уровень предкомпенсация – номер цилиндра, с которого изменяются параметры RLL для того, чтобы компенсировать уменьшение линейной длины дуги сектора записи, находящейся ближе к центру диска.

Интерлив – метод логической нумерации секторов на дорожке не по порядку следования, а по мере готовности контроллера НЖМД к процессу чтения/записи данных на сектор.

В противном случае дорожка будет считываться за число оборотов, равное числу секторов, а не за 3-4 оборота.

13.2. Структура главной загрузочной записи жесткого диска

Главная загрузочная запись MBR – занимает нулевой сектор нулевой дорожки на первой стороне жесткого диска (0/0/1) содержит MBR (Master Boot Records), которая на начальном этапе тестирования системы загружается в ОЗУ по адресу 0000:7C00h. Содержание главной загрузочной записи не зависит от операционной системы, но при установке очередной операционной системы исходная загрузочная запись может быть модифицирована.

С таблицей разделов MBR работает утилита Fdisk (MS DOS), Disk Administrator (NTFS) или эквивалентная утилита иной операционной системы.

Схема MBR следующая: 000 – 1BE (446 байт) – код загрузки и передачи управления на активный раздел диска, содержащий операционную систему.

Таблица разделов жесткого диска (Partition Table):

1BE – 1CE (16 байт) – описание границ первого раздела.

1CE – 1DE (16 байт) – описание границ второго раздела.

1DE – 1EE (16 байт) – описание границ третьего раздела.

1EE – 1FE (16 байт) – описание границ четвертого раздела.

1FE – 1FF (2 байта) – сигнатура (подпись) MBR символы <***55, AA***>.

При организации большего количества разделов таблица продолжается от адреса 1AE, затем 19E и т.д. Если свободного места в загрузочной записи больше нет, – предусмотрена процедура организации дополнительной таблицы разделов в другом секторе жесткого диска. Если по адресу 1FE в нулевом секторе отсутствует сигнатура – символы таблицы ASCII 55h и AAh (№ 85 и 170), то запись MBR считается недействительной и игнорируется BIOS.

Структура элемента таблицы разделов:

1-й байт – флаг загрузки (80h – раздел содержит загружаемую операционную систему, 00h – не активен, т.е. OS в разделе отсутствует);

3 байта – H/S/C начала раздела;

1 байт – код OS;

3 байта – H/S/C конца раздела;

4 байта – относительный (вычисляемый) номер начального сектора;

4 байта – размер раздела в секторах.

Понятно, что при такой организации жесткого диска дефрагментация разделов не допускается.

При загрузке MBR BIOS определяет активные разделы, содержащие операционную систему, и передает процесс загрузки на ближайший раздел для последующей загрузки ядра операционной системы.

Загрузочная структура разделов GPT (GUID Partition Table) – это таблица разделов с GUID (Globally Unique Identifier – 128-битный статистически уникальный идентификатор).

GPT разработана для совершенствования системы разбиения диска и ускорения процесса загрузки операционной системы с использованием новой базовой системы ввода/выада – BIOS UEFI (BIOS Unified Extensible Firmware Interface – Унифицированный Расширяемый Интерфейс встроенного программного Обеспечения).

MBR имеет ряд недостатков:

– ограничение по объему дискового пространства (до 2 Тбайт);

– ограничение по количеству разделов (обычно не более 4);

– таблица MBR есть на диске в одном экземпляре и записана на 0-м секторе, отсутствие резервирования снижает сохранность информации;

– использование физических координат секторов.

Для обеспечения надежности и возможности расширения доступного дискового пространства и была разработана структура разделов GPT.

GPT не отменяет, а дополняет MBR с целью обеспечения преемственности: на одном и том же накопителе можно использовать различные структуры разделов. При инсталляции современной операционной системы обычно появляется запрос на использование типа структуры разделов – MBR или GPT?

Структура GPT (нумерация секторов логическая – LBA):

MBR (0-й сектор).

Заголовок таблицы GPT (1-й сектор LBA [1]).

Таблицы GPT разделов (по 4 раздела на сектор, всего 32 сектора).

Данные GPT разделов накопителя.

Резервные таблицы GPT разделов (32 сектора LBA[N-33] до [N-1]).

Резервный заголовок GPT (последний сектор LBA [N]).

Количество разделов структуры GPT увеличено до 128. Объем накопителя может быть увеличен до 9,4 зеттабайт, т.е. полностью определяется возможностями операционной системы. Для обеспечение большей отказоустойчивости накопителя введено резервирование GPT.

К существенным недостаткам структуры разделов GPT можно отнести то, что она не поддерживается накопителями предыдущих выпусков, поэтому их совместное использование одной и той же Windows операционной системой невозможно. Следует отметить также, что использование резервной копии GPT не гарантирует 100% возможность восстановления разделов и должно учитываться при изменении параметров диска.

13.3. Размещение информации на магнитном диске

Все пространство жесткого диска за исключением MBR разделено на разделы – логические диски, обычно имеющие обозначения C, D, E и т.д. Диск <C> считается основным, на него и рекомендуется устанавливать операционную систему.

Порядок следования устройств, содержащих загружаемые операционные системы, можно изменять в соответствующем разделе SETUP BIOS.

13.3.1. Структура DOS размещения информации

Количество логических дисков может быть сокращено до одного.

Каждый раздел начинается с загрузочной записи и содержит две FAT-таблицы (***File Allocation Table)*** – таблицы описания кластерной структуры файла. Первая таблица считается основной, а вторая – резервной, имеющей зеркальное отражение первой. Вторая таблица FAT используется только для восстановления информации в первой таблице при ее стирании.

Схема размещения информации на жестком диске с использованием FAT-таблиц показана на рисунке 74.

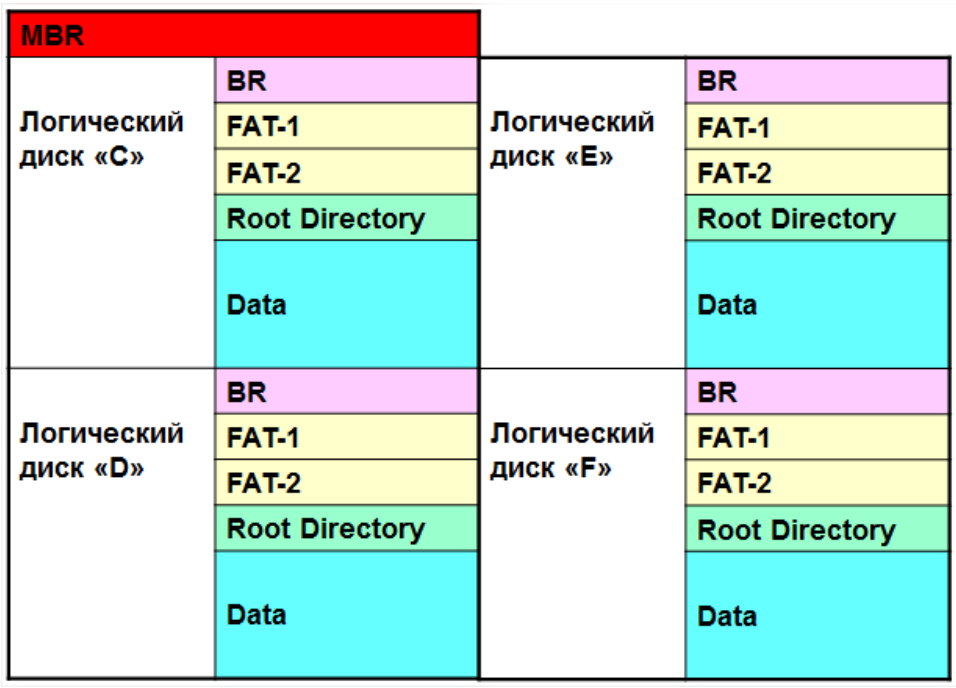


Рисунок 74. – Схема DOS расположения информации на жестком диске

Кроме этого, в каждом разделе имеется корневой директорий, занимающий несколько кластеров, содержащий 32-байтовые элементы, описывающие файлы и папки, находящиеся в данном разделе диска.

Структура загрузочной записи активного логического раздела жесткого диска BR в первых трех байтах содержит команду перехода на начало кода загрузки операционной системы. Остальное содержание загрузочной

записи раздела посвящено технической информации о разделе (блок параметров диска): числе байт в секторе, секторов в кластере, резервных секторов, числе элементов корневого каталога, числе секторов на одну FAT-таблицу, секторов на дорожке, числе головок и т.д. Затем с байта 1Eh начинается код загрузки OS (системный загрузчик). Заканчивается сектор загрузочной записи сигнатурой <55h, AAh>.

Структура элемента корневого каталога логического диска Root Directory включает в себя:

– наименование файла (или каталога);

– расширение, например, exe, dwg, doc, txt, dll, dfx, mp3, avi и т.д.;

– байт атрибутов файла;

– время создания и модификации;

– номер начального кластера в файловой цепочке и атрибуты;

– размер файла в байтах.

Структура байта атрибутов:

0-й бит = 1 – файл только для чтения;

1-й бит = 1 – скрытый файл;

2-й бит = 1 – системный файл;

3-й бит = 1 – метка тома (корневой каталог);

4-й бит = 1 – элемент подкаталога (не файл);

5-й бит = 1 – архивный файл;

7-й бит = 1 – сетевой файл разделяемого доступа.

Структура FAT-таблицы определяет порядок размещения файлов на диске. FAT – это связный список, который используется DOS для определения физического адреса расположения данных на диске, поиска свободного места для новых файлов и указания плохих кластеров.

Первый байт FAT определяет дескриптор носителя (FAT ID), следующие 7 байт содержат 00FFh – заполнитель полей разделителя, остальные поля содержат 2-байтовые элементы следующего назначения:

0000h – свободный (доступный) кластер;

от 0002 до FFEFh – номер кластера, где расположен следующий элемент (часть) файла,

FFF0 – FFF6h – зарезервированные номера,

FFF7h – плохой (недоступный) кластер – <BAD>,

FFFFh – конечный кластер цепочки файла на диске – <EOF>.

Файловая система DOS была разработана для гибких дисков FDD и затем распространена на жесткие диски с учетом MBR, что существенно снизило надежность хранения информации и увеличило время доступа к ней. В настоящее время фирмой Microsoft разработана и используется более надежная файловая система NTFS (файловая система для операционной системы Windows NT).

13.3.2. Структура NTFS размещения информации

Для размещения файлов в таком формате используется один из разделов жесткого диска, информация о котором хранится в MBR. Всего на жестком диске может быть использовано несколько различных файловых систем в различных разделах (на различных логических дисках). Структура размещения информации на жестком диске с использованием главной файловой таблицы представлена на рисунке 75.

Рисунок 75. – Схема NTFS расположения информации на жестком диске

Как и при использовании FAT, основной информационной единицей в NTFS является кластер. Размеры кластеров для томов различной емкости могут существенно отличаться от базовой величины 8 секторов.

При формировании файловой системы NTFS программа форматирования создает файл Master File Table (MTF) и другие области для хранения метаданных. Метаданные используются NTFS для реализации файловой структуры. Первые 16 записей в MFT зарезервированы самой NTFS.

0/1 MFT файловая запись – содержит структуру кластерной цепочки файла на диске (аналогично таблице FAT в DOS-структуре). Более подробно схема файловой записи показана на рисунке 76.

Местоположение файлов метаданных $Mft и $MftMirr записано в загрузочном секторе диска. Если первая запись в MFT повреждена, NTFS считывает вторую запись для нахождения копии первой. Полная копия загрузочного сектора располагается в конце тома. Остальные записи MFT содержат записи для каждого файла и каталога, которые расположены на данном томе.

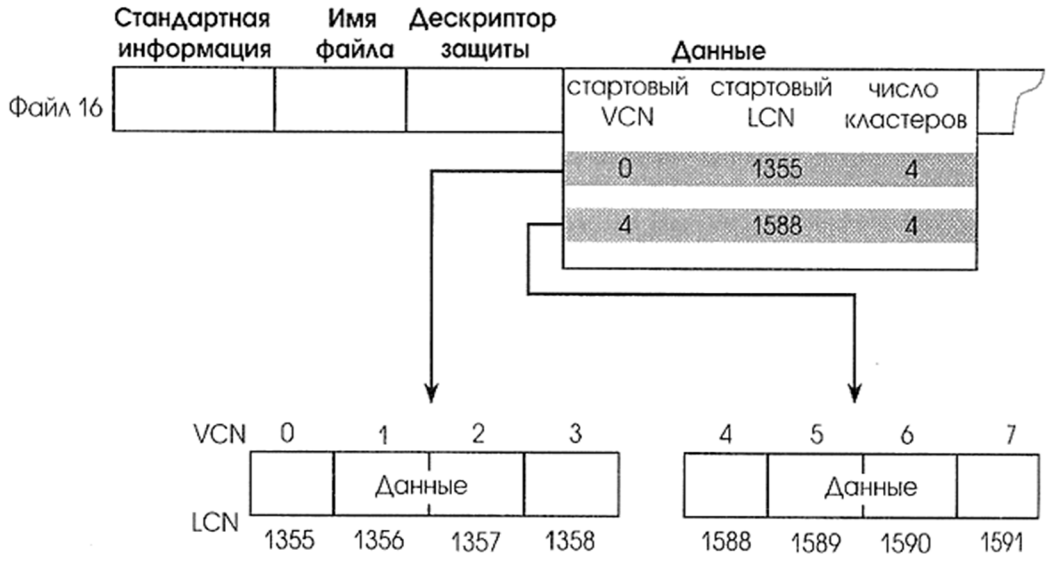


Рисунок 76. – Структура файловой записи MFT (размещения кластеров файла данных на томе NTFS)

2 Файл журнала транзакций (Log file) – содержит список действий, необходимых для восстановления NTFS. Размер зависит от размера тома. Используется Windows 2000 для восстановления файловой системы в случае системных сбоев.

3 Файл состава тома (Volume) – содержит информацию о томе – метку и номер версии.

4 Attribute definitions – таблица имен атрибутов и описания.

5 Root file name index – корневой каталог.

6 Cluster bitmap – информация о том, какие кластеры заняты.

7 Boot sector – содержит код загрузки для загрузочных томов.

8 Bad cluster file – информация о дефектных секторах.

Существует или планируется ввести еще несколько записей (до 15) в MFT, содержащих информацию для различных служб операционной системы (службы квотирования, идентификации, перевода регистра и т.д.).

Обычно один файл использует одну запись в MFT, но если у файла большой набор атрибутов или он становится слишком фрагментированным, то для хранения информации о нем могут потребоваться дополнительные записи. В этом случае первая запись о файле, называемая базовой записью, хранит местоположение других записей. Данные о файлах и каталогах небольшого размера (до 1500 байт) полностью содержатся в первой записи.

Схема взаимодействия NTFS с исполнительными компонентами различных служб операционной системы Windows NT достаточно сложна и требует отдельного рассмотрения. Следует отметить несколько основных служб: регистрация транзакций, драйвер отказоустойчивости FtDisk, драйвер файловой системы NTFS и собственно драйвер диска или RAID системы.

Современная система разметки диска, связанная со структурой BIOS UEFI, – GPT (GUID Partition Table) – не меняет файловые структуры расположения информации на носителе, но позволяет создать до 128 разделов и использовать носители любого объема, т.е. ограничение 2 Тбайта, присущее MBR, устраняется.

13.4. RAID – размещение информации повышенной надежности

RAID (redundant Arrays of inexpensive Disks) – массив недорогих дисков с избыточной информацией для обеспечения ее сохранности.

Основные причины применения RAID:

1. Неприемлемый уровень риска отказа/сбоя устройства с учетом ценности хранимой на нем информации и потерь от простоя всей системы.
2. Недостаточная емкость одиночного накопителя.
3. Недостаточная пропускная способность одиночного накопителя.

Основной смысл создания RAID заключается в распределении порций данных и их контрольных сумм между различными накопителями таким образом, чтобы при выходе из строя одного или нескольких из них система была способна восстановить утраченную информацию и записать ее на новом диске, запущенном взамен вышедших из строя. Также осуществляется выравнивание времени обращения к различным дискам системы, не допуская перегрузки какого-то одного из них.

Схема типового массива RAID показана на рисунке 77.

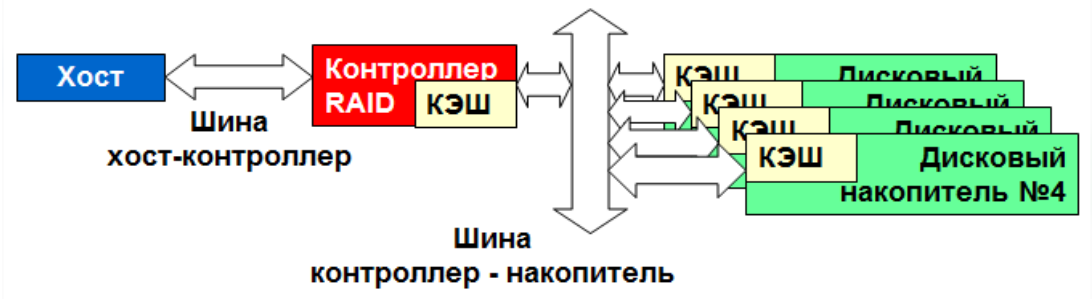


Рисунок 77. – Функциональная схема RAID-массива

Существует несколько вариантов формирования RAID-массива.

RAID 0. Простое перераспределение потоков информации за счет чередования дисков позволяет существенно ускорить процессы чтения/записи информации на дисковые накопители. Занимает минимум 2 диска. Информация равномерно распределяется RAID-контроллером по всем накопителям без дублирования и дополнительной защиты.

Это наиболее простой вариант RAID-систем. Он предназначен только для ускорения обращения к дисковым накопителям (рисунок 78).

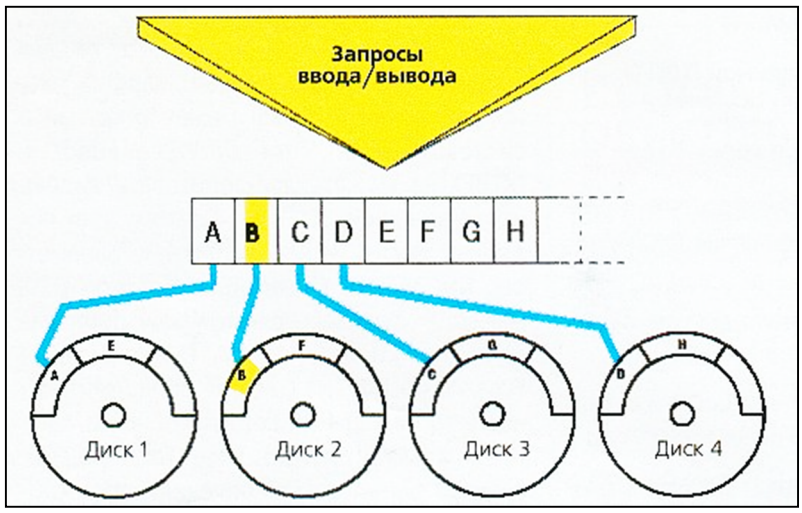


Рисунок 78. – Функциональная схема RAID 0 (чередование дисков)

RAID 1. Простое зеркальное отображение информации (рисунок 79). Занимает минимум 2 диска. Информация одновременно записывается на два дисковых накопителя для обеспечения ее сохранности.

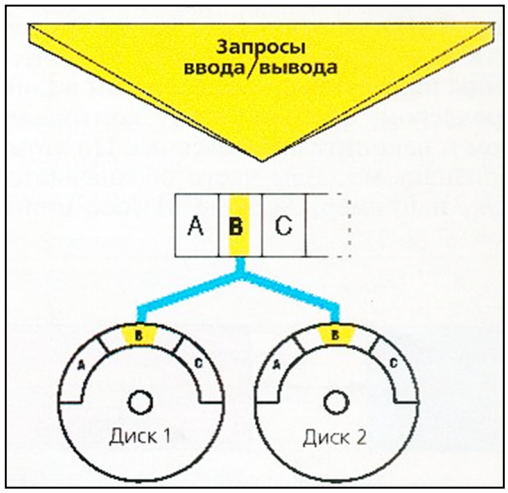


Рисунок 79. – Функциональная схема RAID 1 (зеркалирование дисков)

Надежность в данном случае обеспечивается за счет полного зеркального отображения информации. Этот способ построения массива характеризуется низкой пропускной способностью системы и высокими потерями информационного пространства дисков.

RAID 10 (или 1 + 0). Простое зеркальное отображение двух массивов дисков с перераспределением (чередованием) информации. Занимает минимум 4 диска.

RAID 2. Синхронная запись данных и кодов на все накопители (рисунок 80). Занимает минимум 3 диска. Информация разделяется на кванты малого размера (до 1-го байта) и одновременно с шифрами Хэмминга записывается RAID-контроллером на всех накопителях. Характеризуется высокой пропускной способностью и надежностью, но требует большое количество дисков.

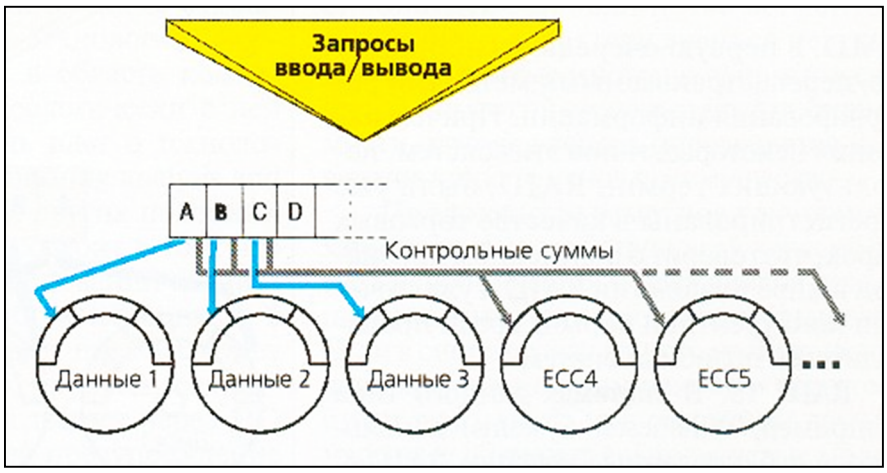


Рисунок 80. – Функциональная схема RAID 2 (синхронная запись)

RAID 3. Вариант аналогичен схеме RAID 2, но для записи кода используется выделенный диск (рисунок 81). Занимает минимум 3 диска. Информация разделяется на кванты малого размера, но вместо кодов Хэмминга используется более простой код шифрования – CRC, а запись кодов выполняется RAID-контроллером на 1 диск.

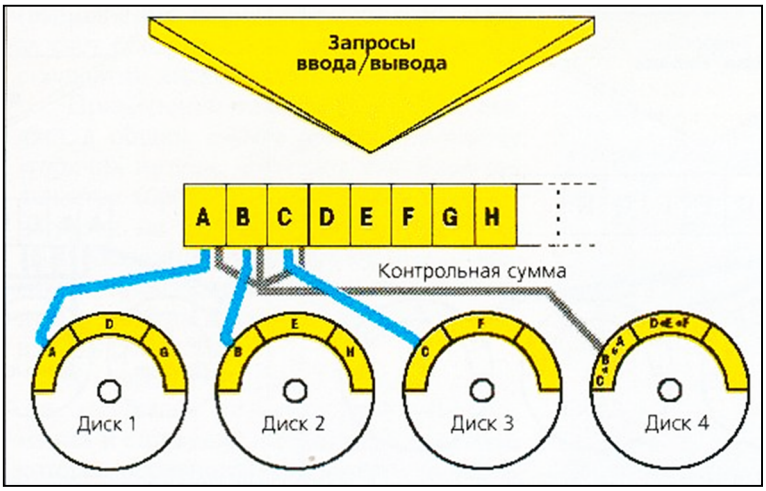


Рисунок 81. – Функциональная схема RAID 3 (синхронная запись)

Такой способ характеризуется более низкой ценой по отношению к RAID 2, но и меньшей надежностью.

RAID 4. Аналогичен варианту схемы RAID 3, но используется для квантов большого размера, что позволяет несколько увеличить пропускную способность для современных дисков большого размера.

RAID 5. Циклическое чередование записи данных на дисках. Занимает минимум 3 диска. Вариант формирования схемы аналогичен варианту RAID 3, но информация, разделенная на кванты малого размера, записывается RAID-контроллером на накопителях с их чередованием.

Характеризуется высокой надежностью, средней пропускной способностью и существенной экономией дискового пространства.

RAID 6. Циклическое чередование записи данных, кодов и контрольных сумм на различных дисках системы (рисунок 82). Занимает минимум 4 диска. Информация кодируется по методу Рида-Соломона, кроме этого, вычисляются контрольные суммы кванта и циклически (чередованием дисков) размещаются на различных накопителях системы.

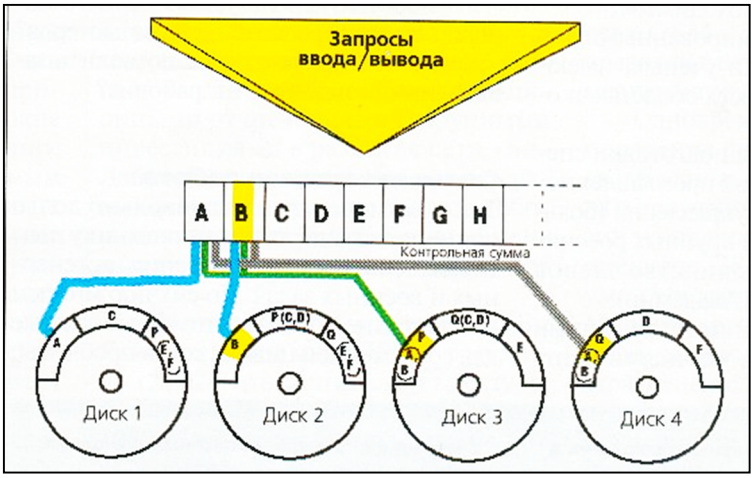


Рисунок 82. – Функциональная схема RAID 6 (чередование записи)

Характеризуется очень высокой надежностью по отношению к остальным вариантам и используется в серверных накопителях.

Перечисленные варианты RAID массивов имеют разные уровни устойчивости к отказам накопителей, показатели производительности и цены (количества дисков в системе), поэтому прямое сравнение их экономической эффективности без учета специфики решаемых задач некорректно. Обычно, BIOS, содержащая встроенный RAID-контроллер, допускает использование вариантов RAID 0, 1, 10. Для систем с более сложной организацией массивов требуется установка отдельного RAID-контроллера.

13.5. Восстановление информации на магнитных дисках

13.5.1. Восстановление информации в файловой системе FAT

Порча или полное разрушение таблицы разделов MBR или загрузочной записи BR обычно является следствием неумелого или небрежного использования утилиты DOS fdisk или аналогичных средств других операционных систем при их установке.

Восстановление этих записей осуществляется вручную с использованием программных утилит фирмы Simantec, например, утилиты Disk Editor, позволяющей работать с физической поверхностью жесткого диска, а не с логическим его содержанием.

Используются ручные методы для восстановления таблицы разделов MBR и содержания BR. Определяется места нахождения загрузочных записей по завершающей сектор сигнатуре <55h, AAh>. Все поля записей запол-

няются специфицированной информацией. В отдельных случаях восстановление таблиц разделов MBR удается получить низкоуровневым сканированием поверхности жесткого диска встроенной утилитой SETAP, если она имеется в составе его утилит, либо аналогичных утилит в составе специальных программных средст обслуживания жестких дисков.

Разрушение MBR при неустойчивом позиционировании магнитных головок жесткого диска является окончательным и не восстанавливается.

13.5.2. Восстановление информации в файловой системе NTFS

Для обслуживания файловой системы NTFS используется многоуровневый драйвер и система сервисов:

– сервис журнала транзакций;

– диспетчер системы ввода/вывода данных;

– диспетчер временной памяти (КЭШ);

– собственно драйвер NTFS;

– драйвер отказоустойчивости NTFS – FtDisk.

Система отказоустойчивости и управления томами позволяет:

– восстанавливать и перемещать данные из плохих (сбойных) кластеров диска в свободные кластеры с последующей корректировкой цепочек кластеров размещения файла в файловых записях;

– отменять транзакции записи данных при внезапном прекращении процесса;

– поддерживать специальное дублирование, зеркалирование и системы избыточности данных MFT;

– поддерживать системы RAID хранения данных высокой надежности;

– осуществлять сжатие (упаковку) данных;

– изменять объем тома (наращивание его) без необходимости последующего переформатирования.

По этой причине дополнительных мер по восстановлению файловой системы NTFS «ручными» методами не требуется.

Необходимость восстановления информации на жестком диске возникает обычно при неумелом обращении с ней, действии вируса или системных неисправностях и сбоях.

1. Сбой операционной системы Windows – устраняется откатом в точки восстановления, которые необходимо периодически обновлять.

2. Для анализа сбоя процесса загрузки системы необходимо создавать загрузочный диск, исполняя один из разделов процесса установки системы.

3. При ошибочном удалении одного или нескольких файлов на магнитном диске существует так называемая «корзина» – поле временного хранения на незанятом дисковом пространстве файлов, помеченных на удаление. По мере заполнения диска эти области будут использоваться OS и восстановление удаленных файлов будет невозможно.

4. При ошибочном удалении файлов на сервере локальной вычислительной сети, как и в предыдущем случае, возможно восстановление этих файлов в течении определенного времени, зависящем от загрузки сервера, с помощью системных утилит, находящихся в распоряжении администратора локальной вычислительной сети, например, Filer или Novell Client.

5. Офисные приложения OS Windows, использующие большие объемы ОЗУ, имеют сервисы периодической записи на жесткий диск содержания буферов ОЗУ, что также помогает сохранять большие объемы информации, например, результаты дневной работы бухгалтера, работающего в Excel, теряющиеся при неожиданном сбое питания или несанкционированном отключении ПЭВМ.

13.5.3. Основные программные средства для восстановления информации на магнитных носителях

Для устранения повреждений таблиц секторов в MBR существуют простые в использовании программы, которые при помощи имеющихся в их составе алгоритмов анализа файловых структур успешно реконструируют MBR. Например, можно рекомендовать условно бесплатные и достаточно популярные программы:

– программа Partition Magic корпорации PowerQuest (в настоящее время относится к корпорации Simantec www.simantec.com, но в сети всегда можно найти доступную версию для любой OS);

– программа Active@Partition Recovery (www.partitionrecovery.com);

– программа R-Studio (www.r-studio.com) – Комплексное средство для восстановления данных и случайно удаленных файлов. R-Studio поддерживает следующие файловые системы: FAT12/16/32, NTFS, NTFS5, UFS1/UFS2 (FreeBSD/OpenBSD/NetBSD), Ext2FS/3FS (Linux) и др.

13.6. Контроллеры дисковых подсистем

13.6.1. Контроллер накопителей на гибких магнитных дисках

Накопитель на гибких магнитных дисках (НГМД) до недавнего времени широко использовался, как запоминающее устройство (рисунок 83).

НГМД включает в себя три основных компонента (рисунок 84):

1. Блок управления приводом вращения диска.

2. Блок управления перемещением и позиционированием магнитных головок.

3. Блок усилителей записи/чтения данных и аппаратная часть интерфейса связи с контроллером НГМД i8272.

Контроллер НГМД интегрирован совместно с контроллером жесткого магнитного диска в состав контроллера IDE.

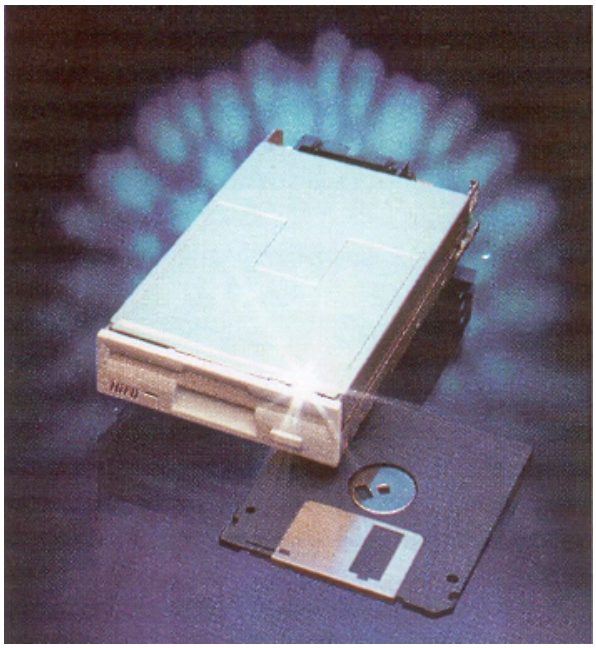


Рисунок 83. – Внешний вид накопителя на гибких магнитных дисках

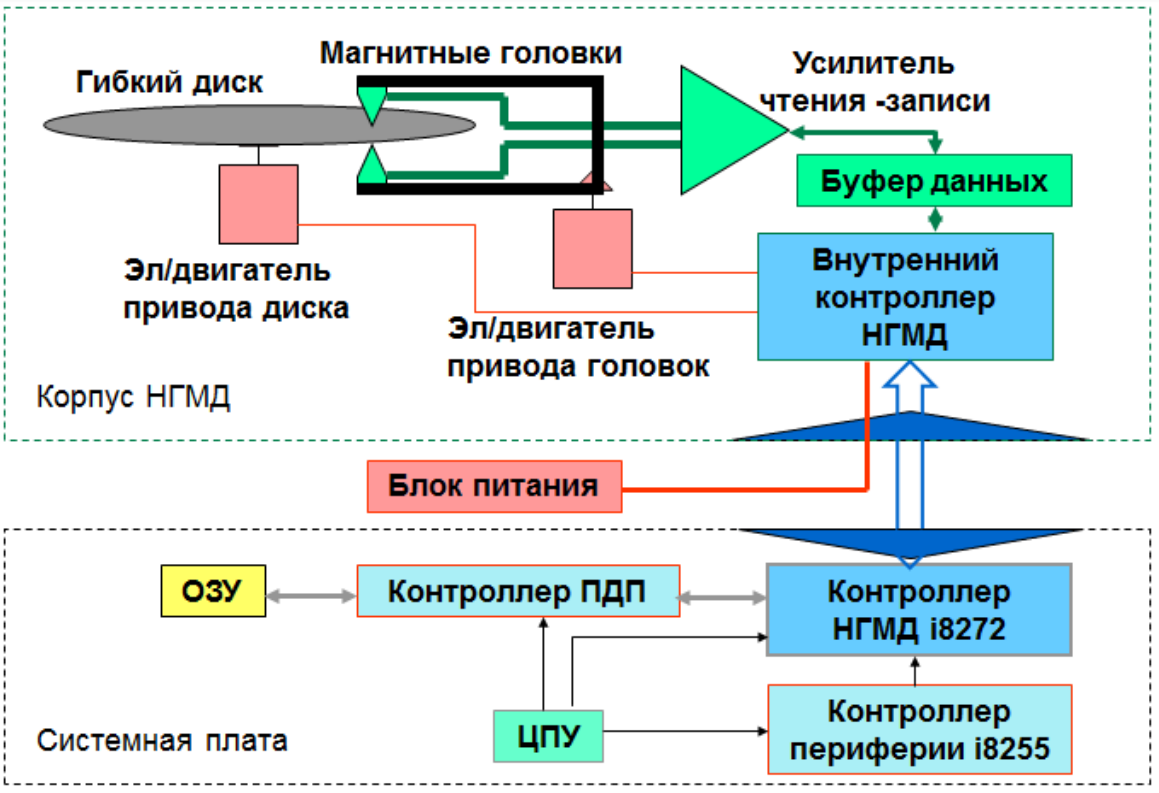


Рисунок 84. – Функциональная схема контроллера НГМД

Основные регистры контроллера:

– Регистр цифрового управления (порт 3F2h):

бит 0 и 1 – выбор устройства – 00-A, 01-B;

бит 2 – сброс/работа контроллера;

бит 3 – разрешение использования ПДП;

бит 4 и 5 – включение/выключение привода диска НГМД.

– Регистр состояния дисковода (порт 3F4h):

бит 0 и 1 – дисковод в режиме перемещения головок;

бит 4 – обрабатывается команда ввода/вывода;

бит 5 – используется режим ПДП;

бит 6 = 1 – вывод данных в направлении ЦПУ, 0 – от ЦПУ;

бит 7 – готовность к передаче данных.

– Регистр передачи команд/данных (порт 3F5h) используется для организации доступа к внутренним регистрам контроллера ST0 – ST3.

– Регистр управления скоростью передачи данных (порт 3F7h): бит 0 и 1 – установки скорости передачи данных: 00 – 500 Кб/с, 01 – 300 Кб/с, 10 – 250 Кб/с, 11 – 125 Кб/с.

В состав команд контроллера входит 15 команд:

1. Чтение данных с диска.
2. Запись данных на диск.
3. Чтение удаленных данных.
4. Запись удаленных данных.
5. Чтение дорожки.
6. Чтение идентификатора диска.
7. Форматирование дорожки.
8. Сканирование дорожки.
9. Сканирование секторов.
10. Позиционирование головки.
11. Калибровка 0-й дорожки.
12. Читать прерванное состояние.
13. Определить параметры НГМД.
14. Читать состояние накопителя.
15. Идентификация недопустимой команды (код ошибки).

Значения основных констант НГМД:

– скорость вращения гибкого диска – 6 об/с;

– время старта НГМД (от подачи команды на чтение/запись до завершения калибровки) – 250 мс;

– время подвода головки от 0-й до заданной дорожки – 35 мс;

– время шага головки (переход между соседними дорожками) – 15 мс.

Для обеспечения работы с командами накопителя на гибких магнитных дисках служит прерывание BIOS 13h с прямым обращением к портам НГМД и прямой адресацией области вывода данных в ОЗУ.

13.6.2. Контроллер накопителей на жестких магнитных дисках

Накопители на жестких магнитных дисках (НЖМД) используются, как основные носители информации в ПЭВМ и серверах. Накопитель НЖМД

имеет от одного до четырех жестких дисков, собранных в пакет, и блок поворотных головок по одной на одну сторону каждого жесткого диска, укрепленных на общем коромысле (рисунок 85).

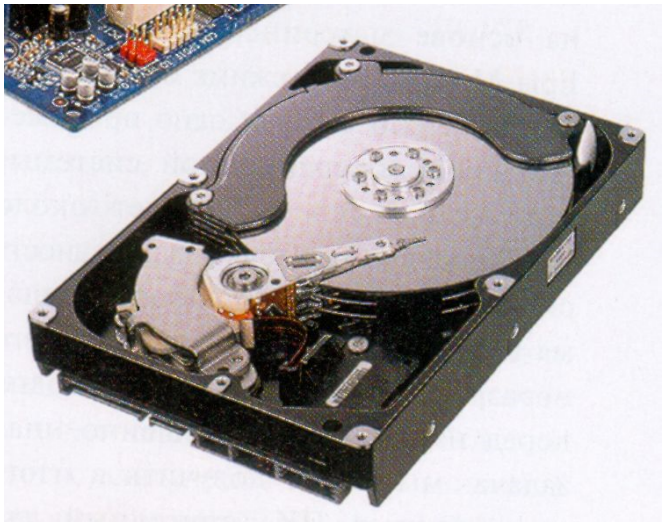


Рисунок 85. – Внешний вид накопителя на жестких магнитных дисках

Структурная схема контроллера НЖМД аналогична схеме контроллера НГМД, но в качестве интерфейса связи используется расширенный интерфейс стандарта ATAPI/EIDE (AT Attachment Packet Interface/Enhanced Integrated Drive Electronics) (parallel-PATA или Serial-SATA), либо усовершенствованный SCSI (II, III, IV) (Small Computer System Interface) (рисунок 86).

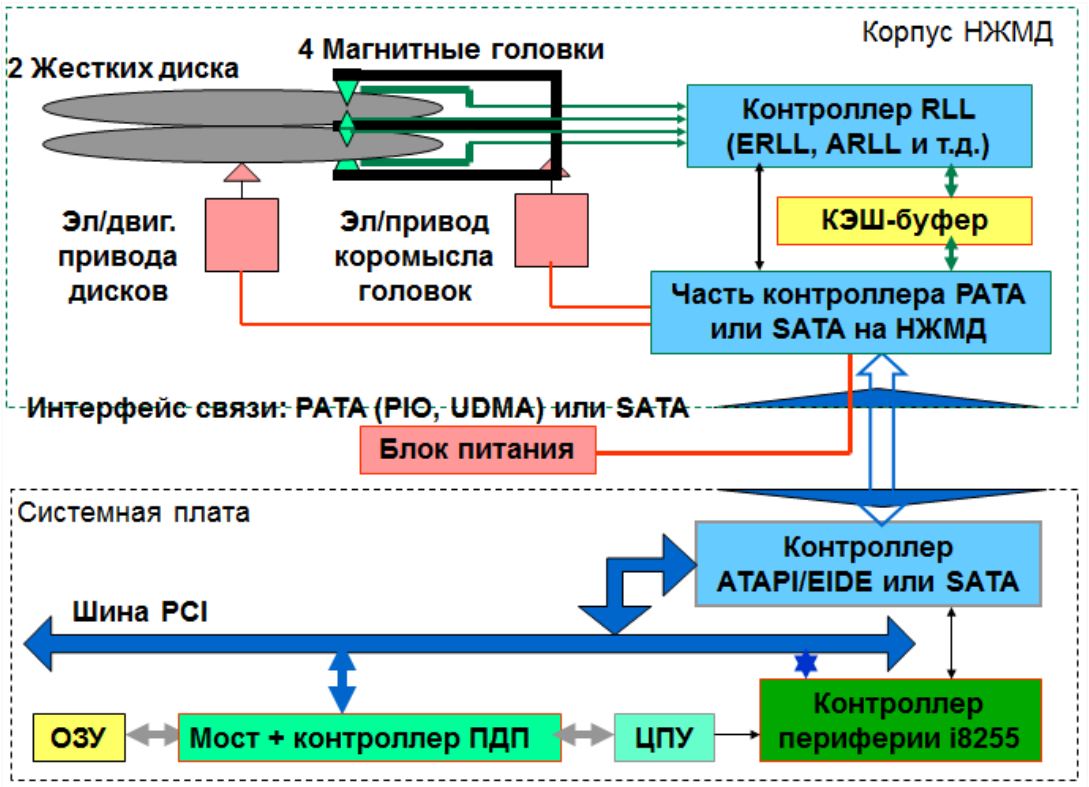


Рисунок 86. – Функциональная схема контроллера НЖМД

Основные регистры команд контроллера НЖМД ATAPI:

Порты PATA: канал № 0 1F0h – 1F7h, 3F6h; канал № 1 170h – 177h, 376h.

Порты SATA: 1-я группа НЖМД F000h – F007h; 2-я группа НЖМД F008h – F00Fh.

Основное назначение портов НЖМД на примере PATA канала № 0:

1F0h – буферный регистр ввода/вывода данных НЖМД.

1F1h – ввод значения старшего цилиндра прекомпенсации ($N_{цил}/4$), чтение признаков ошибок исполнения последней команды:

бит 0 – адресный маркер данных не найден;

бит 1 – ошибка калибровки (на дорожке № 0);

бит 2 – команда отвергнута контроллером;

бит 4 – ID сектора не найден или не существует;

бит 6 – ошибка данных не корректируется;

бит 7 – плохой кластер.

1F2h (r) – счетчик числа секторов для операций чтения/записи, чтение – текущий номер логического сектора для операций чтения/записи.

1F4h + 1F5h – текущий 16-битный номер цилиндра для операций r/w.

1F6h (w) – выбор НЖМД в пределах текущего канала (в примере – 0):

биты 0–3 – номер выбранной магнитной головки;

бит 4 – порядковый номер накопителя (= 0 – 1-й, = 1 – 2-й);

биты 5 и 6 – задание размера сектора (00 – 128, 01 – 256, 10 – 512, 11 – 1024 Кб).

Скорость вращения диска НЖМД – 5000, 7200, 10000, 12000 и более об/мин, емкость – до нескольких терабайт.

Интерфейсы связи IDE (Integrated Drive Electronics) НЖМД с частью контроллера, расположенной на системной плате ПЭВМ, были реализованы с применением параллельной схемы передачи информации по 40- или 80-жильному кабелю (40 дополнительных жил исполняли роль экрана-заземления) с использованием методов пакетной (PIO) через ЦПУ или блоковой (UDMA) через ПДП в ОЗУ передачи информации.

К одному контроллеру можно было подключить только два НЖМД. В дальнейшем интерфейс связи с НЖМД был заменен на последовательный (SATA) с использованием витой пары. В промышленных ЭВМ для ускорения передачи информации от нескольких НЖМД использовались интерфейсы связи SCSI (Small Computer Systems Interfacec) с параллельной 16- или 32-разрядной шиной передачи данных.

Суть интерфейса состоит в том, чтобы обеспечить гибкую схему подключения до 7, как внутренних НЖМД, так и внешних накопителей, и управления этими устройствами с максимальной скоростью их работы как единого, но делимого механизма. Это достигалось за счет создания внешней шины данных SCSI с использованием HOST-контроллера шины и SCSI-контроллеров устройств, подключаемых к этой шине.

Схема контроллера связи SCSI показана на рисунке 87.

