МИНОБРНАУКИ РОССИИ

Государственное образовательное учреждение

высшего профессионального образования

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

Кафедра математического обеспечение и применения ЭВМ

В. А. КИРЬЯНЧИКОВ

Доцент кафедры МОЭВМ

**Архитектура компьютера**

Учебное пособие

Санкт-Петербург

2016

Оглавление

[1 Основные понятия архитектуры и организации компьютеров. 4](#_Toc479439742)

[1.1. Состав компьютера 4](#_Toc479439743)

[1.2. Виды (классы) компьютеров 4](#_Toc479439744)

[1.3. Принцип программного управления и машина фон Неймана 5](#_Toc479439745)

[1.4. Понятия архитектуры, организации и реализации компьютера 5](#_Toc479439746)

[1.5. Многоуровневая организация компьютера 6](#_Toc479439747)

[1.6. Понятие семантического разрыва между уровнями 6](#_Toc479439748)

[1.7. Организация аппаратных средств ВМ 7](#_Toc479439749)

[2 Представление и обработка данных в ВМ 9](#_Toc479439750)

[2.1. Целые числа……………………………………………………………………………………………………...9](#_Toc479439751)

[2.2. Представление и обработка вещественных чисел…………………………………………………………...10](#_Toc479439752)

[2.3. Логические операции над битовыми наборами……………………………………………………………...11](#_Toc479439753)

[2.4. Представление и обработка символов………………………………………………………………………...12](#_Toc479439754)

[2.5. Представление видеоинформации и аудиоинформации……………………………………………………..12](#_Toc479439755)

[3 Организация процессора и основной памяти ВМ 13](#_Toc479439756)

[3.1. Типовая структура процессора и основной памяти 13](#_Toc479439757)

[3.2. Основной цикл работы процессора 14](#_Toc479439758)

[3.3. Организация процессора и памяти в архитекуре Intel X86*.* 15](#_Toc479439759)

[3.3.1. Программно доступные регистры процессора 15](#_Toc479439760)

[3.3.2. Организация стека в архитектуре Intel Х86 16](#_Toc479439761)

[3.3.3. Организация выполняемых программ в MS DOS 16](#_Toc479439762)

[3.3.4. Режимы адресации памяти в архитектуре Intel Х86 17](#_Toc479439763)

[3.3.5. Краткая характеристика системы команд процессоров Intel Х86 19](#_Toc479439764)

[3.3.6 .Арифметическая обработка чисел с использованием математического сопроцессора 19](#_Toc479439765)

[3.3.7. Организация прерываний в процессорах Intel X86 21](#_Toc479439766)

[3.3.8. Эволюция микроархитектуры Intel X86 24](#_Toc479439767)

[4. Управление выполнением команд в компьютерах. 25](#_Toc479439768)

[4.1. Аппаратный способ формирования управляющих сигналов……………………………………………….26](#_Toc479439769)

[4.2. Микропрограммный способ формирования управляющих сигналов……………………………………...27](#_Toc479439770)

[5. Компьютеры с сокращенным набором команд 30](#_Toc479439771)

[6. Организация памяти в компьютере 34](#_Toc479439772)

[6.1. Назначение и основные характеристики памяти 34](#_Toc479439773)

[6.2. Основные среды хранения информации 34](#_Toc479439774)

[6.3. Методы доступа к данным. 35](#_Toc479439775)

[6.4. Память с произвольным доступом (ППД) 35](#_Toc479439776)

[6.5. Блочная организация основной памяти………………………………………………………………………36](#_Toc479439777)

[6.6. Постоянные запоминающие устройства (ПЗУ - ROM)……………………………………………………...37](#_Toc479439778)

[6.7. Ассоциативные запоминающие устройства (АЗУ)………………………………………………………….37](#_Toc479439779)

[6.8. Иерархическая система памяти 38](#_Toc479439780)

[6.9. Организация кэш-памяти……………………………………………………………………………………...39](#_Toc479439781)

[6.10. Организация виртуальной памяти 42](#_Toc479439782)

[6.11. Организация виртуальной памяти в Intel 80386 и более старших моделях. 44](#_Toc479439783)