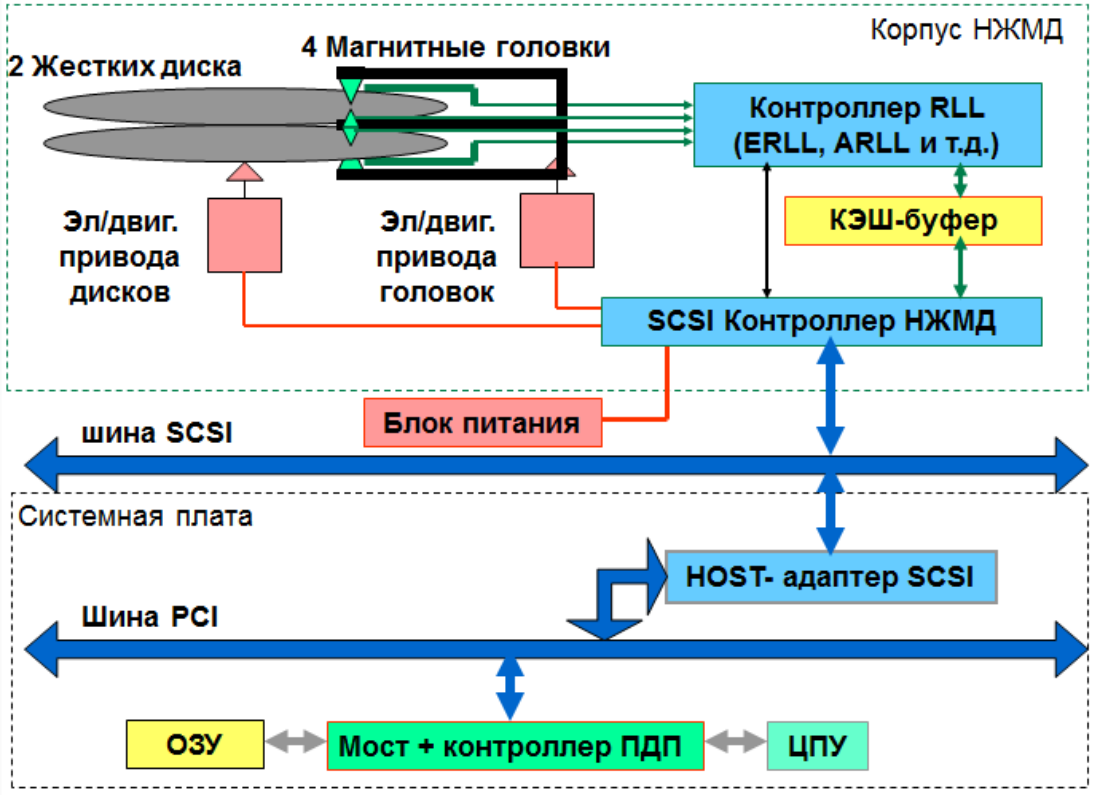


Рисунок 87. – Функциональная схема контроллера SCSI

Сложность схемы интерфейса приводила к существенному удорожанию конструкции, поэтому такой интерфейс подключения НЖМД обычно использовался в промышленных ЭВМ и серверах. В дальнейшем параллельный вид передачи информации в интерфейсе SCSI также был заменен на последовательный.

13.7. Основные типы внешних накопителей

13.7.1. CD-ROM / RW

CD-ROM накопитель на компакт дисках был разработан в 1978 г. Размеры диска $Ø_{нар}$ = 120 мм, $Ø_{вн}$ = 15 мм, толщина – 1,2 мм (рисунок 88).

Запись информации осуществляется по спирали от центрального отверстия к периферии и заканчивается за 5 мм до края диска. Длина спирали ~5 км, она разбита на 333 000 блоков. В каждом блоке 2352 байта. Объем CD-диска – 650 Мб. Стандартная скорость считывания X = 150 Кб /с. Увеличение скорости считывания определяются кратностью этому числу, например, 52X = 52 × 150 = 78000 Кб /с.

Структурная схема контроллера CD аналогична схеме контроллера НЖМД, метод кодирования напоминает RLL, но в качестве носителей информации используются границы контрастных областей (пятен) на дорожках диска.

Рисунок 88. – Внешний вид накопителя на компакт дисках

Расширена система контроля и исправления ошибок чтения (ECC) для чего 288 байт блока используются как контрольные, что позволяет восстанавливать до 1000 байт блока. Контроллер CD так же, как и контроллер НЖМД, содержит КЭШ-буфер для ускорения обмена информацией с ОЗУ.

В качестве интерфейса связи контроллера CD с шиной ЦПУ используется расширенный интерфейс стандарта ATAPI/EIDE (AT Attachment Packet Interface/Enhanced Integrated Drive Electronics) parallelATA.

Поскольку данные на CD-диске не фрагментированы, то их оглавление и поиск не представляют таких трудов, как на НЖМД. В качестве файловой системы на CD используется стандарт ISO 9660 VTOC (Volume Table of Contens), который представляет собой последовательность записей:

Системная область содержит: идентификатор CD-диска и файловой системы, синхронизирующую последовательность, оглавление тома и таблицу размещения данных. Затем идет область данных.

В качестве параметров используются: виток, дуга, блок (аналогично цилиндру, сектору и кластеру для НЖМД).

Существуют другие форматы файловых систем, например, формат CD-DA (Compakt Disk Digital Audio) для аудиозаписей или CDK – диск ф.Kodak и другие частные форматы фирм-производителей видеопродукции.

Следует отметить многосессионный формат записи CD-XA для обеспечения возможности дозаписи CD-диска. В качестве носителя информации используются темные штрихи на светлом фоне. Глубина штриха – 0,12 мкм, ширина штриха – 0,6 мкм, расстояние между витками – 1,6 мкм. Мощность лазера – 0,5 МВт (чтение), 4–14 МВт (запись), tпятна = 250–400 °С.

13.7.2. DVD-ROM / RW

DVD (Digital Video Disk) – дисковод для компакт-дисков с лазерным методом записи-считывания данных. По внешнему виду, основным параметрам конструкции, контроллеру и построению областей хранения данных полностью аналогичен дисководу типа CD. Отличается от CD повышенной

в 6,5 раза плотностью записи данных на диск, достигаемой за счет уменьшения частоты лазера и метода упаковки данных. Объем DVD – 4,7 Гб.

Структурная схема контроллера DVD аналогична схеме контроллера CD, в качестве интерфейса связи также используется расширенный интерфейс ATAPI / EIDE PATA, возможно использование интерфейса SATA или SCSI. В настоящее время широко используется двухсторонняя и двухуровневая запись дисков, внедряется формат HD DVD, что позволяет увеличивать объем диска DVD до 45 Гб, но в целом формат DVD считается уже устаревшим. Сейчас формат DVD уже фактически заменен на формат BR (Blu-ray), обладающий лучшими характеристиками уплотнения данных.

13.7.3. Blu-ray

Blu-ray (синий луч) Disc (BD) – формат оптических дисков, используемый для записи, перезаписи, воспроизведения с повышенной плотностью и хранения больших цифровых объемов данных, включая видео высокой чёткости. Этот формат был разработан консорциумом BDA (Blu-ray Disc Association) совместно с группой ведущих более чем 180 компаний бытовой электроники, персональных компьютеров и медиа-производителей со всего мира.

Формат предлагает емкость:

– однослойный диск Blu-ray (BD) – 23,3 Гб (25 Гб);

– двуслойный диск – 46,6 Гб (50 Гб);

– трёхслойный диск – 100 Гб;

– четырёхслойный диск – 128 Гб;

– ещё продемонстрированы 16- и 20-слойные диски на 400 и 500 Гб.

Как и обычные компакт-диски и DVD-диски, Blu-Ray предоставляет широкий спектр форматов, включая ROM / R / RW. Поддерживаются следующие Blu-Ray форматы:

– BD-ROM – только для чтения HD фильмов, игр, программного обеспечения и т.д.;

– BD-R – записываемый формат для HD видеозаписи и хранения компьютерных данных;

– BD-RE – перезаписываемый формат для HD видеозаписи и хранения компьютерных данных;

– BD/DVD – гибридный формат, который сочетает в себе Blu-ray и DVD на одном диске и может быть воспроизведен как в Blu-ray-плеере, так и в DVD-плеере.

Blu-ray использует файловую систему Universal Disk Format (UDF) 2.50 или 2.60. В соответствии с Blu-ray Disc спецификации, 1x скорость определяется как 36 Мбит/с. Также Blu-ray имеет новую систему коррекции ошибок, более надежную и эффективную, чем у DVD-дисков.

13.7.4. Дисковод на магнитооптических дисках (НМОД)

НМОД использует для записи данных магнитные головки с подогревом области записи лазером (рисунок 89). Запись на холодный диск невозможна, за счет чего достигается высокая степень защиты данных от стирания. НМОД используется в качестве архивирующих накопителей в профессиональных системах записи данных, т.к. информация на МО-дисках может сохраняться до 20 – 25 лет.

Структурная схема контроллера НМОД аналогична схеме контроллера НЖМД с учетом добавления в схему лазера подогрева пятна, совмещенному с магнитной головкой.

Рисунок 89. – Внешний вид накопителя на магнитооптических дисках

Запись осуществляется на одну сторону диска. Кодирование записи выполняется по методу ERLL. в качестве интерфейса связи используется расширенный интерфейс стандарта ATAPI/EIDE (PATA), либо USB.

В бытовых системах не используется по причине высокой стоимости и малой производительности (малой емкости, медленной записи и скорости передачи данных по сравнению с другими типами накопителей).

13.7.5. Стример – накопитель данных на магнитной ленте

Накопитель на магнитной ленте (НМЛ) используется только как архиватор данных из-за его высокой надежности, проверенной временем (срок сохранности данных – до 35 лет) (рисунок 90). НМЛ не допускает использования стандартных файловых систем, т.к. запись данных осуществляется в строго последовательной форме (без дефрагментации). Скорость записи/чтения данных малы по сравнению с другими типами накопителей, поэтому стример применяется только в профессиональных системах хранения данных.

Структурная схема привода НМЛ аналогична схеме контроллера НЖМД, но отсутствует привод головки записи. Сама головка закреплена неподвижно. В качестве контроллера чтения/записи данных используется специализированный контроллер записи, совмещенный с системой ECC.

Рисунок 90. – Внешний вид накопителя на магнитной ленте

Интерфейс связи построен на базе расширенного интерфейса стандарта ATAPI/EIDE (PATA), но чаще встречается интерфейс внешней шины SCSI или USB.

13.7.6. Express Card-накопитель

Express Card-накопители (рисунок 91) используются как элементы дополнительного объема внешней памяти для различных мобильных устройств (цифровых фотоаппаратов и кинокамер, мобильных телефонов, смартфонов, ноутбуков, сканеров штрих-кода, кассовых аппаратов и коммуникаторов).

Рисунок 91. – Внешний вид Express Card-накопителя

Структурная схема контроллера Express Card-накопителя аналогична схеме контроллера ОЗУ, обеспечивает совместимость со стандартом Express Card 1.2. В качестве интерфейса связи используется внешний интерфейс

шины PCI Express 2.0 – слот PCI Express x1, что обеспечивает достаточную скорость обмена информацией ОЗУ с Express Card, т.к. ячейки памяти при записи имеют пониженную скорость обмена информацией (до 2 Мб/с). Объем Express Card увеличен до 64 Гб.

Для подключения Express Card требуется специализированный набор разъемов различной конфигурации и адаптер связи с шиной PCI Express x1.

13.7.7. Flash – накопитель на микросхемах памяти

Твердотельный накопитель Flash используется, как временный мобильный архив вместо гибких дисков, которые, по-видимому, уже никогда не будут использоваться (рисунок 92).

Рисунок 92. – Внешний вид Flach-накопителя

К сожалению, надежность Flash-накопителей невысока и обычно не превышает 10 000 циклов обращения к ячейкам памяти. Скорость передачи информации также существенно ниже, чем в жестких дисках.

Структурная схема контроллера Flash-накопителя аналогична схеме контроллера ОЗУ, ячейки памяти имеют одноуровневую структуру и объединяются в блоки по 4 Кб (для записи) и по 512 байт (для чтения и стирания), поэтому для Flash-накопителей может быть использована любая файловая система. В качестве интерфейса связи используется интерфейс внешней шины USB или USB2. Объем современных Flash-накопителей обычно составляет 8 – 16 Гб при минимальных габаритных размерах.

13.7.8. Твердотельный диск-накопитель

Накопитель SSD (Solid State Disk – твердотельный диск) не содержит в составе своей конструкции подвижных механических элементов (рисунок 93) и, по-видимому, способен превзойти по всем характеристикам жесткие диски.

К сожалению, современные SSD-накопители не обеспечивают необходимый уровень сохранности информации при большом количестве циклов чтения/записи поэтому их не рекомендуется использовать в серверах.

SSD по схеме использования памяти существенно отличается от схемы Flash-накопителей:

– Конструкция ячеек памяти SSD является многоуровневой.

– Повышен ресурс обращений к ячейке (до 100 000 обращений).

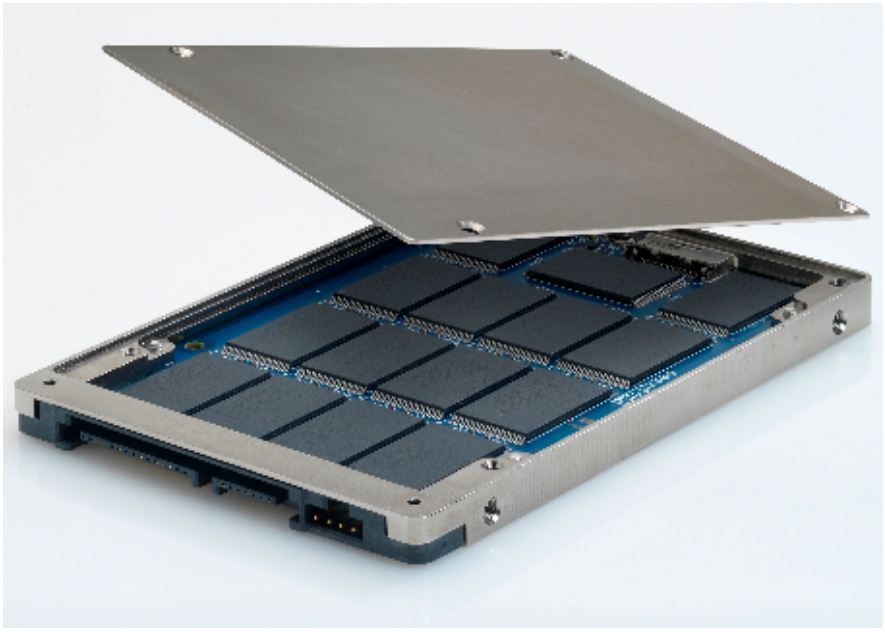


Рисунок 93. – Внешний вид SSD-накопителя

– Структура контроллера обеспечивает равномерную выработку ресурса ячеек памяти, что позволяет обеспечить 5-летнюю гарантию устройства.

– Структура распределения информации – секторная (4 Кб на сектор), что повторяет кластерную структуру жесткого диска и позволяет использовать любую известную файловую систему.

В настоящее время производителями накопителей информации ведется большая работа по поиску новых материалов и разработке новых конструкций накопителей, обеспечивающих увеличение объема, скорости передачи информации, надежности и сроков её хранения.

Тема 14. Архитектура видеосистем ПЭВМ

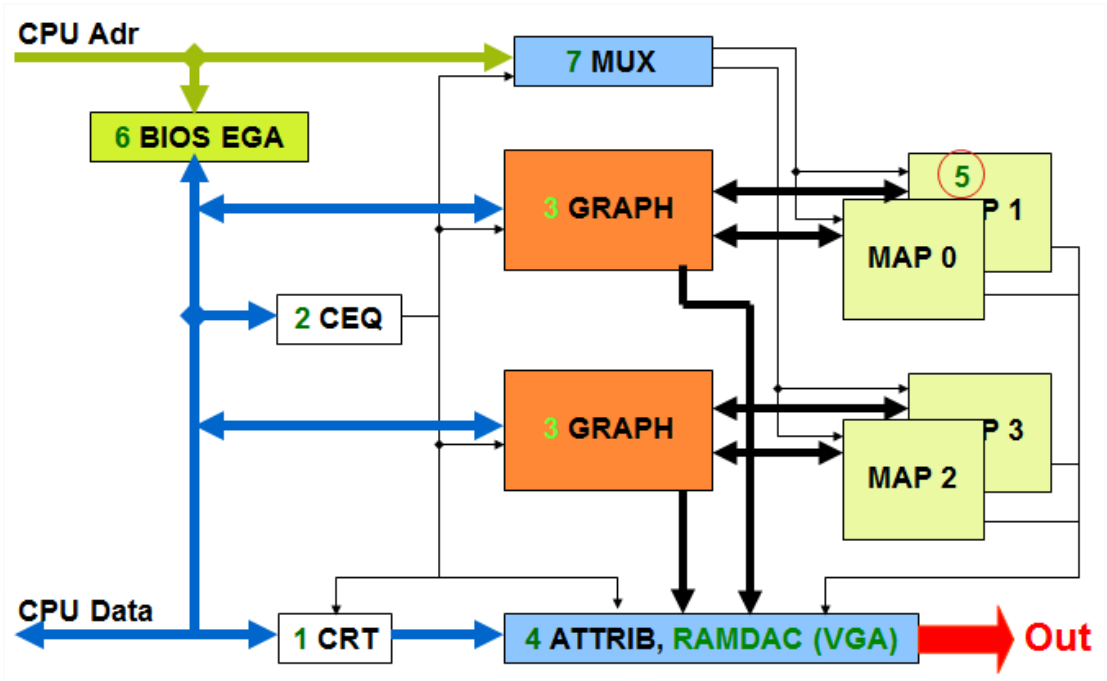
Аппаратные средства для вывода информации на экран включают в себя видеоадаптер (графический контроллер), выполненный для слота расширения шин ISA, PCI, AGP, PCI-e или встроенный на системной плате, а также монитор любого исполнения или телевизор.

Конструктивно видеоадаптер представляет собой самостоятельное устройство, управляемое семейством собственных процессоров (от 2-х до 400 и более), сравнимых по мощности с центральным процессором ПЭВМ. Видеопамять современного графического контроллера достигает размера 4 Гб. В состав видеоадаптера может входить блок TV-тьюнера для прямого приема телепередач, а также блок сопряжения с другими адаптерами для повышения скорости обработки потока видеоинформации (например, технология ATI CrossFireX).

14.1. Графический контроллер EGA

В качестве примера для изучения наследуемой архитектуры видеоконтроллеров в упрощенном виде рассмотрим видеоконтроллер EGA.

Расширенный Графический Адаптер (Enhanced Graphics Adapter – EGA) был разработан фирмой IBM в сентябре 1984 г., как продолжение совершенствования графических контроллеров MDA и CGA. EGA представляет собой графический контроллер, обеспечивающий возможность работы в различных видеорежимах совместно с цветными или монохромными мониторами с цифровыми входами. Кроме того, он обеспечивает возможность работы со световым пером. Графический контроллер EGA может функционировать в текстовом и нескольких графических режимах и обладает возможностью загрузки в видеопамять шрифтов в алфавитно-цифровом режиме. Функциональная схема графического контроллера EGA представлена на рисунке 94.



Рисунке 94. – Функциональная схема графического контроллера EGA

Каждый блок видеоконтроллера представляет собой законченный цифровой автомат, состоящий из операционного блока и управляющего автомата с системной (жесткой) логикой. Назначение блоков видеоадаптера:

– ***Блок управления разверткой изображения (CRT Controller)*** вырабатывает распределенную во времени последовательность сигналов горизонтальной и вертикальной развертки изображения на экране, содержит начальный адрес вывода в видеобуфер, а также координаты и графическое изображение курсора.

– ***Блок синхронизации (Sequencer)*** генерирует тактовые сигналы и сигналы синхронизации доступа к видеопамяти. Данным устройством

также обеспечивается возможность доступа к видеопамяти со стороны центрального процессора в специально выделенные моменты времени в промежутке между интервалами времени, необходимыми для доступа к видеопамяти в процессе регенерации изображения на экране. В этом же блоке содержатся регистры управления записью данных в битовые плоскости.

– ***Графический процессор (Graphics Controller)*** управления данными. В графическом контроллере EGA впервые применили многопроцессорную архитектуру – два графических процессора. Каждый из них выполняет обработку двух своих видеоплоскостей (видеостраниц). В архитектуре VGA используется до 400 графических процессоров. Они предназначены для выполнения элементарных операций с вычисляемой видеоинформацией (векторной графикой) при заполнении видеостраниц по задаваемому центральным процессором алгоритму построения изображения (так называемые графические ускорители).

В графических режимах данные из видеопамяти пересылаются в микросхему контроллера атрибутов последовательно. В текстовых режимах данные пересылаются в параллельной форме в обход графического контроллера.

– ***Контроллер атрибутов (Attribute controller).*** В контроллере атрибутов устанавливается цветовая палитра из 16 цветов, каждый из которых может быть определен независимо от остальных цветов. На вход монитора подается 4-битовый код цвета. Этой же микросхемой выполняются действия по управлению мерцанием и подчеркиванием. Контроллер получает данные из видеобуфера и преобразует их в управляющие сигналы, подаваемые на вход монитора.

– ***Видеобуфер (Display MAP).*** Размер видеобуфера (называемого также видеопамятью или памятью адаптера EGA) равен 64 Кб. Адресное пространство видеобуфера встроено в адресное пространство ОЗУ, поэтому оно доступно центральному процессору, как на чтение, так и на запись, и состоит из 4-битовых видеоплоскостей по 16 Кб. Существует возможность расширения буфера до 128 Кб. На плате расширения установлены разъемы для подключения еще 128 Кб памяти, что позволяет довести размер видеобуфера EGA до 256 Кб. При этом в каждую битовую плоскость добавляется два дополнительных банка памяти по 16 Кб.

С целью совместимости с более ранними моделями видеоадаптеров, адреса видеобуфера могут изменяться. Возможны 4 варианта. Видеобуфер может быть установлен объемом 128 Кб и начинаться с сегментного адреса A0000, объемом 64 Кб с адреса A0000, объемом 32 Кб с адреса B0000 и тем же объемом, но с адреса B8000.

– ***Базовая система ввода/вывода (BIOS EGA)*** находится в памяти специального ПЗУ установленного на плате адаптера. BIOS EGA обычно перезаписывается или встраивается в область ОЗУ с адреса C0000 и объединяется с базовой системой ввода/вывода (BIOS системной платы). Здесь размещаются

шрифты, используемые для генерации символов и управляющие программы видеоадаптера. Размер ПЗУ – 16 Кб (или 32 Кб для видеоадаптера VGA).

– ***Мультиплексорные схемы (MUX)*** для переключения битовых плоскостей при чтении данных видеобуфера блоком управления CRT, центральным процессором или контроллером атрибутов. На плате установлены также два тактовых генератора синхросигналов с частотами 14 и 16 МГц, определяющие частоту вывода точек растра и 4 внешних регистра ввода/вывода, не входящие в состав схемы.

Основной недостаток адаптера EGA – цифровой видеосигнал на выходе, требующий дальнейшей обработки и перевода в аналоговый вид RGB-сигнала с его аппаратной поддержкой. Это требовало установки дополнительных устройств в EGA-мониторах.

14.2. Графический контроллер VGA

Графический адаптер VGA (Video Graphic Array) с точки зрения функциональных возможностей и производительности является постоянно улучшаемой версией графического адаптера EGA. Функционально схемные решения EGA и VGA совпадают. Исключение составляет архитектура контроллера атрибутов, в состав которой введен блок цифроаналогового преобразования выходного сигнала и расширенная область быстродействующей регистровой памяти для размещения 256-цветной перепрограммируемой палитры.

Графический контроллер VGA поддерживает более широкий спектр видеорежимов, особенно при использовании мониторов с изменяемой рабочей частотой. Так же, как и графический адаптер EGA, VGA в своем составе содержит несколько программно-управляемых компонентов: блок управления разверткой видеоизображения на экране монитора (CRT), блок синхронизации (SEQ), графические контроллеры (GRAPH) и контроллер атрибутов. Каждый из этих функциональных блоков адаптера управляется программно. Программы обслуживания VGA в составе базовой системы ввода/вывода (BIOS VGA) доступны по прерыванию 10h. Использование функции с номером 0 данного прерывания позволяет установить адаптер в один из 24 стандартных видеорежимов, поддерживаемых BIOS VGA.

Каждый блок VGA содержит в своем составе ряд регистров, используемых для управления функционированием адаптера.

Для каждого видеорежима в программах BIOS содержится соответствующая таблица значений регистров видеоадаптера, в связи с чем, в большинстве случаев, для установки требуемого видеорежима вместо непосредственной записи в регистры адаптера достаточно воспользоваться программами BIOS.

14.2.1. Программирование режимов использования VGA

Видеорежимы характеризуются следующими параметрами:

– вертикальным разрешением (количеством строк растра на экране);

– горизонтальным разрешением (количеством символов или пикселов в строке);

– представлением данных в буфере;

– атрибутами вывода (цвет, мерцание и т.п.).

Как и видеоадаптер EGA, VGA поддерживает два основных режима использования: алфавитно-цифровой и графический.

Алфавитно-цифровой режим используется в настоящее время как базовый, наладочный, часто называемый режимом DOS, хотя его параметры введены в систему BIOS системной платы и видеоадаптера и от дисковой операционной системы никак не зависят. В этом режиме графические процессора не используются. Вся информация, предназначенная для вывода на экран, формируется центральным процессором в окне видеобуфера с использованием графических примитивов (текселов) таблицы символов ASCII.

Цвета примитивов определяются вспомогательной 16-цветовой палитрой.

Графический режим, при котором используются графические процессоры для заполнения видеоплоскостей, является основным режимом работы графического адаптера VGA.

При программировании графических режимов VGA наиболее важным является управление горизонтальным и вертикальным разрешением изображения на экране дисплея. Любые доступные характеристики режима работы видеоадаптера легко определить, воспользовавшись закладкой.

Минимальное разрешение экрана, доступное для любого графического контроллера VGA – 800 × 600, т.е. 800 пикселов по горизонтали и 600 строк пикселов по вертикали. Более высокое разрешение доступно при соответствующей аппаратной реализации видеоконтроллера и соответствующей разрешающей способности монитора.

Режим работы видеоадаптера VGA программируется таким образом, чтобы время, необходимое для вывода данных из видеобуфера на экран было всегда меньше общего количества времени развертки одного кадра. Это дает возможность вывода на экран бордюра (overscan), окаймляющего собственно выводимое изображение, что позволяет центрировать изображение на экране без выброса луча за края его поля.

Центральным моментом при программировании нестандартных видеорежимом является выбор таких значений временных параметров управления экраном, чтобы не выйти за допустимые пределы частотных характеристик используемого монитора. После выбора значений временных параметров видеорежима можно приступать к программированию основных блоков видеоадаптера VGA, что включает в себя:

– программирование блока управления ЭЛТ;

– программирование блока синхронизации;

– задание частоты генератора пикселов;

– задание высоты символов (в строках растра);

– модификация требуемых переменных BIOS VGA.

Доступ к регистрам VGA этих блоков осуществляется через порты ввода/вывода путем использования команд IN и OUT или специальных функций BIOS, эквивалентных по своим действиям этим командам. Для доступа к программам BIOS используется прерывание с номером 16 (10h).

14.2.2. Регистровая модель видеоконтроллера VGA

Основные регистры графического контроллера VGA представлены следующими блоками.

Регистры блока CRT:

– Регистр состояния видеосигнала: порт 3C2h (только чтение): бит 7 – вертикальный обратный ход луча; порт 3BAh (только чтение):

бит 0 – гашение видеосигнала;

бит 1 – световое перо активировано;

бит 2 – кнопка светового пера активирована;

бит 3 – идентифицирован обратный ход луча.

– Регистры видеоконтроллера: 25 внутренних регистров данных доступны через индексный регистр 3D4h. Чтение/запись этих регистров через порт 3D5h. Назначение внутренних регистров:

00h – 0Fh – константы синхронизации луча, ход, границы, позиция курсора, строки, разрешение.

10h – 11h – обратный ход луча, световое перо.

12h – 18h – работа с видеопамятью, сканирование, синхронизация.

– Регистры управления: порт 3CCh (только чтение) / порт 3C2h (только запись): бит 0 – содержание адресного пространства (mono/color);

бит 1 – доступ к ОЗУ VGA;

бит 4 – отключение видеосигнала;

бит 5 – выбор режима адресации по плоскостям;

биты 2, 3 и 6, 7 – выбор горизонтального и вертикального тактов.

Регистры блока GRAPH:

– Регистры управления: 9 внутренних регистров данных доступны через индексный регистр 3CEh. Чтение/запись этих регистров через порт 3CFh. Назначение регистров:

00h – 4-битовый номер видеоплоскости для заполнения.

01h – флаг заполнения (при установке флага ОЗУ заполняется не данными центрального процессора, а битом-заполнителем).

02h – регистр сравнения и сохранения цвета в плоскостях.

03h – регистр условий модификации данных в видеоплоскостях.

04h – номер плоскости, доступной для чтения.

05h – режим записи в ОЗУ.

06h – выбор адреса видеопамяти.

07h – флаг запрета сравнения видеоплоскостей.

08h – биты маски записи в плоскости: 1 – запись бита из байта данных, 0 – запись из регистра-защелки.

Регистры блока ATRIB:

– Регистры базового цвета: 21 внутренний регистр данных доступны через индексный регистр 3C0h. Чтение/запись этих регистров через порт 3DAh. Назначение регистров:

00h – 0Fh – 16 регистров базовой 4-битовой палитры цветов.

10h – регистр декодирования байта-атрибута (графика/текст).

11h – цвет рамки.

12h – цвет плоскости ОЗУ.

13h – режим разрешения экрана (текстовый/графический).

14h – запоминает и определяет цвета по умолчанию (байт-атрибут).

– Регистры генератор последовательностей: 5 внутренних регистров данных доступны через индексный регистр 3C4h. Чтение/запись этих регистров через порт 3C5h. Назначение регистров:

00h – сброс генератора для сохранения ОЗУ.

01h – режим доступа центрального процессора к видеопамяти.

02h – 4 младших бита доступ к плоскостям видеопамяти (всего 4 (№ 0 – 3) плоскости, каждая вписывается в одно и то же адресное пространство, доступное центральному процессору и замещается побитово).

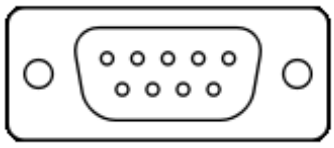
03h – выбор таблицы знакогенератора (всего 8 таблиц).

04h – регистр атрибутов текселов в текстовом режиме.

– Регистры блока доступа к памяти и АЦП (RAMDAC): 255 внутренних 3-байтовых регистров палитры RGB-составляющих оттенков цветов доступны через индексный регистр 3C6h. Чтение трех регистров палитры RGB последовательно через порт 3C7h. Запись трех регистров палитры RGB последовательно через порт 3C8h. Чтение/запись регистров палитры отдельно по составляющим – порт 3C9h.

14.2.3. Разъемы видеоадаптера для подключения монитора

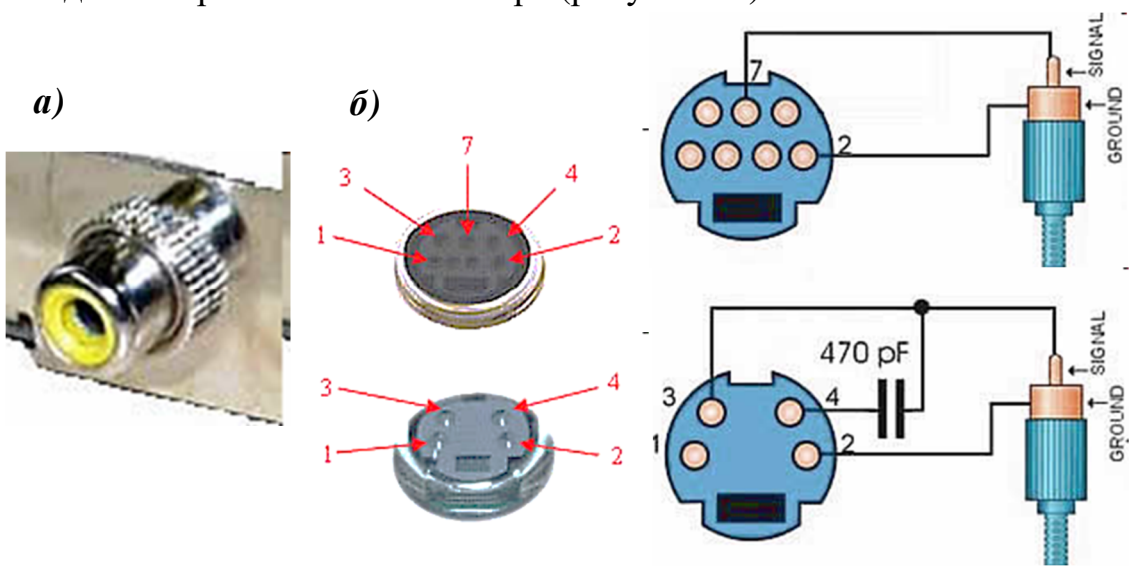
Разъем EGA содержит 9 контактов и от разъема RS232C отличается тем, что на розетке разъема расположены гнезда.



Назначение контактов разъема:

1 – земля;
2 – вторичный красный / интенсивность;
3 – первичный красный;
4 – первичный зелёный;
5 – первичный синий;
6 – вторичный зелёный / интенсивность;
7 – вторичный синий / интенсивность;
8 – строчная синхронизация;
9 – кадровая синхронизация.

Разъем EGA-VGA RCA (Cinch) или «тюльпан» **–** композитное или частотносовмещенное PAL-SECAM соединение. Используется для передачи сигнала в композитном виде. Разъёмы RCA присутствуют, фактически, у каждого современного телевизора (рисунок 95).



а) – «тюльпан», б) – S-video различной комплектации
Рисунок 95. – Разъемы EGA-VGA RCA (Cinch)

Они используются как для передачи видео (обычно цвет разъёмов – жёлтый), так и для передачи аудио (цвет разъёмов – белый и красный).

При передаче сигнала в композитном виде через разъёмы RCA используется полоса пропускания около 3 МГц, следствием чего - относительно невысокая чёткость изображения (не более 300 линий).

К тому же при передаче композитного сигнала по одному физическому каналу в ограниченной полосе частот невозможно полностью разделить яркостную (Y) и цветовую (C) компоненты, что создаёт эффект цветовых перекрёстных искажений (напоминает "сеточку"), особенно хорошо заметных на мелких контрастных деталях.

Разъем EGA-VGA S-video. S-Video обеспечивает заметно лучшее качество, нежели с использованием композитного соединения (см. рисунок 95).

Достигается это тем, что яркостный сигнал (Y), несущий и синхроимпульсы, передаётся отдельно от цветового сигнала (C), в результате чего исчезают цветовые перекрёстные искажения, возникающие при композитном подключении, и повышенной до 6 МГц полосой пропускания, чем обеспечивается чёткость до 500 линий.

Назначение контактов разъема S-video (см. рисунок 95):

1 – земля для яркостного (Y) сигнала.
2 – земля для цветового (C) сигнала.
3 – яркостный (Y) сигнал.
4 – цветовой (C) сигнал.
7 – композитный сигнал.

Разъемы видеоконтроллера VGA показаны на рисунке 96.

Рисунок 96. – Разъемы контроллера VGA для подключения монитора

В настоящее время видеоадаптер VGA для увеличения четкости изображения на экране монитора большой разрешающей способности имеет не только аналоговый выходной сигнал с возможностью изменения напряжения (VGA D-sub), но и несколько каналов типа «витая пара» для передачи цифрового видеосигнала в последовательном виде.

На рисунках 97 – 99 показаны схемы таких разъемов.

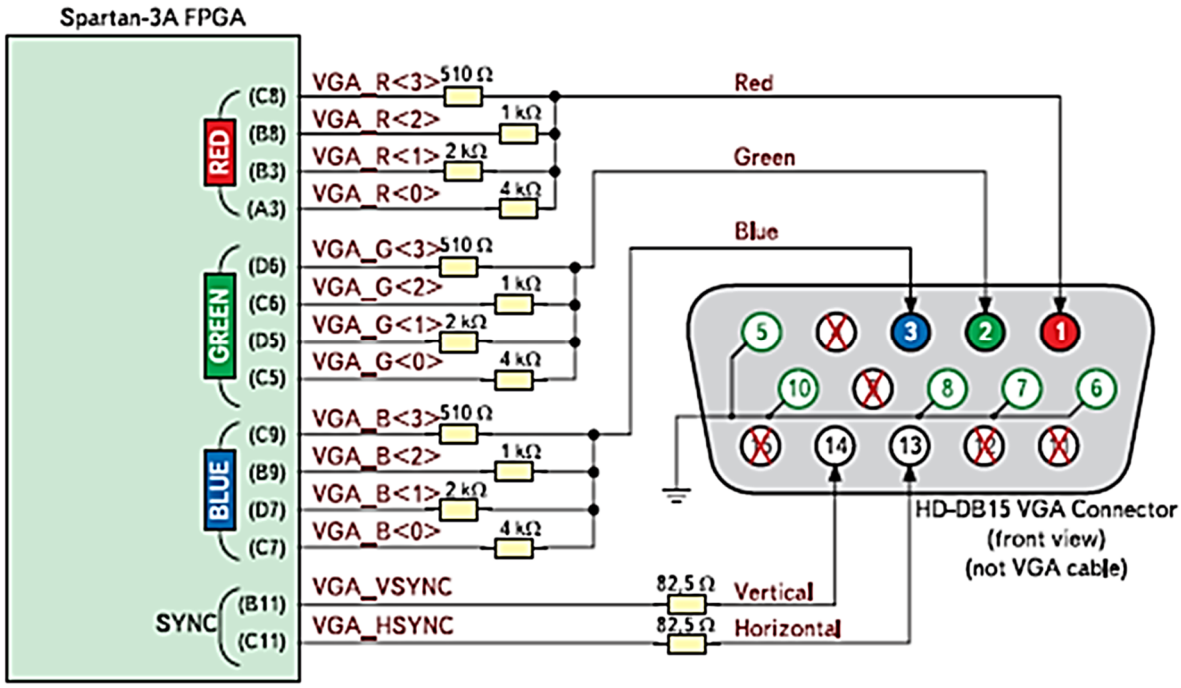


Рисунок 97. – Схема подключения VGA D-sub разъема к микросхеме с программируемой логикой Spartan-3

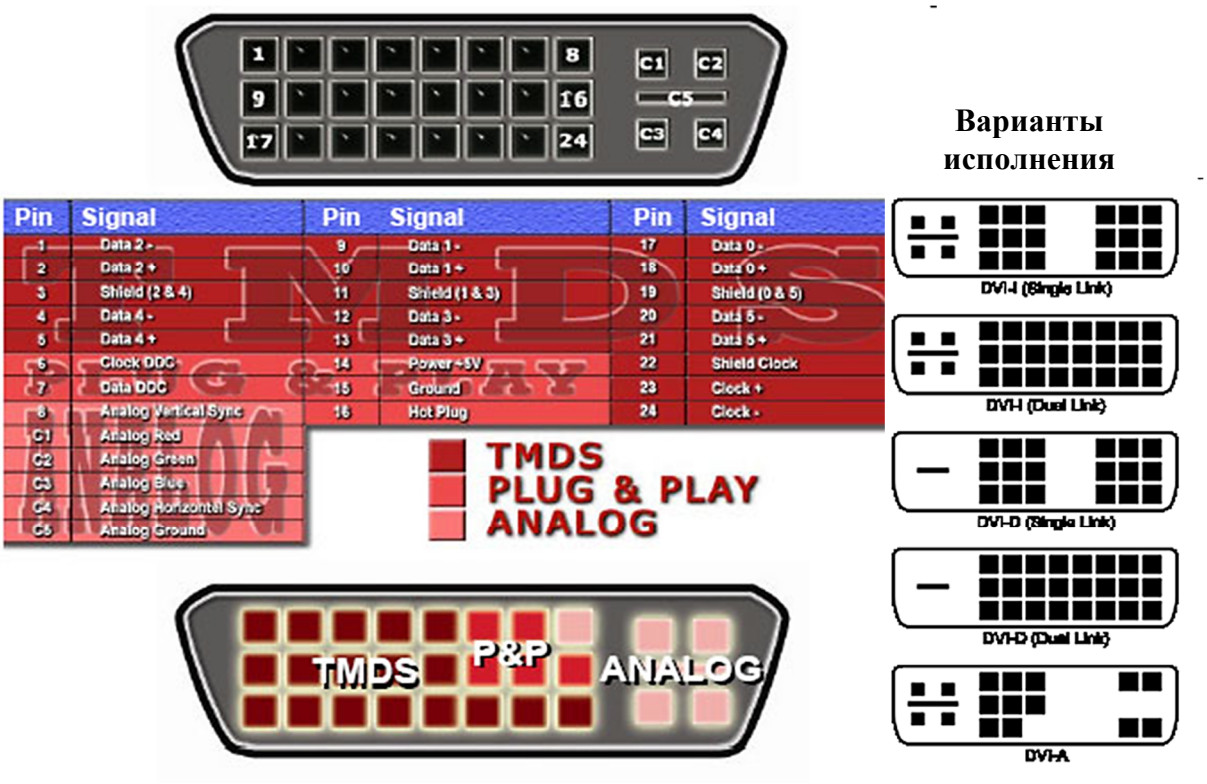


Рисунок 98. – Схема совмещенного разъема видеоконтроллера VGA DVI

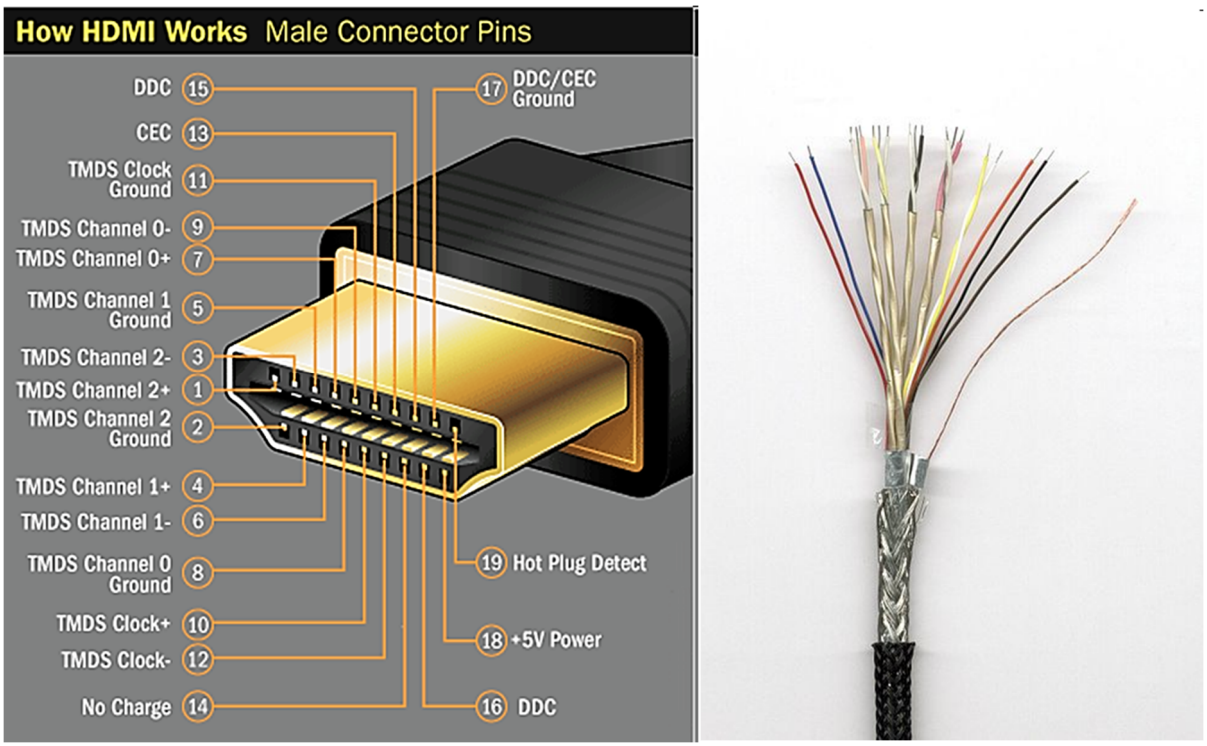


Рисунок 99. – Схема цифрового разъема видеоконтроллера VGA HDMI

14.3. Организация экранной памяти

14.3.1. Страничная организация экранной памяти видеоконтроллера

Центральный процессор в графическом режиме работы VGA формирует цифровое изображение в виде массива *n*-разрядных чисел (определяющих цвет пиксела) объемом по 64 Кб для матрицы пикселов с разрешением M × N и записывает его в ОЗУ либо непосредственно в видеобуфер видеоадаптера, встроенный в адресное пространство ЦП.

Схема оперативной памяти, доступной ЦП для передачи видеоинформации, показана на рисунке 100.

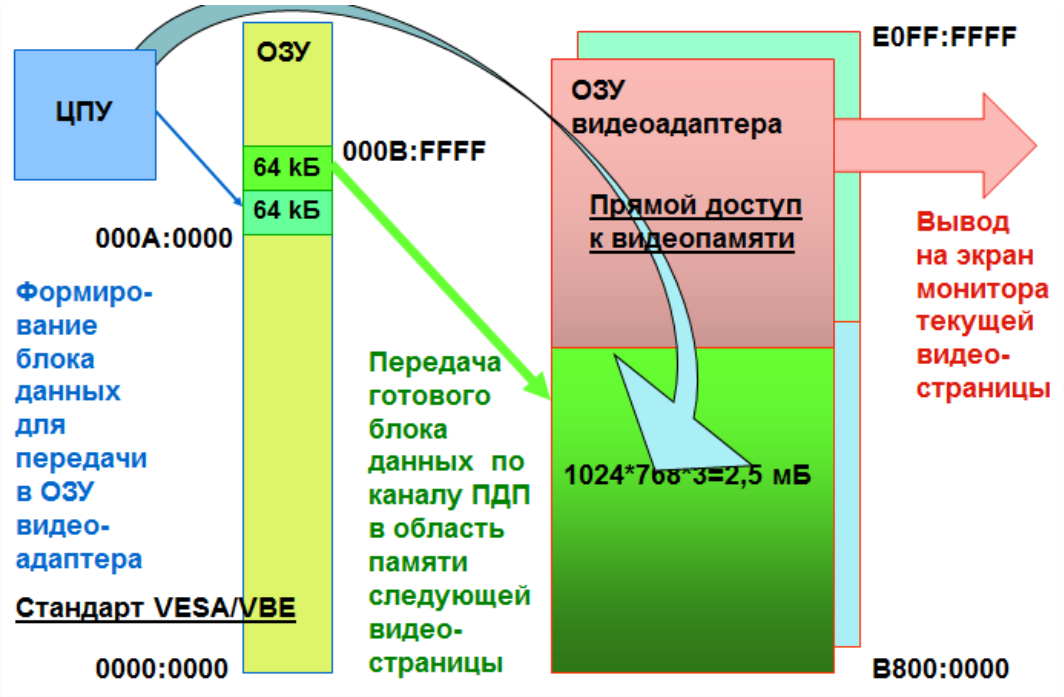


Рисунок 100. – Области ОЗУ для обмена данными с видеоконтроллером

Участок видеопамяти, отведенный для хранения цифрового образа текущего изображения (кадра), называется кадровым буфером (от англ. frame buffer – кадровый буфер). Видеоадаптер последовательно считывает (сканирует) содержимое ячеек кадрового буфера и формирует на выходе видеосигнал, уровень и цвет которого в каждый момент времени пропорционален значению, хранящемуся в отдельной ячейке. Сканирование видеопамяти осуществляется синхронно с перемещением электронного луча (или его образа) по экрану монитора. В результате яркость и цвет каждого пикселя на экране монитора оказывается пропорциональными содержимому соответствующей ячейки памяти кадрового буфера видеомонитора.

По окончании просмотра ячеек, соответствующих одной строке растра, видеоадаптер формирует импульсы строчной синхронизации H-Sync,

инициализирующий обратный ход луча по горизонтали, а по окончании сканирования кадрового буфера – сигнал V-Sync, вызывающий движение электронного луча снизу-вверх (обратный диагональный ход луча) При этом происходит смена или обновление содержания видеобуфера. Время обратного хода луча составляет $1/f_{кадр}$ секунды.

Видеопамять является специальной областью памяти, из которой контроллер CRT организует циклическое чтение содержимого для регенерации изображения на экране монитора. Традиционно для буфера видеоадаптера в карте распределения памяти прикладной задачи выделена область адресов A0000h – BFFFFh (стандарт VESA/VBE 1.0), непосредственно доступная любому процессору x86. Современные графические адаптеры VGA имеют возможность переадресации видеопамяти в область старших адресов (VESA/VBE 2.0), что позволяет в защищенном режиме работать с цельными образами экранов.

Расширение BIOS – это специальное ПЗУ, которое содержит все установки графического адаптера, имеет объем от 16 Кб и встраивается в оперативную память с адреса C0000:00000. Расширение BIOS содержит константы работы с видеоадаптером (характеристики видеорежимов).

Задача графического процессора – заполнение видеоплоскостей под непосредственным управлением центрального процессора. В текстовом режиме данные от центрального процессора поступают в обход графического процессора, непосредственно в контроллер атрибутов.

Цифро-аналоговый преобразователь RAMDAC выполняет преобразование кода цвета пикселя в аналоговый сигнал, подаваемый на вход монитора (ввиду чего монитор должен быть также аналоговым, а не цифровым, как это было при использовании видеоадаптера EGA). Цифро-аналоговый преобразователь RAMDAC содержит в себе 256 регистров, которые и образуют оперативную память (RAM), ввиду чего одновременно доступно только 256 цветов.

14.3.2. Обслуживания экранной памяти видеоконтроллера

Существует несколько способов обслуживания памяти видеобуфера:

1. Использование видеофункций BIOS прерывания 10h.

2. Непосредственная работа с регистрами видеоконтроллера при переходе в защищенный режим работы процессора Intel x86, когда недоступны видеофункции BIOS.

3. Использование дополнительных сопроцессорных устройств, появившихся в процессорах Pentium II (mmx) и Pentium III (xmm).

Команды mmx-расширения центрального процессора (целочисленные). Всего 6 групп команд (инициализации, сравнения, арифметические, сжатия, сдвига, логические и команды пересылки данных) всего 41 команда (76 машинных инструкций), используемые для прямого заполнения CPU видеоплоскостей видеобуфера адаптера с использованием 8-ми 80-битных регистров mmx.

Команды xmm-расширения центрального процессора (для чисел с плавающей запятой). Всего 6 групп команд (сравнения, арифметические, логические, пересылки, преобразования, перераспределения данных, управления и кэширования) всего 149 новых команд, используемые для заполнения видеоплоскостей видеобуфера адаптера с использованием 8-ми 128-битных регистров xmm, введенных в состав CPU, начиная с процессоров Pentium III.

Команды совместимы с аппаратной реализацией видеоадаптеров VGA и могут исполняться непосредственно графическими контроллерами.

2D-ускорители – дополнительные команды ассемблера, сопряженные с развитием архитектуры VGA-адаптеров и GDI (Graphics Device Interface)-драйверов к ним для ускоренной прорисовки двумерных изображений (например, оконной графики в Windows).

3D-системы – доработка CPU, VGA и система дополнительных команд Assembler Pentium II mmx для ускоренной прорисовки трехмерных изображений.

3DNow! – доработка CPU и система дополнительных команд ассемблера процессоров AMD для ускоренной прорисовки трехмерных изображений повышенной четкости.

OpenGL – разработка API-интерфейса (Silicon Graphics) для использования трехмерной графики в приложениях моделирования и рендеринга. Включает 250 функций рисования. Авторы Курт Экли и Марк Сегал.

DirectX – дополнение операционной системы Windows (фирмы Microsoft) наборами API-интерфейсов, обеспечивающих прямой доступ к аппаратным средствам CPU, VGA, аудиоадаптерам и т.д. или их программного эмулирования для обеспечения разработчиков игр возможностью создавать нормально работающие приложения в среде Windows.

Для этой цели разработаны 4 базовых компонента:

– *DirectDraw* – диспетчер видеопамяти.

– *Direct3D* – 3D-графика.

– *DirectInput* – аппаратно независимая система ввода данных.

– *DirectSound* – аппаратно независимая система воспроизведения звука аудиоустройствами различных производителей.

– Дополнительный компонент ***DirectPlay*** позволяет разработчикам игр создавать приложения, работающие на любой аппаратной платформе.

14.4. Основные режимы работы видеосистем

14.4.1. Текстовый режим видеоконтроллера

В текстовом режиме изображение на экране монитора представляет собой множество пикселей и характеризуется разрешением N×M. Однако все пиксели разбиты на группы, которые называются ***текселами*** или символьными позициями размером p×q. В каждом из знакомест может быть

отображен один из 256 символов таблицы ASCII. Таким образом, на экране умещается M/q = Mt символьных строк по N/P = Nt символов в каждой.

Типичным текстовым режимом является режим 80×25 символов.

Изображение символа в пределах каждого знакоместа задается точечной матрицей. Размер матрицы зависит от типа видеоадаптера и текущего видеорежима. Чем больше точек используется для отображения символа, тем выше качество изображения и лучше читается текст.

Точки матрицы, формирующие изображение символа, называют передним планом (foreground), остальные – задним планом или фоном (background).

Если считать, что темной клетке соответствует логическая единица, а светлой – логический нуль, то каждую строку символьной матрицы можно представить в виде двоичного числа. Графическое изображение символа хранится в виде набора двоичных чисел. Для этой цели используется ПЗУ BIOS размещенное на системной плате. Базовый набор символов для DOS размером 8×8 находится по адресу F000:FA6E.

На плате видеоадаптера в составе контроллера атрибутов также имеются свои наборы шрифтов. Аппаратный знакогенератор хранит шрифт, который автоматически используется видеоадаптером сразу же после включения компьютера (обычно это буквы английского алфавита и набор специальных символов). Адресом ячейки знакогенератора является порядковый номер символа. Для кодирования изображения символа используется два байта: первый байт – для задания номера символа в таблице ASCII, второй байт – для указания атрибутов символа (мигания символа, цвета символа, интенсивности цвета символа и цвета фона).

BL	R	G	B	I	R	G	B

Для вывода на экран центральный процессор формирует массив из двух байтовых элементов. Если на экране имеется Nt × Mt тексела, то объем видеопамяти, необходимый для хранения изображения, составит Nt × Mt × 2 байт. Эту область видеопамяти называют ***видеостраницей*** (video page).

Видеостраница является аналогом кадрового буфера в графическом режиме, но имеет значительно меньший объем. В текстовом режиме (80×25 символов) размер видеостраницы составляет 80 × 25 × 2 = 4000 байт. На практике для удобства адресации под видеостраницу отводят 4 Кб = 4096 байт, при этом «лишние» 96 байтов не используются.

Главная особенность текстового режима – адресуемым элементом экрана является не пиксель, а тексел. Т.е. в текстовом режиме нельзя сформировать произвольное изображение в любом месте экрана – можно лишь отобразить символы из заданного набора, причем только в отведенных символьных позициях. Другим существенным ограничением текстового режима является узкая цветовая палитра – не более 16 цветов.

14.4.2. Графический режим работы видеоконтроллера

Изначально графический контроллер при включении ПЭВМ программируется миниоперационной системой BIOS в стандартный текстовый режим с разрешением 80×25, доступный любому видеоконтроллеру. Переключение контроллера в графический режим осуществляется специальным набором команд ассемблера с указанием констант видеорежима либо загрузкой соответствующей программы-драйвера (что предпочтительней).

В графическом режиме содержимое каждой ячейки кадрового буфера (матрицы N × M *n*-разрядных чисел) является кодом цвета соответствующего пикселя экрана. Разрешение экрана при этом также равно N × M. Адресным элементом при этом является минимальный элемент изображения – пиксель. По этой причине графический режим называют также режимом APA (All Point Addressable – все точки адресуемы).

Иногда число *n* называют глубиной цвета. При этом количество одновременно отображаемых цветов равно 2*n*, а размер кадрового буфера, необходимый для хранения цветного изображения с разрешением N×M и глубиной цвета *n*, составит N × M × *n* бит.

Зная предельное разрешение видеоконтроллера, легко определить предельное значение объема видеопамяти, необходимой видеоадаптеру:

$$V = 1920 \times 1380 \times 2 \times 3 \times 4 \text{ страницы} = 64 \text{ Мб}.$$

Остальная память является избыточной и в растровой графике не используется.

Графический режим является основным режимом работы видеосистемы современного персонального компьютера, поскольку в этом режиме на экран монитора можно вывести текст, фотографию, анимацию и видеоролик. В частности, в таком режиме работает видеосистема ПЭВМ под управлением операционных систем Windows, Unix, Palm и других. Однако для эффективной работы в графическом режиме требуется значительный объем видеопамяти и высокопроизводительный компьютер, поэтому данный режим стал основным только с появлением персональных компьютеров на базе центрального процессора Intel Pentium.

Для отображения цвета текселов и пикселов используется палитра цветовых оттенков. Базовая палитра, содержащая 16 4-битных оттенков цветовой гаммы, находится в контроллере атрибутов. Она доступна как в текстовом режиме, так и в графическом. Доступ к палитре осуществляется через индексный регистр 3C0h, номера регистров цвета 00h – 0Fh и содержат 16 регистров формата

0	0	0	0	**I**	**R**	**G**	**B**

Чтение палитры осуществляется через порт 3DAh.

Файловый массив основной палитры графического режима находится в блоке RAMDAC, состоит из 256 трехбайтовых регистров 00h – FFh и доступен через индексный регистр 3C6h. Чтение палитры осуществляется последовательно побайтово через порт 3C7h, а запись – через порт 3C8h. Состав палитры:

0	0	x	x	x	x	x	x	0	0	x	x	x	x	x	x	0	0	x	x	x	x	x	x

Число оттенков: при 8-битном заполнении – 16 777 216. Реально доступно в каждый момент времени 256 цветов.

Для увеличения количества цветовых оттенков палитру необходимо перепрограммировать или использовать режим прямого обозначения цвета. В нем используется не код цветового оттенка, а его полная характеристика, но объем видеокадра при этом увеличивается в 3-4 раза, что в современных VGA-адаптерах вполне допустимо. При этом количество цветовых оттенков может быть увеличено до 32 и даже 64 миллионов.

14.5. Порядок программирования видеоизображения

Смысл программирования видеоизображений заключается в присвоении собственного цвета каждому пикселу, выводимому на экран монитора.

Для базового программирования в стандарте VESA/VBE обычно используют экран с разрешением 320×200 с отображением 256 цветов палитры. При этом получается линейный массив из 64000 однобайтовых элементов. Каждый байт этого массива будет содержать индекс цвета из палитры основных цветовых оттенков. В этом режиме видеобуфер адаптера будет располагаться по адресу A0000 – AF9FF и занимать 64 000 байт, т.е. по одному байту на пиксель.

Сначала весь массив заполняется значением цвета фона или фоновой картинки. Затем экран размещается в координатах адресного пространства видеобуфера, как показано на рисунке 101.

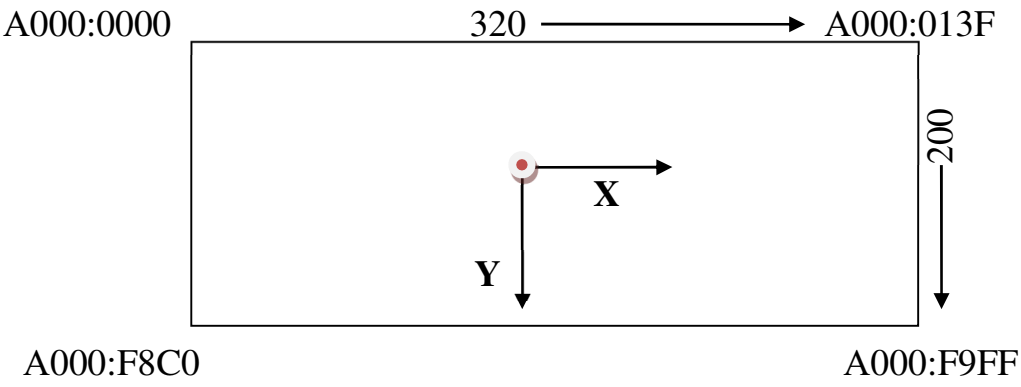


Рисунок 101. – К схеме программирования изображения

Пиксел по адресу (X, У) в координатах: 320 по горизонтали (отсчет X слева направо) и 200 по вертикали (отсчет Y производится сверху вниз) программируется следующим образом:

1. Вычисляется величина смещения линии X – У × 320.

2. К полученному значению добавляется координата X.

3. Полученный результат преобразуется в 16-ричную систему счисления и используется как смещение к параграфу A000:xxxx.

4. В ячейку памяти по этому адресу записывается индекс цвета 0 – 255.

При изменении разрешения экрана необходимо ввести соответствующее масштабирование изображения. В противном случае изображение будет размываться по верхней части экрана.

Стандарт VESA/VBE отображает видеопамять в виде отдельных окон (банков) размерами не более 64 Кб, причем только одно из них может быть активным в заданный момент времени. Переключение таких окон значительно усложняет алгоритмы обработки графики, а также замедляет скорость работы всего приложения. Стандарт VESA/VBE, начиная с версии 2.0, позволяет добиться линейного доступа к видеопамяти в режиме LFB (Linear Flat-frame Buffer) аналогично тому, как это делает DirectDraw.

К сожалению, приложения MS-DOS, использующие режим LFB, не могут работать под управлением ОС семейства Windows. Это значительно ограничивает область применения LFB и практически сводит на нет все его преимущества перед обычными «оконными» режимами VESA/VBE.

Компонент DirectDraw представляет видеопамять в виде непрерывного линейного массива (вектора), напрямую определяющего цвета отображаемых пикселов. Объект DirectDraw представляет собой обычный COM-объект, который отличается от объектов языка Си++ тем, что не может иметь открытых переменных (полей), конструкторов и деструкторов. Для их создания используются специальные функции, а для удаления применяется метод Release().

При выборе языка программирования (транслятора) рекомендуется ориентироваться на Microsoft Visual C++ 6.x, т.к. другие языки имеют более сложный синтаксис вызова функций DirectDraw.

Методы прямого программирования изображений:

1. Создание базового DirectDraw-объекта при помощи функции Direct DrawCreate(), обычно находящейся в динамической библиотеке DDRAW.DLL и объявленной в заголовочном файле ddraw.h:

LPDIRECTDRAW lpDDraw;

hResult = DirectDrawCreate(NULL, &lpDDraw, NULL).

2. Задание полноэкранного режима с исключительным правом доступа:

hResult = pDDraw->SetCooperativeLevel;

(hWnd, DDSCL_EXCLUSIVE | DDSCL_FULLSCREEN).

3. Установка 256-цветового режима с разрешением 640×480 точек:

hResult = lpDDraw->SetDisplayMode(640, 480, 8);

Установка режима 800×600 точек и 65 536 цветов:

hResult = lpDDraw->SetDisplayMode(800, 600, 16).

4. Создание первичной поверхности (Primary Surface):

4.1. Создание структуры типа DDSURFACEDESC:

typedef struct _DDSURFACEDESC {
DWORD dwSize;
DWORD dwFlags;
DWORD dwHeight; и т.д.

4.2. Создание первичной поверхности при помощи метода Create Surface(), принадлежащего объекту DirectDraw:

DDSURFACEDESC ddsd;
DDSCAPS ddsc;
ZeroMemory(&ddsd, sizeof(ddsd));
ddsd.dwSize = sizeof(ddsd); и т.д.

5. Получение прямого доступа к видеопамяти. Для этого блокируется поверхность в памяти при помощи метода Lock():

hResult = lpPrimarySurface->Lock;
(NULL, &ddsd, DDLOCK_WAIT, NULL).

6. ***lpPrimarySurface*** – указатель на область памяти, ассоциированную с поверхностью. Он указывает на начало активной страницы видеопамяти.

Далее следует ввод изображения в пиксельном формате (BPP – байт на пиксел) с использованием любых функций его формирования, поддерживаемых аппаратной реализацией схемы графических процессоров из библиотеки API.

7. После выполнения операций, которые связаны с прямым доступом к памяти, ассоциированной с поверхностью DirectDraw, требуется немедленно разблокировать эту поверхность при помощи метода Unlock():

lpPrimarySurface->Unlock(ddsd.lpSurface).

В противном случае операционная система может зависнуть.

Это методы прямого программирования изображений, которые обычно используются при формировании игрового пространства. На практике обычно применяются различные API-интерфейсы или просто разнообразные графические редакторы, доступные любому пользователю.

Тема 15. Архитектура параллельного порта (интерфейс Centronics)

Параллельный (Centronics) и последовательный (RS232C) порты используются не только для подключения принтера и модема, для которых

были в свое время разработаны (рисунок 102). Простота исполнения и отработанный протокол приема-передачи данных сделали их незаменимыми для подключения к ПЭВМ различных низкоскоростных устройств, применяющихся в промышленности и научных исследованиях.

Рисунок 102. – Внешний вид разъема параллельного порта

Основным назначением интерфейса Centronics (отечественный аналог – ИРПР-М) является подключение к компьютеру принтеров различных типов. Поэтому распределение контактов разъема, назначение сигналов, программные средства управления интерфейсом ориентированы именно на это использование. В то же время с помощью данного интерфейса можно подключать к компьютеру и другие специально разработанные внешние устройства.

Скорость обмена по интерфейсу Centronics – 129 – 200 Кб/с.

Стандартный параллельный порт CPP (Centronics Parallel Port) предназначен только для односторонней передачи информации (команд и данных) от ПЭВМ к принтеру или модему в параллельном виде шириной 8 разрядов. От принтера или модема принимались только коды ошибки или данные в последовательном виде.

Усовершенствованный порт EPP (Enhanced Parallel Port) является двунаправленным, позволяет подключать до 64 устройств и обеспечивает скорость передачи данных с использованием ПДП до 2 Мб/с.

Расширенный порт ECP (Extended Capability Port) позволяет подключить до 128 устройств и поддерживает режим компрессии (сжатия) данных.

На рисунке 103 представлена функциональная схема параллельного порта.

Операционный блок контроллера параллельного обмена представляет собой 3-канальный байтовый интерфейс и позволяет организовать обмен данными в трех режимах:

Режим 0 – синхронный однонаправленный ввод/вывод с использованием четырех каналов: A, B, C1, C2.

Режим 1 – асинхронный однонаправленный ввод/вывод с использованием основных двух каналов: A и B.

Режим 2 – асинхронный двунаправленный ввод/вывод по каналу A.

Программирование режимов работы каналов контроллера осуществляется передачей в буфер управления соответствующего кода.

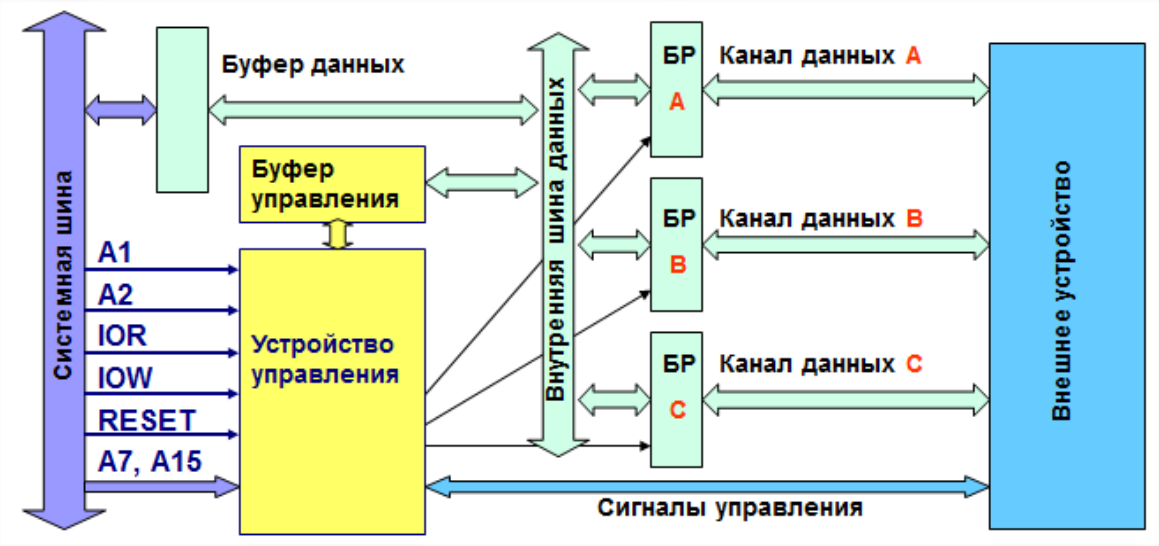


Рисунок 103. – Функциональная схема параллельного порта ПЭВМ

Сигналы Centronics имеют следующее назначение:

D0 ... D7 – шина данных для передачи из компьютера в принтер.

STROBE – сигнал стробирования (сопровождения) данных.

ACK – подтверждение принятия данных и готовности принтера.

BUSY – сигнал занятости принтера обработкой полученных данных и неготовности принять следующие данные.

AUTO FD – сигнал автоматического перевода строки (каретки).

PE – сигнал конца бумаги (режим ожидания).

SLCT – сигнал готовности приемника (принтера).

SLCT IN – сигнал принтеру о том, что последует передача данных.

ERROR – сигнал ошибки принтера.

INIT – инициализация (сброс) принтера и очистка буфера печати.

Формирование и прием сигналов интерфейса Centronics производится путем записи и чтения выделенных для него портов ввода/вывода. В компьютере может использоваться три порта Centronics. Обычно, по умолчанию, это: ***LPT1*** (порт 378h, IRQ5), ***LPT2*** (порт 278h, IRQ7), ***LPT3*** (порт 3BCh).

Базовый адрес порта LPT1 используется для передачи принтеру байта данных. Установленные на линиях данные можно считать из этого же порта в ПЭВМ. Временная диаграмма протокола передачи данных по интерфейсу Centronics представлена на рисунке 104.

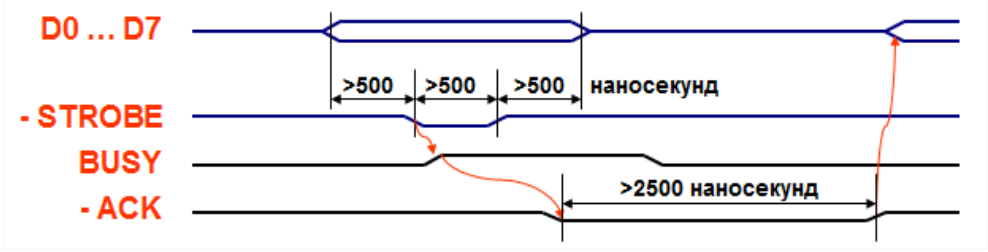


Рисунок 104. – Диаграмма цикла передачи данных по параллельному порту

Перед началом передачи данных контролируется снятие сигналов BUSY и ACK. Затем данные выставляются на шину и формируется сигнал STROBE. За это время принтер должен успеть принять данные и выставить сигнал BUSY, а затем и ACK.

Максимальная длина соединительного кабеля – 1,8 м.

В настоящее время стандарты параллельного порта EPP и ECP включены в общий стандарт IEEE 1284 с добавлением еще двух режимов обмена данными: байтового и полубайтового.

Тема 16. Архитектура последовательного порта (Интерфейс RS-232C)

Интерфейс RS-232C (отечественный аналог – стык С2) предназначен для подключения к компьютеру стандартных внешних устройств (принтера, сканера, модема, мыши и т.д.), а также для связи компьютеров между собой.

Основными преимуществами использования RS-232C по сравнению с Centronics являются возможность передачи на значительно большие расстояния и гораздо более простой соединительный кабель. В то же время работать с ним несколько сложнее: данные в RS-232C передаются в последовательном коде побайтно, а каждый байт обрамляется стартовым и стоповыми битами. Временная диаграмма протокола передачи данных по интерфейсу RS-232C представлена на рисунке 105.

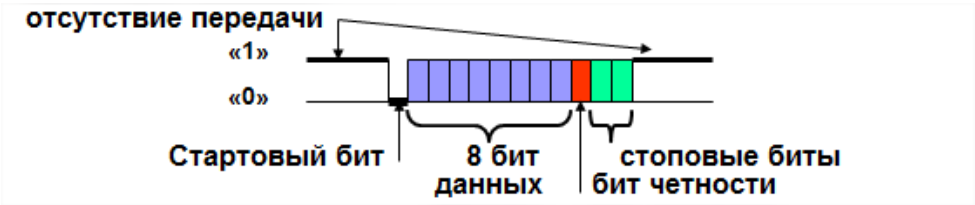


Рисунок 105. – Диаграмма цикла передачи данных по интерфейсу RS-232C

Данные могут передаваться как в одну (полудуплексный режим), так и в обе стороны (дуплексный режим).

Обмен по интерфейсу RS-232C осуществляется по специально выделенным для этого последовательным портам. По умолчанию это:

– ***COM1*** (адреса 3F8h ... 3FFh, прерывание IRQ4),
– ***COM2*** (адреса 2F8h ... 2FFh, прерывание IRQ3),
– ***COM3*** (адреса 3E8h ... 3EFh, прерывание IRQ10),
– ***COM4*** (адреса 2E8h ... 2EFh, прерывание IRQ11).

В состав ПЭВМ могут входить до четырех последовательных портов, работающих в стандарте RS-232C (обычно два или один). Каждое из устройств RS-232C представляет собой самостоятельный контроллер i8250, схема которого показана на рисунке 106, оснащенный 25- или 9-штырьковым разъемом (рисунок 107).

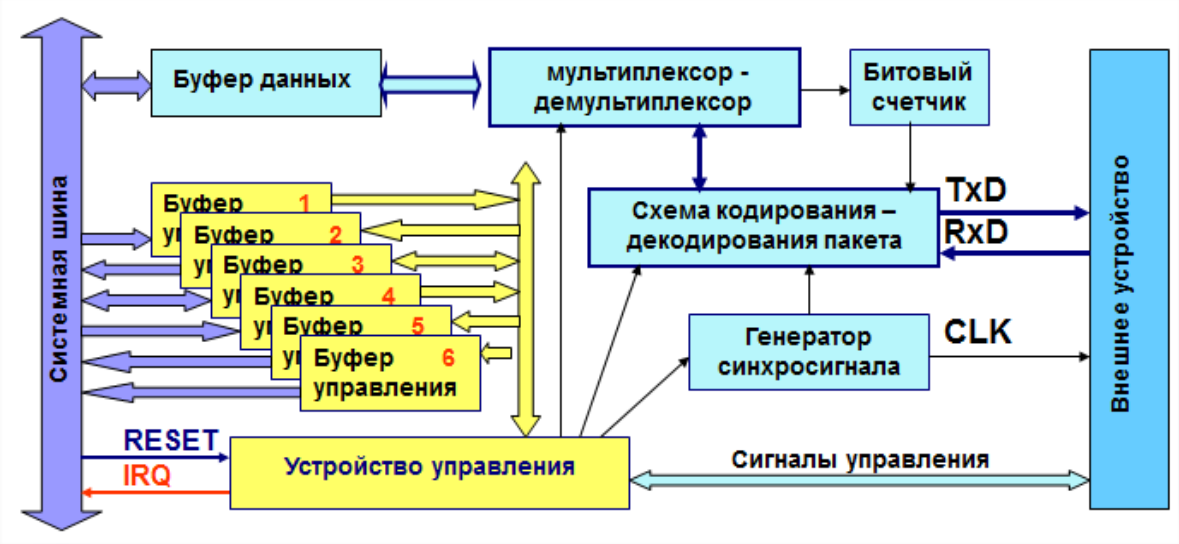
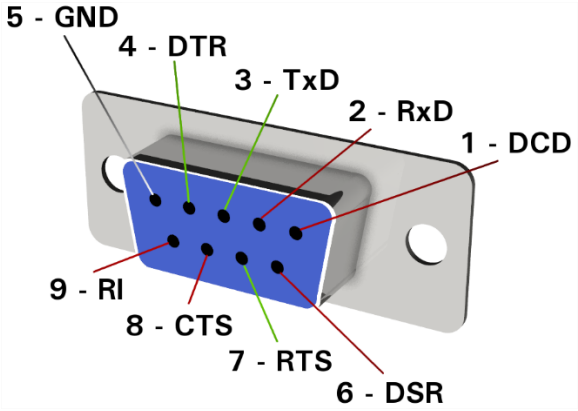


Рисунок 106. – Функциональная схема последовательного порта ПЭВМ



1. DCD – обнаружение несущей данных (принимаемого сигнала);
2. RxD – данные, принимаемые ПЭВМ в последовательном коде;
3. TxD – данные, передаваемые ПЭВМ в последовательном коде;
4. DTR – готовность выходных данных;
5. GND – сигнальное заземление, нулевой провод;
6. DSR – готовность данных для передачи модемом;
7. RTS – сигнал запроса передачи. Активен во время передачи;
8. CTS – сигнал сброса (очистки) для передачи. Активен во время передачи. Определяет готовность приемника;
9. RI – индикатор вызова. Говорит о приеме модемом сигнала вызова по телефонной сети.

Рисунок 107. – Внешний вид разъема последовательного порта

Контроллер порта RS-232C является полностью программируемым устройством. Ему можно задать следующие параметры обмена: количество битов данных и стоп-битов, вид четности и скорость обмена в бодах (бит/с).

Назначение сигналов обращений представлено на рисунке 107, кроме того, сигнал ***FG*** отвечает за защитное заземление (экран).

Конкретные форматы обращений по этим портам можно найти в описаниях микросхем контроллеров последовательного обмена UART (Universal Asynchronous Receiver/ Transmitter), например, для i8250. Наиболее часто используются трех- или четырехпроводная связь (для двунаправленной передачи), схема соединений которой представлена на рисунке 108.

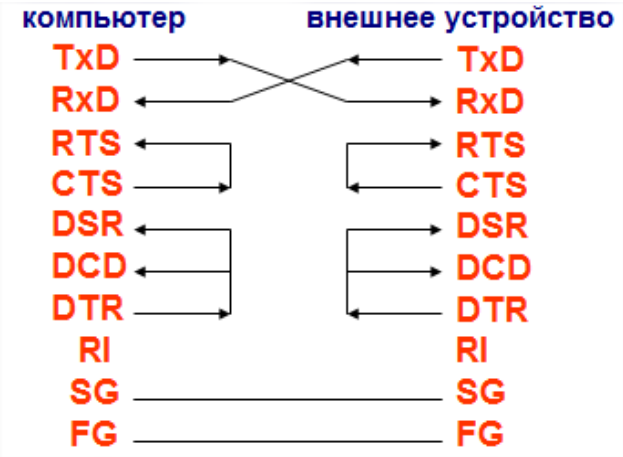


Рисунок 108. – Схема четырехпроводной линии связи для RS-232C

Для двухпроводной линии связи в случае только передачи из компьютера во внешнее устройство используются сигналы SG и TxD.

Такая схема использования интерфейса RS-232C нашла широкое применение её в различных технологических устройствах.

В настоящее время разработано и выпускается промышленностью большое количество сетевого оборудования, предназначенного для сбора данных или управления оборудованием в автоматизированных системах управления технологическими процессами (АСУ ТП). Разработан специальный протокол передачи команд и данных по этому интерфейсу **– *Modbus***. Протокол Modbus широко применяется в промышленности для организации связи между электронными устройствами и может использоваться для передачи данных через последовательные линии связи, оборудованные операционными усилителями / преобразователями сигналов RS-422 и RS-485.

RS-422 – полнодуплексный интерфейс. Прием и передача идут по двум отдельным парам проводов. На каждой паре проводов может быть только по одному передатчику.

RS-485 – полудуплексный интерфейс. Прием и передача идут по одной паре проводов с разделением по времени. В сети может быть много передатчиков, так как они могут отключаются в режиме приема.

Все 10 сигналов интерфейса RS-232C задействуются только при соединении компьютера с модемом по телефонной линии связи. При этом тактовые частоты приемника и передатчика должны быть одинаковыми (расхождение – не более 10%) для этого скорость передатчика (ПЭВМ) может выбираться из ряда: 150, 300, 600, 1200, … 57600, 115200 бит/с.

Следует отметить, что в настоящее время модемы, использующие интерфейс RS-232C, уже не применяются, т.к. для существенного увеличения пропускной способности телефонной линии были заменены на модемы, использующие технологию VDSL (Very High Bit-Rate Digital Subscriber Line) и сетевой протокол TCP|IP для обращения к ПЭВМ. При использовании технологии VDSL скорость передачи данных по телефонной линии, с увеличением частоты несущего сигнала, увеличивается на несколько порядков и может достигать 100 Гб/с, что несравнимо с возможностями интерфейса RS-232C.

Тема 17. Подсистема ввода-вывода (BIOS)

17.1. Назначение, структура и задачи BIOS

BIOS (Basic Input Output System) – часть программного обеспечения ПЭВМ, содержащая миниоперационную систему управления адаптерами внешних устройств, экранными операциями, выполняющую тестирование, а затем начальную загрузку OS и передающую ей управление ПЭВМ.

BOIS обеспечивает стандартный интерфейс, поддерживающий переносимость OS для ПЭВМ с совместимыми процессорами, и состоит из следующих основных компонент:

1. POST – процедуры проверки системных устройств и их ресурсов.
2. ROM-Scan – программа сканирования ОЗУ.
3. SETAP – программный интерфейс, содержащий редактор для просмотра и корректировки констант BIOS.
4. Константы BIOS в CMOS, таблица 256 символов ASCII 8x8 и другие данные.

Все адреса констант документированы и должны сохраняться в последующих версиях BIOS для данной системной платы. Компоненты BOIS записаны в специальной ПЗУ объемом 64 Кб, расположенную на системной плате, и обычно рассматриваются как неотъемлемая часть ПЭВМ, встраиваемая в адресное пространство ОЗУ, начиная с адреса F000:0000.

Основной принцип организации системы ввода/вывода: ЦПУ и ОЗУ образуют ядро ПЭВМ, а различные периферийные устройства, к которым можно отнести любое другое устройство, которое не входит в состав ядра ПЭВМ, сопрягаются с ядром системы с помощью интерфейсов (совокупности шин, сигналов, электрических схем, протоколов передачи данных и команд), входящих в состав ядра OS для организации обмена информацией.

Структура подсистемы POST. Состояние процессора после включения питания предопределено: ***EFLAGS*** = 00000002h; ***EIP*** = 0000FFF0h; ***CS*** = 0F000h; ***PE(CR0)*** = 0. По этому адресу находится команда JMP перехода на процедуру ***POST (Power On Self Test)*** самотестирования и инициализации базовых устройств ПЭВМ.

Алгоритм выполнения процедуры POST показан на рисунке 109.

Рисунок 109. – Схема алгоритма выполнения процедуры POST

Процедуры POST служат для пуска, самотестирования устройств на системной плате, сравнения их статуса с данными CMOS, выделения ресурсов и инициализации: каналы системного интервального таймера (слышен гудок), контроллер прерываний, контроллер прямого доступа, контроллер клавиатуры (загораются индикаторы на клавиатуре), контроллер памяти и т.д., затем инициализируются процедуры самотестирования устройств, имеющих собственные BIOS: видеоконтроллер (на мониторе появляется первое сообщение), контроллеры дисковых накопителей (появляется сообщение), контроллер USB (клавиатура, мышь USB становятся активными), звуковой адаптер, сетевой адаптер и т.д.

Выполняется сканирование оперативной памяти (отображается на мониторе). После успешного завершения процедур тестирования осуществляется поиск boot-устройств, содержащих в буфере данных байт 80h (1000000), т.е. неисполняемую команду, используемую в данном контексте POST для подтверждения активности устройства загрузки OS. Приоритет загрузочных устройств определяется в SETUP CMOS.

При выполнении POST могут генерироваться сообщения об ошибках. BIOS различных производителей имеет различные способы вывода кодов ошибок (звуковые сигналы, индикаторы кодов на системной плате и т.д.). Назначение сигналов и коды ошибок можно найти в справочной литературе или на сайтах производителей BIOS.

17.2. Доступ к переменным и константам BIOS

Осуществляется через меню SETUP BIOS, доступное в начальные моменты загрузки BIOS, с помощью клавиш, указанных в сообщениях, появляющихся в процессе загрузки и исполнения процедур BIOS. Меню SETUP обычно состоит из следующих разделов:

1. Стандартные переменные: дата, время, параметры ОЗУ и накопителей.

2. Дополнительные переменные и установленные модули программного обеспечения BIOS для тестирования аппаратных средств, устройств ядра ПЭВМ и ближайшего окружения.

3. Параметры остальных устройств, интегрированных в состав системной (материнской) платы ПЭВМ и варианты распределения ресурсов (слотов расширений шин, прерываний и каналов прямого доступа). Здесь же определяется порядок опроса периферийных устройств, которые могут содержать загрузочные модули операционных систем.

4. Параметры интерфейсов устройств, подключаемых к системной плате, определяющих варианты энергосбережения.

5. Размеры констант, определяющих параметры центрального процессора (частота, напряжение энергопитания ядра центрального процессора и ОЗУ), а также их предельные величины для сигнализации или отключения.

6. Набор параметров BIOS, загружаемый по умолчанию (в случае ошибок ручного набора параметров пользователем).

7. Ввод пароля для входа в редактор переменных – SETUP BIOS.

8. Ввод пароля для продолжения загрузки BIOS и выполнения POST-процедур после включения ПЭВМ (пользовательский пароль).

Значение паролей может быть снято системным обнулением BIOS.

17.3. Система Plug & Play автоопределения устройств ПЭВМ

На начальном этапе формирования архитектуры ПЭВМ программа BIOS использовалась как жесткий конфигуратор всех устройств на системной плате и устройств расширения архитектур (внешних устройств). Это приводило к многочисленным конфликтам устройств, претендующих на одни и те же ресурсы, которые приходилось устранять вручную при наладке ПЭВМ. Отказ от жесткого закрепления ресурсов за устройствами из состава архитектуры ПЭВМ и создание системы самоопределения устройств, получившей название ***Plug & Play*** (**PnP** буквально «включай и работай») стал большим шагом вперед в развитии архитектуры современных ЭВМ.

Основные принципы построения системы PnP были сформулированы и частично внедрены в 1974 г. для шины MCA (Micro Channel Architecture).

Основные принципы построения системы PnP:

1. Ресурсы ядра ПЭВМ (порты доступа и их разрядность, номера прерываний, адресное пространство ОЗУ для обмена информацией, каналы

прямого доступа) не являются жестко распределенными, а присваиваются по требованию в процессе работы BIOS.

2. Каждое периферийное (по отношению к ядру системы) устройство имеет описание набора требований в своем BIOS.

3. В составе BIOS PnP имеется программа – системный конфигуратор, которая присваивает номера периферийным устройствам, составляет паспорта (описания) этих устройств и выделяет необходимые ресурсы, с учетом недопущения конфликтов, при необходимости производит оптимизацию (перераспределение) ресурсов. Паспорта устройств сохраняются в реестре и передаются загружаемой операционной системе без изменения.

4. После загрузки операционной системы для периферийных устройств загружаются соответствующие системные драйвера. Осуществляется повторная проверка безконфликтной работы устройств. Современные BIOS – BIOS EFI (Extensible Firmware Interface) уже не требуют повторной проверки безконфликтной работы устройств, что существенно ускоряет процесс загрузки OS.

5. При отключении периферийного устройства или подключения нового операционная система автоматически перераспределяет освободившиеся ресурсы, определяет параметры нового устройства, проверяет его и предоставляет необходимые ресурсы без перезагрузки OS (на лету).

PnP – спецификация архитектуры аппаратных средств ПЭВМ, используемая соответствующими операционными системами для их конфигурирования и исключения конфликтов устройств между собой.

Основной компонент PnP – все оборудование, подключаемое к шинам, содержит энергонезависимые регистры POS (Programmable Option Select), где хранится конфигурация устройства и требуемые ресурсы.

Дополнительный компонент – файлы OS описания устройств, драйверов к ним и требуемых ресурсов (ini-файлы или реестр OS).

Программы BIOS PnP, бесконфликтно распределяющие ресурсы.

Схема инициализации устройств при загрузке BIOS показана на рисунке 110.

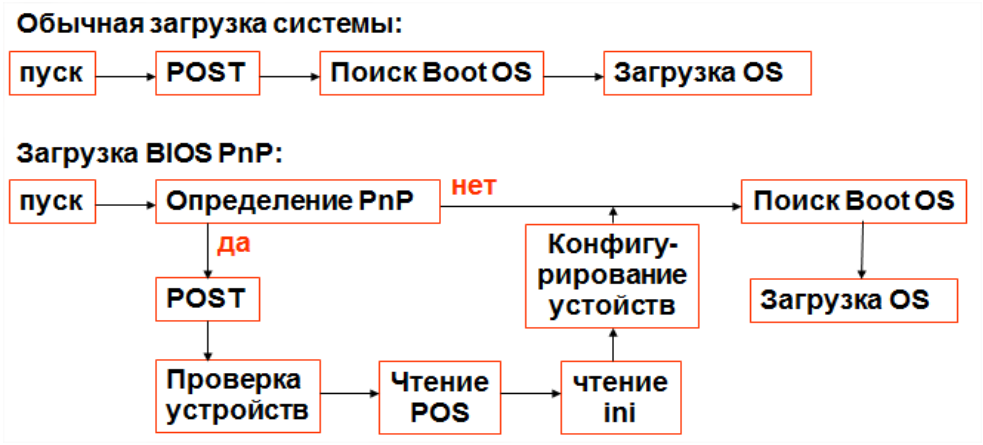


Рисунок 110. – Схема загрузки обычной BIOS и BIOS PnP

Для реализации принципов построения системы PnP, учитывая, что в составе архитектуры ПЭВМ встречается достаточно много устаревших устройств с жестко закрепленными за ними системными ресурсами, которые изменять не целесообразно (контроллер клавиатуры, системный интервальный таймер, контроллеры ПДП и т.д.), в реальной системе PnP используется следующий порядок распределения ресурсов:

1. При проверке POST определяются устройства «не PnP».
2. Устройствам «не PnP» ресурсы выделяются в первую очередь согласно спецификационных требований, т.к. эти устройства неперенастраиваемые.
3. При обнаружении конфликтов BIOS PnP генерирует уведомление о необходимости устранения конфликтов вручную.
4. Затем осуществляется итерационное конфигурирование устройств PnP.
5. Используются методы изоляции устройств друг от друга (присваивается идентификатор и серийный номер), после этого устройству присваивается дескриптор (Handle).

Присвоение идентификатора связано с используемой устройством шины и осуществляется специальной программой из состава OS – энумератором шины, которая является новым типом драйвера контроллера шины.

Номера идентификаторов являются уникальными для каждого устройства и неизменными для каждой последующей перезагрузки OS, например: ***PnP 0000*** – контроллер прерываний AT, ***PnP 0100*** – системный интервальный таймер, ***PnP 0C04*** – математический сопроцессор, ***PnP 0A03*** – контроллер шины PCI и т.д.

Подсистема ввода/вывода ПЭВМ и ядро OS решает следующие задачи:

1. Реализация вычислительной системы переменной конфигурации.
2. Параллельная работа программ в памяти и процедур ввода/вывода.
3. Упрощение процедур ввода/вывода, обеспечение их программной независимости от конфигурации конкретного периферийного устройства.
4. Обеспечение автоматического распознавания ядром ЭВМ периферийных устройств, многообразия их состояний (готовности, отсутствия носителя, ошибок чтения/записи и т.д.).
5. Интеллектуализация интерфейса, налаживание диалога между ядром и периферийными устройствами.
6. Переносимость и независимость OS от аппаратной платформы и ядра ПЭВМ.