[6.12. Защита памяти в процессоре Intel 80386 46](#_Toc479439784)

[*7.* Организация работы с внешней памятью 47](#_Toc479439785)

[7.1. Типы, виды, свойства дисковых накопителей информации…………………………………………………47](#_Toc479439786)

[7.2. Магнитные дисковые накопители……………………………………………………………………………..47](#_Toc479439787)

[7.3. Основные физические и логические параметры ЖМД……………………………………………………...48](#_Toc479439788)

[7.4. Контроллеры жестких дисков………………………………………………………………………………….49](#_Toc479439789)

[7.5. Логическое хранение и кодирование информации…………………………………………………………..49](#_Toc479439790)

[7.6. Интерфейсы жестких дисков…………………………………………………………………………………..51](#_Toc479439791)

[7.7. Работа накопителя………………………………………………………………………………………………51](#_Toc479439792)

[7.8. Внешняя память на CD и DVD дисках……………………………………………………………………….52](#_Toc479439793)

[8. Принципы организации RAID массивов………………………………………………………………54](#_Toc479439794)

[8.1. Основные принципы построения RAID массивов…………………………………………………………..54](#_Toc479439795)

[8.2. Одиночные уровни RAID……………………………………………………………………………………...55](#_Toc479439796)

[8.3. Составные уровни RAID массивов……………………………………………………………………………59](#_Toc479439797)

[8.4. Сравнительные результаты 61](#_Toc479439798)

[9. Системные и локальные шины 62](#_Toc479439799)

[9.1. Общие положения и требования к шинам 62](#_Toc479439800)

[9.2. Основные виды, характеристики и параметры шин………………………………………………………….63](#_Toc479439801)

[9.3. Стандарты шин 65](#_Toc479439802)

[10. Организация системы ввода-вывода в ВМ 69](#_Toc479439803)

[10.1. Назначение и основные требования к системе ввода-вывода ВМ 69](#_Toc479439804)

[10.2. Архитектура систем ввода-вывода 69](#_Toc479439805)

[10.3. Способы выполнения операции передачи данных 71](#_Toc479439806)

[10.4. Структуры контроллеров ВУ для различных режимов передачи данных 73](#_Toc479439807)

[11. Программные средства управления вводом-выводом (ПС УВВ) 75](#_Toc479439808)

[11.1. Состав ПС УВВ 75](#_Toc479439809)

[11.2. Основные компоненты процедуры управления ввода-вывода общего вида 76](#_Toc479439810)

[11.3 Состав и реализация устанавливаемого драйвера символьного типа 76](#_Toc479439811)

[Список литературы 80](#_Toc479439812)

[ПРИЛОЖЕНИЯ 81](#_Toc479439813)

# Основные понятия архитектуры и организации компьютеров.

# *1.1. Состав компьютера*

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



Процессор – основное устройство управления и обработки данных. Память – набор устройств для хранения информации, используемой в процессе работы машины. Внешние устройства обеспечивают взаимодействие машины с внешней средой (в том числе с пользователями). Шины связи и протоколы взаимодействия реализуют физическую среду и алгоритмы обмена данными между различными устройствами.

Операционные системы - являются основными средствами управления выполнением программ и распределения ресурсов машины между процессами. Инструментальные ПС (программные среды) включают все средства, необходимые для разработки программ: редакторы, компиляторы, отладчики и др. Системы промежуточного ПО (Middleware) – ИТС (инструментальные технологические среды) – позволяют реализовать определенную технологию разработки программных средств: COM, DCOM, CORBA, RMI и др. Проблемно-ориентированные ПС предназначены для решения задач определенной области применения: MathLab (научные вычисления), AutoCAD (конструирование), PhotoShop (графический редактор), 3D Studio (машинная графика) и др. Утилиты – сервисные средства, облегчающие взаимодействия пользователя с компьютером: архиваторы, программы форматирования, антивирусные программы и др.