Пути решения этих задач:

1. ***Модульность*** – новые периферийные устройства не вызывают существенных изменений архитектуры и вписываются в существующее адресное пространство, каналы и порты доступа.
2. ***Унификация по формату передаваемых данных и команд*** вне зависимости от используемых внутренних машинных языков микроопераций.
3. ***Унифицированный интерфейс*** по разрядности шины, набору линий сигналов управления и протоколам обмена.

4. ***Унификация по адресному пространству***, доступному ядру ПЭВМ, и каналам доступа к нему со стороны центрального процессора для операций ввода/вывода информации в пределах этого адресного пространства.

Современная система PnP состоит из следующих компонентов:

1. ***BIOS стандарта PnP (BIOS EFI)***, которая обеспечивает:

– уведомления – сообщение пользователю об обнаружении нового устройства;

– конфигурирование – изоляция устройства до присвоения ID;

– поддержка данных – информация завершения POST на специальной RAM.

2. ***Система драйверов-энумераторов шины PCI.*** Контроллер шины получает информацию из RAM об устройстве или из реестра для устройств «не PnP» и присваивает уникальный номер Vendor_ID.

3. ***Дерево аппаратных средств и реестр.*** Ветвь в реестре OS под названием «HKEY_LOKAL_MACHINE\ HARDWARE», которая состоит из типов аппаратных устройств.

4. ***Windows или другая OS PnP.*** Фирма Intel предлагает спецификацию PnP всем разработчикам OS.

5. ***Драйверы устройств PnP.*** Спецификация PnP предполагает не только наличие доступной для BIOS информации об устройстве в RAM этого устройства, но и динамически подгружаемый драйвер этого устройства. Существует интерфейс прикладного программирования (API) для создания таких драйверов для новых устройств стандарта PnP. Их загрузка должна регистрироваться диспетчером конфигурации и отвечать за выделенные ресурсы (сдавать их при выгрузке).

6. ***Арбитр ресурсов (служба OS PnP).*** Основные функции: обновление реестра, помещая туда новейшую информацию о выделении ресурсов на стадии загрузки; переназначение ресурсов «на лету» любым устройствам PnP, конфигурация которых изменилась. Арбитр ресурсов работает в контакте с диспетчером конфигурации, который в любой момент может запросить у арбитра ресурсов освобождения ресурса с последующим предоставлением его другому устройству.

7. ***Диспетчер конфигурации (служба OS PnP)*** отвечает за процесс конфигурирования всей системы в целом. Он непосредственно взаимодействует как с BIOS, так и с реестром, координируя процесс конфигурирования в ходе событий:

– когда BIOS отправляет ему список устройств «не PnP» на системной плате при загрузке, которые имеют жестко закрепленные за ними ресурсы;

– когда он получает извещение об изменении конфигурации от BIOS или энумераторов шин, которую использует для идентификации всех устройств на конкретной шине, а также требования каждого устройства о выделении ресурсов. Эта информация заносится в реестр.

8. ***Пользовательский интерфейс (API).*** Основное требование для пользовательских приложений, запускаемых в OS PnP, – они не должны иметь явных обращений к ресурсам устройств (портам ввода/вывода, прерываниям или дискам). Необходимо заменять эти обращения формальными обращениями к соответствующим устройствам.

17.4. BIOS UEFI

Система ввода/вывода BIOS разрабатывалась одновременно с дисковой операционной системой DOS в 1980 г., и если к настоящему времени операционная система перерабатывалась и модернизировалась многократно, то BIOS в этом отношении практически не изменилась, хотя были разработаны и используются множество дополниельных компонент:

– система управления энергоснабжением ACPI;

– система распознавания оборудования Plug & Play;

– графический интерфейс пользователя;

Дальнейшее совершенствование системы ввода/вывода было организовано фирмой Intel еще в 1998 г. Была разработана BIOS EFI (Extensible Firmware Interface), однако разработчики архитектур материнских (системных) плат поддержали это начинание и организовали форум и соответствующий индустриальный стандарт UEFI только в 2008 г. UEFI был поддержан и разработчиками операционных систем.

Основным недостатком традиционной BIOS принято считать сложность загрузки операционной системы в связи с отсутствием связи результатов работы процедур CMOS и систем поддержания реестра аппаратных средств операционных систем, а также ограниченность MBR в части предельного размера дискового пространства (не более 2 Тб).

Новая базовая система ввода/выада BIOS UEFI (BIOS Unified Extensible Firmware Interface – Унифицированный Расширяемый Интерфейс встроенного программного Обеспечения) лишена этих недостатков, имеет улучшенный графический итерфейс пользователя с поддержкой манипулятора мышь, не использует статическую загрузочную запись MBR и порт 80h для загрузки операционной системы, поэтому аппаратный загрузчик может располагаться где угодно и подчиняется лишь директивам EFI.

UEFI может работать в 32-битном или 64-битном режимах, её адресное пространство гораздо больше, чем у традиционной BIOS, поскольку она использует усовершенствованную структуру разделов GPT. В UEFI встроено множество других функций. Она поддерживает безопасный запуск Secure Boot, в котором можно проверить, что загрузку ОС не изменила никакая вредоносная программа. Она может поддерживать работу по сети, что позволяет проводить удалённую настройку и отладку параметров BIOS.

Для доступа к экрану настроек UEFI не обязательно прерывать процесс загрузки BIOS и OS, как это принято в традиционной BIOS, настройку BIOS UEFI можно сделать и через загрузочное меню Windows.

К недостаткам BIOS UEFI следует отнести специальные требования к аппаратному обеспечению компьютеров, поддерживающему UEFI. Однако для обеспечения преемственности операционных систем большинство версий UEFI поддерживают эмуляцию традиционных BIOS.

Тема 18. Общие сведения об операционных системах

18.1. Общее понятие об операционной системе

Чтобы полностью овладеть всеми возможностями своего компьютера, необходимо знать и понимать его операционную систему. Назначение операционной системы заключается в обеспечении удобства управления компьютером. Любая операционная система, в полном смысле этого термина, является первой и наиболее важной программой любого компьютера. Как правило, она является и наиболее сложной, используемой только для управления самим компьютером.

Основная часть работы операционной системы заключается в выполнении огромного количества рутинных операций контроля, проверки достоверности, вычисления значений физических адресов и т.д. Она предназначена, чтобы скрыть от пользователей большое количество сложных и ненужных им деталей процесса управления аппаратной частью. Как правило, операционная система состоит из нескольких частей:

1) система BIOS в ПЗУ ПЭВМ;
2) главная загрузочная запись;
3) аппаратный загрузчик операционной системы;
4) сканер и конфигуратор аппаратных средств ПЭВМ;
5) ядро операционной системы и командный монитор;
6) файлы конфигурации ini или реестр OS;
7) диспетчеры объектов и устройств;
8) драйверы устройств.

В состав современных OS обычно входят только последние четыре части, первые же четыре представляют утилиты BIOS стандарта EFI. Схема архитектуры современной операционной системы Windows NT/2000/XP/7/8/9/10 представлена на рисунке 111.

На этой схеме представлены следующие компоненты и уровни.

Пользовательский режим. Большая часть приложений, запускаемых пользователем, работает в пользовательском режиме. Все эти приложения обладают ограниченным доступом к операционной системе, благодаря чему

при возникновении неполадок в программе приложения ядро ОС остается надежно защищенным и продолжает нормально функционировать.

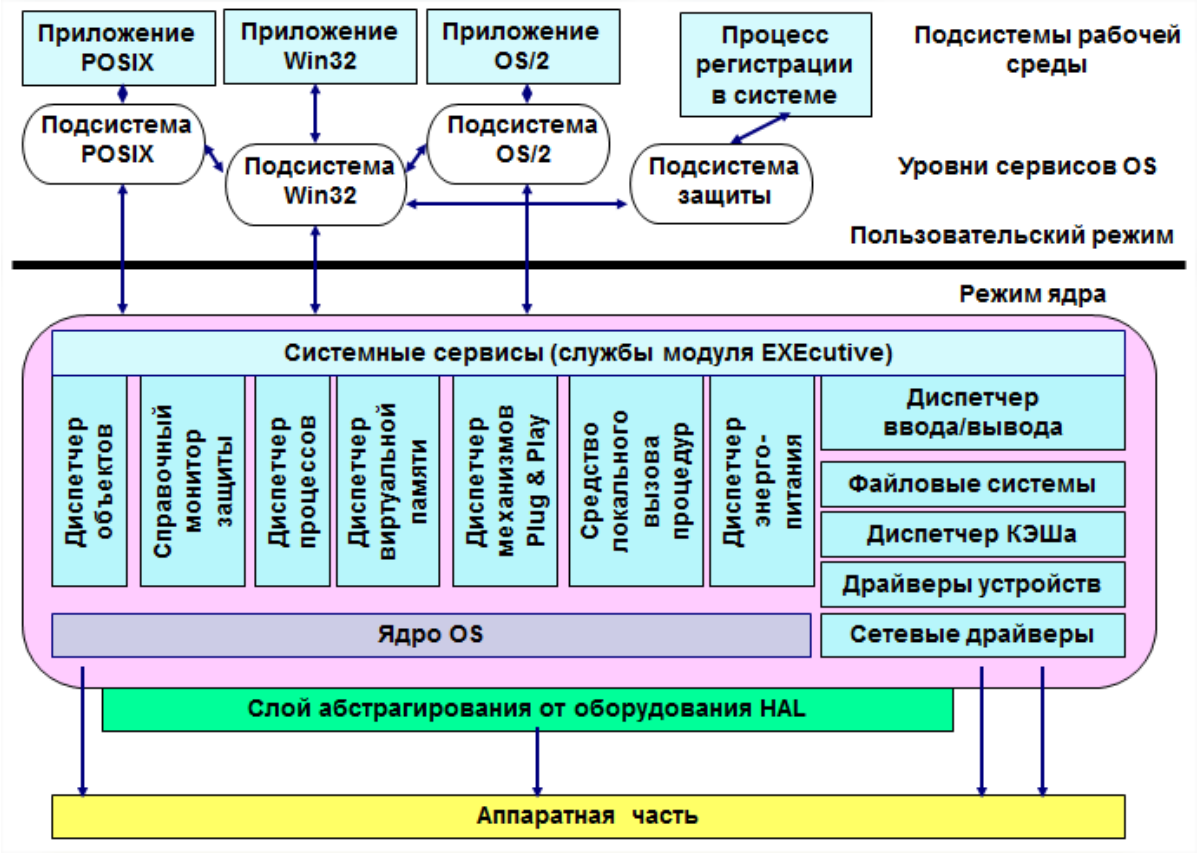


Рисунок 111. – Схема архитектуры современной операционной системы

Пользовательское приложение работает в рамках изолированного адресного пространства, предоставляемого операционной системой. При обращении к аппаратным устройствам (принтеру) службами ядра (диспетчером ввода/вывода) запускается соответствующий драйвер. Службы Windows (например, Task Scheduler, Messenger, Alerter и др.) работают в пользовательском режиме в специальном пользовательском контексте безопасности индивидуального рабочего окружения каждой конкретной прикладной задачи.

Режим ядра. Процессы, работающие в этом режиме, обладают наивысшим уровнем привилегий. Они работают в одном и том же адресном пространстве и могут напрямую обращаться к оборудованию компьютера, включая такие важные устройства, как центральный процессор или видеоадаптер. В этом же режиме функционируют драйверы устройств, все системные диспетчеры, модуль Microkernel, все службы модуля EXEcutive, а также системные сервисы уровня абстракции оборудования HAL (Hardware Abstraction Layer).

Модуль EXEcuteve. Так обозначаются в Windows NT/2000/XP программные компоненты, работающие в режиме ядра. Эти компоненты включают в себя жизненно-важные службы ОС, такие как управление памятью,

вводом/выводом, системой безопасности, механизмами взаимодействия процессов, кэшированием, а также системой управления объектами.

Модуль EXEcutive загружается в процессе начальной загрузки ОС и является частью файла Ntoskrnl.exe.

Модуль Microkernel. Управляет переключением процессора между выполнением разных потоков, а также обрабатывает системные прерывания и исключения. Этот модуль синхронизирует работу нескольких процессоров на многопроцессорных аппаратных платформах. В отличие от остального кода ОС, этот модуль никогда не перемещается в виртуальную память, т.к. его компоненты должны иметь фиксированные физические адреса.

Модуль Microkernel также является частью файла Ntoskrnl.exe.

Уровень абстракции оборудования HAL (Hardware Abstraction Layer) позволяет ОС работать на аппаратных платформах различной конфигурации и количества процессоров, не требуя ее перекомпилирования. Обычно HAL разрабатывается производителем аппаратной платформы.

Модуль HAL находится в файле hal.dll и загружается в процессе начальной загрузки ОС.

Процессы и потоки.

Процесс – это программа, которую можно запустить в рабочей среде ОС. Каждый процесс обладает своим адресным пространством, одним или несколькими программными потоками, а также идентификатором безопасности SID (Security ID), соответствующим учетной записи в контексте безопасности, которой этот процесс функционирует.

Поток – это составляющая часть процесса (что-то вроде процесса внутри процесса), т.е. программный код, выполнением которого занят процессор. В любой момент времени один процессор может выполнять только один программный поток. Переключение процессора между выполнением нескольких программных потоков осуществляется средствами ОС.

Каждый процесс может включать в себя несколько программных потоков. Например, работая в программе Explorer, можно открыть новое окно, для управления этим окном создается новый поток. Это означает, что в системе существует только один экземпляр процесса Windows Explorer, но в рамках этого процесса работает одновременно несколько программных потоков.

Симметричная многопроцессорная архитектура и наращиваемость ОС. ОС Windows NT/2000/XP обладают встроенной поддержкой многопроцессорных систем SMP (Symmetric Multiprocessing). Это означает, что если в системе установлено больше одного процессора, то поток, требующий выполнения, будет выполняться первым освободившимся процессором. Переключение процессоров между потоками осуществит модуль ***Microkernel***.

Службы и приложения пользовательского режима. В пользовательском режиме работают процессы трех типов с различным уровнем привилегий:

1. ***Системные процессы.*** Это процессы, управляющие рабочей средой пользовательского режима: ***Winlogon*** – подключение пользователей

к системе, ***Service Controller*** – сервис подключения служб, ***Session Manager*** – диспетчер сеанса. Эти процессы запускаются модулем ***Ntoskrnl.exe*** в процессе начальной загрузки и используют учетную запись ***LocalSystem***.

2. ***Службы Windows.*** Службы (***Alerter, Computer Browser*** и др.) запускаются процессом ***Service Controller*** (файл ***services.exe***) для прикладной задачи. Они функционируют как отдельные потоки в рамках процесса services.exe и не отображаются диспетчером задач в качестве отдельных процессов. Большинство служб не использует контекст безопасности ***LocalSystem***.

3. ***Пользовательские прикладные программы.*** Т.е. прикладные программы, запускаемые пользователем в пользовательском режиме. Каждая такая программа функционирует в рамках своего собственного виртуального адресного пространства, обычно не превышающего 1 Гб.

Процесс ассоциируется с подсистемой рабочего окружения ***environment subsystem***, который принимает от него вызовы, адресуемые ***API (Application Program Interface)*** и преобразует их в команды, адресованные модулю ***EXEcutive***.

Для поддержки DOS служит подсистема времени исполнения ***Csrss.exe***, которая запускается автоматически при формировании вызова.

18.2. Порядок загрузки операционной системы

Загрузка любой операционной системы после завершения операций инициализации аппаратных средств предусматривает следующие этапы:

– инициализация загрузчика OS (Boot loader process);

– выбор операционной системы (если предусмотрен выбор);

– повторное сканирование аппаратных средств, если не используется стандарт BIOS EFI или ACPI;

– загрузка ядра OS и его инициализация.

После завершения POST BIOS передает управление на первое внешнее устройство (согласно установленному приоритету), имеющее в буфере данных байт 80h. Считывание следующего байта из этого порта активирует программу аппаратной загрузки OS: т.е. будет выполнена процедура, записанная в начале раздела MBR, которая позволит найти адрес первого активного раздела в таблице разделов (по значению байта 80h), и загрузит следующую за ним команду в табличной записи JMP(xx) на исполнение.

Адрес команды JMP является адресом аппаратного загрузчика NTLDR (для операционной системы Windows NT/2000/XP), IBMBIO.COM (для DOS) или аналогичной программы для других OS (например UNIX, Linux и т.д.).

NTLDR загрузит режим плоского 32-разрядного адресного пространства и запустит минифайловую систему, совместимую с FAT16, FAT32 и NTFS. Затем читает ***Boot.ini*** в корневом каталоге и предложит выбор OS к загрузке. После выбора Windows XP выполняет программу ***Ntdetect.com***,

чтобы собрать информацию о всех физических устройствах, подключенных к ЦП. Затем ***NTLDR*** загружает в ОЗУ и запускает ядро операционной системы – программу ***Ntoskrnl.exe***, которая принимает данные сканирования.

Файлы, необходимые для успешного запуска ОС Windows XP представлены в таблице 5.

Таблица 5. – Основные файлы загрузки OS Windows XP

Наименование файла	Расположение файла
NTLDR	Корневой каталог Windows
Boot.ini	Корневой каталог Windows
Bootsect.dos (для выбора ОС)	Корневой каталог Windows
Ntdetect.com	Корневой каталог Windows
Ntbootdd.sys (только для SCSI)	Корневой каталог Windows
Ntoskrnl.exe	%SystemRoot%\System32
Hal.dll	%SystemRoot%\System32
Улей реестра \system	%SystemRoot%\System32\Config
драйверы устройств	%SystemRoot%\System32\Drivers

Процедура запуска системы закончится неудачей, если хотя бы один из указанных файлов не будет найден или окажется поврежденным.

Файл ***Bootsect.dos*** содержит копию первого сектора раздела альтернативной ОС (в данном случае DOS), аналогично можно построить загрузчики для ОС UNIX и Linux, а затем отредактировать файл ***Boot.ini***:

C:\ BOOTSECT.UNX=“UNIX”

C:\ BOOTSECT.LNX=“Linux”

Развитие архитектуры операционных систем ПЭВМ началось с первой версии DOS небольшой фирмы Microsoft.

Загрузочная запись ДОС версии 1.00 имеет одно незначительное отличие от всех остальных версий: вместо имени Роберта О'Рира – разработчика первой версии ДОС, которую он сделал по прототипу (Key DOS) за два месяца, во всех последующих версиях стоит название фирмы – «Microsoft». В настоящее время разработка новой версии ОС занимает 6 – 10 лет и в ее реализации участвует коллектив из 200 – 300 человек. Фирма Microsoft стала практически монополистом на рынке ОС ПЭВМ. Из альтернативных ОС для ПЭВМ можно указать лишь Unix и Linux BSD.

Основное направление совершенствования, которое реализовано в настоящее время для многоядерных процессоров нового поколения архитектуры Itanium, – создание нового интерфейса между ОС и встроенным программным обеспечением аппаратных платформ – UEFI, который предназначен для замены системы BIOS. Сейчас UEFI, а также полномасштабный стандарт энергосбережения ACPI поддерживаются многими разработчиками системных плат и операционных систем.

Для обепечения загрузки OS через BIOS UEFI фирмой Microsoft разработан новый загрузчик для операционной системы Windows, вошедший в состав ОС Windows 8 и последующих (***bootmgr*** вместо ***NTLDR).*** Файлы ***Boot.ini*** и ***Ntdetect.com*** исключены из системы, т.к. информация, которая хранилась в них, находится в BOOT, а все аппаратное окружение полностью соответствует спецификации UEFI. Для изоляции MBR, GPT, Boot и загрузчика от неосторожных действий пользователя при инсталляции OS создается изолированный (скрытый) системный раздел на жестком диске размером 100 Мб и 300 Мб для резервирования системы и последующего восстановления. Соответствующим образом откорректирован и реестр Windows.

Процесс загрузки операционной системы также усовршенствован. При загрузке OS под управлением BIOS UEFI после выполнения POST-процедур (отличие кодов ошибок UEFI незначительно), прежде всего, проверяется структура разметки GPT. В таблицах GPT определяются все разделы типа EFI (на них содержатся загрузчики с расширением .efi) и подгружаются в память с передачей управления первому загрузчику. Возможна загрузка OS и непосредственно из BIOS UEFI. Загрузка OS при этом не сопровождается повторной проверкой и распределением ресурсов системы, что существенно ускоряет процесс загрузки.

Тема 19. Тенденции развития

19.1. Центральные процессоры

19.1.1. Повышение тактовой частоты

Чтобы увеличить тактовую частоту при имеющихся материалах, используется следующее: более совершенный технологический процесс с более низкими стандартами проектирования; увеличение количества слоев металлизации; более совершенная система с меньшим количеством каскадов и более совершенными транзисторами, а также более плотное расположение функциональных блоков.

Таким образом, все производители микропроцессоров перешли на технологию CMOS, хотя Intel, например, использовала БиКМОП для первых представителей семейства Pentium. Известно, что высокочастотные биполярные и КМОП-схемы имеют примерно одинаковые показатели тепловыделения, но КМОП-схемы технологически более продвинуты, что определяет их преобладание в микропроцессорах.

Уменьшение размеров транзисторов, сопровождающееся снижением напряжения питания с 5 до 2,5 – 3 В и ниже, увеличивает быстродействие и уменьшает тепловыделение. Все производители микропроцессоров используют стандарты проектирования от 0,35 – 0,25 микрона до 0,18 и 0,12 микрона

и стремятся использовать уникальную технологию 0,07 микрона. При минимальном размере частей внутренней структуры интегральных схем 0,1 – 0,2 мкм достигается оптимальный уровень, ниже которого все характеристики транзистора быстро ухудшаются. Почти все свойства твердого тела, в том числе его электропроводность, резко меняются и «сопротивляются» дальнейшей миниатюризации, возрастание сопротивления связей происходит в геометрической прогрессии. Потери даже на более коротких линиях внутренних соединений такого размера «съедают» до 90% сигнала по уровню и мощности.

В этом случае начинают проявляться эффекты квантовой связи, в результате чего твердотельное устройство становится системой, действие которой основано на коллективных электронных процессах. Проектная норма 0,05 – 0,1 мкм является нижним пределом твердотельной микроэлектроники, основанной на классических принципах синтеза схем.

Уменьшение длины межсоединений важно для увеличения тактовой частоты, поскольку значительная часть длительности такта составляет время, необходимое для прохождения сигналов через проводники внутри кристалла. Например, в Alpha 21264 были предприняты специальные меры для кластерной обработки, чтобы найти взаимодействующие компоненты микропроцессора.

Проблема уменьшения длины межсоединений на кристалле с использованием традиционных технологий решается за счет увеличения количества слоев металлизации.

19.1.2 Увеличение объема и пропускной способности подсистемы памяти

Возможные решения для увеличения пропускной способности подсистемы памяти включают создание одно- или многоуровневого КЭШа, в дополнение к увеличению пропускной способности интерфейсов между процессором и КЭШем, и конфликтующей с этим увеличением пропускной способности между процессором и основной памятью. Улучшение интерфейсов достигается за счет увеличения пропускной способности шины (увеличения частоты шины и/или её ширины) и введения дополнительных шин, которые разрешают конфликты между процессором, КЭШ-памятью и основной памятью. В последнем случае одна шина работает на частоте процессора с КЭШ-памятью, а вторая – на частоте основной памяти. При этом частоты работы второй шины, например, равны 66, 66, 166 МГц для микропроцессоров Pentium Pro-200, Power PC604E-225, Alpha 21164-500, работающих на тактовых частотах 300, 225, 500 МГц соответственно. При ширине шин 64, 64, 128 разрядов это обеспечивает пропускную способность интерфейса с основной памятью 512, 512, 2560 Мб/с, соответственно.

Общая тенденция увеличения размеров КЭШ-памяти реализуется поразному:

– внешние КЭШ-памяти данных и команд с двухтактовым временем доступа объемом от 256 Кб до 2 Мб со временем доступа 2 такта в HP PA-8000;

– отдельный кристалл КЭШ-памяти второго уровня, размещенный в одном корпусе в Pentium Pro;

– размещение отдельных КЭШ-памяти-команд и КЭШ-памяти-данных первого уровня объемом по 8 Кб и общей для команд и данных КЭШ-памяти второго уровня объемом 96 Кб в Alpha 21164. Наиболее распространенным решением является размещение отдельных КЭШей первого уровня для данных и инструкций на чипе с возможностью создания КЭШа второго уровня вне кристалла. Например, Pentium II использует встроенный КЭШ первого уровня для команд и данные по 16 Кб, каждая из которых работает на тактовой частоте процессора, и внешний КЭШ второго уровня, работающий на половинной тактовой частоте.

19.1.3. Увеличение количества параллельно работающих исполнительных устройств

Каждое семейство микропроцессоров следующего поколения демонстрирует увеличение количества функциональных исполнительных механизмов и улучшение их характеристик, как временных (сокращение числа проходов конвейера и уменьшение продолжительности каждой фазы), так и функциональных, вводя MMX-расширения команд и т.д. В настоящее время процессоры могут выполнять до шести операций за такт. Однако число операций с плавающей запятой за такт ограничено двумя для R10000 и Alpha 21164, а HP PA-8500 выполняет четыре операции за такт.

Для загрузки функциональных исполнительных устройств используются переименование регистров и прогнозирование переходов, устраняющие зависимости между командами по данным и управлению, буферы динамической переадресации.

Широко используется архитектура с длинным словом команды – VLIW. Так, архитектура /D-64, разработанная Intel и HP, использует комбинацию нескольких инструкций в одной команде (EPIC). Это упрощает процессор и ускоряет выполнение команд. Процессоры с архитектурой /D-64 могут адресовать до 4 Гб памяти и работать с 64-битными данными. Архитектура /L-64 используется в микропроцессоре Merced, обеспечивая производительность до 6 Гфлопс для операций с одинарной точностью и до 3 Гфлопс с повышенной точностью на частоте 1 ГГц.

19.1.4. Системы на одном кристалле и новые технологии

В настоящее время системы, выполненные на одном кристалле, – SOC (System On Chip) – получили широкое развитие. В сферу применения SOC входят игровые приставки и телекоммуникации. Такие кристаллы требуют применения новейших технологий.

Основное технологическое новшество в области SOC было сделано корпорацией IBM, которая в 1999 г. смогла реализовать относительно недо-

рогой процесс объединения логической части микропроцессора и оперативной памяти на одном кристалле. В частности, в новой технологии используется так называемый дизайн памяти с врезанными ячейками (trench cell). В этом случае конденсатор, который хранит заряд, помещается в определенное углубление в кристалле кремния. Это позволяет разместить более 24 тысяч элементов, почти в 8 раз больше, чем в обычном микропроцессоре и в 2 – 4 раза больше, чем в микросхемах памяти для ПК. Следует отметить, что, хотя кристаллы, объединяющие логические схемы и память, ранее производились, например, такими компаниями, как Toshiba, Siemens AG и Mitsubishi, подход, предложенный IBM, выгодно отличается по стоимости. Кроме того, это не влияет на производительность.

Использование новой технологии открывает широкие перспективы для создания более мощных и миниатюрных микропроцессоров и помогает создавать компактные, высокоскоростные и недорогие электронные устройства: маршрутизаторы, компьютеры, контроллеры жестких дисков, мобильные телефоны, игровые приставки.

19.2. Оперативная память

История SDRAM началась в 1992 г., а в 2000 г. она превзошла почти все другие разновидности DRAM на рынке. Промышленная группа JEDEC стандартизировала интерфейс для SDRAM в 1993 г., поэтому при использовании памяти разных производителей проблем, как правило, не возникает.

Обычная SDRAM может принимать одну команду и отправлять одно слово данных за такт. Со временем JEDEC определила стандарт для двойной скорости передачи данных (DDR). Он по-прежнему принимает одну команду за такт, но передает или читает два слова за один такт. Он может передавать одно слово по восходящему фронту тактового сигнала, а другое – по нисходящему. На практике это означает, что внутри на одну команду он читает два слова, что позволяет внутреннему таймеру работать медленнее, чем I/O. Поэтому, если тактовая частота ввода / вывода составляет 200 МГц, таймер внутренний может работать на частоте 100 МГц, и во время передачи данных он все равно будет передаваться по два слова на каждый такт I/O.

В результате они изобрели стандарт DDR2, который перестраивает память так, чтобы она работала с четырьмя внутренними словами, а затем отправляла или получала четыре слова одновременно. Тактовая частота не изменилась, поэтому задержка увеличилась. DDR3 снова удвоил свой внутренний размер данных, соответственно увеличив задержку.

DDR4 пошел другим путем. Он не удваивал внутреннюю шину памяти, но сделал перемежающийся доступ к банкам внутренней памяти для увеличения пропускной способности. Уменьшение напряжения также позволяет увеличить тактовую частоту. DDR4 появился в 2012 г.

Увеличение пропускной способности примерно совпало с увеличением количества ядер в процессорах. Таким образом, хотя чистая пропускная способность увеличилась, пропускная способность на ядро на типичной машине долгое время не менялась. Фактически, учитывая быстрое увеличение числа ядер на процессоре, его среднее значение уменьшается. Итак, пришло время для нового стандарта.

Стандарт DDR5 был определен в 2017 г. Сообщается, что пропускная способность DDR5-3200 SDRAM в 1,36 выше, чем у DDR4-3200 и, возможно, даже больше. Кроме того, размер предварительной выборки, по меньшей мере, снова удваивается (таблица 6).

Таблица 6. – Характеристики различных типов памяти

Тип	Год выпуска	Пропускная способность	Контактов на чипе	Напряжение (В)	Предвыборка
SDR	1993	1,6 ГБ/с	168	3,3	1n
DDR	2000	3,2 ГБ/с	184	2,5/2,6	2n
DDR2	2003	8,5 ГБ/с	240	1,8	4n
DDR3	2007	17 ГБ/с	240	1,35/1,5	8n
DDR4	2014	25,6 ГБ/с	380	1,2	8n
DDR5	2019	32 ГБ/с	380	1,1	8/16n
HBM2	2016	307 ГБ/с	2860	1,25/1,35	16n
GDDR6	2016	72 ГБ/с	180	1,35	16n

Как видно из таблицы 6, за 26 лет пропускная способность по сравнению с исходной памятью SDR выросла в 20 раз. Предварительная выборка из 16 слов выглядит особенно интересной, поскольку она позволяет чипу заполнять типичный КЭШ ПК за раз.

Также есть спецификация LP-DDR5 для опции памяти с низким энергопотреблением для таких устройств, как смартфоны. LP-DDR4 позволяет выбирать между двумя частотными вариантами, чтобы была возможность жертвовать скоростью ради энергопотребления. LP-DDR5 имеет три различных варианта настройки. Также есть стандарты GDDR – уже существующие в GDDR6 – для обработки графики и других высокоскоростных приложений. В долгосрочной перспективе LP-DDR5 сможет работать с пропускной способностью 6,4 Гбит/с на бит I/O, а GGDR6 сможет работать с сотнями гигобайт в секунду, в зависимости от ширины слова.

19.3. Хранилища данных

19.3.1. Перспективы твердотельных накопителей

В накопителях SSD, как правило, используется флеш-память NAND трех основных разновидностей – SLC, MLC и TLC. Память SLC (одноуровневая ячейка) хранит один бит данных, MLC (многоуровневая ячейка) – два

бита, TLC (трехуровневая ячейка) – три. Существует также 3D NAND, который, хотя и имеет другую компоновку, работает по тем же принципам. Одной из тенденций в технологии флеш-памяти является четырехуровневая NAND (четырехъярусная ячейка, QLC). Его ячейки имеют 16 уровней заряда, что означает, что он может хранить четыре бита данных.

В прошлом память QLC-NAND считалась практически невозможной из-за короткого срока службы ячеек. Переход от плоской структуры QLC-NAND (2D) к 3D-NAND позволил использовать старые технологии, в которых ячейки памяти были больше. Это привело к созданию нового, более надежного типа ячеек с большим количеством циклов перезаписи. QLC NAND в 3D-версии позволяет выпускать более быстрые и емкие твердотельные накопители.

Многие производители, включая Intel, Samsung и Toshiba, пошли по пути создания многослойных структур NAND 3D для увеличения емкости флеш-накопителей. Они стараются создавать ячейки многоуровневыми, чтобы увеличить плотность записи. В 2013 г. производственные процессы позволили создать только 24 слоя, в 2014 г. – 36, в 2015 г. – 48, затем 64. Например, первые анонсированные в 2017 г. чипы Toshiba 3D QLC NAND имеют емкость 96 Гб и используют 64 слоя. Память, созданная с использованием технологии 3D NAND, доказала свою эффективность.

19.3.2. Будущее HDD

История HDD – это история совершенствования и инноваций. С момента своего создания в 1956 г. размер жесткого диска уменьшился в 57 000 раз, емкость же увеличилась в 1 млн раз, а стоимость снизилась в 2000 раз. Цена за 1 Гб за 60 лет упала в 2 млрд раз.

Производители жестких дисков добились больших успехов, уменьшив размер пластины и, следовательно, время поиска, увеличив плотность записи и улучшив технологию чтения/записи. Количество головок чтения/записи увеличилось, появились новые интерфейсы шины и уменьшилось трение благодаря заполнению корпуса гелием.

В 2005 г. была внедрена технология перпендикулярной записи, которая позволила достичь плотности более 100 Гбит на 1 кв. дюйм. Технология размеченного хранения данных (Bit Patterned Media Recording, BPMR), предложенная компанией Toshiba в 2010 г., также способствовала повышению плотности записи. BPMR с помощью нанолитографии разбивает магнитную среду, уменьшая «размеры» бита. Ожидается, что к 2025 г. благодаря использованию технологий BPMR и HAMR удастся довести плотность записи до 10 Тбайт на 1 кв. дюйм.

Технология Shingled Magnetic Recording (SMR) также служит для увеличения плотности хранения. Суть технологии заключается в том, что но-

вый записываемый трек перекрывает часть ранее записанного трека. Предыдущая дорожка уменьшается, то есть плотность записи увеличивается. Этот подход был выбран потому, что из-за физических ограничений записывающие головки не могут быть такого же малого размера, как считывающие головки. Еще в 2002 г. компания Seagate успешно продемонстрировала магнитную запись с нагревом носителя (Heat-Assisted Magnetic Recording, HAMR), которая осуществляется с помощью лазера. Это увеличивает плотность и может в конечном итоге привести к созданию диска размером 20 Тб.

Western Digital утверждает, что ее конкурирующая технология микроволновой магнитной записи (MAMR) увеличит емкость диска до 40 Тб к 2025 г. Некоторые эксперты отрасли и производители дисков прогнозируют еще большее увеличение плотности записи, что обеспечит создание дисков до 100 Тб в следующем десятилетии. В дополнение к нагреву и микроволнам, для увеличения плотности записи предполагается использовать экстремальное охлаждение.

Таким образом, жесткие диски являются довольно многообещающими устройствами хранения. В ближайшем будущем должны появиться гибридные накопители нового поколения (использующие твердотельные накопители в качестве КЭШа жестких дисков) с улучшенными показателями цена / производительность, и использование герметичных гелиевых дисков. Гибридизация позволит сосуществовать двум типам накопителей и дополнять друг друга.

Многие из технологий HDD, упомянутые выше, появились давно, но ещё не достигли фазы массовой реализации, в то время как SSD сделали огромный скачок по всем параметрам.

ЛИТЕРАТУРА

1. Жмакин, А.П. Архитектура ЭВМ : учеб. пособие / А.П. Жамкин. – 2-е изд., перераб. и доп. – СПб. : БХВ-Петербург, 2010. – 352 с. + CD-ROM.

2. Колесниченко, О.В., Аппаратные средства PC / О.В. Колесниченко, И.В. Шишигин. – 4-е изд. перераб. и доп. – СПб. : БХВ-Петербург, 2003. – 800 с. (Энциклопедия аппаратных ресурсов персональных компьютеров)

3. Гук, М. Анатомия персонального компьютера / М. Гук, В. Юров. – СПб. : Питер, 2001. – 512 с.

4. Джордейн, Р. Справ. программиста персональных компьютеров типа IBM PC XT/AT / Р. Джордейн. – М. : Финансы и статистика, 1992. – 544 с.

5. Леонтьев, В.П. Новейшая энциклопедия компьютера 2011 / В.П. Леонтьев. – М. : ОЛМА Медиа Групп, 2010. – 960 с.

6. Касаткин, А.И. Управление ресурсами : справ. пособие / А.И. Касаткин. – Минск : Выш. шк., 1993. – 432 с.

7. Касаткин, А.И. Системное программирование / А.И. Касаткин. – Минск : Выш. шк., 1993. – 300 с.

8. Новиков, Ю.В. Аппаратура локальных сетей. Функции, выбор, разработка / Ю.В. Новиков, Д.Г. Карпенко. – М. : Эком, 1999. – 174 с.

9. Юров, В. ASSEMBLER. Учеб. курс / В. Юров, С. Хорошенко. – СПб. : Питер, 1999. – 672 с.

10. Кулаков, В. Программирование на аппаратном уровне. Спец. справ. / В. Кулаков. – СПб. : Питер, 2003. – 847 с.

11. Несвижский, В. Программирование аппаратных средств в WINDOWS / В. Несвижский. – СПб. : БХВ-Петербург, 2004. – 880 с.

12. Финогенов, К.Г. Основы разработки прикладных виртуальных драйверов [Электронный ресурс] / К.Г. Финогенов // КомпьютерПресс. – 2001. – №№ 3–12. – URL: http://www.cpress.ru.

13. Максимов, Н.В. Архитектура ЭВМ и вычислительных систем / Н.В. Максимов, Т.Л. Партыка, И.И. Попов. – М. : Форум, 2008. – 512 с.

14. Таненбаум, Э. Архитектура компьютера / Э. Таненбаум, Т. Остин. – 6-е изд. – СПб. : Питер, 2013. – 816 с.

15. Patterson, D. Computer architecture a quantitative approach / D. Patterson, J. Hennecy. – 4-th ed. – Elsevier, 2007. – 704 p.

16. Озеров, С. Новые шины [Электронный ресурс] / С. Озеров, А. Карабуто. – Ч. 1.: PCI Express – общая концепция и возможности. – Режим доступа: http://daily.sec.ru/publication.cfm?rid=45&pid=10862, http://daily.sec.ru/publication.cfm?rid=45&pid=10863.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Лабораторный практикум построен на базе программной модели ЭВМ, разработанной Жмакиным А.П. [1] и используемой в технических вузах в качестве базового учебного комплекса.

Лабораторная работа № 1.
Архитектура ЭВМ и система команд

Цель лабораторной работы: знакомство с интерфейсом модели ЭВМ, методами ввода и отладки программы, действиями основных классов команд и способов адресации.

Порядок выполнения: необходимо ввести в память ЭВМ, в соответствующее окно, текст программы (рис. 1.1), скомпилировать, выполнить в режиме **Шаг** некоторую последовательность команд (определенную вариантом задания) и зафиксировать в отчете все изменения на уровне программно-доступных объектов ЭВМ, происходящие при выполнении этих команд. Ответить на поставленный вопрос по номеру варианта задания.

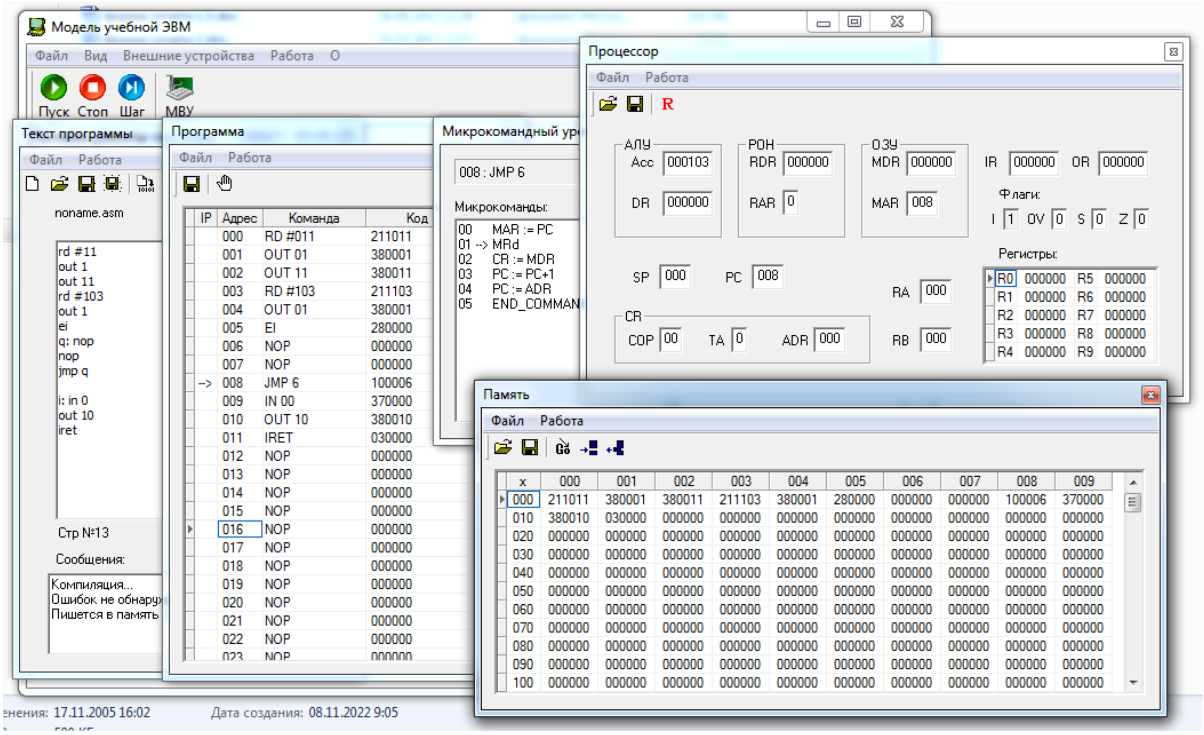


Рисунок 1.1. – Функциональная структура учебной ЭВМ

1. Общие положения

Для решения с помощью ЭВМ некоторой задачи должна быть разработана программа. Программа на языке ЭВМ представляет собой последовательность команд. Код каждой команды определяет выполняемую опера-

цию, способ адресации и адрес. Выполнение программы, записанной в памяти ЭВМ, осуществляется последовательно по командам в порядке возрастания адресов команд или в порядке, определяемом командами передачи управления.

Для того чтобы получить результат выполнения программы, пользователь должен:

– ввести программу в память ЭВМ;

– определить, если это необходимо, содержимое ячеек ОЗУ и РОН, содержащих исходные данные, а также регистров IR и BR;

– установить в РС стартовый адрес программы;

– перевести модель в режим **Работа.**

Каждое из этих действий выполняется посредством интерфейса модели, представленного на рисунке 1.1.

Ввод программы может осуществляться как в машинных кодах непосредственно в память модели, так и в мнемокодах в окно **Текст программы** с последующим ассемблированием.

Команды в память учебной ЭВМ вводятся в виде шестиразрядных десятичных чисел. Формат команды показан на рисунке 1.2.

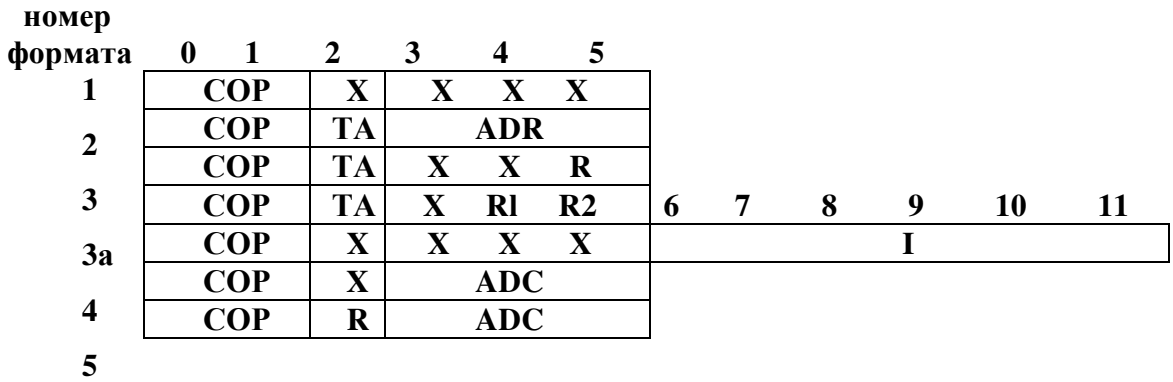


Рисунок 1.2. – Форматы машинных кодов команд учебной ЭВМ

Полный перечень кодов способов адресации команд, а также их машинных кодов операций представлен в таблицах 2-А и 3-А приложения.

В настоящей лабораторной работе программирование ЭВМ выполняется в машинных кодах.

2. Пример исполнения

Дана последовательность мнемокодов (таблица 1.1), которую необходимо преобразовать в машинные коды команд согласно схеме, представленной

на рисунке 1.2, занести в ОЗУ ЭВМ, выполнить в режиме **Шаг** и зафиксировать изменение состояний программно-доступных объектов ЭВМ в таблице 1.2.

Таблица 1.1 – Команды и коды

Последовательность	**Значения**				
Команды Assemblera	RD#20	WR30	ADD #5	WR@30	JNZ 002
Машинные коды	21 1 020	22 0 030	23 1 005	22 2 030	12 0 002

Введем полученные коды последовательно в ячейки ОЗУ, начиная с адреса 000. Выполняя команды в режиме **Шаг**, будем фиксировать изменения программно доступных объектов (в данном случае это Acc, PC и ячейки ОЗУ 020 и 030) в таблице 1.2.

Таблица 1.2. – Содержимое регистров

PC	**Acc**	**M(30)**	**M(20)**	**PC**	**Acc**	**M(30)**	**M(20)**
000	000000	000000	000000	004			000025
001	000020			002			
002		000020		003	000030		
003	000025			004			000030

3. Задание

1. Ознакомиться с архитектурой учебной модели ЭВМ.

2. Записать в ОЗУ «программу», состоящую из пяти команд (варианты задания выбрать из таблицы 1.3).

3. Команды разместить в последовательных ячейках памяти.

Таблица 1.3. – Варианты задания

№	**IR**	**Команда 1**	**Команда 2**	**Команда 3**	**Команда 4**	**Команда 5**
1	000007	IN	MUL #2	WR 10	WR @10	JNS 001
2	X	RD #17	SUB #9	WR 16	WR @16	JNS 001
3	100029	IN	ADD #16	WR 8	WR @8	JS 001
4	X	RD #2	MUL #6	WR 11	WR @11	JNZ 00
5	000016	IN	WR8	DIV #4	WR @8	JMP 002
6	X	RD #4	WR 11	RD @11	ADD #330	JS 000
7	000000	IN	WR9	RD @9	SUB #1	JS 001
8	X	RD 4	SUB #8	WR8	WR @8	JNZ 001
9	100005	IN	ADD #12	WR 10	WR @10	JS 004
10	X	RD 4	ADD #15	WR 13	WR @13	JMP 001
11	000315	IN	SUB #308	WR 11	WR @11	JMP 001
12	X	RD #988	ADD #19	WR 9	WR @9	JNZ 001
13	000017	IN	WR 11	ADD 11	WR @11	JMP 002
14	X	RD #5	MUL #9	WR 10	WR @10	JNZ 001

4. При необходимости, установить начальное значение в устройство ввода IR.

5. Определить те программно-доступные объекты ЭВМ, которые будут изменяться при выполнении этих команд.

6. Выполнить в режиме **Шаг** введенную последовательность команд, фиксируя изменения значений объектов, определенных в п. 4, в таблице (форма таблицы 1.2).

7. Если в программе образуется цикл, необходимо просмотреть не более двух повторений каждой команды, входящей в тело цикла.

4. Содержание отчета

1. Формулировка варианта задания.
2. Машинные коды команд, соответствующих варианту задания.
3. Результаты выполнения последовательности команд в форме таблицы 1.2.
4. Ответ на вопрос под номером задания.

Контрольные вопросы:

1. Из каких основных частей состоит ЭВМ и какие из них представлены в модели?
2. Что такое система команд ЭВМ?
3. Какие классы команд представлены в модели?
4. Какие действия выполняют команды передачи управления?
5. Какие способы адресации использованы в модели ЭВМ? В чем отличие между ними?
6. Какие ограничения накладываются на способ представления данных в модели ЭВМ?
7. Какие режимы работы предусмотрены в модели и в чем отличие между ними?
8. Как записать программу в машинных кодах в память модели ЭВМ?
9. Как просмотреть содержимое регистров процессора и изменить содержимое некоторых?

Лабораторная работа № 2.
Командный цикл процессора

Цель лабораторной работы: изучение реализации командного цикла процессора на уровне микрокоманд и микроопераций.

Реализация программы в ЭВМ сводится к последовательному выполнению команд. Каждая команда, в свою очередь, выполняется как последовательность микрокоманд, реализующих элементарные действия над операционными элементами процессора. Каждая микрокоманда на аппаратном уровне инициируется устройством управления микрокоманд, содержащем набор соответствующих микропрограмм.

В программной модели учебной ЭВМ предусмотрен **Режим микрокоманд,** в котором действие командного цикла реализуется и отображается на уровне микрокоманд. Список микрокоманд текущей команды выводится в специальном окне **Микрокомандный уровень** (см. рисунки 1.1 и 2.1). Для этого его необходимо вывести на панель из меню **Вид** модели и активировать его из меню **Работа**, как это показано на рисунке 2.1.

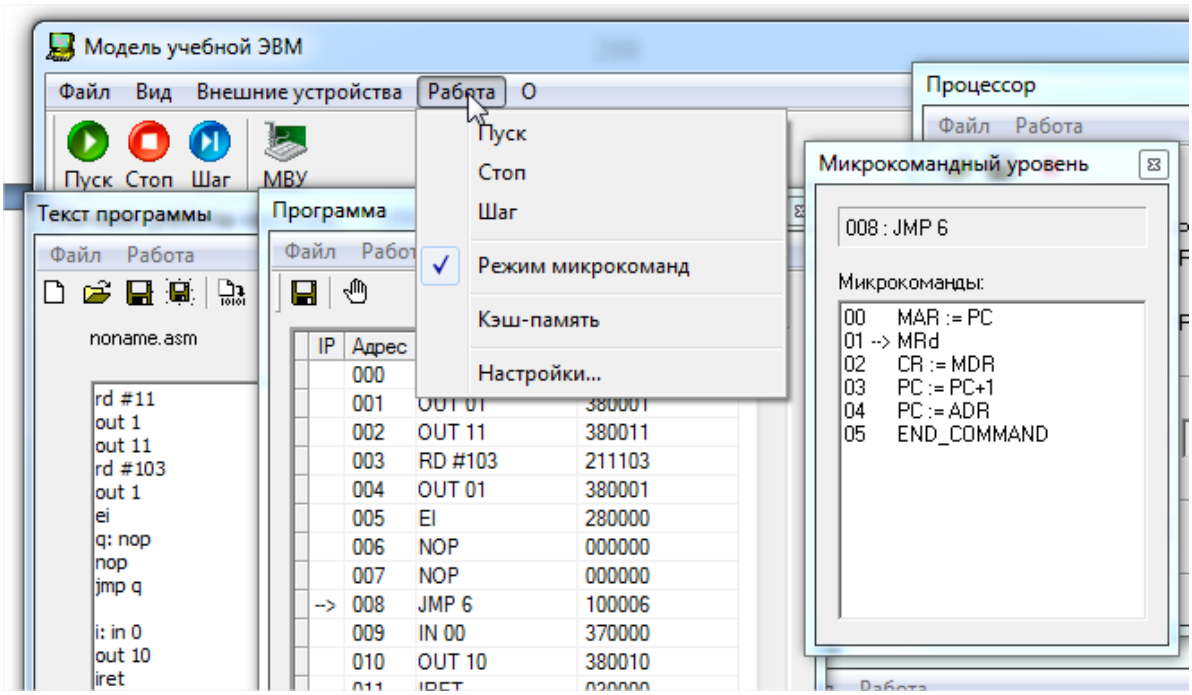


Рисунок 2.1. – Окно «Микрокомандный уровень»

Задание 1

Выполнить снова последовательность команд по варианту задания лабораторной работы № 1 (см. таблицу 1.3), но уже в режиме **Шаг.** Зарегистрировать изменения состояния процессора и памяти в форме таблицы 2.1, как показано в примере.

Таблица 2.1. – Состояние модели ЭВМ на уровне микрокоманд

Адрес (PC)	Мнемокод	Микрокоманда	ОЗУ		CR		
			MAR	MDR	COP	TA	ADR
000	RD #20	MAR := PC	000	000000	00	0	000
		MRd	000				
		CR := MDR		211020			
		PC :=PC + 1			21	1	020
001		Acc := 000.ADR					
	WR 30	MAR:=PC					
		MRd	001				
		CR := MDR		220030			
		PC :=PC + 1			22	0	030
002		MAR := ADR					
		MDR := Acc	030				
		MWr		000020			
	ADD #5	MAR :=PC					
		MRd	002				
		CR :=MDR		231005			
		PC :=PC+1			23	1	005

Окончание таблицы 2.1

CR			АУ		Ячейки	
COP	TA	ADR	Acc	DR	020	030
00	0	000	000000	000000	000000	000000
21	1	020				
			000020			
22	0	030				
						000020
23	1	005				

Задание 2

Записать последовательность микрокоманд для следующих команд модели учебной ЭВМ:

- **ADD R3**
- **ADD @R3**
- **ADD @R3+**
- **ADD -@R3**

- **JRNZ R3, M**
- **MOV R4, R2**
- **JMP M**
- **CALL M**
- **RET: PUSH R3**
- **POP R5**

Контрольные вопросы:

1. Какие микрокоманды связаны с изменением состояния аккумулятора?
2. Какие действия выполняются в модели по микрокоманде MRd?
3. Какие действия выполняются в модели по микрокоманде RWr?
4. Попробуйте составить микропрограмму (последовательность микрокоманд, реализующих команду) для несуществующей команды «умножение модулей чисел».
5. Что изменится в работе процессора, если в каждой микропрограмме микрокоманду увеличения программного счетчика PC:= PC + 1 переместить в самый конец микропрограммы?

Лабораторная работа № 3.
Программирование разветвляющегося процесса

Цель лабораторной работы: освоение методов программирования вычислений с ветвлением процесса вычислений по условию изменения адреса и методов отладки программы.

Для реализации алгоритмов, пути в которых зависят от исходных данных, используют команды условной передачи управления.

1. Пример исполнения

В качестве примера рассмотрим программу вычисления функции

$$\begin{cases} (x+11)^2 - 125, & \text{при } x \geq 16, \\ y = \dfrac{x^2 + 72x - 6400}{-168}, & \text{при } x < 16. \end{cases}$$

причем X вводится с устройства ввода IR, результат У выводится на OR.

Блок-схема алгоритма решения задачи представлена на рисунке 3.1.

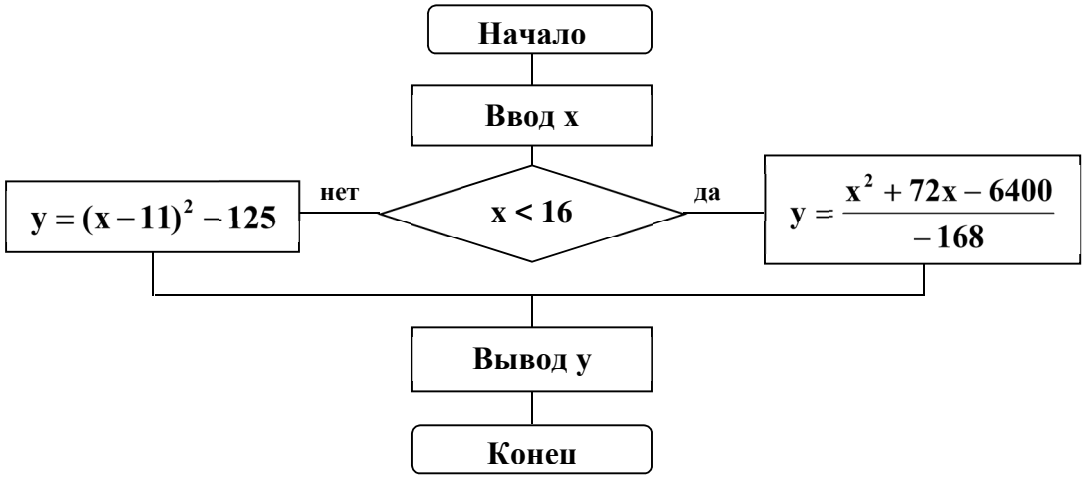


Рисунок 3.1. – Блок-схема алгоритма программы

В данной лабораторной работе используются двухсловные команды с непосредственной адресацией, позволяющие оперировать отрицательными числами и числами по модулю, превышающими 999, в качестве непосредственного операнда.

Оценив размер программы примерно в 20 – 25 команд, отведем для области данных ячейки ОЗУ, начиная с адреса 030. Составленная программа с комментариями представлена в виде таблицы 3.1.

Таблица 3.1. – Пример программы

Адрес	Команда		Комментарий
	Мнемокод	Код	
000	IN	01 0 000	Ввод **x**
001	WR 30	22 0 030	Размещение **x** в ОЗУ(ОЗО)
002	SUB #16	24 1 016	Сравнение с границей— (**x** -16)
003	JS 010	13 0 010	Переход по отрицательной разности
004	RD 30	21 0 030	Вычисления по первой формуле
005	SUB #11	24 1 011	
006	WR 31	22 0 031	
007	MUL 31	25 0 031	
008	SUB #125	24 1 125	
009	JMP 020	10 0 020	Переход на вывод результата
010	RD 30	21 0 030	Вычисления по второй формуле
O11	MUL 30	25 0 030	
012	WR 31	22 0 031	
013	RD 30	21 0 030	
014	MUL #72	25 1 072	
015	ADD 31	23 0 031	
016	ADI 106400	43 0 000	
017		**10 6400**	
018	DIVl 100168	46 0 000	
019		**10 0168**	
020	OUT	02 0 000	Вывод результата
021	HLT	09 0 000	Стоп

2. Задание

1. Разработать программу вычисления и вывода значения функции:

$$\mathbf{y} = \begin{cases} F_i(x), & \text{при } x \geq a, \\ F_j(x), & \text{при } x < a. \end{cases}$$

для вводимого из IR значения аргумента **x**.

Функции и допустимые пределы изменения аргумента приведены в таблице 3.2, варианты заданий – в таблице 3.3.

Таблица 3.2. – Функций $\mathbf{F_{i,j}(x)}$.

k	$\mathbf{F_k(x)}$	*k*	$\mathbf{F_k(x)}$
1	$\frac{\mathbf{x+17}}{\mathbf{1-x}}$; $2 \le x \le 12$	5	$\frac{\mathbf{(x+2)^2}}{\mathbf{15}}$; $50 \le x \le 75$
2	$\frac{\mathbf{(x+3)^2}}{\mathbf{x}}$; $1 \le x \le 50$	6	$\frac{\mathbf{2x^2+7}}{\mathbf{x}}$; $1 \le x \le 30$
3	$\frac{\mathbf{1000}}{\mathbf{x+10}}$; $-50 \le x \le -15$	7	$\frac{\mathbf{x^2+2x}}{\mathbf{10}}$; $-50 \le x \le 50$
4	$\mathbf{(x+3)^3}$; $-20 \le x \le 20$	8	$\frac{\mathbf{8100}}{\mathbf{x^2}}$; $1 \le x \le 90$

Таблица 3.3. – Варианты задания.

Номер варианта	*i*	*j*	*a*	Номер варианта	*i*	*j*	*a*
1	2	1	12	8	8	6	30
2	4	3	-20	9	2	6	25
3	8	4	15	10	5	7	50
4	6	1	12	11	2	4	18
5	5	2	50	12	8	1	12
6	7	3	15	13	7	6	25
7	6	2	11	14	1	4	5

2. Исходя из допустимых пределов изменения аргумента функций (см. таблицу 6) и значения параметра ***a*** для своего варианта задания (см. таблицу 7) выделить на числовой оси ***Ox*** области, в которых функция **y** вычисляется по представленной в п. 1 формуле, и недопустимые значения аргумента. На недопустимых значениях аргумента программа должна выдавать на OR максимальное отрицательное число: 199 999.

3. Ввести текст программы в окно **Текст программы,** при этом возможен набор и редактирование текста непосредственно в окне **Текст программы** или загрузка текста из файла, подготовленного в другом редакторе.

4. Ассемблировать текст программы, при необходимости исправить синтаксические ошибки.

5. Отладить программу. Для этого:

а) записать в IR значение аргумента x > *a* (в области допустимых значений);

б) записать в PC стартовый адрес программы;

в) проверить правильность выполнения программы (т. е. правильность результата и адреса останова) в автоматическом режиме. В случае наличия ошибки выполнить пп. 5, *г)* и 5, *д);* иначе перейти к п. 5. *е)*;

г) записать в PC стартовый адрес программы;

д) наблюдая выполнение программы в режиме **Шаг**, найти команду, являющуюся причиной ошибки; исправить ее; выполнить пп. *5, а)* – 5, *в)*;

е) записать в IR значение аргумента x < *a* (в области допустимых значений); выполнить п. 5, *б)* и 5, *в)*;

ж) записать в IR недопустимое значение аргумента x и выполнить пп. 5, *б)* и 5, *е)*.

6. Для выбранного допустимого значения аргумента **x** наблюдать выполнение отлаженной программы в режиме **Шаг** и записать в форме таблицы 3.1 содержимое регистров ЭВМ перед выполнением каждой команды.

3. Содержание отчета

Отчет о лабораторной работе должен содержать следующие разделы:

1. Формулировка варианта задания.
2. Блок-схема алгоритма решения задачи.

3. Размещение данных в ОЗУ.

4. Программа в форме таблицы 3.1.

5. Последовательность состояний регистров ЭВМ при выполнении программы в режиме **Шаг** для одного значения аргумента.

6. Результаты выполнения программы для нескольких значений аргумента, выбранных самостоятельно.

Контрольные вопросы:

1. Как работает механизм косвенной адресации?

2. Какая ячейка будет адресована в команде с косвенной адресацией через ячейку 043, если содержимое этой ячейки равно 102 347?

3. Как работают команды передачи управления?

4. Что входит в понятие «отладка программы»?

5. Какие способы отладки программы можно реализовать в модели?

6. Для чего вводятся ограничения изменения аргумента и что происходит за их пределами?

Лабораторная работа № 4.
Программирование цикла с переадресацией

Цель лабораторной работы: освоение различных методов адресации и адресации с использованием счетчиков.

При решении задач, связанных с обработкой массивов, возникает необходимость изменения исполнительного адреса при повторном выполнении некоторых команд. Эта задача может быть решена путем использования косвенной адресации.

1. Пример исполнения

Разработать программу вычисления суммы элементов массива чисел C_1, C_2, ..., C_n. Исходными данными в этой задаче являются: n – количество суммируемых чисел и C_1, C_2, ..., C_n – массив суммируемых чисел. Заметим, что должно выполняться условие $n > 1$, т.к. алгоритм предусматривает, по крайней мере, одно суммирование. Кроме того, предполагается, что суммируемые числа записаны в ОЗУ подряд, т.е. в ячейки памяти с последовательными адресами. Результатом является сумма S.

Составим программу для вычисления суммы со следующими конкретными параметрами: число элементов массива – 10, элементы массива расположены в ячейках ОЗУ по адресам 040, 041, 042, ..., 049. Используемые для решения задачи промежуточные переменные имеют следующий смысл: A_i – адрес числа C_i, $i \in \{1, 2, ..., 10\}$; ОЗУ(A_i) – число по адресу A_i, S – текущая сумма; k – счетчик цикла, определяющий число повторений тела цикла.

Распределение памяти таково. Программу разместим в ячейках ОЗУ, начиная с адреса 000, примерная оценка объема программы – 20 команд; промежуточные переменные: A_i – в ячейке ОЗУ с адресом 030, k – по адресу 031, S – по адресу 032. Блок-схема алгоритма программы показана на рисунке 4.1, текст программы с комментариями приведен в таблице 4.1.

Таблица 4.1. – Текст программы примера

Адрес	Команда	Примечание
1	2	3
000	RD #40	Загрузка начального адреса массива 040
001	WR 30	в ячейку 030
002	RD #10	Загрузка параметра цикла k = 10 в ячейку 031
003	WR 31	
004	RD #0	Загрузка начального значения суммы S = 0
005	WR 32	в ячейку 032
006	Ml: RD 32	Добавление
007	ADD @30	к текущей сумме
008	WR 32	очередного элемента массива
009	RD 30	Модификация текущего

Окончание таблицы 4.1.

1	2	3
010	ADD #1	адреса массива
011	WR 30	(переход к следующему адресу)
012	RD 31	Уменьшение счетчика
013	SUB #1	(параметра цикла)
014	WR 31	на 1
015	JNZ Ml	Проверка параметра цикла и переход при к≠0
016	RD 32	Вывод
017	OUT	Результата
018	HLT	Стоп

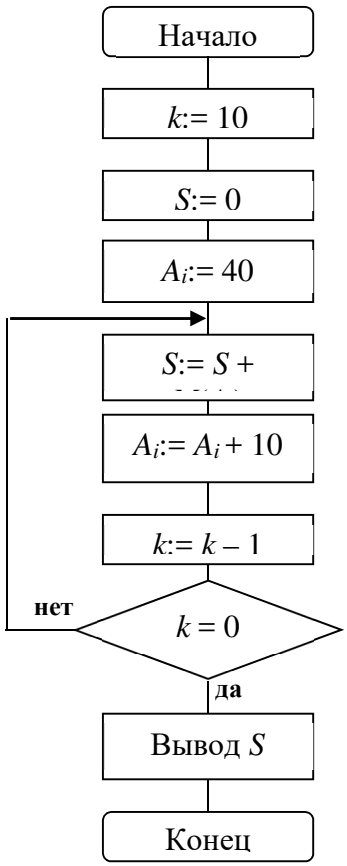


Рисунок 4.1. – Блок-схема алгоритма программы для примера

2. Задание

1. Написать программу определения заданной характеристики последовательности чисел $C_1, C_2, ..., C_n$. Варианты заданий приведены в таблице 4.2.

2. Записать программу в мнемокодах, введя ее в поле окна **Текст программы.**

3. Сохранить набранную программу в виде текстового файла и произвести ассемблирование мнемокода программы.

4. Загрузить в ОЗУ необходимые константы и исходные данные.

5. Отладить программу.

Таблица 4.2. – Варианты задания

Номер варианта	Характеристика последовательности чисел C_1, C_2, ..., C_n
1	Количество четных чисел
2	Номер минимального числа
3	Произведение всех чисел
4	Номер первого отрицательного числа
5	Количество чисел, равных C_1
6	Количество отрицательных чисел
7	Максимальное отрицательное число
8	Номер первого положительного числа
9	Минимальное положительное число
10	Номер максимального числа
11	Количество нечетных чисел
12	Количество чисел, меньших C_1
13	Разность сумм четных и нечетных элементов массива
14	Отношение сумм четных и нечетных элементов массива

Примечание. – Под четными (нечетными) элементами массивов понимаются элементы массивов, имеющие четные (нечетные) индексы. Четные числа – элементы массивов, делящиеся без остатка на 2 (т.е. проверка наличия остатка дает 0).

3. Содержание отчета:

1. Формулировка варианта задания.
2. Блок-схема алгоритма решения задачи.
3. Распределение памяти (размещение в ОЗУ переменных, программы и необходимых констант).
4. Текст программы с комментариями.
5. Значения исходных данных и результата выполнения программы.

Контрольные вопросы:

1. Как организовать цикл в программе?
2. Что такое параметр цикла?
3. Как происходит выход из цикла?
4. Как поведет себя программа, если в ней будет отсутствовать команда WR31 по адресу 014?
5. Как поведет себя программа, если метка M1 будет поставлена по адресу 005?
6. Как поведет себя программа, если метка M1 будет поставлена по адресу 007?

Лабораторная работа № 5.
Работа с подпрограммами и стеком

Цель лабораторной работы: изучение организации программ с использованием подпрограмм и стека.

В программировании часто встречаются ситуации, когда одинаковые действия необходимо выполнять многократно в разных частях программы (например, вычисление функции sin *x).* При этом с целью экономии памяти не следует многократно повторять одну и ту же последовательность команд – достаточно один раз написать так называемую *подпрограмму* (в терминах языков высокого уровня – процедуру), обеспечить правильный вызов этой подпрограммы и возврат в точку вызова по завершению подпрограммы.

Для *вызова* подпрограммы необходимо указать ее начальный адрес в памяти и передать (если необходимо) параметры – те исходные данные, с которыми будут выполняться предусмотренные в подпрограмме действия. Адрес подпрограммы указывается в команде вызова CALL, а параметры могут передаваться через определенные ячейки памяти, регистры или стек.

Возврат в точку вызова обеспечивается сохранением адреса текущей команды (содержимого регистра PC) при вызове и использованием в конце подпрограммы команды возврата RET, которая возвращает сохраненное значение адреса возврата в PC.

Для реализации механизма вложенных подпрограмм (возможность вызова подпрограммы из другой подпрограммы и т.д.) адреса возврата целесообразно сохранять в стеке. *Стек* («магазин») – особым образом организованная безадресная память, доступ к которой осуществляется через единственную ячейку, называемую *верхушкой стека*. При записи слово помещается в верхушку стека, предварительно все находящиеся в нем слова смещаются вниз на одну позицию; при чтении извлекается содержимое верхушки стека (оно при этом из стека исчезает), а все оставшиеся слова смещаются вверх на одну позицию. Такой механизм напоминает действие магазина стрелкового оружия (отсюда и второе название). В программировании называют такую дисциплину обслуживания LIFO (Last In First Out, последним пришел – первым вышел) в отличие от дисциплины типа *очередь* – FIFO (First In First Out, первым пришел – первым вышел).

В обычных ОЗУ нет возможности перемещать слова между ячейками, поэтому при организации стека перемещается не массив слов относительно неподвижной верхушки, а верхушка относительно неподвижного массива. Под стек отводится некоторая область ОЗУ, причем адрес верхушки хранится в специальном регистре процессора – указателе стека SP.

В стек можно поместить содержимое регистра общего назначения по команде PUSH или извлечь содержимое верхушки в регистр общего назначения по команде POP. Кроме того, по команде вызова подпрограммы CALL

значение программного счетчика PC (адрес следующей команды) помещается в верхушку стека, а по команде RET содержимое верхушки стека извлекается в PC. При каждом обращении в стек указатель SP автоматически модифицируется.

В большинстве ЭВМ стек «растет» в сторону меньших адресов, поэтому перед каждой записью содержимое SP уменьшается на 1, а после каждого извлечения содержимое SP увеличивается на 1. Таким образом, SP всегда указывает на верхушку стека.

В процессе организации циклов мы будем использовать новые возможности системы команд модели ЭВМ, которые позволяют работать с новым классом памяти – сверхоперативной (регистры общего назначения – РОН). В реальных ЭВМ доступ в РОН занимает значительно меньшее время, чем в ОЗУ; кроме того, команды обращения с регистрами короче команд обращения к памяти. Поэтому в РОН размещаются наиболее часто используемые в программе данные, промежуточные результаты, счетчики циклов, косвенные адреса и т.п.

В системе команд учебной ЭВМ для работы с РОН используются специальные команды, мнемоники которых совпадают с мнемониками соответствующих команд для работы с ОЗУ, но в адресной части содержат символы регистров RO – R9.

Кроме обычных способов адресации (прямой и косвенной) в регистровых командах используются два новых – постинкрементная и преддекрементная (см. таблицу 2-А). Кроме того, к регистровым относится команда организации цикла JRNZ R, M. По этой команде содержимое указанного в команде регистра уменьшается на 1, и если в результате вычитания содержимое регистра не равно 0, то управление передается на метку M. Эту команду следует ставить в конце тела цикла, метку M – в первой команде тела цикла, а в регистр R помещать число повторений цикла.

1. Пример исполнения

Даны три массива чисел. Требуется вычислить среднее арифметическое их максимальных элементов. Каждый массив задается двумя параметрами: адресом первого элемента и длиной.

Очевидно, в программе трижды необходимо выполнить поиск максимального элемента массива, поэтому следует написать соответствующую подпрограмму. Параметры в подпрограмму будем передавать через регистры: R1 – начальный адрес массива, R2 – длина массива.

Рассмотрим конкретную реализацию этой задачи. Пусть первый массив начинается с адреса 085 и имеет длину 14 элементов, второй – 100 и 4, третий – 110 и 9. Программа будет состоять из основной части и подпрограммы. Основная программа задает параметры подпрограмме, вызывает ее

и сохраняет результаты работы подпрограммы в рабочих ячейках. Затем осуществляет вычисление среднего арифметического и выводит результат на устройство вывода. В качестве рабочих ячеек используются регистры общего назначения R6 и R7 – для хранения максимальных элементов массивов. Подпрограмма получает параметры через регистры R1 (начальный адрес массива) и R2 (длина массива). Эти регистры используются подпрограммой в качестве регистра текущего адреса и счетчика цикла, соответственно. Кроме того, R3 используется для хранения текущего максимума, а R4 – для временного хранения текущего элемента. Подпрограмма возвращает результат через аккумулятор.

В таблице 5.1 приведен текст основной программы и подпрограммы. Обратите внимание: цикл в подпрограмме организован с помощью команды JRNZ, а модификация текущего адреса – средствами постинкрементной адресации.

Таблица 5.1. – Программа примера

Команда	**Примечания**	**Команда**	**Примечания**
Основная программа		**Подпрограмма MAX**	
RD #85	Загрузка	M: RD @R1	Загрузка
WR R1	параметров	WR R3	первого элемента в R3
RD #14	первого	L2: RD @R1+	Чтение элемента и модификация адреса
WR R2	массива	WR R4	Сравнение
CALL M	Вызов подпрограммы	SUB R3	и замена,
WR R6	Сохранение результата	JS LI	Если R3 < R4
RD #100	Загрузка	MOV R3, R4	
WR R1	параметров	LI: JRNZ R2, L2	Цикл
RD #4	второго	RD R3	Чтение результата в Acc
WR R2	массива	RET	Возврат
CALL M	Вызов подпрограммы		
WR R7	Сохранение результата		
RD #110	Загрузка		
WR R1	параметров		
RD #9	третьего		
WR R2	массива		
CALL M	Вызов подпрограммы		
ADD R7	Вычисление		
ADD R6	среднего		
DIV #3	арифметического		
OUT	Вывод результата		
HLT	Стоп		

2. Задание

Составить и отладить программу учебной ЭВМ для решения следующей задачи: задается **три массива** в памяти начальными адресами и длинами. Варианты параметров обработки массивов даны в таблице 5.2. Вычислить и вывести **среднее арифметическое** параметров этих массивов.

Таблица 5.2. – Варианты задания

№ варианта	Характеристика последовательности чисел C_1, C_2, ..., C_n
1	Количество чисел, равных C_1
2	Максимальное отрицательное число
3	Разность сумм четных и нечетных элементов массива
4	Количество нечетных чисел
5	Количество чисел, меньших C_1
6	Минимальное положительное число
7	Количество четных чисел
8	Номер максимального числа
9	Отношение сумм четных и нечетных элементов массива
10	Произведение всех чисел
11	Количество отрицательных чисел
12	Номер первого положительного числа
13	Номер минимального числа
14	Номер первого отрицательного числа

3. Содержание отчета:

1. Формулировка варианта задания.
2. Блок-схема алгоритма основной программы.
3. Блок-схема алгоритма подпрограммы.
4. Распределение памяти (размещение в ОЗУ переменных, программы и необходимых констант).
5. Тексты программы и подпрограммы с комментариями.
6. Значения исходных данных и результата выполнения программы.

Контрольные вопросы:

1. Как работает команда MOV R3, R7?
2. Какие действия выполняет процессор при реализации команды CALL?
3. Как осуществляется возврат и подпрограммы?
4. Как поведет себя программа примера 4, если в ней вместо команд CALL M использовать команды JMP M?
5. После начальной установки процессора (сигнал Сброс) указатель стека SP устанавливается в 000. По какому адресу будет производиться запись в стек первый раз, если не загружать SP командой WRSP?
6. Как используя механизмы постинкрементной и преддекрементной адресации организовать дополнительный стек в произвольной области памяти, не связанный с SP?

Лабораторная работа № 6.
Программирование внешних устройств

Цель лабораторной работы: изучение способов организации взаимодействия процессора и внешних устройств (ВУ) в составе ЭВМ.

Выше отмечалось, что связь процессора и ВУ может осуществляться в синхронном или асинхронном режиме.

Синхронный режим используется для ВУ, всегда готовых к обмену данными. В нашей модели такими ВУ являются дисплей и тоногенератор – процессор может обращаться к этим ВУ, не анализируя их состояние (правда дисплей блокирует прием данных после ввода 128 символов, формируя флаг ошибки).

Асинхронный обмен предполагает анализ процессором состояния ВУ, которое определяет готовность ВУ выдать или принять данные или факт осуществления некоторого события, контролируемого системой. К таким устройствам в нашей модели можно отнести клавиатуру и блок таймеров.

Анализ состояния ВУ может осуществляться процессором двумя способами:

1) в программно-управляемом режиме;

2) в режиме прерывания.

В первом случае предполагается программное обращение процессора к регистру состояния ВУ с последующим анализом значения соответствующего разряда слова состояния. Такое обращение следует предусмотреть в программе с некоторой периодичностью, независимо от фактического наступления контролируемого события (например, нажатие клавиши).

Во втором случае при возникновении контролируемого события ВУ формирует процессору запрос на прерывание программы, по которому процессор и осуществляет связь с ВУ.

1. Задание

Свой вариант задания (таблица 6.1) требуется выполнить двумя способами – сначала в режиме программного контроля, далее модифицировать программу таким образом, чтобы события обрабатывались в режиме прерывания программы. Поскольку «фоновая» (основная) задача для этого случая в заданиях отсутствует, роль ее может сыграть «пустой цикл» типа:

```
M:  NOP
    NOP
    JMP  M
```

Таблица 6.1. – Варианты задания

№	Задание	Используемые ВУ	Пояснения
1	2	3	4
1	Ввод пятиразрядных чисел в ячейки ОЗУ	Клавиатура	Программа должна обеспечивать ввод последовательности ASCII-кодов десятичных цифр (не длиннее 5), перекодировку в «8421», упаковку в десятичное число (первый введенный символ – старшая цифра) и размещение в ячейке ОЗУ. ASCII-коды нецифр игнорировать
2	Программа ввода символов с клавиатуры с выводом на дисплей	Клавиатура, дисплей, таймер	Очистка буфера клавиатуры после ввода 50 символов или каждые 10 с
3	Вывод на дисплей трех текстов, хранящихся в памяти, с задержкой	Дисплей, таймер	Первый текст выводится сразу при запуске программы, второй через 15 с, третий – через 20 с после второго
4	Вывод на дисплей одного из трех текстовых сообщений, в зависимости от нажатой клавиши	Клавиатура, дисплей	<1> – вывод на дисплей первого, текстового сообщения; <2> – второго, <3> – третьего. Остальные символы – нет реакции
5	Выбирать из потока ASCII-кодов только цифры и выводить их на дисплей	Клавиатура, дисплей, тоногенератор	Вывод каждой цифры сопровождается коротким звуковым сигналом
6	Выводить на дисплей каждый введенный с клавиатуры символ, причем цифру выводить «в трех экземплярах»	Клавиатура, дисплей, тоногенератор	Вывод каждой цифры сопровождается троекратным звуковым сигналом
7	Селективный ввод символов с клавиатуры	Клавиатура, дисплей	Все русские буквы, встречающиеся в строке ввода, – в верхнюю часть экрана дисплея (строки 1–4), все цифры – в нижнюю часть экрана (строки 5–8), остальные символы не выводить
8	Вывод содержимого заданного участка памяти на дисплей посимвольно с заданным промежутком времени между выводами символов	Дисплей, таймер	Остаток от деления на 256 трех младших разрядов ячейки памяти рассматривается как ASCII-код символа. Начальный адрес памяти, длина массива вывода и промежуток времени – параметры подпрограммы
9	Программа ввода символов с клавиатуры с выводом на дисплей	Клавиатура, дисплей	Очистка буфера клавиатуры после ввода 35 символов

Окончание таблицы 6.1.

1	2	3	4
10	Выводить на дисплей каждый введенный с клавиатуры символ, причем заглавную русскую букву выводить «в двух экземплярах»	Клавиатура, дисплей, таймер	Очистка буфера клавиатуры после ввода 48 символов, очистка экрана каждые 15 с
11	Вывод на дисплей содержимого группы ячеек памяти в числовой форме (адрес и длина группы – параметры подпрограммы)	Дисплей, таймер	Содержимое ячейки распаковывается (с учетом знака), каждая цифра преобразуется в соответствующий ASCII-код и выдается на дисплей. При переходе к выводу содержимого очередной ячейки формируется задержка 10 с
12	Определить промежуток времени между двумя последовательными нажатиями клавиш	Клавиатура, таймер	Результат выдается на OR. (Учитывая инерционность модели, нажатия не следует производить слишком быстро)

2. Задания повышенной сложности

1. Разработать программу-тест на скорость ввода символов с клавиатуры. По звуковому сигналу включается клавиатура и таймер на T секунд. Можно начинать ввод символов, причем каждый символ отображается на дисплее, ведется подсчет количества введенных символов (после каждых 50 дается команда на очистку буфера клавиатуры, после 128 – очищается дисплей). Переполнение таймера выключает клавиатуру и включает сигнал завершения ввода (можно тон этого сигнала сопоставить с количеством введенных символов). Параметр T вводится из IR. Результат S – средняя скорость ввода (символ/с) выдается на OR. Учитывая, что модель учебной ЭВМ оперирует только целыми числами, можно выдавать результат в формате $S \times 60$ символов/мин.

2. Разработать программу-тест на степень запоминания текста. Три различных вариантах текста выводятся последовательно на дисплей на T_1 секунд с промежутками T_2 секунд. Далее эти тексты (то, что запомнилось) вводятся с клавиатуры (в режиме ввода строки) и программно сравниваются с исходными текстами. Выдается количество (процент) ошибок.

3. Разработать программу-калькулятор. Осуществлять ввод из буфера клавиатуры последовательности цифр, упаковку (см. задание 1 в таблице 6.1). Разделители – знаки бинарных арифметических операций и =. Результат переводится в ASCII-коды и выводится на дисплей.

3. Порядок выполнения работы

1. Запустить программную модель учебной ЭВМ и подключить к ней определенные в задании внешние устройства (меню: **Внешние устройства \ Менеджер ВУ).**

2. Написать и отладить программу, предусмотренную заданием, с использованием программного анализа флагов готовности ВУ. Продемонстрировать работающую программу преподавателю.

3. Изменить отлаженную в п. 2 программу таким образом, чтобы процессор реагировал на готовность ВУ с помощью подсистемы прерывания. Продемонстрировать работу измененной программы преподавателю.

4. Содержание отчета:

1. Текст программы с программным анализом флагов готовности ВУ.
2. Текст программы с обработчиком прерывания.

Контрольные вопросы:

1. При каких условиях устанавливается и сбрасывается флаг готовности клавиатуры Rd?
2. Возможно ли в блоке таймеров организовать работу всех трех таймеров с разной тактовой частотой?
3. Как при получении запроса на прерывание от блока таймеров определить номер таймера, достигшего состояния 99 999 (00 000)?
4. Какой текст окажется на экране дисплея, если после нажатия в окне обозревателя дисплея кнопки **Очистить** и загрузки по адресу CR (11) константы #10 вывести по адресу DR (10) последовательно пять ASCII-кодов русских букв А, Б, В, Г, Д?
5. В какой области памяти модели ЭВМ могут располагаться программы-обработчики прерываний?
6. Какие изменения в работе отлаженной вами второй программы произойдут, если завершить обработчик прерываний командой RET, а не IRET?

Лабораторная работа № 7.
Принципы работы КЭШ-памяти

Цель лабораторной работы: проверить работу различных алгоритмов замещения данных в кэш-памяти при различных режимах записи.

1. Задание

В качестве задания предлагается некоторая короткая «программа» (таблица 7.2), которую необходимо выполнить с подключенной КЭШ-памятью (размером 4 и 8 ячеек) в шаговом режиме для следующих двух вариантов алгоритмов замещения (таблица 7.1).

Таблица 7.1. – Пояснения к вариантам задания

Номера вариантов	Режим записи	Алгоритм замещения
1, 7, 11	Сквозная	СЗ, без учета бита записи
	Обратная	О, с учетом бита записи
2, 5, 9	Сквозная	БИ, без учета бита записи
	Обратная	О, с учетом бита записи
3, 6, 12	Сквозная	О, без учета бита записи
	Обратная	СЗ, с учетом бита записи
4, 8, 10	Сквозная	БИ, без учета бита записи
	Обратная	БИ, с учетом бита записи

Таблица 7.2. – Варианты задания

№ вар	Номера команд программы						
	1	2	3	4	5	6	7
1	RD #12	WR 10	WR @10	ADD 12	WR R0	SUB 10	PUSH R0
2	RD #65	WR R2	MOV R4, R2	WR 14	PUSH R2	POP R3	CALL 002
3	RD #16	SUB #5	WR 9	WR 09	WR R3	PUSH R3	POP R4
4	RD #99	WR R6	MOV R7, R6	ADD R7	PUSH R7	CALL 006	POP R8
5	RD #11	WR R2	WR -@R2	PUSH R2	CALL 005	POP R3	RET
6	RD #19	SUB #10	WR 9	ADD #3	WR @9	CALL 006	POPR4
7	RD #6	CALL 006	WR 11	WR R2	PUSH R2	RET	JMP 002
8	RD #8	WR R2	WR @R2+	PUSH R2	POP R3	WR -@R3	CALL 003
9	RD #13	WR 14	WR @14	WR @13	ADD 13	CALL 006	RET
10	RD #42	SUB #54	WR 16	WR @16	WR R1	ADD @R1+	PUSH Rl
11	RD #10	WR R5	ADD R5	WR R6	CALL 005	PUSH R6	RET
12	JMP 006	RD #76	WR 14	WR R2	PUSH R2	RET	CALL 001

Не следует рассматривать заданную последовательность команд, как фрагмент программы. Некоторые конструкции, например, последовательность команд PUSH R6, RET в общем случае не возвращает программу в точку вызова подпрограммы. Такие группы команд введены в задание для того, чтобы обратить внимание студентов на особенности функционирования стека.

Напомним, что программа определяется как последовательность команд, выполнение которых позволит получить некий *результат.*

2. Порядок выполнения работы

1. Написать и ввести в модель учебной ЭВМ текст своего варианта программы (см. таблицу 7.2), ассемблировать его и сохранить на диске в виде txt-файла.

2. Установить параметры КЭШ-памяти размером 4 ячейки, выбрать режим записи и алгоритм замещения в соответствии с первой строкой своего варианта из таблицы 7.1.

3. В шаговом режиме выполнить программу, фиксируя после каждого шага состояние КЭШ-памяти.

4. Для одной из команд записи (WR) перейти в режим **Такт** и отметить, в каких микрокомандах происходит изменение КЭШ-памяти.

5. Для КЭШ-памяти размером 8 ячеек установить параметры в соответствии со второй строкой своего варианта из таблицы 7.1 и выполнить программу в шаговом режиме еще раз, фиксируя последовательность номеров замещаемых ячеек КЭШ-памяти.

3. Содержание отчета:

1. Вариант задания – текст программы и режимы КЭШ-памяти.

2. Последовательность состояний КЭШ-памяти размером 4 ячейки при однократном выполнении программы (команды 1 – 7).

3. Последовательность микрокоманд при выполнении команды WR с отметкой тех микрокоманд, в которых возможна модификация КЭШ-памяти.

4. Для варианта КЭШ-памяти размером 8 ячеек – последовательность номеров замещаемых ячеек КЭШ-памяти для второго варианта параметров КЭШ-памяти при двукратном выполнении программы (команды 1 – 7).

Контрольные вопросы:

1. В чем смысл включения КЭШ-памяти в состав ЭВМ?
2. Как работает КЭШ-память в режиме обратной записи?
3. Как работает КЭШ-память в режиме сквозной записи?
4. Как зависит эффективность работы ЭВМ от размера КЭШ-памяти?

Лабораторная работа № 8.
Алгоритмы замещения строк КЭШ-памяти

Цель лабораторной работы: изучение влияния параметров КЭШ-памяти и выбранного алгоритма замещения на эффективность работы системы.

Эффективность в данном случае оценивается числом КЭШ-попаданий по отношению к общему числу обращений к памяти. Учитывая разницу в алгоритмах в режимах *сквозной* и *обратной записи,* эффективность использования КЭШ-памяти вычисляется выражениям (2) и (3) описания модели, соответственно, для сквозной и обратной записей.

Очевидно, эффективность работы системы с КЭШ-памятью будет зависеть не только от параметров КЭШ-памяти и выбранного алгоритма замещения, но и от класса решаемой задачи. Так, линейные программы должны хорошо работать с алгоритмами замещения типа *очередь,* а программы с большим числом условных переходов, зависящих от случайных входных данных, могут давать неплохие результаты с алгоритмами *случайного замещения*. Можно предположить, что программы, имеющие большое число повторяющихся участков (часто вызываемых подпрограмм и/или циклов) при прочих равных условиях обеспечат более высокую эффективность применения КЭШ-памяти, чем линейные программы. Разумеется, на эффективность напрямую должен влиять размер КЭШ-памяти.

Для проверки высказанных выше предположений выполняется настоящая лабораторная работа.

1. Задание

В данной лабораторной работе все варианты задания одинаковы: исследовать эффективность работы КЭШ-памяти при выполнении двух разнотипных программ, написанных и отлаженных вами при выполнении лабораторных работ № 2 и № 4.

2. Порядок выполнения работы

1. Загрузить в модель учебной ЭВМ отлаженную программу из лабораторной работы № 2.

2. В меню **Работа** установить режим **Кэш-память.**

3. В меню **Вид** выбрать команду **Кэш-память,** открыв тем самым окно **Кэш-память,** в нем нажать первую слева кнопку на панели инструментов, открыв диалоговое окно **Параметры кэш-памяти,** и установить следующие параметры КЭШ-памяти: размер – 4, режим записи – сквозная, алгоритм замещения – случайное, без учета бита записи (W).

4. Запустить программу в автоматическом режиме; по окончании работы просмотреть результаты работы КЭШ-памяти в окне **Кэш-память**, вычислить значение коэффициента эффективности *K* и записать в ячейку таблицы 8.1, помеченную звездочкой (*).

5. Выключить КЭШ-память модели (**Работа | Кэш-память**) и изменить один из ее параметров – установить флаг с учетом бита записи (в окне **Параметры кэш-памяти).**

6. Повторить п. 4, поместив значение полученного коэффициента эффективности в следующую справа ячейку таблицы 8.1.

7. Последовательно меняя параметры КЭШ-памяти, повторить пп. 3–5, заполняя все ячейки таблицы 8.1.