# *1.2. Виды (классы) компьютеров*

В настоящее время распространены следующие классы компьютеров:

- микрокомпьютеры – встраиваемые микропроцессоры со специальным ПО, используемые как программируемые контроллеры для промышленного оборудования (Embedded Computer); с середины 1990-х г.г. широкое распространение получили процессоры архитектуры ARM (Advanced RISC Machines), используемые в [смартфонах](http://ru.wikipedia.org/wiki/%D0%A1%D0%BC%D0%B0%D1%80%D1%82%D1%84%D0%BE%D0%BD),  [интернет-планшетах](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%BF%D0%BB%D0%B0%D0%BD%D1%88%D0%B5%D1%82) и других мобильных и энергоэффективных устройствах;

- персональные компьютеры (на основе процессоров Pentium, многоядерных процессоров Core Duo, Core i3, i5 и i7) – ВМ, предназначенные для работы одного пользователя;

- рабочие станции (Sun Work Stations) – ВМ большей производительности, чем ПК; имеют проблемную ориентацию или управляют несколькими ПК, объединенными в простую (локальную) сеть;

- средние и большие машины (Host computers) – вычислительные системы из нескольких процессоров (CDC6600, HP9000, Series800, SGI); это системы, которые предназначены для решения сложных задач обработки данных или управления большим числом ПК, объединенных в глобальную сеть;

- суперкомпьютеры – системы с параллельной архитектурой векторно-матричного типа ( Illiac IV, VP-2000, Эльбрус ), имеющие сверхвысокую производительность обработки данных.

# *1.3. Принцип программного управления и машина фон Неймана*

Фон Нейман впервые предложил в 40-х годах XX века концепцию хранимой программы, основные принципы которой заключаются в следующем:

1. ***Двоичное кодирование:*** вся информация (как команды, так и данные) кодируется двоичными цифрами 0 и 1, поскольку двоичное кодирование по теории информации близко к оптимальному, а кроме того, легче реализовать элементы с двумя устойчивыми состояниями (магнитные сердечники, триггеры).
2. ***Программное управление***: команды программы, так же как и данные, хранятся в памяти машины; хранимая программа позволяет выполнять команды в естественном порядке следования либо осуществлять произвольный переход от одной команды к другой.
3. ***Однородность памяти:*** Вид хранимой информации (команды или данные непосредственно в памяти) неразличим, а зависит от последующего использования; команды могут обрабатываться так же, как и числовые данные (модификация команд – сейчас не поощряется), либо порождать в процессе обработки другие команды (трансляция – широко применяется).
4. ***Адресность:*** в команде указываются не сами данные, а адреса их размещения в памяти.

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

1. наличие единого вычислительного устройства, включающего один процессор, память и некоторые внешние устройства;
2. использование линейной структуры адресации памяти со словами фиксированной длины;
3. централизованный принцип управления выполнением программы по последовательному алгоритму;
4. низкий уровень машинных команд, позволяющих выполнять только элементарные операции.

Для таких машин «узким» местом», ограничивающим производительность, является память и каналы связи: как данные, так и команды должны последовательно выбираться из памяти и передаваться между устройствами. Для повышения производительности в фон неймановских машинах применяются:

- увеличение разрядности обработки данных (16 бит🡪32 и 64 бит);

- активное использование конвейеризации при выборке и обработке команд;

- активное использование кэш-памяти (Cache – тайник, скрытый), т.е. модулей памяти, которые являются буферными между процессором и оперативной памятью.

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

# *1.4. Понятия архитектуры, организации и реализации компьютера*

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

- состав и характеристики процессоров, включая системы команд;

- состав и характеристики устройств памяти и ВУ;

- состав программных средств разработки ПО;

- вид ОС и режимы обработки данных.

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

- представление и форматы данных;

- уровни памяти и их взаимодействие;

- состав и форматы машинных команд;

- систему прерываний;

- способы обмена данными.

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

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

# *1.5. Многоуровневая организация компьютера*

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

Таблица 1.1

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

*Системы промежуточного ПО*:

1. Инструментальные среды программирования (Delphi, Visual C, С++ Builder)

2. Инструментальные технологии программирования (DCOM, CORBA, RMI, ECLIPSE)

*Особенности многоуровневой организации*:

1. Каждый верхний уровень интерпретируется одним или несколькими нижними уровнями.