Совет: При очередном запуске программы не забывайте устанавливать процессор модели в начальное состояние, нажимая кнопку **R** в окне **Процессор!**

8. Повторить все действия, описанные в пп. 1–7 для программы из лабораторной работы № 4, заполняя вторую таблицу по форме таблицы 8.1.

Таблица 8.1. – Результаты эксперимента

Способ	**Сквозная запись**					
Алгоритм	**Случайное замещение**		**Очередь**		**Бит U**	
Размер	**без W**	**с W**	**без W**	**с W**	**без W**	**с W**
4	*					
8						
16						
32						
Способ	**Обратная запись**					
Алгоритм	**Случайное замещение**		**Очередь**		**Бит U**	
Размер	**без W**	**с W**	**без W**	**с W**	**без W**	**с W**
4						
8						
16						
32						

3. Содержание отчета:

1. Две таблицы по форме таблиц 5.1 и 8.1 с результатами моделирования программ из лабораторных работ № 5 и № 8 при разных режимах работы КЭШ-памяти.

2. Выводы, объясняющие полученные результаты.

Контрольные вопросы:

1. Как работает алгоритм замещения *очередь* при установленном флажке **С учетом бита записи** в диалоговом окне **Параметры кэш-памяти?**

2. Какой алгоритм замещения будет наиболее эффективным в случае применения КЭШ-памяти большого объема (в КЭШ-память целиком помещается программа)?

3. Как скажется на эффективности алгоритмов замещения учет значения бита записи W при работе КЭШ-памяти в режиме обратной записи?

4. Как скажется на эффективности алгоритмов замещения учет значения бита записи W при работе КЭШ-памяти в режиме сквозной записи?

5. Для каких целей в структуру ячейки КЭШ-памяти включен бит использования?

6. Как устанавливается и сбрасывается бит использования?

Лабораторная работа № 9.
Контроллер прерываний ПЭВМ (изучение функций прерывания 21h)

Цель работы: изучить прерывание 21h и возможности его применения для ввода информации с клавиатуры, вывода на экран, а также получить навыки работы с файлами через описатели.

Краткие теоретические сведения

Любая программа может инициировать синхронное программное прерывание просто путем выполнения команды INT. MS-DOS использует для взаимодействия со своими модулями и прикладными программами прерывания от 20h до 3Fh. (Например, доступ к диспетчеру функций MS-DOS осуществляется выполнением Int 21h.) Программы BIOS, хранящиеся в ПЗУ, и прикладные программы IBM PC используют другие прерывания, с большими или меньшими номерами. Приведем примеры некоторых программных прерываний:

Номер	Описание
10h	Обслуживание видеоадаптера
14h	Работа с асинхронным последовательным адаптером
16h	Обслуживание клавиатуры
17h	Обслуживание принтера
21h	Используется MS-DOS, предоставляет пользователю множество функций для работы с клавиатурой, дисплеем, дисковой подсистемой и т.д.

Вывод на экран

Вывод информации в ассемблерных программах осуществляется обычно при помощи сервисных функций DOS (прерывание 21h) или BIOS (прерывание 10h). Процесс вывода состоит в следующем:

− определенные регистры микропроцессора загружаются выводимой информацией или адресом буфера, содержащего выводимую информацию;

− в регистр AH заносится номер используемой для операции вывода функции;

− инициируется прерывание.

Ниже представлен перечень функций прерывания 21h и 10h, использующихся для вывода информации.

Прерывание 21h

Функция 02h

Вывод на дисплей.

Вход. AH = 02h

DL = выводимый символ

Выход. Нет

Описание. Посылает символ из DL на стандартный вывод. Обрабатывает символ Backspace (ASCII 8), перемещая курсор влево на одну позицию и оставляя его в новой позиции.

Функция 05h

Вывод на принтер.

Вход. AH = 05h

DL = символ, записываемый на стандартный принтер

Выход. Нет

Описание. Посылает символ в DL на стандартное устройство принтера, обычно LPT1. Команда DOS MODE может перенаправить этот вывод в последовательный порт.

Функция 09h

Выдать строку.

Вход. AH = 09h

DS:DX = адрес строки, заканчивающейся символом '$'.

Выход. Нет

Описание. Строка, исключая завершающий ее символ '$', посылается на стандартный вывод. Символы Backspace обрабатываются как в функции 02h. Обычно, чтобы перейти на новую строку, включают в текст пару CR/LF (ASCII 13h и ASCII 0ah). Строки, содержащие '$', можно выдать через 40h.

Прерывание 10h

Функция 02h

Вход. AH = 02h

BH = видео страница

DH, DL = строка, колонка (считая от 0)

Выход. Нет

Описание. Устанавливает курсор в позицию DH, DL. Установка курсора на строку 25 делает курсор невидимым.

Функция 09h

Писать символ/атрибут в текущей позиции курсора.

Вход. AH = 09h

BH = номер видео страницы

AL = записываемый символ

CX = счетчик (сколько экземпляров символа записать)

BL = видео атрибут (текст) или цвет (графика)

Выход. Нет

Описание. Выводит на экран в текущую позицию курсора символ с заданным атрибутом.

Функция 0ah

Писать символ в текущей позиции курсора.

Вход. AH = 0ah

BH = номер видео страницы

AL = записываемый символ

CX = счетчик (сколько экземпляров символа записать)

Выход. Нет

Описание. Выводит на экран в текущую позицию курсора заданный символ.

Функция 13h

Вывод строки.

Вход. AH = 13h

ES:BP – выводимая строка

CX = длина строки (подсчитываются только символы)

DH, DL = позиция (строка, колонка) начала вывода

BH = номер страницы

AL = код подфункции:

0 = атрибут в BL; курсор без изменения

1 = атрибут в BL; курсор в конец строки

2 = формат строки: char, attr, ...; курсор без изменения

3 = формат строки: char, attr, ...; курсор в конец строки

Выход. Нет

Описание. Выдает строку в позиции курсора. Символы 0dh (CarRet), 0ah (LineFeed), 08h (Backspace) и 07h (Beep) трактуются как команды управления и не высвечиваются.

Некоторые функции прерывания 10h используют для вывода атрибут символа. Определение атрибутов символа приведено в лабораторной работе №1.

Приведенный ниже фрагмент программы иллюстрирует процесс вывода строки на экран с использованием прерывания 21h

```
.DATA
STR DB 'Hello!$' ;Описание строки
…
.CODE
…
MOV AH, 09H; Выбор функции прерывания
MOV DX, OFFSET STR; Занесение в DX адреса выводимой строки
INT 21H
…
```

Ввод с клавиатуры

Процесс ввода информации в ассемблерных программах осуществляется аналогично выводу:

− в регистр AH заносится номер функции ввода;

− инициируется прерывание, после выполнения которого определенные регистры процессора содержат либо введенную информацию, либо адрес буфера с введенной информацией.

Ниже описан ряд функций прерывания 21h, используемых для ввода информации с клавиатуры.

Функция 01h

Ввод с клавиатуры.

Вход. AH = 01h

Выход. AL = символ, полученный с клавиатуры

Описание. Считывает (ожидает) символ со стандартного входного устройства. Отображает этот символ на стандартное выходное устройство (эхо). Ввод расширенных клавиш ASCII (F1 – F12, PgUp, курсор и т.п.) требует двух обращений к этой функции. Первый вызов возвращает AL = 0. Второй вызов возвращает в AL расширенный код ASCII.

Функция 07h

Нефильтруемый консольный ввод без эха.

Вход. AH = 07h

Выход. AL = символ, полученный с клавиатуры

Описание. Считывает (ожидает) символ со стандартного входного устройства и возвращает этот символ в AL. Не фильтрует. Не проверяет на Ctrl-Break, Backspace и т.п. Необходимо вызывать дважды для ввода расширенного символа ASCII.

Функция 08h

Консольный ввод без эха.

Вход. AH = 08h

Выход. AL = символ, полученный с клавиатуры

Описание. Считывает (ожидает) символ со стандартного входного устройства и возвращает этот символ в AL. При обнаружении Ctrl-Break выполняется прерывание Int 23h. Необходимо вызывать дважды для ввода расширенного символа ASCII.

Функция 0ah

Буферизированный ввод строки.

Вход. AH = 0ah

DS:DX = адрес входного буфера (смотри ниже)

Выход. Буфер содержит ввод, заканчивающийся символом CR (ASCII 0dh)

Описание. При входе буфер по адресу DS:DX должен быть оформлен следующим образом:

MAX	?	…	?	?	?	?	?	?

MAX – максимально допустимая длина ввода (от 1 до 254) При выходе буфер заполнен данными следующим образом:

MAX	LEN	…	T	E	X	T	…	0dh

LEN – действительная длина данных без завершающего CR (здесь – 0dh).

Символы считываются со стандартного ввода вплоть до CR (ASCII 0dh) или до достижения длины MAX-1. Если достигнут MAX-1, включается консольный звонок для каждого очередного символа, пока не будет введен возврат каретки CR (нажатие Enter). Второй байт буфера заполняется действительной длиной введенной строки, не считая завершающего CR. Последний символ в буфере – всегда CR (который не засчитан в байте длины). Символы в буфере (включая LEN) в момент вызова используются как «шаблон». В процессе ввода действительны обычные клавиши редактирования: Esc выдает «\» и начинает с начала, F3 выдает буфер до конца шаблона, F5 выдает «@» и сохраняет текущую строку как шаблон, и так далее. Большинство расширенных кодов ASCII игнорируются. При распознавании Ctrl-Break выполняется прерывание Int 23h (буфер остается неизменным).

Функция 0ch

Ввод с очисткой

Вход. AH=0ch

AL = номер функции ввода (01h, 06h, 07h, 08h или 0ah)

Выход. Нет

Описание. Очищает буфер опережающего ввода стандартного ввода, а затем вызывает функцию ввода, указанную в AL. Это заставляет систему ожидать ввод очередного символа. Следующие значения допустимы в AL: 01h – ввод с клавиатуры; 06h – ввод с консоли; 07h – нефильтрующий без эха; 08h – ввод без эха; 0ah – буферизованный ввод.

Приведенный ниже фрагмент программы иллюстрирует буферизированный ввод строки:

```
.DATA
BUF DB 30,00,30 DUP ('$'),'$'
…
.CODE
…
MOV AH, 0AH; занесение в AH номера функции
LEA DX, BUF; загрузка DX адресом буфера BUF
INT 21H
. . .
```

При организации буфера BUF (размер буфера 30 байт), он весь заполняется символами «$», что удобно, если введенную строку в дальнейшем необходимо выводить на экран или в файл (при условии, конечно, что для ввода буфер будет использоваться лишь однократно).

Работа с файлами через дескрипторы (описатели)

Если программы, написанные на языках высокого уровня, могут открыть файл без выполнения подготовительных действий (они выполняются автоматически), то ассемблерные программы должны создать специальные области данных, которые используются при операциях ввода/вывода. Используется два метода доступа к файлам: метод управляющего блока файла (FCB) и метод дескриптора файла (Handle). С помощью метода FCB можно получить доступ только к файлам, находящимся в текущем каталоге. Метод дескриптора файла позволяет получить доступ к любому файлу, независимо от того, какой каталог является текущим.

Начиная с DOS версии 2.0 в набор функций прерывания 21h включены UNIX-подобные файловые функции. Идея их состоит в том, что, когда программа открывает файл, DOS возвращает 16-битовое значение «описателя файла» (дескриптора файла) (handle). После этого, когда программа читает, позиционирует, пишет или закрывает файл, она ссылается на него через описатель. Одно из самых больших удобств – то, что можно обращаться к некоторым устройствам так, как будто это дисковые файлы, через зарезервированные описатели DOS:

Handle	Наименование и описание
0	Стандартное устройство ввода (обычно клавиатура)
1	Стандартное устройство вывода (обычно экран)
2	Стандартное устройство ошибок (всегда CON – экран для сообщений)
3	Стандартное устройство AUX (1-й посл. Порт)
4	Стандартный принтер LPT1

Ниже приведен перечень наиболее часто используемых функций прерывания 21h для работы с файлами через описатели.

Функция 3ch

Создать файл.

Вход. AH = 3ch

DS:DX = адрес строки ASCIIZ с именем файла

CX = атрибут файла

Выход. AX = код ошибки, если CF установлен, и описатель файла, если ошибки нет.

Описание. Файл создается в указанном (или умалчиваемом) оглавлении и открывается в режиме доступа «чтение/запись». Если файл уже существует, то при открытии файл усекается до нулевой длины. Если атрибут

файла – «только чтение», открытие отвергается (атрибут можно изменить функцией 43h).

Функция 5bh

Создать новый файл (не должен существовать).

Вход. AH = 5bh

DS:DX = адрес строки ASCIIZ с именем файла

CX = атрибут файла

Выход. AX = код ошибки, если CF установлен, и описатель файла, если ошибки нет

Описание. Этот вызов идентичен функции 3ch, с тем исключением, что он вернет ошибку, если файл с заданным именем уже существует.

Функция 5ah

Создать уникальный файл.

Вход. AH = 5ah

DS:DX = адрес строки ASCIIZ с путем (заканчивается \)

CX = атрибут файла

Выход. AX = код ошибки, если CF установлен, и описатель файла, если ошибки нет. DS:DX (не изменяется) становится полным ASCIIZ-именем нового файла.

Описание. Открывает (создает) файл с уникальным именем в оглавлении, указанном строкой ASCIIZ, на которую указывает DS:DX. Описание пути должно быть готово к присоединению в его конец имени файла. Необходимо обеспечить минимум 12 байт в конце строки. После возврата строка DS:DX будет дополнена именем файла. DOS создает имя файла из шестнадцатеричных цифр, получаемых из текущих даты и времени. Если имя файла уже существует, DOS продолжает создавать новые имена, пока не получит уникальное имя.

Функция 3dh

Открыть файл.

Вход. AH = 3dh

DS:DX = адрес строки ASCIIZ с именем файла

AL = режим открытия

Выход. AX = код ошибки, если CF установлен, и описатель файла, если ошибки нет

Описание. В момент открытия файл должен существовать. Файл открывается в выбранном режиме доступа (AL = 0 – для чтения; AL = 1 – для записи; AL = 2 – для чтения и записи) и указатель «чтения/записи» устанавливается в 0.

Функция 3eh

Закрыть файл.

Вход. AH = 3eh

BX = описатель файла

Выход. AX = код ошибки, если CF установлен

Описание. BX содержит описатель файла (handle), возвращенный при открытии. Файл, представленный этим описателем, закрывается, его буфер сбрасывается, а оглавление обновляется корректными размером, временем и датой.

Функция 41h

Удалить файл.

Вход. AH = 41h

DS:DX = адрес строки ASCIIZ с именем файла

Выход. AX = код ошибки, если CF установлен

Описание. Имя файла не может содержать обобщенные символы («?» и «*»). Файл удаляется из заданного оглавления заданного диска. Если файл имеет атрибут только чтение, то перед удалением необходимо изменить этот атрибут через функцию 43h.

Функция 42h

Установить указатель чтения/записи (можно также узнать размер файла).

Вход. AH = 42h

BX = описатель файла

CX:DX = смещение указателя: (CX * 65536) + DX

AL = 0 переместить к началу файла + CX:DX

AL = 1 переместить к текущей позиции + CX:DX

AL = 2 переместить к концу файла – CX:DX

Выход. AX = код ошибки, если CF установлен

DX:AX = новая позиция указателя файла (если нет ошибки)

Описание. Перемещает логический указатель чтения/записи к нужному адресу, с которого начнется очередная операция чтения или записи. Вызов с AL = 2, CX = 0, DX = 0 возвращает длину файла в DX:AX. DX здесь старшее значащее слово: действительная длина (DX * 65536) + AX.

Функция 3Fh

Читать из файла/устройства.

Вход. AH = 3fh

BX = описатель файла

DS:DX = адрес буфера для чтения данных

CX = число считываемых байт

Выход. AX = код ошибки, если CF установлен

AX = число действительно прочитанных байт

Описание. CX байт данных считываются из файла или устройства с описателем, указанным в BX. Данные читаются с текущей позиции указателя чтения/записи файла и помещаются в буфер вызывающей программы, адресуемый через DS:DX.

Всегда необходимо сравнивать возвращаемое значение AX (число прочитанных байт) с CX (запрошенное число байт):

− если AX = CX, (и CF сброшен) – чтение было корректным без ошибок;

− если AX = 0 – достигнут конец файла (EOF);

− если AX < CX (но ненулевой), то возможны два варианта: при чтении с устройства – входная строка имеет длину AX байт; при чтении из файла – в процессе чтения достигнут EOF.

Функция 40h

Писать в файл/устройство.

Вход. AH = 40h

BX = описатель файла

DS:DX = адрес буфера, содержащего данные

CX = число записываемых байт

Выход. AX = код ошибки, если CF установлен

AX = число действительно записанных байт

Описание. CX байт данных записывается в файл или на устройство с описателем, заданным в BX. Данные берутся из буфера, адресуемого через DS:DX и записываются, начиная с текущей позиции указателя чтения/записи файла. Необходимо всегда сравнивать возвращаемое значение AX (число записанных байт) с CX (запрошенное число байт для записи): если AX = CX, запись была успешной; если AX < CX, встретилась ошибка (скорее всего, переполнение).

Некоторые функции в качестве параметра используют атрибут файла. Атрибут – это один байт битовых флагов, связанный с каждым файлом и находящийся в элементе оглавления для файла. В атрибуте определены следующие биты:

X	X	A	D	V	S	H	R

R – только чтение (нельзя обновлять или удалять);

H – скрытый;

S – системный;

V – метка тома;

D – элемент подоглавления;

A – архивный;

X – не используются.

ASCIIZ строка, содержащая имя файла, должна иметь вид:
«‘d:\путь\имя_файла’,0»

Если диск и/или путь опущены, они принимаются по умолчанию. После вызова функции описатель файла должен быть сохранен для последующих операций. Количество описателей в системе регламентируется файлом CONFIG.SYS.

Приведенный ниже пример иллюстрирует процесс работы с файлом через описатели в ассемблерной программе.

```
.data
buf db ‘d:\Users\1.txt’,0
.code
…
;Создание файла
mov ah,3ch        ;номер функции создания файла
mov cx,0     ;атрибуты создаваемого файла
                  lea dx,buf        ;в DS:DX – адрес
             ASCIIZ строки с
                                  ;именем создавае-
             мого файла
int 21h           ;вызов функции создания файла
jc no_create      ;Проверка флага переноса, если
                  ;установлен – обработка ошибки
. . . ;Работа с файлом

no_create:        ;обработчик ошибки создания
```

Задания к выполнению лабораторной работы выдаются преподавателем непосредственно перед занятием

Примеры вариантов заданий

1. Чтение данных входного текстового файла Input.txt и запись в выходной файл Output.txt содержимого входного с удалением повторений символов. Например, при содержимом входного файла aafhkdjjjlll должен быть создан выходной файл с содержимым afhkdjl.

2. Чтение данных входного текстового файла Input.txt и запись в выходной файл Output.txt содержимого входного с удалением символов, описанных как массив в сегменте данных программы. Например, при содержимом входного файла aafhkdjjjlll и массиве, содержащем символы afd, должен быть создан выходной файл с содержимым hkjjjlll.

3. Ввод символов с клавиатуры с записью введенной строки в файл Text.txt с удалением прописных букв. После запуска программы ожидается ввод строки. Введенные символы дописываются в конец существующего

файла. Если файл не существует, его необходимо создать. При нажатии клавиши Enter файл закрывается и программа завершает работу.

4. Ввод символов с клавиатуры и запись каждого второго символа в выходной файл Output.txt. После запуска программы ожидается ввод строки. Введенные символы дописываются в конец существующего файла. Если файл не существует, его необходимо создать. При нажатии клавиши Enter файл закрывается и программа завершает работу.

5. Чтение входного текстового файла и вывод на экран всех символов, кроме прописных. Выход из программы – по нажатию клавиши Esc.

6. Ввод символов с клавиатуры. На экране должны отображаться только строчные английские буквы, они же записываются в текстовый файл. Выход из программы – по нажатию клавиши Enter.

7. Вывод на экран кодовой таблицы символов и запись ее в текстовый файл.

Содержание отчета:

1. Тема и цель работы.
2. Задание лабораторной работы.
3. Блок-схема программы.
4. Листинг программы.
5. Выводы по работе.

Контрольные вопросы:

1. Каков порядок вызова прерываний?
2. Как передаются аргументы функций?
3. Как возвращается результат работы функции?
4. Функции для работы с клавиатурой?
5. Какие существуют методы для работы с файлами?
6. Какие основные функции для работы с файлами через дескрипторы?

Лабораторная работа № 10.
Часы реального времени
(Программирование часов реального времени)

Цель работы: изучить структуру часов реального времени и структуру памяти CMOS, а также порядок работы с этим устройством и памятью.

Краткие теоретические сведения

Основное назначение часов – это непрерывный отсчет времени суток даже в тот период, когда компьютер выключен, а также извещение процессора о наступлении определенного времени (режим будильника). Часы реального времени реализованы на базе микросхемы типа Motorola MC 146818 (RT/CMOS RAM Chip) и обеспечивают непрерывный отсчет времени при выключении питания на компьютере за счет батареи. Эта микросхема включает также память, содержимое которой сохраняется при исключении питания. Аппаратура часов реального времени использует только первые 14 байт этой памяти, а остальные предназначены для сохранения конфигурации компьютера. Часы позволяют генерировать три типа прерываний по линии 0 второго контроллера прерываний (IRQ 8), что соответствует в IBM PC прерыванию 70h:

1. периодические прерывания с интервалом 976.562 мкс;
2. прерывание от будильника;
3. прерывание по окончанию обновления значения часов.

1. Регистры часов

Как уже говорилось ранее, часы используют первые 14 байт памяти микросхемы типа Motorola MC146818, которые будем в дальнейшем называть регистрами часов. Чтобы считать любой байт памяти микросхемы, в том числе и данные из регистра часов необходимо выполнить следующие шаги: записать адрес считываемого регистра в порт 70h; считать данные регистра из порта 71h.

Например, следующие команды используются для чтения регистра 0 часов реального времени:

```
mov     al,0  ; Записать адрес регистра
out     70h,al
jmp     short $+2
in      al,71h      ; Считать регистр
```

Аналогично, для записи значения в регистр часов требуется занести адрес модифицируемого регистра в порт 70h, после чего вывести в порт 71h записываемое значение. Например:

```
mov     al,0            ; Записать адрес регистра
out     70h,al
jmp     short $+2
mov     al,20h          ; Занести новое значение
out     71h,al
```

Следует отметить, что чтение и запись данных порта 71h должны выполняться немедленно после записи номера регистра в порт 70h. Кроме того, в процессе операций ввода-вывода с регистрами часов должны быть запрещены прерывания, что позволит избежать модификации порта 70h процедурой обработки прерывания до чтения или записи данных.

Адресация регистров часов приведена в таблице:

Адрес регистра	Назначение регистра
00h	Секунды
01h	Секунды для будильника
02h	Минуты
03h	Минуты для будильника
04h	Часы
05h	Часы для будильника
06h	День недели
07h	Дата
08h	Месяц
09h	Год
0Ah	Регистр состояния 1
0Bh	Регистр состояния 2
0Ch	Регистр состояния 3
0Dh	Регистр состояния 4
32h	Столетие

Регистры времени 00h – 09h и 32h содержат время в двоично-десятичном формате (например, дата 26 хранится в виде 26h). Столетие задастся в виде первых двух цифр полного номера года (например, 19h). На машинах PS/2 байт столетия расположен по адресу 37h.

Регистр состояния 1 имеет следующий формат:

7	6	5	4	3	2	1	0
UIP	DV			RS			

Бит UIP определяет момент обновления показаний часов и может быть установлен в 0 - часы доступны для чтения или - 1 - аппаратура выполняет обновление показаний часов. Для правильного чтения или записи значений часов эти операции необходимо выполнять, когда значение бита UIP равно 0. Биты DV задают значение частоты для обновления показаний часов. Для правильного функционирования часов данные биты должны быть установлены в значение 010, что соответствует частоте 32.768 КГц. Это единственная частота, обеспечивающая правильное время в часах. Биты RS позволяют выбрать делитель выходной частоты. Для правильного функционирования эти биты должны быть усыновлены в значение 0110, что соответствует выходной частоте прямоугольных колебаний 1.024 КГц и интервалу времени между периодическими прерываниями 976.562 мкс.

Формат **регистра состояния 2** приведен ниже:

7	6	5	4	3	2	1	0
UPD	PIE	AIE	UIE	SQWE	DM	CF	DSE

Бит UPD используется для обновления показания часов. Когда он устанавливается в 1, внутренний цикл обновления часов прекращается, и программа может инициализировать любой регистр часов. Для запуска часов надо установить этот бит в 0. Бит PIE управляет генерацией периодических прерываний *t* интервалом 976.562 мкс. Когда он установлен в 0, прерывания запрещены. При установке этого бита в 1, периодические прерывания разрешены. Аналогично, бит AIE задает, разрешены ли прерывания от будильника, а бит UIE разрешает или запрещает прерывания при завершении цикла обновления показаний часов. Бит SQWE разрешает или запрещает генерацию прямоугольных колебаний (см. биты RS в регистре состояния 1). Функция генерации прямоугольных колебаний не используется в компьютере и бит устанавливается в 0. DM определяет формат времени и даты. Если этот бит установлен в 0, то используется двоично-десятичный формат. При единичном значении бита данные представляются в двоичном виде.

Для IBM PC этот бит установлен в 0.

Бит CF задаст формат представления времени. Значение 0 указывает, что используется 12-часовой формат, а значение 1 – 24-часовой формат. В компьютерах IBM PC используется 24-часовой формат. Бит DSE установлен в 1, если в часы заносится летнее время и программы, пользующиеся показаниями часов, осуществляют пересчет времени. Обычно эта функция не используется, и бит установлен в 0.

Регистр состояния 3 доступен только по чтению. Он содержит текущий статус прерывания от часов и имеет следующий формат:

7	6	5	4	3		2	1	0
INT	PI	AI	UI		Резерв			

Бит INT указывает на наличие прерывания (0 – нет прерывания, 1 – выработано прерывание). Биты PI, AI и UI уточняют тип прерывания, выработанного часами. Единичное значение бита PI указывает, что выработано периодическое прерывание. Аналогично, единичное значение бита AI сигнализирует о генерации прерывания от будильника, а бита UI – прерывания об окончании обновления показаний часов. Обработчик прерывания по линии IRQ8 должен прочитать регистр состояния 3, иначе следующее прерывание не будет выработано.

Регистр состояния 4 доступен только по чтению и имеет следующий формат:

7	6	5	4	3	2	1	0
VRB	Резерв						

В этом регистре состояния задействован только бит VRB который сигнализирует о состоянии аккумуляторной батареи. Единичное значение говорит о нормальном питании. Нулевое значение указывает, что батарея разряжена и данные в часах недостоверны.

2. Программирование часов

Помимо чтения текущего времени, с помощью часов можно выполнять функции отсчета времени, используя периодическое прерывание, или получать управление в заранее определенное время, используя функцию будильника. Следует отметить, что все эти функции поддерживаются средствами BIOS. Поэтому, как правило, непосредственного программирования часов не требуется. Остановимся только на общих принципах программирования часов реального времени:

– самая простая функция часов – это установка и чтение времени. Чтение времени необходимо производить только между интервалами обновления часов, т.е. когда бит UIP в регистре состояния 1 равен 0. Установку времени и даты лучше выполнять, запретив сначала работу часов, т.е. установив в регистре состояния 2 единичное значение бита UPD. После этого можно загрузить все значения в часы и выполнить в необходимый момент их пуск, установив бит UPD в нулевое значение. Следует отметить, что Стандартные функции BIOS не позволяют установить значение в поле «день недели». Поэтому, обычно, это значение часов недостоверно;

– функция будильника позволяет вырабатывать прерывание при совпадении значений часов и будильника. Запись значения будильника можно производить только между интервалами обновления часов, т.е. когда бит UIP в регистре состояния 1 равен 0. Для получения прерывания от будильника необходимо разрешить генерацию этого прерывания, установив единичное значение бита AIE в регистре состояния 2. При возникновении этого прерывания устанавливается в единичное значение бит AI регистра состояния 3;

– средства часов реального времени позволяют генерировать периодические прерывания с интервалом 976.562 мкс. Для разрешения этого прерывания необходимо установить 1 для бита PIE регистра состояния 2. При возникновении прерывания устанавливается в единичное значение бита PI в регистре состояния 3;

– прерывание по окончании цикла обновления часов вырабатывается каждую секунду при продвижении значения часов. Обычно это прерывание не используется для программирования. Для разрешения этого прерывания необходимо установить бит UIE регистра состояния 2 в единичное значение. При возникновении прерывания устанавливается в единичное значение бит UI регистра состояния 3.

Функции BIOS прерывания 1Ah

Прочитать показания часов реального времени

На входе: AH = 02h.

На выходе: CH = часы в BCD-формате (например,13h означает 13 часов);
CL = минуты в BCD-формате;
DH = секунды в BCD-формате;
CF = CY = 1, если часы реального времени не установлены.

Установить часы реального времени

На входе: AH = 03h;
CH = часы в BCD-формате (например,13h означает 13 часов);
CL = минуты в BCD-формате;
DH = секунды в BCD-формате;
DL = 1, если необходимо использовать летнее время

На выходе: не используются.

Прочитать дату из часов реального времени

На входе: AH = 04h.

На выходе: CH = столетие в BCD-формате;
CL = год в BCD-формате (например, CX=1991h означает 1991 год);
DH = месяц в BCD-формате;
DL = число в BCD-формате;
CF = CY = 1, если часы реального времени не установлены.

Установить дату в часах реального времени

На входе: AH = 05h;
CH = столетие в BCD-формате;
CL = год в BCD-формате (например, CX=1991h означает 1991 год);
DH = месяц в BCD-формате;
DL = число в BCD-формате;

На выходе: не используются.

Установить будильник

На входе: AH = 06h;
CH = часы в BCD-формате;
CL = минуты в BCD-формате;
DH = секунды в BCD-формате.

На выходе: CF = CY = 1, если часы реального времени не установлены.

Эта функция позволяет установить будильник на заданное время. Когда будильник «зазвенит», будет вызвано прерывание INT 4Ah (это прерывание вызывают модули BIOS после прихода аппаратного прерывания от часов реального времени IRQ8, т.е. прерывания с номером 70h). Программа, использующая функцию будильника, должна подготовить обработчик прерывания INT 4Ah, завершающий свою работу выполнением команды IRET. Программа может установить только один будильник.

Сброс будильника

На входе: AH = 07h.

На выходе: не используются.

Эта функция позволяет сбросить будильник, например, для того чтобы установить его заново на другое время.

Примерный вариант задания

Написать программу, наглядно демонстрирующую работу функций прерывния 1Ah, а также изменение и чтение состояния регистров часов через порты 70h / 71h.

Содержание отчета:

1. Тема и цель работы.
2. Задание для лабораторной работы.
3. Блок-схема программы.
4. Листинг программы.
5. Выводы по работе.

Контрольные вопросы:

1. Каково назначение часов реального времени?
2. Как произвести чтение и запись в регистры часов?
3. Каково назначение регистров состояния?
4. Какие функциии BIOS-прерывания управления часами реального времени существуют?
5. Что необходимо сделать для обработки (получения) сигнала будильника?

Лабораторная работа № 11.
Контроллер клавиатуры ПЭВМ
(Программирование буфера клавиатуры)

Цель работы: изучение структуры буфера клавиатуры и его использование.

Краткие теоретические сведения

Принципы работы клавиатуры

Клавиатура выполнена, как правило, в виде отдельного устройства, подключаемого к компьютеру тонким кабелем. Малогабаритные компьютеры Lap-Top используют встроенную клавиатуру. Внутри клавиатуры находится специализированная микросхема (контроллер клавиатуры), которая отслеживает нажатия на клавиши и посылает номер нажатой клавиши в персональный компьютер.

Если рассмотреть сильно упрощенную принципиальную схему клавиатуры, представленную на рисунке 11.1, можно заметить, что все клавиши находятся в узлах матрицы.

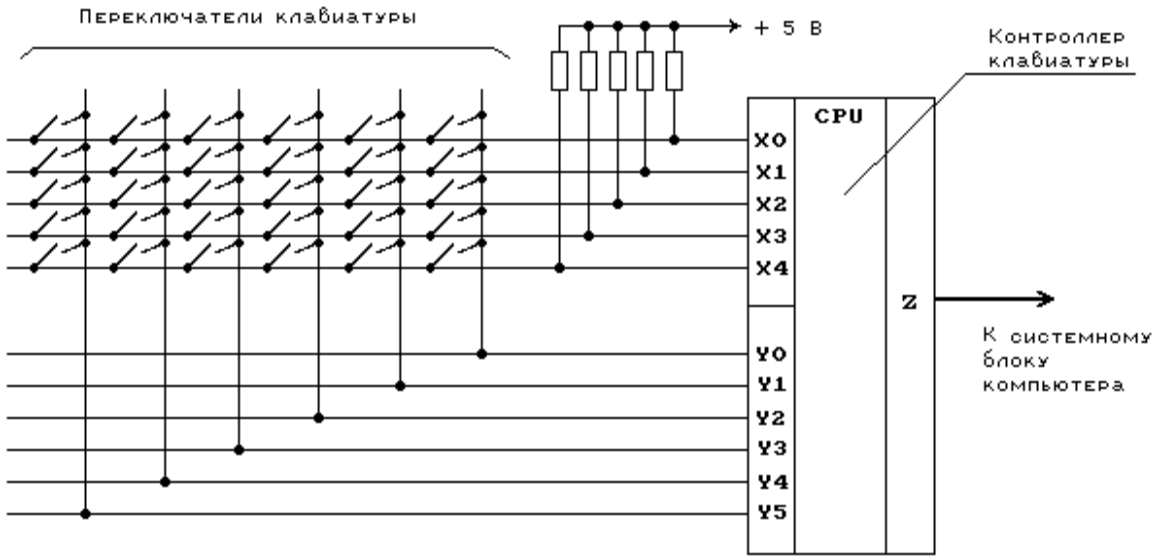


Рисунок 11.1. – Упрощенная схема клавиатуры

Все горизонтальные линии матрицы подключены через резисторы к источнику питания +5 В. Клавиатурный компьютер имеет два порта – выходной и входной. Входной порт подключен к горизонтальным линиям матрицы (X0 – X4), а выходной – к вертикальным (Y0 – Y5).

Устанавливая по очереди на каждой из вертикальных линий уровень напряжения, соответствующий логическому 0, клавиатурный контроллер опрашивает состояние горизонтальных линий. Если ни одна клавиша не

нажата, уровень напряжения на всех горизонтальных линиях соответствует логической 1 (т.к. все эти линии подключены к источнику питания +5 В через резисторы). Если оператор нажмет на какую-либо клавишу, то соответствующая вертикальная и горизонтальная линии окажутся замкнутыми. Когда на этой вертикальной линии контроллер установит значение логического 0, то уровень напряжения на горизонтальной линии также будет соответствовать логическому 0.

Как только на одной из горизонтальных линий появится уровень логического 0, клавиатурный контроллер фиксирует нажатие на клавишу. Он посылает в персональный компьютер запрос на прерывание и номер клавиши в матрице. Аналогичные действия выполняются и тогда, когда оператор отпускает нажатую ранее клавишу.

Номер клавиши, посылаемый клавиатурным процессором, однозначно связан с распайкой клавиатурной матрицы и не зависит напрямую от обозначений, нанесенных на поверхность клавиш. Этот номер называется скан-кодом (Scan Code). Слово scan («сканирование»), подчеркивает тот факт, что клавиатурный контроллер сканирует клавиатуру для поиска нажатой клавиши.

Но программе нужен не порядковый номер нажатой клавиши, а соответствующий обозначению на этой клавише ASCII-код. Этот код не зависит однозначно от скан-кода, т.к. одной и той же клавише могут соответствовать несколько значений ASCII-кода. Это зависит от состояния других клавиш. Например, клавиша с обозначением «1» используется и для ввода символа «!» (если она нажата вместе с клавишей SHIFT). Поэтому все преобразования скан-кода в ASCII-код выполняются программным обеспечением внутри ПК. Как правило, эти преобразования выполняют модули BIOS. Для использования символов кириллицы эти модули расширяются клавиатурными драйверами.

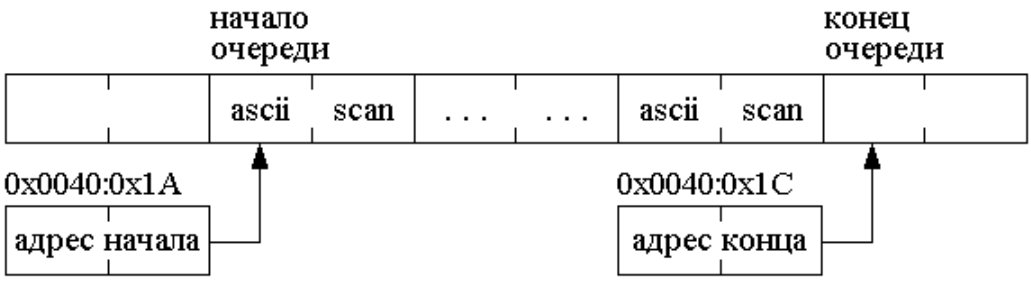
Если нажать на клавишу и не отпускать её, клавиатура перейдет в режим автоповтора. В этом режиме в центральный компьютер автоматически через некоторый период времени, называемый периодом автоповтора, посылается код нажатой клавиши. Режим автоповтора облегчает ввод с клавиатуры большого количества одинаковых символов.

Структура буфера клавиатуры

Клавиатура является основным средством ввода информации при работе на компьютере. В компьютерах IBM PC взаимодействие программ с клавиатурой организовано в асинхронном режиме. При нажатии клавиши на клавиатуре код нажатой клавиши поступает в буфер клавиатуры и хранится там до тех пор, пока программа не затребует его. Если скорость нажатия клавиш и скорость обработки нажатия клавиши программой отличаются, то или программа ожидает поступления кода очередной клавиши

в буфер, или коды нажатых клавиш накапливаются в буфере клавиатуры. Хранение информации в буфере организовано в виде очереди (FIFO – First Input First Output), буфер клавиатуры находится в оперативной памяти компьютера и доступен для обычных операций чтения и записи.

Очередь буфера клавиатуры реализована следующим образом:



Каждой нажатой клавише в очереди соответствует ячейка размером в 2 байта, в первом байте записан ascii-код, а во втором – scan-код нажатой клавиши. Следует учитывать, что при считывании слова из буфера клавиатуры scan-код оказывается в старшем байте, а ascii-код – в младшем байте слова, в соответствии с порядком следования байтов, принятом для IBM PC. В двух словах оперативной памяти по адресам 0x0040:0x1A и 0x0040:0x1C хранятся смещения адресов начала (головы) и конца (хвоста) очереди буфера клавиатуры. Следует обратить внимание, что эти смещения указаны относительно адреса 0040:0000h. Сам буфер размером 32 байта (16 слов) расположен в компьютере IBM PC/XT по адресу 0000h:041Eh и организован в виде кольцевой очереди. Следует отметить, что рост буфера (клавиши нажимаются, а символы из буфера не считываются) происходит в сторону хвоста.

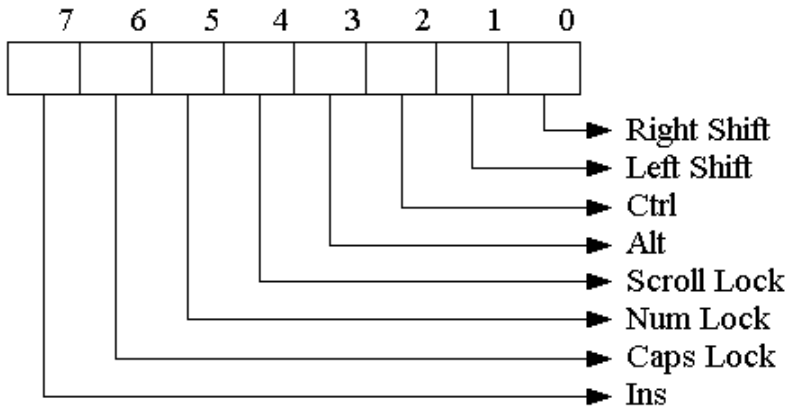
В компьютерах моделей IBM PC/AT и IBM PS/2 расположение клавиатурного буфера задается содержимым двух слов памяти с адресами 0000h:0480h (смещение адреса начала буфера относительно 0040h) и 0000h:0482h (смещение конца буфера относительно 0040h). Обычно эти ячейки памяти содержат значения, соответственно, 001Eh и 003Eh, что соответствует расположению клавиатурного буфера в IBM PC/AT и IBM PS/2 его расположению в IBM PC/XT.

Указателями на начало и конец клавиатурного буфера обычно управляют обработчики прерываний INT 09h и INT 16h. Программа извлекает из буфера коды нажатых клавиш, используя различные функции прерывания INT 16h. Заметим, что вы можете удалить все символы из буфера клавиатуры, установив оба указателя на начало буфера. Однако есть более предпочтительный способ с использованием, например, прерывания BIOS INT 16h.

Для чтения scan-кода последней нажатой клавиши (даже в том случае, если буфер уже заполнен) можно использовать 60-й порт. При этом следует учитывать, что многократное обращение к порту будет возвращать один и тот же результат до тех пор, пока не будет нажата следующая клавиша на клавиатуре.

При переполнении внутреннего буфера клавиатуры или буфера, расположенного в области данных BIOS программа-обработчик прерывания INT 09h генерирует звуковой сигнал.

Кроме буфера клавиатуры и ячеек памяти с указателями головы и хвоста очереди, с обработкой нажатия клавиш клавиатуры связано ещё несколько элементов в области памяти данных BIOS. Одним из этих элементов является флаг состояния клавиатуры, располагающийся в области переменных BIOS по адресу 0000:0417h. Его размер составляет 1 байт, отдельные биты флага содержат информацию о состоянии клавиш Shift, Ctrl, Alt, Scroll Lock, Num Lock, Caps Lock и Ins. Соответствие перечисленных клавиш и битов флага показано на рисунке:



Другие элементы памяти, связанные с клавиатурой, используются достаточно редко и мы не будем их рассматривать.

Примерный вариант задания

Написать программу, наглядно демонстрирующую работу буфера клавиатуры. Необходимо отображать содержимое буфера в табличной форме с указанием текущего положения «головы» и «хвоста». Предусмотреть возможность посимвольной очистки буфера.

Содержание отчета:

1. Тема и цель работы.
2. Задание для лабораторной работы.
3. Блок-схема программы.
4. Листинг программы.
5. Выводы по работе.

Контрольные вопросы:

1. Какова структура буфера клавиатуры?
2. В чем различие scan-кода и ascii-кода?
3. Как очичтить буфер клавиатуры?
4. Какой порт используется для чтения нажатой клавиши?
5. Что описывает флаг состояния клавиатуры?

Лабораторная работа № 12.
Архитектура дисковых систем (Исследование работы устройства чтения и записи на гибком магнитном диске)

Цель работы – изучить физическую и логическую организацию накопителя на гибких магнитных дисках и прерывание Int 13h BIOS для работы с контроллером накопителя на гибком магнитном диске.

Теоретические сведения

Физическая организация накопителя на гибком магнитном диске. Дискета представляет собой круглую пластину, покрытую с двух сторон магнитным материалом. Информация записывается раздельно на две стороны дискеты. В дисководе к поверхности дискеты прижимаются две (по одной с каждой стороны) головки чтения/записи. Пара головок может перемещаться вдоль радиуса дискеты, занимая на нем ряд фиксированных положений. При вращении дискеты точки ее поверхности, которые могут находиться в контакте с головками, образуют концентрические окружности, именуемые дорожками. Дорожки делятся на сектора. Цилиндр – совокупность всех дорожек, расположенных на разных поверхностях и равноудаленных от оси вращения. С точки зрения адресации понятия «дорожка» и «цилиндр» являются синонимами. Обмен информацией с дискетой происходит через прямой доступ к памяти. Сектор – это наименьший объем (блок) информации, передаваемый за одну операцию чтения/записи. Объем сектора – 512 байт. Кластер – минимальный объем дискового пространства, который может быть выделен для размещения файла. Все файловые системы, используемые для работы с жесткими дисками, основаны на кластерах, которые состоят из одного или нескольких смежных секторов. Чем меньше размер кластера, тем более эффективно используется дисковая память. Если размер кластера не задан во время форматирования, он выбирается операционной системой в зависимости от объема диска. Стандартные значения подобраны таким образом, чтобы снизить потерю дискового пространства и степень возможной фрагментации тома. Кластеры также называют единичными блоками. Адрес сектора состоит из трех составляющих:

- номер дорожки (нумерация дорожек начинается с 0);
- номер головки (нумерация головок начинается с 0);
- номер сектора на дорожке (нумерация секторов начинается с 1).

При записи на диск больших объемов информации необходимо свести к минимуму затраты по переключению на следующий сектор диска. Поэтому принят такой порядок определения следующего сектора при последовательной записи (чтении). Следующим считается сектор, расположенный следующим на той же дорожке, под той же головкой (при записи сектора головки за счет вращения диска установятся над следующим сектором, так

что затраты времени на переключения практически нулевые). При заполнении всей дорожки следующим принято считать первый сектор дорожки, расположенный на том же цилиндре под следующей головкой (этот сектор будет находиться под головками, а электрическое переключение на другую головку – процесс быстрый). И только при заполнении всего цилиндра меняется номер дорожки в адресе, этот процесс требует механического перемещения головок и, следовательно, больших затрат времени.

Логическая организация накопителя на гибких магнитных дисках. Логическая структура дискеты включает в себя следующие элементы:

- Boot – сектор;
- FAT (2 копии);
- корневой каталог;
- область данных.

Boot-сектор является самым первым сектором логического диска (номер логического сектора – 0). Для дискеты его физический адрес – дорожка 0, головка 0, сектор 1. При загрузке системы с дискеты именно этот сектор считывается в память программой POST и выполняет дальнейшую загрузку. Boot-сектор содержит информацию о физических параметрах диска и о размещении на нем системной информации, используемой далее системой при работе с диском, поэтому даже если дискета и несистемная, то есть загрузка с нее не выполняется, на ней все равно имеется Boot-сектор с данными о носителе. Формат Boot-сектора различен для версий DOS до 4.0 и от 4.0 и выше. Далее приводится описание Boot-сектора для DOS 4.0 и выше.

Первые 3 байта сектора содержат команду JMP, обеспечивающую переход на программу загрузки. Следующие 8 байт – символьный идентификатор программного средства, производившего форматирование дискеты. Если дискета форматировалась системной утилитой FORMAT, там записан номер версии операционной системы.

Поле SectSize (2 байта) – размер сектора, всегда содержит значение 512.

Поле ClustSize (1 байт) показывает, сколько секторов содержится в одном кластере – единице распределения дисковой памяти.

Поле ResSect (2 байта) содержит число резервных секторов, расположенных до начала FAT, обычно это 1 – перед FAT имеется только Boot-сектор. Поле FatCnt (1 байт) – число копий FAT, всегда 2.

Поле RootSize (2 байта) показывает размер корневого каталога, чтобы узнать число секторов в корневом каталоге надо содержимое этого поля разделить на 16.

Поле TotSecs (4 байта) – общее число секторов на диске для диска, отформатированного в DOS 4.0 или выше, содержит 0, если число секторов больше 65535, в этом случае число секторов может быть выбрано из поля LongTotSecs.

Байт Media идентифицирует тип носителя. Возможны такие его значения: 0xFF – 2 стороны, 8 секторов на дорожке; 0xFE – 1 сторона, 8 секторов на дорожке; 0xFD – 2 стороны, 9 секторов на дорожке; 0xFC – 1 сторона, 9 секторов на дорожке; 0xF9 – 2 стороны, 15 секторов на дорожке; 0xF8 – жесткий диск.

Поле FatSize (2 байта) – число секторов в одной копии FAT.

Поле TrkSecs (2 байта) – число секторов на дорожке.

Поле HeadCnt (2 байта) – число поверхностей.

Поля HidnSecL и HidnSecH для DOS 4.0 и выше можно интерпретировать как одно поле: dword HidnSec (4 байта) – это количество скрытых секторов. Для DOS ниже 4.0 используется 2-байтное число скрытых секторов – HidnSecL, и на этом область данных Boot-сектора кончается.

Поле Drive – номер дисковода, на котором диск форматировался.

Поле DOS4_flag содержит код 0x29, если диск форматировался в DOS 4.0 и выше.

Серийный номер тома – случайное число, записываемое в поле VolNum при форматировании, может использоваться в дальнейшем для идентификации диска, как и метка тома (поле VolLabel).

Поле FatForm содержит символьную последовательность 'FAT12' или 'FAT16', в зависимости от формата FAT. После этой системной информации в Boot-секторе записана программа начальной загрузки.

Для версий DOS более ранних, чем 4.0 блок системной информации в Boot-секторе имеет меньший размер – до поля HidnSecL включительно. Поля от SectSize по FatSize включительно образуют так называемый Блок Параметров BIOS (BPB – BIOS Parameter Block), он формируется в оперативной памяти драйвером диска и в дальнейшем используется при всех операциях с данным диском. В DOS 4.0 и выше принят Расширенный BPB – от поля SectSize до LongTotSecs включительно.

Формат Boot-сектора зависит не от версии DOS, установленной на той ПЭВМ, на которой дискета читается, а от версии на той ПЭВМ, где дискета форматировалась. Поскольку формат ранних версий полностью перекрывается форматом поздних, совместимость по носителям сохраняется.

FAT (File Allocation Table – таблица размещения файлов). Следом за Boot-сектором на диске расположена FAT. С точки зрения распределения дискового пространства диск разбит на кластеры. Кластер представляет собой группу последовательно расположенных секторов. Если DOS получает запрос на выделение дискового пространства, она выделяет сразу целый кластер. Размер кластера является компромиссом между двумя противоречивыми требованиями: с точки зрения экономии дискового пространства выгоден малый размер кластера, так как для маленького файла (размером даже в один байт) выделяется целый кластер, большая часть которого не используется. Но, с другой стороны, при малом размере кластера их на диске

получится очень много и управление ими усложняется. FAT представляет собой «карту» дискового пространства – массив элементов, каждый из которых соответствует одному кластеру диска. Номер элемента соответствует номеру кластера.

Поскольку область диска, содержащая системную информацию, распределена предварительно, она в FAT не отражается. Первые два элемента FAT не используются, (первый байт содержит код, совпадающий с полем Media Boot-сектора), нумерация кластеров области данных диска начинается, таким образом, с 2. Размер элемента FAT может быть 12 или 16 бит.

Описание содержимого полей далее дается для 12-битного формата FAT, в скобках указываются значения для 16-битного формата. Значения элементов FAT от 2 до 0xFEF (0xFFEF) включительно – информационные значения. Такое значение – номер следующего кластера, распределенного данному файлу. Таким образом, FAT обеспечивает списковую структуру распределения – каждый ее элемент содержит указание на следующий элемент. Поскольку кластеры распределяются файлу по мере его заполнения, файл не обязательно занимает смежные кластеры, FAT может обеспечить связывание в цепочку разнесенных по диску кластеров. Другие значения элементов зарезервированы для системной информации. Значение 0 идентифицирует свободный кластер. Значения элементов от 0xFF0 до 0xFF6 (от 0xFFF0 до 0xFFF6) – резервные кластеры. Значение 0xFF7 (0xFFF7) – сбойный кластер. Значения от 0xFF8 до 0xFFF (от 0xFFF8 до 0xFFFF) – признак конца цепочки, последнего кластера файла.

Корневой каталог. Как уже упоминалось, на диске хранятся две идентичные копии FAT, расположенные одна за другой. Следом за FAT размещается корневой каталог. Корневой каталог – главный каталог диска, с которого начинается дерево подкаталогов диска. В FAT12 и FAT16 для него выделено 16 Кб для хранения 512 элементов. В системе FAT32 корневой каталог является файлом произвольного размера. Корневой каталог состоит из элементов каталога, которые состоят из следующих полей.

Поле fname (11 бит = 8 бит для короткого имени + 3 бита для расширения) содержит имя файла или подкаталога, причем для файла байты 0 – 7 поля содержат имя, а 8 – 10 – расширение. Неиспользуемые байты поля заполнены пробелами. Если файл удаляется из каталога, то в соответствующем элементе первый символ имени заменяется кодом 0xE5, это оставляет возможность восстановления случайно удаленного файла, если, однако, за это время Элемент Каталога не будет использован для нового файла. Код 0 в первом символе имени означает, что элемент свободен и никогда не использовался. При внесении в каталог нового файла система сначала использует элементы удаленных файлов, и лишь затем – свободные.

Поле attr (1 байт) – поле атрибутов содержит признаки, характеризующие файл. Формат байта атрибутов файла показан на рисунке 12.1.

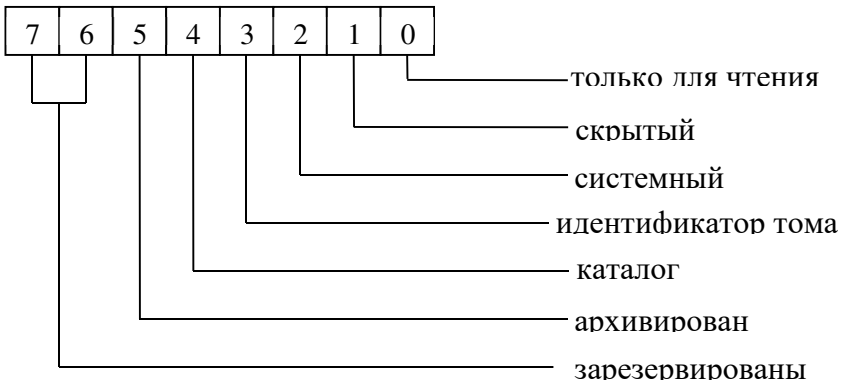


Рисунок 12.1. – Формат байта атрибутов файла

Разряды байта атрибутов устанавливаются в 1 в случае, если у файла имеется соответствующее свойство:

- бит 0 – файл только для чтения;
- бит 1 – скрытый файл;
- бит 2 – системный файл;
- бит 3 – идентификатор тома;
- бит 4 – каталог;
- бит 5 – архивированный файл;
- биты 6 и 7 – зарезервированы, должны быть установлены в 0.

Поля time (2 байта) и date (2 байта) содержат время и дату последней модификации файла. Формат времени: ЧЧЧЧЧММММММССССС (Часы, Минуты, Секунды); формат даты: ГГГГГГГММММДДДДД (Год, Месяц, День).

Поле cl (2 байта) – номер первого кластера, распределенного файлу, то есть начало той цепочки, которая продолжается в FAT.

Поле size (4 байта) – размер файла в байтах. В DOS не предусмотрены какие-либо специальные признаки конца файлов. Применяемый в некоторых случаях символ ^Z (код 26) интерпретируется как признак конца конкретными программами, но не DOS. DOS же определяет конец файла по его размеру, получаемому из этого поля элемента каталога. Для подкаталога это поле содержит 0.

Почти все, сказанное выше о корневом каталоге, справедливо и для подкаталогов. Разница между ними и корневым каталогом состоит в их размещении. Корневой каталог размещается на определенном месте (вслед за FAT) и имеет фиксированный размер. Подкаталогам дисковая память распределяется как файлам, то есть, при нехватке места в подкаталоге ему выделяется новый кластер, и это выделение отражается в FAT. Таким образом, размер подкаталога не ограничивается. В любом подкаталоге обязательно присутствуют два элемента с именами «.» и «..» и атрибутами «подкаталог».

Первый из них – ссылка на самого себя, второй – ссылка на предшествующий узел дерева подкаталогов.

Прерывание INT 13h BIOS для работы с контроллером накопителя на гибком магнитном диске. Регистр – внутренняя ячейка памяти микропроцессора, используемая для выполнения текущих операций. Так как доступ к регистру происходит во много раз быстрее, чем к ячейкам оперативной памяти, размещение промежуточных результатов в регистрах дает значительный выигрыш в скорости исполнения программы. Микропроцессор имеет большое число двухбайтовых внутренних регистров, каждому из которых присвоено «имя», или мнемоника. Все регистры можно условно разделить на три группы:

- регистры общего назначения – AX, BX, CX, DX;
- регистры-указатели – DI, SI, BP, SP, IP;
- сегментные регистры – CS, DS, SS, ES.

Регистры общего назначения. Регистры общего назначения можно рассматривать как четыре 16-битовых или восемь 8-битовых регистров. В первом случае регистры имеют имена AX, BX, CX, DX эти регистры образованны из 8-битовых регистров AL, AH, BL, BH, CL, CH, DL, и DH. Здесь L и H означают младшие и старшие байты 16-битовых регистров. Например, регистры AL и AH образуют соответственно младший и старший байты регистра AX.

Сегментные регистры CS, DS, SS, ES. Программы и данные хранятся в отдельных областях памяти. Эти области (или сегменты) могут иметь объем до 64 Кб. Микропроцессор может иметь дело одновременно с четырьмя сегментами. Начальные адреса этих сегментов хранятся в его четырех сегментных регистрах.

Регистр дополнительного сегмента ES (extra segment) указывает на текущий дополнительный сегмент, который используется при выполнении операций над строками.

Язык Си поддерживает исключительно удобный механизм доступа к внутренним регистрам микропроцессора через псевдопеременные, ссылка на которые в Си-программе компилируется в ссылку на внутренний регистр микропроцессора:

_AL, _AH, _BL, _BH, _CL, _CH, _DL, _DH;
_AX, _BX, _CX, _DX;
_DI, _SI, _BP, _SP, _IP;
_CS, _DS, _SS, _ES.

Для того чтобы записать в регистр необходимое значение достаточно присвоить псевдопеременной это значение, например: _AX = 0x07.

Прерывания. В самом общем виде это наличие в аппаратуре специальных средств, с помощью которых выполнение текущей программы приостанавливается и процессор переходит к так называемой программе обслу-

живания прерывания. Механизм прерываний позволяет организовать выполнение тех или иных функций ядра и быструю реакцию процессора на возникновение каких-то внешних событий.

Часто обработчикам программных прерываний требуется передать какие-то значения, задающие конкретное действие, характеристики ситуации и т.п., и получить какие-то результаты по завершении исполнения прерывания. Для такого обмена данными используются внутренние регистры процессора. Прерывания подобного вида используются в данной лабораторной работе. Перед вызовом прерывания необходимо передать определенные параметры, значения которых заносятся в регистры.

Описание функций 13-го прерывания BIOS для работы с контроллером на гибком магнитном диске

Функция 00h – привести в исходное состояние дисковую систему. Приводит в исходное состояние контроллер диска и подключенные к нему дисководы, позиционер головок чтения/записи перемещается к цилиндру 0 и подготавливает систему к операциям ввода-вывода на диске.

При вызове AH = 00h, DL = дисковод (00h, 01h, …, 80h, 81h, …).

При возврате, если функция выполнена успешно, флаг переноса сброшен (CF = 0). Если функция не выполнена, флаг переноса установлен (CF = = 1) и AH = состояние (см. Int 13н с функцией 01h).

Функция 00h вызывает сброс и рекалибровку дискового контроллера, позиционер головки устанавливаются на нулевой цилиндр. Если в адресе дисковода старший бит (бит 7) установлен в 1, выполняется сброс контроллера НМД. Сброс рекомендуется выполнять после того, как произошла ошибка при выполнении других операций, таких, как чтение или запись. После сброса можно попытаться повторить операцию. Адрес дисковода 0 соответствует первому флоппи-диску A, 1 – второму B и т.д. Адреса 80, 81 соответствуют первому и второму физическим накопителям на жестком магнитном диске. Функцию следует вызывать после сбоя при выполнении запросов на чтение, запись, верификацию или форматирование на гибком диске перед повторением операции. Если при вызове функции DL = 80h (т.е. выбирается жесткий диск), приводятся в исходное состояние контроллеры гибкого и жесткого дисков.

Функция 01h – получить состояние дисковой системы. Возвращает состояние последней дисковой операции.

При вызове AH = 01h, DL = дисковод (00h, 01h, …, 80h, 81h, …).

При возврате AH = 00H, AL = состояние дисковода после завершения последней операции.

Эта функция может быть использована для анализа результата выполнения дисковой операции и получения кода ошибки. Передаваемый в регистре AL код ошибки (таблица 12.1, Г – только для гибкого диска; Ж – только

для жесткого диска) функция берет из области данных BIOS – из байта с адресом 0000:0441h.

При использовании жесткого диска код ошибки 11h (ошибка скорректированных данных) указывает на обнаружение восстановимой ошибки в процессе предшествующего чтения сектора (Int 13h с функцией 02h).

Таблица 12.1. – Коды ошибок дисковой операции

Код ошибки	Описание
00H	Отсутствие ошибки
01H	Недопустимая команда
02H	Не найдена адресная метка
03H	Диск защищен от записи (Г)
04H	Не найден сектор
05H	Сброс в исходное состояние не выполнен (Ж)
06H	Гибкий диск снят (Г)
07H	Дефектная таблица параметров (Ж)
08H	Нарушение границ ПДП (Г)
09H	ПДП пересек границу 64 Кбайт
0AH	Флаг дефектного сектора (Ж)
0BH	Флаг дефектной дорожки (Ж)
0CH	Не найден тип носителя (Г)
0DH	Недопустимое число секторов при форматировании
0EH	Обнаружена метка адреса управляющих данных - уровень арбитража ПДП вышел из диапазона (Ж)
0FH	Ошибка ПДП
10H	Невосстановимая ошибка циклического контроля (ECC)
11H	Ошибка данных скорректирована кодом проверки корректировки (Ж)
20H	Отказ контроллера
40H	Сбой поиска дорожки
80H	Тайм-аут диска (отсутствие ответа)
AAH	Дисковод не готов (Ж)
BBH	Неопределенная ошибка (Ж)
CCH	Отказ записи (Ж)
E0H	Ошибка регистра состояния (Ж)
FFH	Операция опознания не выполнилась (Ж)

Функция 02h – прочитать сектор. Читает в память один или более секторов. Эта функция позволяет прочитать один или несколько секторов диска в буфер, находящийся в оперативной памяти. Необходимо задать для начального сектора номер дорожки, головки и самого сектора.

Для НГМД номер дорожки и сектора задается следующим образом: биты регистра CX 5 ... 0 задают номер сектора, а биты 15 ... 6 – номер дорожки. Перед чтением необходимо подготовить таблицу параметров дискеты или диска (для операций с НГМД).

При вызове AH = 02h, AL = число секторов, CH = цилиндр (дорожка), CL = сектор, DH = головка, DL = дисковод (00h, 01h, …, 80h, 81h, …), ES: BX = сегмент: относительный адрес буфера.

При возврате, если функция выполнена успешно, флаг переноса сброшен (CF = 0) и AH = 00h, AL = число переданных секторов. Если функция не выполнена, флаг переноса установлен (CF = 1) и AH = состояние (см. Int 13h с функцией 01h).

При использовании жестких дисков код ошибки 11h свидетельствует о том, что ошибка чтения была скорректирована алгоритмом ECC (проверки и корректировки ошибки); в этом случае регистр AL содержит длину посылки. Возвращенные данные, скорее всего, правильны, хотя имеется незначительная вероятность неправильной корректировки. Если была запрошена многосекторная пересылка, операция завершилась передачей сектора, содержащего ошибку чтения. При использовании гибких дисков ошибка может возникнуть, если к моменту запроса выключен мотор. ПЗУ BIOS не организует автоматического ожидания раскрутки мотора до требуемой скорости перед попыткой выполнения операции чтения. Запрашивающая программа должна установить в исходное состояние дисковую систему (Int 13h с функцией 00h) и трижды повторить операцию перед тем, как прийти к заключению, что ошибка вызвана чем-то другим.

Функция 03h – записать сектор. Записывает из памяти на диск один или более секторов. Функция записи секторов аналогична предыдущей функции, за исключением направления перемещения данных. В этом случае данные записываются из буфера в сектора диска.

При вызове AH = 03h, AL = число секторов, CH = цилиндр, CL = сектор, DH = головка, DL = дисковод (00h, 01h, …, 80h, 81h, …).

При возврате, если функция выполнена успешно, флаг переноса сброшен (CF = 0) и AH = 00H, AL = число переданных секторов. Если функция не выполнена, флаг переноса установлен (CF = 1) и AH = состояние (см. Int 13h с функцией 01h).

Замечания. При использовании гибких дисков ошибка может возникнуть, если к моменту запроса выключен мотор. ПЗУ BIOS не организует автоматического ожидания раскрутки мотора до требуемой скорости перед попыткой выполнения операции записи. Запрашивающая программа должна установить в исходное состояние дисковую систему (Int 13h с функцией 00h) и трижды повторить операцию перед тем, как прийти к заключению, что ошибка вызвана чем-то другим.

Функция 04h – верифицировать сектор – верифицирует адресные поля одного или более секторов. Эта операция не пересылает данные ни в память, ни из памяти. С помощью этой функции можно убедиться, что указанные сектора существуют и их можно прочесть. Данные проверяются по методу избыточного циклического контроля (CRC). Адрес буфера не нужен, так как чтения данных в оперативную память при проверке секторов не происходит.

При вызове AH = 04 h, AL = число секторов, CH =цилиндр (дорожка), CL = сектор, DH = головка, DL= дисковод (00h, 01h, …, 80h, 81h, …).

При возврате если функция выполнена успешно, флаг переноса сброшен (CF = 0) и AH = 00h, AL = число верифицированных секторов. Если функция не выполнена, флаг переноса установлен (CF = 1) и AH = состояние (см. Int 13h с функцией 01h). Эту функцию можно использовать для проверки, установлен ли диск в дисководе гибких дисков. Если используется компьютер с BIOS, выпущенной ранее 11.15.85, регистры ES:BX должны указывать на буфер соответствующего размера, как и при выполнении операции чтения.

Функция 05h – форматировать дорожку. Инициализирует поля адресов секторов и дорожки на указанной дорожке диска. Функция форматирования предназначена для начального формирования структуры дорожки диска, она разрушает все имеющиеся на дорожке данные. С помощью функции 05h можно за один раз отформатировать только одну дорожку с указанным номером. Для этой функции необходимо задать буфер формата.

Перед вызовом функции форматирования регистры ES:BX должны содержать полный адрес буфера формата. Для дискет перед форматированием этот буфер должен представлять собой заполненный массив 4-байтовых элементов: номера дорожки, головки, сектора и кода размера сектора. Код размера сектора может иметь следующие значения:

– 128 байт на сектор;
– 256 байт на сектор;
– 512 байт на сектор;
– 1024 байт на сектор.

Количество элементов в массиве должно быть равно количеству создаваемых на дорожке секторов, т.е. для каждого сектора буфер формата должен содержать один описывающий его 4-байтовый элемент.

При форматировании флоппи-дисков с помощью этой функции таблица параметров дискеты должна содержать правильное значение количества секторов на дорожке и другие параметры.

При вызове AH = 05h, AL = чередование (жесткие диски PC/XT), CH = = цилиндр, DH = головка, DL= дисковод (00h, 01h, …, 80h, 81h, …), ES:BX = сегмент: относительный – адрес списка адресных полей.

При возврате, если функция выполнена успешно, флаг переноса сброшен (CF = 0) и AH = 00h. Если функция не выполнена, флаг переноса установлен (CF = 1) и AH = состояние (см. Int 13h с функцией 01h)

На гибких дисках список адресных полей состоит из последовательности 4-байтовых записей, по одной записи на сектор, имеющих следующий формат (таблица 12.2).

Таблица 12.2. – Формат адресного поля для гибкого диска

Байт	Содержимое
0	Цилиндр
1	Головка
2	Сектор
3	Код размера сектора: 00h, если 128 байт на сектор 01h, если 256 байт на сектор 02h, если 512 байт на сектор (стандарт) 03 h, если 1024 байт на сектор

На гибких дисках число секторов на дорожку берется из таблицы параметров гибкого диска в BIOS, адрес которой записан в векторе прерывания Int 1eh. Если эта функция используется для формирования гибких дисков на машинах PC/AT или PS/2, ей должен предшествовать вызов Int 13h с функцией 17H для выбора носителя, предназначенного для форматирования. При использовании жестких дисков два старших бита 10-битного номера цилиндра помещаются в два старших бита регистра CL. При использовании жестких дисков машин PC/XT-286, PC/AT и P5/2 регистр ES:BX указывает на 512-байтовый буфер, содержащий пары байтов для каждого сектора физического диска. Нулевой байт содержит 00h – для работоспособного сектора, 80h – для дефектного сектора. Первый байт содержит номер сектора.

Например, для форматирования дорожки с 17 секторами и чередованием, равным 2, ES:BX должен указывать на следующий 34-байт массив в начале 512-байт буфера:

db 00h, 01h, 00h, 0ah, 00h, 02h, 00h, 0bh, 00h, 03h, 00h, 0ch
db 00h, 04h, 00h, 0dh, 00h, 05h, 00h, 0eh, 00h, 06h, 00h, 0fh
db 00h, 07h, 00h, 10h, 00h, 08h, 00h, 11h, 00h, 09h

Функция 08h – получить параметры дисковода. С помощью этой функции программа может определить тип дисковода, количество дисководов, обслуживаемых первым дисковым контроллером, и другие параметры дисковода, которые нужны программе для организаций доступа к диску на физическом уровне. Возвращает различные параметры указанного дисковода.

При вызове AH = 08h, AL = дисковод (00h, 01h, …, 80h, 81h, …).

При возврате если функция выполнена успешно, флаг переноса сброшен (CF = 0) и BL = тип дисковода (гибкие диски PC/AT и PS/2):

– 01h, если 360 Кб, 40 дорожек, 5,25";
– 02h, если 1,2 Мб, 80 дорожек, 5,25";
– 03h, если 720 Кб, 80 дорожек, 3,5";
– 04h, если 1,44 Мб, 80 дорожек, 3,5".

CH = младшие 8 бит максимального номера цилиндра, CL = биты 6 – 7: два старших бита максимального номера цилиндра, биты 0 – 5: максимальный

номер сектора, DH = максимальный номер головки, DL = число дисководов, ES:DI = сегмент: относительный адрес таблицы параметров дисковода.

Если функция не выполнена, флаг переноса установлен (CF = 1) и AH = = состояние (см. Int 13h с функцией 01h).

На машинах PC и PC/XT эта функция поддерживается только на жестких дисках. Значение, возвращаемое в регистре DL, отражает действительное число физических дисководов, прикрепленных к адаптеру запрошенного дисковода.

Пример программы, использующей функцию 03h Int 13h записи из памяти на диск одного сектора

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include <stdlib.h>
#include <time.h>
//Массив данных для записи в сектор
static char data_write[512];
//Значение сегмента и смещения буфера в памяти
char *seg,*off;
int disc;
void main(void) {
disc=0x00;//дисковод A
//Выполнить функцию "сброс" для дисковода
{
_AH=0x00;
_DL=disc;
geninterrupt(0x13);
}
//Сегмент: относительный адрес буфера
//формирование смещения в памяти блока данных для
записи
off=(char*) FP_OFF(data_write);
//формирование сегмента в памяти блока данных для
записи
seg=(char*) FP_SEG(data_write);
//формирование указателя в памяти блока данных для
записи
ptr1=(char*) MK_FP(seg, off);
 //Запись в сектор данных по адресу ES:BX
  do{
  _BX = (int) off;
  _ES = (int) seg;
```

```
  _AH = 0x03;
  _AL = 1;
  _CH = 0x01;
  _CL = 0x1;
  _DH = 0x00;
  _DL = disc;
  geninterrupt(0x13);
    }while((_AH!=0x0 || _AL==0));
  }
```

Чтение элемента FAT. Выборку элемента 12-битной FAT можно вести по следующему алгоритму: выбирается слово по смещению i*1,5 от начала FAT, где i – номер кластера; если i четное, содержимое элемента FAT составляют младшие 12 бит этого слова, в противном случае – старшие 12 бит. Для 16-битной FAT слово по смещению i*2 содержит элемент FAT.

```
//выделение памяти под FAT
byte *buff = (byte*) realloc(buff, FatSize*512);

//здесь код чтения FAT в buff

//форматная распечатка FAT
word ss;
for(i=0;i<10; i++);
   if (fat12) {
   int m=(i*3)/2; ss=*(word *)(buff+m);
   if(i%2) /*нечетный элемент */ ss>>=4;
   else /* четный элемент */ ss&=0x0fff;
   if (ss>0x0fef) printf(" %03xH ",ss);
   else printf("%5d ",ss);
   }
  else {
   m=i*2; ss=*(word *)(buff+m);
   if (ss>0xffef) printf("%04xH ",ss);
   else printf("%5d ",ss);
}
```

Порядок выполнения работы

Согласно одному из следующих вариантов задания написать и отладить программу на языке Си или Ассемблер, работающую с накопителем на гибких магнитных дисках, используя прерывание 13h:

1. Верифицирование, запись сектора на дискету и чтения сектора с дискеты, а также форматирование дорожки дискеты.

2. Вывод следующей информации: количество кластеров на диске, количество секторов на кластер, количество байт на сектор, количество плохих кластеров.

3. Чтение содержимого корневого каталога: имя файла, атрибут, дата, время, номер кластера, размер.

Контрольные вопросы:

1. Какова физическая организация накопителей на жестком и гибком дисках?
2. Из каких составляющих состоит адрес сектора (для чтения/записи)?
3. Каково назначение boot-сектора?
4. Каково назначение FAT?
5. Какие функции (номер прерывания и функции) имеются в сервисе BIOS для работы с файловой системой дисков?
6. В чем различие FAT12, FAT16 и FAT32?

Лабораторная работа № 13.
Архитектура видеосистемы ПЭВМ
(изучение структуры текстового видеобуфера)

Цель работы: приобретение навыков компиляции программ на ассемблере, изучение структуры текстового видеобуфера и программирование его с использованием прямой адресации памяти.

Краткие теоретические сведения

Разработка программы на языке ассемблера включает **четыре этапа**:

1-й этап. Подготовка исходного текста программы и оформление его в виде текстового файла с помощью какого-нибудь редактора в формате DOS с расширением .asm.

2-й этап. Ассемблирование программы с применением транслятора TASM, результатом которого является объектный файл с расширением .obj. Если в процессе трансляции будут обнаружены ошибки, то объектный файл не создаётся, а формируется сообщение об ошибках. Ошибки необходимо устранить, после чего требуется повторная трансляция. Объектный файл (двоично-кодированное представление программы) не может быть запущен на исполнение, так как в нём не содержится информация о загрузке сегментов программы в памяти компьютера.

3-й этап. Компоновка программы производится компоновщиком (редактором связей) Turbo Linker и заключается в объединении объектных модулей в один исполняемый файл с назначением стартового адреса программы. Исполняемый файл имеет расширение .exe. Второй и третий этапы определяют процесс подготовки исполнительного файла программы, называемого «трансляцией».

4-й этап состоит в отладке программы с использованием отладчика Turbo Debugger – основного инструмента при изучении форматов команд, их кодирования, а также представления переменных программы в памяти.

Рассмотрим процесс трансляции программы на примере простой интерактивной программы *Hello.asm*. Для ассемблирования файла *Hello.asm* в командной строке наберите: *tasm hello.asm* и нажмите клавишу *Enter*. В результате будет создан объектный файл с этим же именем *hello.obj*. Заметим, что расширение имени файла *.asm* вводить не обязательно, т.к. TASM принимает это по умолчанию. На экране вы увидите следующее:

```
    Turbo Assembler Version 3.0 Copyright (C) 1988,1996
Borland International
    Assembling file:             Hello.asm
    Error messages:              None
    Warning messages:         None
    Passes                              1
    Remaining memory: ***K
```

Из сообщения следует, что ассемблирование завершено без ошибок и отсутствия предупреждений. Предупреждение не является ошибкой, однако его игнорирование может привести к неприятностям в дальнейших этапах работы с программой, поэтому лучше своевременно реагировать на данный тип замечаний.

Для компоновки программы введите в командную строку: *tlink hello.obj*. Здесь, так же, как и при ассемблировании, расширение имени *.obj* не является обязательным. По завершению компоновки будет сформирован файл *hello.exe* с выводом на экран сообщения:

```
Turbo Linker Version 4.0. Copyright (C) 1991 Bor-
land International
```

Теперь программу *hello.exe* можно запустить на исполнение, результатом которого будет вывод на экран сообщения:

```
Hello, world
```

Курсор будет мерцать после последнего символа в ожидании нажатия любой клавиши. При нажатии на клавишу произойдет завершение программы. Для значительного упрощения процедуры компиляции программы рекомендуется создать .bat-файл (например файл compile.bat) содержащий всего две строки:

```
tasm имя_файла.asm
tlink имя_файла.obj
```

Этот файл должен располагаться в одной директории с исходным текстом программы и файлами tasm.exe и tlink.exe. В случае удачной компиляции (исходный код набран без ошибок) в директории появятся файлы:

```
имя_файла.map
имя_файла.obj
имя_файла.exe
```

Структура видеобуфера

Современные видеоконтроллеры поддерживают разнообразные текстовые и графические режимы. Текстовые режимы различаются по разрешению (число отображаемых символов по горизонтали и вертикали) и цветовой палитре (монохромный или 16-цветовой режим). Для графических режимов основным признаком классификации является количество одновременно отображаемых цветов и, соответственно, количество бит видеопамяти, отводимое на каждую точку (пиксел) изображения. Различают следующие типы графических режимов:

– монохромный (1-битное кодирование);

– 16-цветовой EGA/VGA (4-битное кодирование);

– 256-цветовой SVGA (8-битное кодирование);

– HiColor (16-битное кодирование);

– TrueColor (24-битное / 32-битное кодирование).

Графические режимы VGA (SVGA) сильно устарели, а текстовые продолжают успешно применяться.

Всё, что изображено на мониторе – графика, текст – одновременно присутствует в памяти, встроенной в видеоадаптер. Для того чтобы изображение появилось на мониторе, оно должно быть записано в память видеоадаптера. В текстовом режиме для VGA-совместимых систем для видеопамяти отводится адресное пространство, начинающееся с логического адреса B800h:0000h и заканчивающееся адресом BF00h:0FFFh. Данная область разбивается на 8 секторов по числу видеостраниц (4 Кб на страницу). Таким образом, постраничное деление адресного пространства видеопамяти в текстовом режиме имеет следующий вид:

– B800h:0000h – страница 0, смещение в диапазоне 0000h – 0FFFh;

– B900h:0000h – страница 1, смещение в диапазоне 0000h – 0FFFh;

– .....

– BF00h:0000h – страница 7, смещение в диапазоне 0000h – 0FFFh.

На экране отображается видеобуфер, соответствующий активной странице. **По умолчанию используется страница 0.**

В текстовых режимах для изображения каждого символа отводится 2 байта: байт с ASCII-кодом символа и байт с его атрибутом. При этом по адресу B800h:0000h находится байт с кодом символа (левый верхний угол экрана), а в B800h:0001h – атрибут этого символа; B800h:0002h – код второго символа, а в B800h:0003h – атрибут второго символа и т.д. по столбцам и строкам. Структура байта атрибутов приведена на рисунке 13.1.

Рисунок 13.1. – Структура байта атрибутов

Из рисунка 13.1 следует, что каждый символ может принимать любой из 16 возможных цветов, определяемых сочетанием младших 4-х битов. Биты 4 – 6 байта атрибутов задают цвет фона под данным символом. Последний бит 7, в зависимости от режима видеоадаптера, определяет либо яркость фона под данным символом (тогда фон также может принимать 16 разных цветов), либо мерцание символа (*устанавливается DOS по умолчанию).*

При загрузке машины устанавливается стандартная палитра, коды цветов которой приведены в таблице 13.1. Рассмотрим некоторые примеры. Так,

в режиме мерцания значение старшего полубайта атрибута 8h обозначает не серый фон, а чёрный при мерцающем символе, цвет которого по-прежнему определяется младшим полубайтом; значение старшего полубайта 0Ch – красный фон при мерцающем символе. Переключение назначения бита 7 осуществляется подфункцией 03h функции 10h прерывания Int 10h.

Таблица 13.1. – Коды цветов стандартной палитры

Код	Цвет	Код	Цвет
0h	Чёрный	8h	Серый
1h	Синий	9h	Голубой
2h	Зелёный	0Ah	Салатовый
3h	Бирюзовый	0Bh	Светло-бирюзовый
4h	Красный	0Ch	Розовый
5h	Фиолетовый	0Dh	Светло-фиолетовый
6h	Коричневый	0Eh	Жёлтый
7h	Белый	0Fh	Ярко- белый

В текстовом режиме 80x25 (80 столбцов, 25 строк) двухбайтовые коды символов записываются в видеобуфер в том порядке, в каком они должны появиться на экране: первые 80*2 байт соответствуют первой строке экрана, вторые 80*2 байт – второй и т.д. При этом переход на следующую строку экрана определяется не управляющими кодами возврата каретки и перевода строки, а размещением кода в другом месте видеобуфера. Для того чтобы из программы получить доступ к видеобуферу, надо занести в один из сегментных регистров данных сегментный адрес видеобуфера. После этого, задавая те или иные смещения, можно выполнить запись в любые места (ячейки) видеобуфера. Вычислить смещение ячейки в координатах «строка-столбец» *(row, clm)* можно так:

*VidAdd r = (row*160) + (clm*2)*

Для установки текстового режима 80x25 (режим № 3) можно использовать нулевую функцию прерывания видеосервиса 10h:

```
mov ah,0 ;номер функции
mov al,3 ;номер режима
int 10h  ;вызов прерывания видеосервиса
```

Для вывода информации в видеобуфер существуют наборы функций DOS и BIOS, но их использование часто неоправдано из-за низкой производительности, в этом случае используют прямой доступ к памяти видеобуфера для отображения или чтения из него информации. Для того, чтобы организовать такой доступ необходимо, для начала, загрузить в сегментный регистр соответствующий адрес видеобуфера, например:

```
mov ax,0B800h
mov es,ax
```

Заметьте, что сегментный регистр нельзя инициализировать напрямую, а только с использованием промежуточного регистра общего назначения. Затем необходимо использовать какой-либо индексный или базовый регистр для указания смещения, по которому будет записываться или считываться информация. Например, для чтения символа из верхнего левого угла в регистр AL можно написать:

```
mov  si,0
mov  al,es:[si]
```

И соответственно для записи в верхний левый угол латинской буквы A (код 41h, таблица 13.2) красным цветом можно написать:

```
mov si,0  ;задание смещения символа в видеобуфере
mov al,41h ;задание кода символа
mov es:[si],al ;вывод символа
inc si        ;увеличение si на единицу
mov ah,4h          ;красный цвет символа, черный фон
mov es:[si],al ;вывод атрибутов символа
```

С учетом того, что AX – 2-байтный регистр, включающий AH и AL (байтные регистры), этот код можно переписать следующим образом:

```
mov si,0 ;задание смещения левого верхнего угла
mov ax,0441h  ;задание атрибутов и кода символа
mov es:[si],ax ;вывод атрибутов и кода символа в
;левый верхний угол
```

С учетом вышесказанного, полный исходный текст для вывода буквы A на экран в левом верхнем углу будет выглядеть следующим образом:

```
.model small
.386
.data                  ;сегмент данных
.stack            ;сегмент стека
.code                  ;сегмент кода
main proc              ;точка входа в главную функцию
mov ah,0 ;номер функции установки ;видеорежима
mov al,3 ;номер видеорежима (текстовый ;80x25)
int 10h           ;вызов прерывания видеосервиса
mov ax,0b800h     ;загрузка сегмента видеобуфера
mov es,ax         ;загрузка сегмента видеобуфера
mov si,0 ;установка смещения символа в ;буфере
mov ax,0441h ;загрузка кода символа и ;атрибута
mov es:[si],ax    ;вывод символа на экран
```