2. Каждый из уровней можно проектировать независимо.

3. Модификация нижних уровней не влияет на реализацию верхних.

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

# *1.6. Понятие семантического разрыва между уровнями*

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

1) для традиционных ВМ, считающихся машинами со сложным набором команд (CISC), используется специализация машин, при которой операторы проблемно-ориентированных языков могут непосредственно выполняться аппаратными средствами машины; платой за повышение производительности является увеличение сложности устройства управления и отход от универсальности; примерами могут служить: аппаратная реализация графических преобразований; аппаратная реализация операций с векторами и матрицами;

2) переход к ВМ с сокращенным набором команд (RISC), характеризующихся ограниченным списком простых команд, оперирующих в основном данными, размещенными в регистрах; в результате реализация операторов ЯВУ на основе команд RISC-процессора оказывается почти столь же эффективной, что и аппаратная реализация, но не усложняет устройства управления.

# *1.7. Организация аппаратных средств ВМ*

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

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

ВУ1

ЦП ОП

ВУn

ЦП – центральный процессор

ОП – оперативная память

ВУ – внешнее устройство

Рис. 1.2

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

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

ВУ

ВУ

Контроллер

Канал 1 (Мультиплексный)

ЦП ОП Канал 2 (Селекторный) Контроллер ВУ

Канал 3 (Мультиплексный)

Контроллер Контроллер

ВУ ВУ

ВУ ВУ

Рис. 1.3

Центральным элементом машины является память ОП, которая хранит программы ЦП и каждого из каналов, являющихся процессорами ввода-вывода, работающими параллельно с ЦП по собственной программе, выбираемой из ОП. Различают каналы: селекторный – управляет только одним ВУ и применяется для подключения быстрых устройств; мультиплексный – управляет несколькими более медленными ВУ.

При данной организации все еще очень большое количество связей. Большая специализация процессоров различного типа затрудняет их интегральное исполнение. Такая организация применялась в машинах третьего и частично четвертого поколений.

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

А

Р

Б Общая шина (ОШ)

ЦП И

Т

Р

Память К1 МЛТП

ВУ1 К1 К2 К3

ВУ ВУ ВУ

Рис.1.4

Магистрально-модульная организация компьютера предполагает выделение общего универсального канала (магистрали связи между элементами системы – модулями) и определения общих правил взаимодействия. В центре ВМ – центральный процессор, управляющий информационной связью между устройствами, подключенными к магистрали (ВУ и память). ***Магистраль,*** называемая также ***Общей шиной*** (ОШ), представляет собой множество проводов. По одной группе проводов (шина данных) передается обрабатываемая информация, по другой (шина адреса) – адреса памяти или ВУ, к которым обращается процессор. Есть еще третья часть магистрали – шина управления, по ней передаются управляющие сигналы (например, сигнал запуска операции в устройстве и др.). Всякая информация, передаваемая от процессора к другим устройствам по шине данных, сопровождается адресом, передаваемым по адресной шине. Это может быть адрес ячейки в оперативной памяти или адрес ВУ.

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

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

В транзакции типа "Чтение" по шине сначала посылается в память адрес вместе с соответствующими сигналами управления, иницирующими чтение. Память отвечает, возвращая на шину данные с соответствующими сигналами управления. Транзакция типа "Запись" требует, чтобы ЦП или ВУ послало в память адрес и данные и не ожидает возврата данных. Обычно ЦП вынужден простаивать во время интервала между посылкой адреса и получением данных при выполнении чтения, но часто он не ожидает завершения операции при записи данных в память.

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

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

Данная организация ВМ предложена для разгрузки шины, связывающей процессор с памятью, и как следствие – повышения производительности и надежности работы ВМ.