```
mov ah,0          ;функция ожидания ввода символа
int 16h           ;для организации задержки
mov ax,4c00h ;номер функции завершения ;программы
int 21h           ;завершение программы
main endp             ;конец главной функции
end main          ;точка входа в программу
```

Кодовая таблица ASCII (таблица 2) представляет собой полный набор символов, доступных в операционной системе DOS.

Таблица 13.2. – Кодовая таблица ASCII

Задания к выполнению лабораторной работы выдаются преподавателем непосредственно перед занятием

Примерный вариант задания

Разработать программу, выполняющую прорисовку на экране заданной фигуры в текстовом режиме 80x25 с использованием символов псевдографики.

Содержание отчета:

1. Тема и цель работы.
2. Задание для лабораторной работы.
3. Блок-схема программы.
4. Листинг программы.
5. Выводы по работе.

Контрольные вопросы:

1. Из каких этапов состоит разработка программы на ассемблере?
2. Какие программы используется для компиляции ассемблерного кода?
3. Что такое ассемблирование программы?
4. Что такое компоновка программы?
5. Что такое видеобуфер?
6. Какова структура видеобуфера в текстовом режиме?
7. Как задается цвет фона и цвет символа?

Лабораторная работа № 14.
Система ввода/вывода
(Программирование манипулятора типа «мышь»)

Цель работы: изучение функций, предоставляемых прерыванием 33h для работы с устройством ввода «мышь».

Краткие теоретические сведения

Для программирования устройства вода типа «мышь» предоставляются функции прерывания 33h. Рассмотрим их подробно.

Функция 00h – Сброс драйвера мыши. Сбрасывает (инициализирует) драйвер мыши.

Входные данные: AX = 0000h.

Выходные данные: AX = состояние мыши, AX = FFFFh: драйвер мыши установлен, AX = 0000h: ошибка, драйвер мыши не установлен, BX = число кнопок мыши.

Программа инициализации выполняет следующие задачи:

1. Перемещает маркер мыши в центр экрана и стирает его изображение на экране. После разрешения вывода маркера маркер мыши по умолчанию имеет вид инверсного прямоугольника. Этот маркер всегда воспроизводится на нулевой экранной странице независимо от текущего видеорежима. Областью перемещения мыши становится весь экран.
2. Устанавливает обработчик событий (event handler) (по умолчанию не устанавливается).
3. Устанавливает эмуляцию светового пера (по умолчанию не устанавливается).
4. Задает скорость перемещения маркера мыши. По умолчанию относительная скорость равно 8 микки (1 микки = 1/200 дюйма) на 8 горизонтальных элементов и 16 микки на 16 вертикальных.
5. Задает максимальную скорость мыши (по умолчанию равна 64 микки в секунду).

Функция 01h – Вывод маркера мыши. Выводит на экран маркер мыши. Этот маркер отображает любое движение мыши, перемещаемой пользователем.

Входные данные: AX = 0001h.

Выходные данные: Отсутствуют.

Эта функция увеличивает на единицу значение внутреннего счетчика, который определяет: должен ли маркер мыши быть виден на экране. После инициализации драйвера мыши функцией 00h этот счетчик содержит -1 (т.е. маркер мыши не виден). Когда после обращения к функции 01h значение этого счетчика становится нулевым, маркер мыши появляется на экране.

Драйвер мыши отображает перемещение мыши даже тогда, когда маркер не воспроизводится на экране. После обращения к этой функции маркер может появляться не в том месте, в каком он находился в момент удаления маркера в результате обращения к функции 00h или 02h.

Функция 02h – Удаление маркера мыши. Удаляет маркер мыши с экрана.

Входные данные*:* AX = 0002h.

Выходные данные: Отсутствуют.

Эта функция уменьшает на единицу значение внутреннего счетчика, который определяет: должен ли маркер мыши быть виден на экране. Если счетчик имеет значение 0, то маркер мыши воспроизводится на экране; если счетчик имеет значение -1, то маркер удаляется с экрана. Драйвер мыши отображает перемещение мыши даже тогда, когда маркер не воспроизводится на экране. После обращения к этой функции маркер может появляться не в том месте, в каком он находился в момент удаления маркера в результате обращения к функции 00h или 02h.

Функция 03h – Чтение положения маркера/состояния кнопок. Возвращает текущее положение маркера мыши и текущее состояние кнопок мыши.

Входные данные: AX = 0003h.

Выходные данные: BX = состояние кнопок мыши, бит 0 = 1: нажата левая кнопка, бит 1 = 1: нажата правая кнопка, бит 2 = 1: нажата средняя кнопка, биты 3 – 15: не используются; CX = координата X (горизонтальная координата маркера); DX = координата Y (вертикальная координата маркера).

Координаты, возвращаемые в регистрах CX и DX, являются координатами элементов изображения на виртуальном экране мыши, а не физическими координатами на реальном экране. Если на мыши имеются только две кнопки, то информация о центральной кнопке не имеет значения.

Функция 04h – Перемещение маркера мыши. Перемещает активный маркер мыши в указанную точку экрана.

Входные данные: AX = 004h, CX = координата X (горизонтальная координата маркера), DX = координата Y (вертикальная координата маркера).

Выходные данные: Отсутствуют.

Примечания: Координаты, возвращаемые в регистрах CX и DX, являются координатами элементов изображения на виртуальном экране мыши, а не физическими координатами на реальном экране. Если указанная в обращении позиция находится за пределами диапазона перемещения маркера мыши, заданного функциями 07h и 08h, то функция корректирует координаты таким образом, что маркер остается внутри диапазона. Маркер перемещается в новую позицию даже в том случае, если он не воспроизводится на экране. После того, как воспроизведение маркера мыши снова будет разрешено, он появится в новой позиции.

Функция 05h – Определение числа нажатий кнопки мыши. Информирует вызывающую программу о том, сколько раз была нажата указанная кнопка мыши с момента последнего обращения к функции 05H. Функция 05H также сообщает вызывающей программе координаты маркера на экране в момент последнего нажатия кнопки.

Входные данные: AX = 0005h; BX = кнопка мыши; BX = 0: левая кнопка мыши; BX = 1: правая кнопка мыши; BX = 2: средняя кнопка мыши.

Выходные данные: BX = состояние всех кнопок мыши: бит 0 = 1: нажата левая кнопка; бит 1 = 1: нажата правая кнопка; бит 2 = 1: нажата средняя кнопка; биты 3–15: не используются. BX = кнопки мыши, которые были нажаты с момента последнего обращения к функции. CX = горизонтальная координата в момент последнего нажатия. DX = вертикальная координата маркера в момент последнего нажатия.

Координаты, возвращаемые в регистрах CX и DX, являются координатами элементов изображения на виртуальном экране мыши, а не физическими координатами на реальном экране. При обращении к этой функции счетчик числа нажатий указанной клавиши сбрасывается в ноль.

Функция 06h – Определение числа отпусканий кнопки мыши. Информирует вызывающую программу о том, сколько раз была отпущена указанная кнопка мыши с момента последнего обращения к функции 06h. Функция 06h также сообщает вызывающей программе координаты маркера на экране в момент последнего отпускания кнопки.

Выходные данные*:* AX = 0006h; BX = кнопка мыши; BX = 0: левая кнопка мыши; BX = 1: правая кнопка мыши; BX = 2: средняя кнопка мыши.

Выходные данные*:* BX = состояние всех кнопок мыши: бит 0 = 1, бит 1 = = 1, бит 2 = 1; BX = кнопки мыши, отпущенные с момента последнего обращения; CX = горизонтальная координата маркера в момент последнего отпускания кнопки; DX = вертикальная координата маркера в момент последнего отпускания кнопки.

Координаты, возвращаемые в регистрах CX и DX, являются координатами элементов изображения на виртуальном экране мыши, а не физическими координатами на реальном экране. При обращении к этой функции счетчик числа нажатий указанной клавиши сбрасывается в ноль.

Функция 07h – Задание диапазона перемещения по горизонтали. Определяет диапазон перемещения маркера мыши по горизонтали. После того, как диапазон установлен, пользователь не может вывести маркер мыши за его пределы.

Входные данные*:* AX = 0007h; CX = минимальная горизонтальная координата маркера; DX = максимальная горизонтальная координата маркера.

Выходные данные*:* Отсутствуют.

Координаты, передаваемые в регистрах CX и DX, описывают положение элементов изображения на виртуальном экране мыши, а не физические координаты на реальном экране. Если в момент обращения к функции 07h маркер мыши находится за пределами устанавливаемого диапазона, то драйвер мыши автоматически перемещает его внутрь диапазона. Если значение DX меньше значения CX, то эти параметры меняются местами.

Функция 08h – Задание диапазона перемещения мыши по вертикали. Определяет диапазон перемещения маркера мыши по вертикали. После того, как диапазон установлен, пользователь не может вывести маркер мыши за его пределы.

Входные данные*:* AX = 0008h; CX = минимальная вертикальная координата маркера; DX = максимальная вертикальная координата маркера.

Выходные данные*:* Отсутствуют.

Координаты, передаваемые в регистрах CX и DX, описывают положение элементов изображения на виртуальном экране мыши, а не физические координаты на реальном экране. Если в момент обращения к функции 08h маркер мыши находится за пределами устанавливаемого диапазона, то драйвер мыши автоматически перемещает его внутрь диапазона. Если значение DX меньше значения CX, то эти параметры меняются местами.

Функция 09h – Описание маркера мыши (в графическом режиме). Описывает внешний вид маркера мыши в графическом режиме, а также битовое поле, корректирующее элементы изображения вокруг маркера мыши.

Входные данные*:* AX = 0009h; BX = ширина маркера, начиная с левого края битового поля; CX = высота маркера, начиная с верхнего края битового поля; EX = адрес сегмента битового поля; DX = смещение битового поля.

Выходные данные*:* Отсутствуют.

Битовое поле состоит из 64 байтов, из которых первые 32 являются результатом операции AND, а остальные 32 байта – результат операции OR с текущими элементами изображения.

Функция 0ah – Описание маркера мыши (в тестовом режиме). Описывает битовую маску, определяющую внешний вид маркера в текстовом режиме.

Входные данные*:* AX = 000ah; BX = тип маркера; BX = 0: программный; BX = 1: аппаратный; CX = маска AND (программный маркер) или начальная линия (аппаратный маркер); DX = маска XOR (программный маркер) или конечная линия (аппаратный маркер).

Выходные данные*:* Отсутствуют.

Если выбран программный маркер, то код символа, находящегося под маркером, и байт атрибутов этого символа логически умножаются (AND) на маску, заданную в регистре CX, а затем выполняется операция «исключающее или» (XOR) между результатом умножения и маской в регистре DX.

Для байта атрибутов эти операции выполняются со старшим байтом регистров CX и DX (CH и DH), а для кода символа – с младшим байтом (CL и DL). Аппаратный маркер имеет такую же форму, как обычный текстовый курсор. В монохромном режиме значения начальной и конечной линий изменяются в диапазоне от 0 до 13. В цветном режиме значение линий изменяется от 0 до 7.

Функция 0bh – Определение величины перемещения. Определяет расстояние между текущим положением мыши и положением мыши в момент последнего обращения к функции 0bh.

Входные данные*:* AX = 000bh.

Выходные данные*:*CX = расстояние от последней точки по горизонтали (в микки); DX = расстояние от последней точки по вертикали (в микки).

Эти значения должны интерпретироваться как числа со знаком. Положительные значения указывают на перемещение в нижнюю или правую часть экрана, отрицательные – в верхнюю или левую часть экрана. Расстояния выражены в микки (1 микки = 1/200 дюйма), а не в элементах изображения.

Функция 0ch – Задание обработчика событий. Задает адрес обработчика событий, вызываемого драйвером мыши в случае определенных событий, имеющих отношение к мыши.

Входные данные*:* AX = 000ch; CX = события, вызывающие обращение к обработчику (маска событий): бит 0: перемещение мыши, бит 1: нажатие левой кнопки мыши, бит 2: отпускание левой кнопки мыши, бит 3: нажатие правой кнопки мыши, бит 4: отпускание правой кнопки мыши, бит 5: нажатие средней кнопки мыши, бит 6: отпускание средней кнопки мыши, биты 7–15: не используются; ES = адрес сегмента обработчика; DX = смещение обработчика.

Выходные данные*:* Отсутствуют.

Драйвер мыши обращается к обработчику событий через ассемблерную команду call типа FAR, и потому обработчик событий должен заканчиваться командой RET типа EAR. Ни один из регистров процессора не должен быть возвращен в вызывающую программу с измененным содержанием. Драйвер мыши передает обработчику событий следующую информацию через регистры процессора:

– AX = маска событий. Биты этой маски соответствуют событиям, указанным в регистре CX при установке обработчика событий. Кроме того, могут быть установлены в единицу и другие биты, поскольку эта маска отражает текущее состояние драйвера мыши и не ограничивается событиями, выбранными при установке обработчика событий.

– BX = состояние кнопок мыши: бит 0 = нажата левая кнопка мыши; бит 1 = нажата правая кнопка мыши; бит 2 = нажата средняя кнопка мыши.

– CX = горизонтальная координата маркера мыши.

– DX = вертикальная координата маркера мыши.
– SI = величина последнего перемещения мыши по горизонтали.
– DI = величина последнего перемещения мыши по вертикали.
– DS = сегмент драйвера мыши.

Координаты, передаваемые в регистрах CX и DX, описывают положение элементов изображения на виртуальном экране мыши, а не физические координаты на реальном экране. Значения в регистрах SI и DI выражены в микки. Эти значения должны интерпретироваться как числа со знаком. Положительные значения указывают на перемещение в нижнюю или правую часть экрана, а отрицательное – на перемещение в верхнюю или левую часть экрана.

Функция 0fh – Задание скорости маркера. Устанавливает соотношение между микки и элементами изображения на экране. Это соотношение определяет чувствительность мыши и скорость перемещения по экрану.

Входные данные*:* AX = 000fh; CX = число микки по горизонтали; DX = число микки по вертикали.

Выходные данные*:* Отсутствуют.

Значения регистров CX и DX могут изменяться в диапазоне от 1 до 32 767. По умолчанию скорость задается равной 8 микки по горизонтали и 16 микки по вертикали. Таким образом, по горизонтали маркер двигается вдвое быстрее, чем по вертикали. Обращение к функции 00h (сброс драйвера мыши) отменяет любые установленные значения скорости и заменяет их значениями по умолчанию.

Функция 10h – Определение области исключения. Описывает любую область экрана как область исключения. При входе в область исключения маркер мыши исчезает.

Входные данные*:* AX = 0010h; CX = координата X, верхний левый угол области исключения; DX = координата Y, верхний левый угол области исключения; SI = координата X, правый нижний угол области исключения; DI = координата Y, правый нижний угол области исключения.

Выходные данные*:* Отсутствуют.

Координаты, передаваемые в регистрах CX, DX, DI и SI описывают положение элементов изображения на виртуальном экране мыши, а не физические координаты на реальном экране. Обращение к функции 00h (сброс драйвера мыши) или к функции 01h (вывод маркера мыши) отменяет координаты области исключения.

Функция 13h – Задание предельной скорости для удвоения скорости маркера. Эта функция задает предельное значение скорости мыши, при которых происходит удвоение скорости. Если скорость перемещения мыши превышает определенный предел, то драйвер мыши удваивает скорость маркера путем удвоения значения соотношения между микки и элементами изображения на экране.

Входные данные*:* AX = 0013h; DX = предельная скорость, выраженная в микки в секундах.

Выходные данные*:* Отсутствует.

1 микки = 1/200 дюйма. Чтобы предотвратить удвоение скорости мыши, можно установить более высокий предел. Скорость свыше 5000 микки в секунду достичь практически невозможно.

Функция 1ah – Задание чувствительности мыши. Определяет соотношение между физическим перемещением и перемещением маркера мыши. Определяет также максимальную скорость, при которой происходит удвоение скорости мыши.

Входные данные*:* AX = 001ah; BX = число микки по горизонтали; CX = число микки по вертикали; DX = предельная скорость для удвоения скорости мыши.

Выходные данные*:* Отсутствуют.

Значения регистров CX и DX могут изменяться от 1 до 32 767. По умолчанию устанавливается 8 микки по горизонтали и 16 микки по вертикали. Таким образом, по горизонтали маркер движется вдвое быстрее, чем по вертикали. Чтобы предотвратить удвоение скорости мыши, можно установить более высокий предел. Скорость свыше 5000 микки в секунду достичь практически невозможно. Обращение к функции 00h (сброс драйвера мыши) отменяет установленные ранее значения скорости и заменяет их значениями по умолчанию.

Функция 1bh – Определение чувствительности мыши. Возвращает параметры, установленные ранее в результате обращения к функциям 1ah, 0Fh или 13h.

Входные данные*:* AX = 001bh.

Выходные данные*:* BX = число микки по горизонтали; CX = число микки по вертикали; DX = предельное значение скорости для удвоения скорости мыши.

Функция 1ch – Задание интенсивности аппаратных прерываний мыши. Определяет частоту считывания аппаратным обеспечением мыши текущего положения мыши и состояния ее кнопок.

Входные данные*:* AX = 001ch; BX = интенсивность прерываний: бит 0: прерывание отсутствует; бит 1: 30 прерываний в секунду; бит 2: 50 прерываний в секунду; бит 3: 100 прерываний в секунду; бит 4: 200 прерываний в секунду; биты 5 – 15: не используются.

Выходные данные*:* Отсутствуют.

Эта функция может быть использована только для подключенной к порту мыши (Inport mouse). Если в регистре BX установлены в единицу несколько битов, то действует только самый младший. Разрешение мыши возрастает с увеличением интенсивности прерываний. Увеличение числа прерываний от мыши снижает скорость выполнения основной программы.

Функция 1fh – Деактивизация драйвера мыши. Переводит в неактивное состояние текущий драйвер мыши и возвращает адрес программы обработки прерывания, которая использовалась для прерывания 33h.

Входные данные*:* AX = 001fh.

Выходные данные*:* AX = код ошибки: AX = FFFFH: ошибка; AX = = 001FH: ошибка; ES = адрес сегмента использовавшегося обработчика событий; BX = смещение использовавшегося обработчика событий.

Обращение к этой функции отключает все установленные ранее активные программы обработки драйверов мыши. Исключением является программа обработки прерывания 33h, но вызывающая программа может записать в этот вектор прерывания первоначальное значение, поскольку соответствующий адрес возвращается в регистрах ES:BX.

Функция 20h – Активизация драйвера мыши. Активизирует драйвер мыши, отключенный ранее функцией 1fh.

Входные данные*:* AX = 002op.

Выходные данные*:* Отсутствуют.

Функция 21h – Сброс драйвера мыши. Инициализирует драйвер мыши, запрещает маркер мыши и установленный на данный момент обработчик событий.

Входные данные*:* AX = 0021h.

Выходные данные*:* AX = состояние ошибки: AX = FFFFh: ошибка; AX = 0021h: без ошибок; BX = число кнопок мыши.

В отличие от функции 00h эта функция не выполняет полного аппаратного сброса устройства.

Функция 24h – Определение типа мыши. Определяет тип установленной мыши и номер версии драйвера мыши.

Входные данные: AX = 0024h.

Выходные данные: BH = целая часть номера версии; BL = дробная часть номера версии; CH = тип мыши:

CH = 1: параллельная мышь;

CH = 2: последовательная мышь;

CH = 3: подключен к порту мышь;

CH = 4: мышь PS/2;

CH = 5: мышь фирмы "Хьюлетт Пакард";

CL = номер IRQ;

CL = 0:PS/2;

CL = 2,3,4,5 или 7: номер IRQ в PC.

Если номер версии драйвера равен, например, 6.24, то значение 6 возвращается в регистре BH, а 24 – в регистре BL.

Задания к выполнению лабораторной работы выдаются преподавателем непосредственно перед занятием

Примеры вариантов заданий:

1. Вывод координат мыши при нажатии левой клавиши мыши. По нажатию правой – выход из программы.

2. Рисование кривой линии с помощью мыши. Рисование осуществляется белым цветом при перемещении манипулятора «мышь» с нажатой левой клавишей мыши. При нажатой правой клавише – рисование осуществляется черным цветом (стирание). Выход – по нажатию клавиши «q».

3. Рисование прямоугольника с помощью мыши. Координаты левой верхней вершины определяются нажатием левой кнопки мыши, а правой нижней – правой кнопки.

4. Вывод координат в процессе перемещения мыши. Процесс вывода координат мыши включается нажатием левой, а выключается нажатием правой кнопки мыши. Выход – по нажатию клавиши «q».

Содержание отчета:

1. Тема и цель работы.
2. Задание для лабораторной работы.
3. Блок-схема программы.
4. Листинг программы.
5. Выводы по работе.

Контрольные вопросы:

1. Какое прерывание используется для работы с драйвером мыши на низком уровне?
2. Как получить текущие координаты мыши?
3. Как получить статус кнопок мыши?
4. Как установить обработчик событий мыши?
5. Что такое чувствительность мыши?
6. Что такое область исключения?

Приложение

Система команд модели ЭВМ

Таблица 1-А. – Команды программной модели учебной ЭВМ

Ст.? / Мл.?	0	1	2	3	4
0	NOP	JMP	–	MOV	–
1	IN	JZ	RD	RD	RDI
2	OUT	JNZ	WR	WR	–
3	IRET	JS	ADD	ADD	,ADI
4	WRRB	JNS	SUB	SUB	SBI
5	WRSP	JO	MUL	MUL	MULI
6	PUSH	JNO	DIV	DIV	DIVI
7	POP	JRNZ	–	IN	–
8	RET	INT	EI	OUT	–
9	HLT	CALL	DI	–	–

Таблица 2-А. – Типы способов адресации, их коды и обозначение

Обозначение	Код	Тип адресации	Пример команды
	0	Прямая (регистровая)	ADD 23 (ADD R3)
#	1	Непосредственная	ADD #33
@	2	Косвенная	ADD @33
[ ]	3	Относительная	ADD [33]
@R	4	Косвенно-регистровая	ADD @R3
@R+	5	Индексная с постинкрементом	ADD @R3+
-@R	6	Индексная с преддекрементом	ADD -@R3

Примечание. В таблицах 1.А –2.А приняты следующие обозначения:

– DD – данные, формируемые командой в качестве (второго) операнда: прямо или косвенно адресуемая ячейка памяти или трехразрядный непосредственный операнд;

– R* – содержимое регистра или косвенно адресуемая через регистр ячейка памяти;

– ADR* – два младших разряда ADR поля регистра CR;

– V – адрес памяти, соответствующий вектору прерывания;

– M(*) – ячейка памяти, прямо или косвенно адресуемая в команде;

– I – пятиразрядный непосредственный операнд со знаком.

Таблица 3-А. – Система команд программной модели учебной ЭВМ

КОП	Мнемокод	Название	Действие
1	2	3	4
00	NOP	Пустая операция	Нет
01	IN	Ввод	Acc ← IR
02	OUT	Вывод	OR ← Acc
03	IRET	Возврат из прерывания	FLAGS.PC ← □ (S□); IN□(S□)
04	WRRB	Загрузка RB	RB ← CR[ADR]
05	WRSP	Загрузка SP	SP ← CR[ADR]
06	PUSH	Поместить в стек	DEC(SP); M(SP) ← R

Окончание таблицы 3-А.

1	2	3	4
07	POP	Извлечь из стека	R → M(SP); INC(SP)
08	RET	Возврат	PC → M(SP); INC(SP)
09	HLT	Стоп	Конец командных циклов
10	JMP	Безусловный переход	PC ← CR[ADR]
И	JZ	Переход, если 0	if Acc = 0 then PC ← CR[ADR]
12	JNZ	Переход, если не 0	if Acc ≠ 0 then PC ← CR[ADR]
13	JS	Переход, если отрицательно	if Acc < 0 then PC ← CR[ADR]
14	JNS	Переход, если положительно	if Acc > 0 then PC ← CR[ADR]
15	JO	Переход, если переполнение	if |Acc| > 99999 then PC ← CR[ADR]
16	JNO	Переход, если нет переполнения	if |Acc| ≤ 99999 then PC ← CR[ADR]
17	JRNZ	Цикл	DEC(R); if R > 0 then PC ← CR[ADR]
18	INT	Программное прерывание	DEC(SP); M(SP) ← FLAGS.PC; PC ← M(V)
19	CALL	Вызов подпрограммы	DEC(SP); M(SP) ← PC; PC ← CR(ADR)
20	Нет	–	–
21	RD	Чтение	Acc ← DD
22	WR	Запись	M(*)← Acc
23	ADD	Сложение	Acc ← Acc + DD
24	SUB	Вычитание	Acc ← Acc − DD
25	MUL	Умножение	Acc ← Acc x DD
26	DIV	Деление	Acc ← Acc / DD
27	Нет	–	–
28	EI	Разрешить прерывание	IF ← 1
29	DI	Запретить прерывание	IF ← 0
30	MOV	Пересылка	R1 ← R2
31	RD	Чтение	Acc ← R*
32	WR	Запись	R* ← Acc
33	ADD	Сложение	Acc ← Acc + R*
34	SUB	Вычитание	Acc ← Acc − R*
35	MUL	Умножение	Acc ← Acc x R*
36	DIV	Деление	Acc ← Acc / R*
37	IN	Ввод	Acc ← ВУ(CR[ADR*])
38	OUT	Вывод	ВУ(CR[ADR*]) ← Acc
39	Нет	–	–
40	Нет	–	–
41	RDI	Чтение	Acc <-1
42	Нет	–	–
43	ADI	Сложение	Acc ← Acc + I
44	SBI	Вычитание	Acc ← Acc −1
45	MULI	Умножение	Acc ← Acc x I
46	DIVI	Деление	Acc ← Acc / 1

МЕТОДИКА ВЫПОЛНЕНИЯ КУРСОВОЙ РАБОТЫ

1. Требования к оформлению курсой работы

Текстовая часть курсовой работы оформляется на листах формата А4 в соответствии с ГОСТ 2.105-95 ЕСКД «Общие требования к текстовым документам» и представляется сброшюрованной в папке-скоросшивателе.

Порядок брошюровки следующий:

1) титульный лист, номер страницы не указывается;

2) задание на курсовую работу, оформленное на стандартном бланке, номер страницы не указывается;

3) содержание (оглавление) – оформляется на листе со стандартной рамкой и основной надписью по форме для текстовых документов в соответствии с ГОСТ 2.104, номер страницы – 3-й (на всех последующих листах – нумерация страниц по порядку);

4) введение – страницы не нумеруются, оформляется на листах со стандартной рамкой (весь последующий текстовый материал также);

5) основная часть, соответствующая заданию курсовой работы;

6) заключение – страницы не нумеруются;

7) список использованной литературы – страницы не нумеруется;

Графический материал оформляется в виде приложений.

Объем пояснительной записки составляет ориентировочно 30 – 40 страниц формата А4. Шрифт записки – «Times New Roman», 14, через 1,5 интервала.

Графическая часть выполняется на листах формата А3. Использование скриншотов не допускается.

2. Критерии оценки курсовой работы

При приеме и проверке работы преподаватель-руководитель в целом оценивает, насколько ее автор:

– в полном объеме справился с разработкой темы;

– самостоятельно выполнил схемы в приложении;

– последовательно, логически связанно построил изложение;

– продемонстрировал владение содержанием работы, доказал умение аргументировано отстаивать свою точку зрения;

– изучил достаточное количество теоретических материалов, включая новейшие источники;

– написал работу грамотно, техническим языком, придерживаясь научного стиля изложения;

– правильно, аккуратно оформил работу в соответствии с требованиями.

Работа не принимается (возвращается на доработку) если:

– студент не проявил самостоятельности при подготовке работы, работа свелась к простому копированию доступных источников или, что недопустимо – дублированию работы других студентов;

– приложения № 1 или № 2 выполнены не самостоятельно, а использована вставка рисунка, скриншота или фотографии, найденных в интернете схем;

– отсутствуют подписи студента на чертежах и титульном листе;

– не полностью выполнено задание на курсовую работу (пропущены отдельные пункты задания) или допущены грубые ошибки;

– содержание работы не соответствует заданию;

– отсутствует титульный лист;

– отсутствует задание на курсовую работу;

– не оформлено содержание курсовой работы с указанием листов;

– отсутствует папка-скоросшиватель;

– во время защиты студент не смог отстоять основные результаты работы, не проявил должного уровня владения темой.

Студент, не сдавший или не защитивший курсовую работу, не допускается к экзамену по данной дисциплине.

Курсовая работа в виде отпечатанного материала, сброшюрованная и подписанная студентом, сдается на кафедру энергетики и электроники (секретарю кафедры, ауд. 416н).

Для проверки работы в процессе выполнения, текст работы можно высылать руководителю работы по адресу его электронной почты.

3. Исходные данные курсовой работы

Исходными данными являются: фирма-производитель и тип системной платы, функциональное устройство, имеющееся на системной плате, и используемое программное обеспечение для него, которые повариантно выбираются из таблицы. Номер варианта определяется порядковым номером фамилии студента в списке журнала группы.

Таблица. – Исходные данные курсовой работы

№ варта	*Фирма-производитель системной платы	*Тип системной платы	Заданное устройство на системной плате	Программы для обслуживания устройства
1	2	3	4	5
1	ASUS	Z9PR-D12	Контроллер VGA	функции BIOS
2	ASRock	B75M-DGS	Контроллер USB +Flash	функции Win32 API
3	MSI	H87M-P32	Контроллер USB +Gamepad	функции Win32 API
4	GigaByte	GA-Z87P-D3	интервальный таймер	функции BIOS

Окончание таблицы.

1	2	3	4	5
5	Intel	DH61CRB3	Контроллер HDD SATA	функции Win32 API
6	ASRock	Q87M Vpro	Контроллер VGA	функции BIOS
7	ASRock	Z77E-ITX	Контроллер Lan Ethernet	функции Win32 API
8	ASUS	P8B-C/4L	Контроллер Lpt	функции BIOS
9	ASUS	P9X79-E WS	Контроллер USB +Mause	функции Win32 API
10	GigaBite	G1.Sniper M5	Контроллер AC'97 Audio	функции Win32 API
11	ASRock	Z87 Extreme4	Контроллер VGA	функции BIOS
12	GigaByte	GA-Z87X-UD3H	Контроллер HDD IDE	функции BIOS
13	MSI	Z87-G3	Контроллер VGA	функции BIOS
14	ASUS	Z87-K	Контроллер клавиатуры	функции BIOS
15	ASUS	P9D-V	Контроллер прерываний	функции BIOS
16	ASUS	P8Z77-V LX	Контроллер RS232C	функции BIOS
17	ASRock	G41M-VS3	Контроллер CMOS BIOS	функции BIOS
18	ASUS	Z8PE-D12x	Контроллер VGA	функции BIOS
19	Intel	DH77EB	Контроллер AC'97 Audio	функции Win32 API
20	ASRock	FM2A55M-VG3	Контроллер FDD	функции BIOS
21	Intel	S5500BCR	Контроллер мыши PS/2	функции BIOS
22	Intel	S3420GPV	Контроллер RS232C	функции BIOS
23	Intel	DBS1200BTSR	Контроллер прерываний	функции BIOS
24	BioStar	A57A	Контроллер Lan Ethernet	функции Win32 API
25	BioStar	G41D3C	Контроллер HD Audio	функции Win32 API
26	BioStar	TZ77XE4	интервальный таймер	функции BIOS
27	MSI	970A-G46	Контроллер клавиатуры	функции BIOS
28	EliteGroup	A960M-M3	Контроллер RAID	функции BIOS
29	EliteGroup	X77H2-A3	Контроллер SCASI	функции BIOS
30	EliteGroup	G31T-M9	Контроллер шины PCI	функции BIOS

Примечание. *– фирма-производитель системной платы и ее тип уточняются (актуализируются) при выборе задания студентом. Данные выбираются из прилагаемого списка.

4. Содержание расчетно-пояснительной записки

Введение. Указывается назначение устройства из состава архитектуры вычислительных систем и шины связи с устройствами ввода-вывода ПЭВМ, для которых разрабатывается интерфейс.

Раздел 1. Обзор системной (материнской) платы (МП): описание центрального процессора (ЦП), схемы оперативного запоминающего устройства (ОЗУ), используемой шинной архитектуры и установленных устройств ввода-вывода, анализ ресурсов, требующихся для их работы (используемые прерывания, доступное адресное пространство, порты доступа и т.д.).

Раздел 2. Исследование принципиальной схемы заданного устройства и его функциональной структуры, а также сопряжения с используемой шиной для связи с ЦП.

Раздел 3. Описание протоколов обмена данными исследуемого устройства через шину сопряжения с ЦП ПЭВМ или через систему ввода-вывода с подключаемыми внешними устройствами (ВУ).

Раздел 4. Рассмотрение основных программ или функций BIOS для обеспечения интерфейса связи с заданным устройством.

Заключение. Описываются перспективы использования и развития интерфейса данного типа.

Приложения: 1) общая схема используемой системной (материнской) платы; 2) функциональная схема заданного устройства.

5. Общие рекомендации по разработке разделов расчетно-пояснительной записки

При формировании содержания пояснительной записки следует ориентироваться прежде всего на обеспечение общей логики построения материала курсовой работы, принимая за основу структуру содержания, приведенную в задании. Некоторые разделы пояснительной записки можно расширять, вводить подразделы, но полностью исключать разделы не разрешается. Конечное содержание пояснительной записки должно быть согласовано с руководителем курсовой работы в процессе ее выполнения.

Прежде всего, необходимо выполнить поиск, накопление, проработку и систематизацию исходного материала по теме курсовой работы.

Во **Введении** необходимо кратко сформулировать цель и основные задачи разрабатываемого устройства из состава архитектуры ВС.

5.1. Рекомендации по формированию раздела 1

Общая техническая характеристика системной платы для заданной модели берется из технического руководства или данных сайта производителя (электронные адреса некоторых производителей системных плат приведены в Списке фирм-производителей материнских плат).

Студент самостоятельно разрабатывает и рисует структурную схему системной платы, пользуясь руководством по материнской плате или ее техническими характеристиками. Все поясняющие надписи на схеме даются *только на русском языке*.

Дополнительно разрешается приводить фотографии системной платы «в плане» и со стороны разъемов ввода-вывода.

В разделе приводятся конструктивные параметры слота для установки центрального процессора. Определяется наличие мультипроцессорной системы. Рассматривается линейка процессоров, совместимых с заданной моделью системной платы, а также границы повышения производительности центрального процессора (оверклокинга), если он доступен в SETUP.

Приводятся характеристики ОЗУ и КЭШ-памяти, установленной на материнской плате, кратко излагается принцип её работы. Указывается тип

используемой памяти (SD RAM, DDR, DDR2 и т.д.) и предельные технические характеристики (объем, напряжение питания и т.д.) Дается характеристика системы двухканального доступа. Приводится порядок настройки памяти, если он доступен в SE1TUP.

Рассматриваются варианты внутренних шин, установленных на системной плате (ISA, AGP, PCI, PCI-E и других). Указывается разрядность шины, адресное пространство, частота, пропускная способность, мультиплексирование.

Приводится краткое описание интерфейсов ввода-вывода: RS232C, паллельный порт LPT, USB, FDD, ATA IDE, SATA (поддерживаемые типы RAID-массивов), PS/2, InfraRed, оптический выход, DVI, HDMI и т.д., наличие и характеристики встроенных устройств (видеоадаптера, сетевого адаптера, звукового адаптера и т.д.), блоки питания и возможности управления электропитанием ASPI.

Графическое изображение структурной (слотовой) схемы системной платы с необходимыми пояснениями выносится в приложение № 1.

5.2. Рекомендации по формированию раздела 2

Исследование заданного устройства из состава архитектуры вычислительной системы или его разработка должна начинаться с построения его функциональной схемы. Приэтом разработчик должен понимать, что все функции разрабатываемого устройства – это лишь различные составляющие общей функциональной структуры сложной вычислительной системы, характеризующиеся одним из многочисленных способов передачи информации от периферийных устройств к центральному процессору через систему ввода-вывода ПЭВМ.

Каждое из заданных устройств имеет в своем составе устройство управления, представляющее микропрограммный автомат, реализующий различные варианты асинхронного обмена цифровой информацией. Используемая система ввода-вывода ПЭВМ обладает рядом принципиальных особенностей, которые должно поддерживать заданное устройство и его интерфейс. Эти принципиальные особенности необходимо изложить в данном разделе и показать их реализацию на принципиальной схеме устройства:

– Каждое устройство имеет схему буферизации и схему подключения к внешней и (или) внутренней шине.

– Каждое используемое устройство ввода-вывода в обязательном порядке обладает адресом порта ввода-вывода данных.

– Каждое устройство ввода-вывода подключено к контроллеру прерываний и имеет свой постоянный или присваиваемый BIOS номер аппаратного прерывания.

– Некоторые устройства ввода-вывода (наиболее современные из них) подключены к устройствам арбитража и используют схемы транзакций, поддерживаемые соответствующими драйверами современных операционных систем (например, контроллеры SATA).

– Некоторые устройства используют возможности прямого доступа к памяти и подключены к контроллеру прямого доступа к оперативной памяти ПЭВМ, в связи с чем располагают доступным адресным пространством, а также системой выбора каналов прямого доступа к нему (например, контроллеры FDD).

Само заданное устройство, в зависимости от его функциональной структуры, может поддерживать все возможности, предоставляемые ему шиной и соответствующим устройством ввода-вывода ПЭВМ, а может использовать только некоторые из них, что также необходимо отразить в пояснительной записке.

Интерфейс устройства, кроме стандартной шины, подключаемой к системе ввода-вывода ПЭВМ, может содержать промежуточные схемы и (или) шины передачи информации. Например, использовать радиоканал, инфракрасный луч света или систему усиления сигнала типа «токовая петля» (RS-485). Описание дополнительных схем передачи информации и принципов их работы также необходимо привести в пояснительной записке.

Безусловно, разработка устройства сопряжения потребует навыков и знаний проектирования системотехнической аппаратуры, что не является непременным атрибутом данной работы (см. декларируемые цели проекта). Поэтому принципиальная схема заданного устройства может быть представлена в общем схемном решении (без представления электрической схемы устройства). Принципиальная схема устройства может содержать:

– блок микропрограммного управления;
– блок генератора синхросигнала;
– блок буферизации данных;
– блок организации асинхронного обмена;
– блок присвоения и селектирования (распознавания) адресов;
– блок присвоения и использования номера прерывания;
– блок организации прямого доступа к памяти;
– другие блоки, определяемые спецификой заданного устройства.

Устройство может быть спроектировано для нескольких шин.

Так, например, простой манипулятор мышь может быть подсоединен к ПЭВМ с использованием нескольких типов интерфейсов: RS-232C, PS/2, USB, InfraRED, радиочастотного и т.д. При этом основная конструкция манипулятора остается без изменений, меняется только функциональная структура устройства связи и программное обеспечение приема информации, поскольку меняются порты доступа и протоколы обращения к ним.

Графическое изображение функциональной схемы заданного устройства выносится в приложение № 2.

5.3. Рекомендации по формированию раздела 3

Схема передачи данных, представленных в цифровом виде (все предлагаемые к разработке интерфейсы содержат такие схемы), предполагает наличие шины и протокола передачи данных по ним.

Итак, информация от разрабатываемого устройства через устройство сопряжения передается на шину. Устройство сопряжения может входить в состав периферийного устройства, либо контроллера шины, а может быть оформлено в виде самостоятельного устройства. В любом случае, оно является неотъемлемой (аппаратной) частью разрабатываемого устройства.

Шина со стороны центрального процессора обязательно должна быть подключена к одному из многочисленных устройств системы ввода-вывода ПЭВМ. Другого способа подключения устройств к центральному процессору просто не существует. Протокол передачи данных описывает механизмы синхронизации источника и приемника данных, а также формат пакетов передаваемых данных и порядок контроля сохранности информации в пакетах. Шина или интерфейс не могут существовать сами по себе – все они служат для подключения любых устройств сопряжения, предназначенных для ввода или вывода информации пользователем этих устройств, при этом:

– Шина, по которой передаются данные в систему ввода-вывода, может быть последовательной и иметь в своем составе систему мультиплексор-демультиплексор, либо быть параллельной.

– Шина может поддерживать синхронный или асинхронный режим передачи данных.

– Шина может содержать ряд сигналов оповещения и управления периферийными устройствами.

– Устройства поддержки шины (устройства сопряжения) могут содержать системы присвоения номеров подключаемых периферийных устройств (энумераторы шин) для их последующего распознавания или системы маршрутизации информационных потоков, поддерживаемые соответствующими программами-драйверами операционных систем.

В курсовом проекте должны быть достаточно четко систематизированы и представлены протоколы обмена (передачи данных) в виде временных диаграмм с описанием всех сигналов оповещения и управления вне зависимости от того, является шина мультиплексированной или нет.

Раздел должен содержать описание используемых протоколов обмена данными по шинам, а также описание сигналов оповещения и управления.

5.4. Рекомендации по формированию раздела 4

Разработка прикладной программы, обеспечивающей работу заданного устройства не является целью настоящей работы, поэтому нет необходимости приводить полный листинг прикладной программы в тексте расчетно-пояснительной записки, достаточно привести лишь наиболее значимые фрагменты.

Все процессы ввода-вывода информации в ПЭВМ сопровождаются (организуются) программами-драйверами, поддерживающими типовые протоколы обмена. Программы-драйверы могут идти в комплекте программного обеспечения операционной системы, либо быть разработанными самостоятельно с использованием соответствующего инструментария формирования модулей для конкретных операционных систем (например, DDK – Device Development Kit для операционных систем Windows).

Прикладная программа использует массивы данных, формируемые драйверами обмена, в заданном адресном пространстве с заданной периодичностью либо помещает туда данные для передачи. При наличии стандартного программного обеспечения (например, библиотек Win32 API или функций BIOS (DOS), можно ограничиться описанием их назначения и приведением в тексте записки примеров их использования для организации работы исследуемого устройства.

Список фирм-производителей материнских плат:

1. Системные платы фирмы ASRock:
URL: http://www.asrock.com/support/CPU.ru.asp.
URL: http://www.asrock.com/mb/index.ru.asp?s=n.
2. Системные платы фирмы ASUSTeK (ASUS):
URL: http://support.asus.com/cpusupport/cpusupport.aspx.
3. Системные платы фирмы Biostar:
URL: http://www.biostar.com.tw/app/en/mb/index.php
4. Системные платы фирмы EliteGroup (ECS)
URL: http://eu.ecs.com.tw/ECSWebSite/Support/Support_CPU_List.aspx?CategoryID=1&MenuID=52&LanID=6.
5. Системные платы фирмы Gigabyte (Giga-byte):
URL: hhttp://www.gigabyte.ru/products/mb/.
6. Системные платы фирмы Intel:
URL: http://processormatch.intel.com/CompDB/.
7. Системные платы фирмы SuperMicro:
URL: http://www.supermicro.com/products/motherboard/ (для процессоров Intel). В меню "Motherboards" слева выберите тип процессора.
URL: http://www.supermicro.com/Aplus/motherboard/ (для процессоров AMD). В меню "A+ Motherboards" слева выберите тип процессора.
8. Системные платы фирмы Micro-Star (MSI, MicroStar):
URL: http://www.microstar.ru/program/products/mainboard/mbd/prombdcpu_support.php?kind= 1&CHIP=Socket%20A&ID=2.
9. Системные платы фирмы TYAN:
URL: http://www.tyan.com/support_download_cpu.aspx.
10. Системные платы фирмы Foxconn:
URL: http://www.foxconnchannel.com/product/Motherboards/compatibility_ detail.aspx?model_id=&type_id=en-us0000001.
11. Системные платы фирмы Epox:
URL: http://www.epox.ru/data/download/cpu/epox-cpu-support.zip.
12. Системные платы фирмы Elpina:
URL: http://www.pcchips.com.tw/PCCWebSite/Support/CPU_Support.aspx? CategoryID=1&MenuID=69&LanID=0&ln=6.

ВОПРОСЫ ДЛЯ ПОДГОТОВКИ К ЭКЗАМЕНУ

1. Основные определения информации. Определение и схема цифрового автомата. Такт, тактовый интервал.

2. Понятие алгоритма. Принципы Джона фон Неймана построения ЭВМ. Общее и Неймановское определение ЭВМ.

3. Блочная базовая схема ЭВМ. Описание. Неймановская (Принстонская) и Гарвардская архитектуры ЭВМ. Основные отличия.

4. Физический носитель нуля и единицы. VT-диаграмма. Описание зон диаграммы.

5. Двоичное кодирование простых чисел (основные форматы, пределы использования). Смещенный двоичный код. Схема нормализованного 2-х байтового представления двоичного вещественного числа с плавающей запятой.

6. Кодирование символов (принцип, примеры). Основные таблицы символов. Схема преобразования кодов кириллицы из таблицы ANSI в ASCII.

7. Общие понятия об управляющем автомате с «жесткой» логикой и автомате с программой в памяти. Определения. Схемы. Способы адресации микрокоманд.

8. Методы низкоуровневого микропрограммирования. Схемы. Достоинства и недостатки.

9. Проектирование и программирование микропрограммного автомата. Концепция и основные этапы.

10. Структурная схема микропрограммного автомата. Порядок построения микропрограммы для него.

11. Командный цикл центрального процессора. Определения. Основные этапы. Схема.

12. Базовая схема центрального микропроцессора. Назначение регистров и сигналов.

13. Программная регистровая модель ПЭВМ. Шесть групп программно доступных регистров процессора и сопроцессоров.

14. Схема и назначение основных шестнадцати регистров центрального процессора. Схема регистра флагов.

15. Характеристики систем машинных команд. Способы адресации операндов. Классификация машинных команд.

16. Общий формат машинной команды в объектных кодах. Схема построения машинного кода.

17. Таблицы кодов операций (основная и дополнительные). Порядок работы с кодами. Программы-отладчики.

18. Расширение базовой архитектуры ПЭВМ. Математический сопроцессор, процессора SSE. Регистровая схема. Системы команд ассемблера. Эволюция архитектуры.

19. Классификация команд ассемблера: основные для базовой архитектуры и дополнительные системы команд для расширений архитектуры (матсопроцессора, 3DNow!, MMX и XMM).

20. Организация оперативной памяти ПЭВМ. Концепция многоуровневой памяти. Определения. Схема.

21. Четыре режима адресации памяти центрального процессора. Характеристики. Схема.

22. Схема сегментной адресации памяти в режиме прямой адресации (плоская модель памяти).

23. Схема сегментного распределения памяти в защищенном режиме. Содержание дескриптора сегмента.

24. Схема логического распределения памяти по адресам 00000h – FFFFFh (плоская память режима 8086).

25. Схема формирования эффективного, линейного и физического адресов в режиме прямой адресации.

26. Схема формирования эффективного, линейного и физического адресов в режиме защищенных адресов.

27. Защищенный режим адресации. Назначение системных регистров. Схема уровней привилегий, защита.

28. Адресуемая память (схема). Способы адресации операндов в машинной команде.

29. Ассоциативная и стековая память (схемы). Принцип работы. Область использования.

30. Типы памяти (классификация). Контроллер ОЗУ (схема и основные сигналы управления).

31. Понятие шины и магистрали, состав шины. Характеристики и этапы развития шины PCI.

32. Схема наследуемой шинной архитектуры XT. Основные типы современных шинных архитектур.

33. Формирования шинного интерфейса для внешних устройств. Схема. Порядок работы.

34. Мостовая схема буферизации и изменение формата данных. Задачи буферизации данных.

35. Основные задачи прерывания выполнения программы. Общая схема механизма прерывания программы в режиме прямой адресации. Порядок восстановления прерванной программы. Типы прерываний.

36. Схема обработки прерывания в защищенном режиме. Исключения и прерывания. Типы исключений.

37. Таблица векторов прерываний. Содержание стека в защищенном режиме. Состав дескриптора IDT.

38. Схема контроллера прерываний. Назначение основных регистров. Порядок программирования.

39. Четыре режима формирования приоритетов ПКП, два режима завершения прерываний ПКП.

40. Контроллер прямого доступа к памяти. Назначение. Основные задачи. Принципы работы.

41. Общая функциональная схема реализации контроллера ПДП. Порядок её работы.

42. Основные сигналы контроллера ПДП i8327A. Порты доступа. Порядок программирования.

43. Четыре режима работы контроллера ПДП i8327A. Основные типы передачи информации.

44. Системный интервальный таймер 8254. Схема, назначение каналов, сигналы и функционирование.

45. Режимы использования каналов интервального таймера. Основные характеристики режимов.

46. Схемы и конкретные режимы использования каналов 0, 1 и 2 системного интервального таймера.

47. Часы реального времени. Порты доступа и регистры часов. Структурная схема и функционирование.

48. Частота генератора часов. Формат BCD и схема его использования в ПЭВМ. Константы CMOS SETUP.

49. Схема и назначение системы управления энергопитанием ПЭВМ. Интерфейс системы.

50. Основные таблицы переменных ACPI. Порядок доступа к переменным. Конфигурирование системы.

51. Система прерываний, используемая ACPI. Схема. Порядок работы.

52. Контроллер клавиатуры. Структурная схема, функционирование и порты доступа.

53. Обработка прерывания от клавиатуры. Схема.

54. Схема процедуры обработки скэн-кода. Адрес и содержание констант клавиш-переключателей клавиатуры.

55. Схема буфера клавиатуры. Содержание байтов буфера. Порядок программирования буфера клавиатуры.

56. Основные и дополнительные коды клавиатуры. Escape-последовательности. Alt-ввод.

57. Архитектура дисковой подсистемы ПЭВМ (основные термины).

58. Структура файловой системы MS DOS (FAT-32) размещения информации на магнитном диске (схема).

59. Состав MBR, BR, Root и FAT (Общие понятия).

60. Структура файловой системы NTFS. Схема взаимодействия с компонентами ОС Windows NT.

61. RAID-массивы. Схемы вариантов, назначение вариантов, области использования.

62. Контроллер НГМД 8272. Схема. Регистры. Система команд. Значения основных констант.

63. Контроллер НЖМД. Схема. Регистры контроллера. Характеристики интерфейсов связи.

64. Методы кодирования информации на магнитных дисках (диаграммы). Интерлинг и предкомпенсация.

65. Основные типы современных накопителей информации и их характеристики (объем, скорость обмена).

66. Видеоконтроллер EGA/VGA. Схема. Назначение отдельных блоков и их функционирование.

67. Видеоконтроллер EGA/VGA. Основные режимы использования. Регистры. Порядок программирования.

68. Страничная организация экранной памяти (схема). Области ПЗУ ЭВМ для обмена видеоданными.

69. Состав байта-атрибута символа в текстовом режиме. Палетты – виды, состав и адреса доступа.

70. Пикселы. Порядок программирования видеоизображения. Понятие о 3D, Direct X.

71. Параллельный интерфейс Centronics. Схема контроллера. Сигналы. Диаграмма протокола передачи.

72. Последовательный интерфейс RS232C. Схема контроллера. Сигналы. Формат передачи данных.

73. Подсистема ввода/вывода BIOS. Задачи. Схема процедуры POST. Область адресов портов. Порядок использования.

74. Доступ к переменным и константам BIOS. Структура системного редактора Setup. Схема распределения ресурсов ВС.

75. Основные отличия современных подсистем ввода/вывода BIOS EFI (UEFI). Новый порядок распределения ресурсов и GPT-разметка диска.

76. Система PnP автоопределения внешних устройств ПЭВМ. Задачи. Принципы построения.

77. Основные принципы построения современных системы P&P. Основные компоненты и язык программирования.

78. Операционные системы ПЭВМ. Основные понятия. Состав ОС. Схема ОС Windows NT.

79. Основные модули ОС Windows NT/2000/XP. Порядок загрузки ОС Windows NT/2000/XP через MBR и BIOS UEFI.

80. Аппаратная зависимость ОС и способы ее преодоления. Порядок загрузки и конфигурирования ОС. Файлы конфигурации и реестр ОС.

81. Тенденции и перспективы развития современных архитектур. Многоядерные и многопроцессорные ПЭВМ. Дальнейшее расширение системы plug & play и BIOS.

82. Минимизация ПЭВМ: ПЭВМ, ноутбуки, Netбуки и планшеты. Трансиверы, медиаплееры и медиацентры. Общие понятия о системах barebon ПЭВМ. Перспективы развития.

СОДЕРЖАНИЕ

Учебное издание

ПИТОЛИН Владимир Евгеньевич
СУРТО Сергей Геннадьевич

**АРХИТЕКТУРА ПЭВМ
И ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ**

*Рекомендовано УМО по образованию в области информатики
и радиоэлектроники в качестве пособия для специальностей
1-40 02 01 «Вычислительные машины, системы и сети»,
1-36 04 02 «Промышленная электроника»*

Редактор: *И. Н. Чапкевич*

Подписано в печать 08.11.22. Формат 60×84 1/16. Бумага офсетная 70 г/м^2.
Цифровая печать. Усл. печ. л. 16,62. Уч.-изд. л. 19,59. Тираж 30 экз. Заказ 660.

Издатель и полиграфическое исполнение –
учреждение образования «Полоцкий государственный университет
имени Евфросинии Полоцкой».

Свидетельство о государственной регистрации
издателя, изготовителя, распространителя печатных изданий
№ 1/305 от 22.04.2014, перерегистрация от 24.08.2022.

ЛП № 02330/278 от 27.05.2004.

Ул. Блохина, 29, 211440, г. Новополоцк.