ВУ ВУ

ВУ ВУ

Контроллер 1 Контроллер N

Шина памяти

Память ЦП

Шина ввода / вывода

Шина прямого доступа к памяти (ПДП)

Контроллер ПДП ВУ

ВУ

Рис.1.5

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

ADD AX, 100 ; адресуется ячейка памяти 100

IN AX, 100 ; адресуется внешнее устройство с номером 100

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

# Представление и обработка данных в ВМ

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

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

## Целые числа

Представляются в позиционной системе счисления в виде

 ,

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

αi =[0, s-1] - разрядный коэффициент

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

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

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

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



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

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

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

***Обработка целых чисел****.*

Различают беззнаковое и знаковое (с учетом знака) представления и обработку ЦЧ. Беззнаковые двоичные ЦЧ длиной n бит изменяются в диапазоне [0, Nmax], где Nmax = 2n – 1 . Например:

n = 4 - диапазон [0, 15] , n = 8 - диапазон [0, 255] , n = 16 - диапазон [0, 65535],

n = 32 - диапазон [0, ≈ 232  = 4\*230 ≈ 4\* (103) 3 = 4 \*109].

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

При использовании ЦЧ с учетом знака диапазон их изменения разбивается на две части: положительные числа ( для n-битного числа - [0, 2n-1 – 1] ) и отрицательные числа [- 2n-1 , 0]. Если результат арифметической обработки положительного числа превышает 2n-1 – 1, то возникает переполнение (положительное), а для отрицательных чисел переполнение (отрицательное) возникает, если результат меньше (- 2n-1). ЦЧ с учетом знака могут представляться в 3-х формах: прямой код и два инверсных кода: обратный и дополнительный. Во всех 3-х формах старший разряд считается знаковым и для положительных чисел равен 0, а для отрицательных равен 1.

Положительные числа во всех 3-х формах представляются одинаково «0&информационные разряды числа», где &-конкатенация (присоединение). Так, для 4-х -битных ЦЧ с диапазоном изменения [-8,7] число 5 будет иметь двоичное представление 0101. Представления отрицательных чисел различны.

Прямой код (ПК) отрицательного ЦЧ имеет вид «1&информационные разряды числа». Соответственно, число [-5] в ПК будет иметь вид [-5]пк = 1101.

Замечание: В прямом коде существуют: положительный ноль, равный 0000 и отрицательный ноль, равный 1000.

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

При работе с 16-ричными числами следует учитывать, что инверсные значения 16-ричных цифр имеют вид:

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

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

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

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

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

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

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

При выполнении операций умножения и деления используется представление ЦЧ в прямом коде.

Знак результата формируется как «исключающее ИЛИ» или «сумма по mod2» знаков операндов. Умножение выполняется следующим образом An \* Bn = C2n . При этом переполнение не возникает. Деление выполняется следующим образом A2n / Bn = {C1n (n-битное частное)иC2n (n-битный остаток)}.Знак частного вычисляется как «сумма по mod2» знаков делимого и делителя, знак остатка равен знаку делимого. Переполнение фиксируется, если частное не размещается в n – разрядах при делении на 0 или если A > 2n \* B.

**П**ри решении экономических задач часто возникает потребность в обработке десятичных чисел. Для этого используется их двоично-десятичное представление, при котором каждый десятичный разряд числа представляется двоичной тетрадой (4 битами) цифр с весами 8421. Значения тетрады 1010 ÷ 1111, соответсвующие 16-ричным цифрам A ÷ F, считаются запрещенными и их получение при сложении данных требует коррекции результата. Коррекция выполняется путем прибавления к результату так называемого «десятичного заполнителя» 610 = 01102 . Например, при сложении десятичных чисел 8 + 5 =13, представленных в двоично-десятичном формате имеем

1000 + 0101 = 1101 (запрещенный код) +0110 = 0001 0011 = 1310

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

## Представление и обработка вещественных чисел.

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

N = m \* S P , где S – основание СС, m – мантисса, P – порядок.

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

N = 5.41 \*100 = 0.541\*101 = 541\*10-2 = 0.00541\*103

Для повышения точности ВЧ обычно используют их **нормализованное** представление, при котором на мантиссу накладывается ограничение 1/S ≤ m < 1 (представление N = 0.541\*101  - нормализованное) .

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | | 1 2 8 | | 9 31 |
| Sm | | Sp | P | m |
|  | порядок | | | Мантисса |

Здесь Sm – знак мантиссы или знак числа, Sp – знак порядка.

При использовании короткого формата имеем диапазон изменения ВЧ [ - Nmax, Nmax] , где

Nmax = 1\* 2Pmax = 2128 = ( 210 )12.8  ≈ ( 103 )12.8 ≈ 1038

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

δ = 2 -23 = ( 210 ) -2.3  ≈ ( 103 ) -2.3 ≈ 10 -7 ,

то есть обеспечивается точность на уровне 7 десятичных разрядов.

Для длинного формата ВЧ соответственно получим диапазон [- 10308 , 10308 ] и точность - 17 десятичных разрядов.

В целом следует отметить, что форма представления ВЧ с плавающей точкой позволяет значительно увеличить диапазон обрабатываемых ВЧ за счет некоторой потери точности из-за того, что часть разрядов числа отводится под порядок.

Короткий формат при невысокой точности значений позволяет размещать в памяти большее число операндов и выполнять операции с небольшими затратами времени, а длинный формат обеспечивает возможность вычислений с повышенной точностью.

Выполнение операций алгебраического сложения над ВЧ состоит из трех этапов:

1. Выравнивание порядков операндов – выпоняется в сторону большего порядка: увеличение на 1 меньшего порядка сопровождается сдвигом на 1 раздряд вправо мантиссы числа с меньшим порядком; этот процесс продолжается пока порядки операндов не станут равны.
2. Суммирование мантисс – выполняется по правилам сложения целых чисел.
3. Нормализация результата.

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

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

При выполнении операций умножение или деление порядки операндов участвуют в алгебраическом сложении как целые числа (суммируются при умножении или вычитаются при делении), а мантиссы перемножаются или делятся также как целые числа. Считаем, что исходные операнды представлены в нормализованной форме. Тогда при умножении может возникнуть нарушение номализации вправо на 1 разряд, которое исправляется сдвигом мантиссы результата на 1 разряд влево и вычитанием 1 из порядка. Также при умножении может возикнуть переполнение, если Р1 + Р2 > Рmax или потеря значимости, если Р1 + Р2 < ‒ Рmax . При делении может возникнуть переполнение при нулевой мантиссе делителя или, если Р1 - Р2 > Рmax , а также потеря значимости при исправлении нарушения номализации влево или, если Р1 - Р2 < ‒ Рmax .

## Логические операции над битовыми наборами

Логические операции над битовыми наборами выполняются поразрядно в соответствии с таблицей 2.1.

Таблица 2.1

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

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

## Представление и обработка символов.

Представление символьной информации основано на сопоставлении каждому символу определенной числовой (обычно двоичной) комбинации. Совокупность символов и назначенных им числовых кодов образует ***таблицу кодировки.*** В настоящее время известно много таблиц кодировки, но их объединяет весовой принцип, также называемый принцип лексикографического упорядочения кодов символов. Суть его состоит в том, что коды цифр возрастают по мере увеличения цифры, а коды буквенных символов увеличиваются в алфавитном порядке. Так, код буквы «Б» на 1 больше кода буквы «А». Это упрощает обработку символьной информации, в частности, облегчает сортировку, упорядочивание и поиск символьных данных.

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

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

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

## Представление видеоинформации и аудиоинформации.

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

* + 1. **Видеоинформация.**

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

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

Векторное представление описывает изображение не пикселами, а кривыми – сплайнами. Сплайн – гладкая кривая, проходящая через несколько (2-4) опорных точек, задающих форму сплайна. В векторной графике распространены сплайны на основе кривых Безье. Любую элементарную кривую можно построить, зная четыре коэффициента Р0, Р1, Р2, Р3, соответствующих четырем точкам на плоскости. Перемещение точек влечет изменение формы кривой. Для многих видов изображений их описание с помощью математических формул является более простым способом благодаря легкости масштабирования и существенно меньшим требованиям к памяти. Недостатком векторных изображений является необходимость разбиения изображений на множество составляющих его примитивов (линия, прямоугольник, эллипс) и более низкое качество. К распространенным векторным форматам относятся CDR, DXF, PS, SVG, VSD.

* + 1. **Аудиоинформация**

Аудиоинформация имеет чисто динамический характер и связана с представлением звуковых сигналов, частоты которых лежат в диапазоне 15 Гц ÷ 20 КГц, являющихся по своей природе непрерывными (аналоговыми). Поэтому для использования в ВМ аудиоинформация должна быть представлена в виде значений (выборок), взятых через определенные интервалы времени, и преобразована в цифровую форму с помощью аналого-цифрового преобразователя (АЦП). После обработки в ВМ для воспроизведения звуковые сигналы должны быть восстановлены в аналоговую форму с помощью цифро-аналогового преобразователя (ЦАП). Для обеспечения высокого качества воспроизведения требуется большое количество выборок и соответственно большая емкость памяти для хранения аудиоинформации. Это приводит к необходимости использования различных методов сжатия данных для сокращения требуемого объема памяти. К распространенным форматам представления аудиоинформации следует отнести AVI, WAV, AIF, MPEG.

# Организация процессора и основной памяти ВМ

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

**Процессор** выполняет две функции:

- обработка данных в соответствии с заданной программой;

- управление всеми устройствами машины.

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

- непосредственно самих значений данных;

- адресов расположения данных в памяти;

- сведений для определения адресов размещения данных в памяти.

Формирование исполнительного адреса – этап перехода от сведений об адресе к самому адресу. В адресной части могут быть сведения об отсутствии операндов (нуль-адресная или безадресная команда ) и адресации от одного (одноадресная команда) до трех операндов (трехадресная команда).

# *3.1. Типовая структура процессора и основной памяти*

Типовая структура центральной части ВМ – процессора и основной памяти представлена на рис. 3.1.

А

Д

У

Операционная часть {Θj} Управляющая часть {Θi} Память

АЛУ Внутренние Рг РАП

{У j } {Уi}

РОНы Рг СС РДП

ПСч Накопитель

Рг команд Контроллер памяти

Рг УиС

ДешКОПиРА

ФормировательУС

ЦП ГТ

Рис. 3.1

На схеме, показанной на рис. 3.1, использованы следующие компоненты процессора и памяти:

* АЛУ – арифметико-логическое устройство выполняет операции по обработке данных;
* РОНы – регистры общего назначения (от 8 до нескольких сотен штук) – сверхбыстрая память малой емкости для хранения операндов;
* Рг СС – регистр слова состояния. Содержит текущее состояние процессора, в который входит уровень приоритета текущей программы, биты условий {Θj}завершения последней команды, режим обработки текущей команды. Возможны следующие режимы обработки (в порядке возрастания уровня приоритета):

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

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

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

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

# *3.2. Основной цикл работы процессора*

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

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

Большинство этапов для выполнения требуют как минимум один цикл памяти.

Для ускорения выборки и обработки команд программы используется конвейерный способ выполнения этих этапов (при выполнении очередного этапа текущей команды одновременно происходит выполнение предыдущего этапа следующей команды), схематично показанный на рис. 3.2. Разумеется, если одна команда изменяет содержимое какой-либо ячейки памяти, а следующая использует ее новое значение, то последняя не может начать исполняться, пока результат не будет сохранен.

Б1 Б2 Б3 Б4 Б5

Сохранение результата

Исполнение операции

Выборка операндов

Декодирование команды

Выборка команды

|  |  |
| --- | --- |
| Б1 | 7  6  5  4  3  2  1 |
| Б2 | 6  5  4  3  2  1 |
| Б3 | 5  4  3  2  1 |
| Б4 | 4  3  2  1 |
| Б5 | 3  2  1 |

Время

Рис.3.2.

После выполнения основного цикла процессора, как правило, происходит проверка запроса программного прерывания. Вообще, в процессе выполнения команд программы возможно появление запросов прерываний двух видов:

* запрос программного прерывания, который обслуживается процессором путем выполнения специальной программы – обработчика прерываний, требует сохранения текущего состояния управляющих и операционных узлов процессора и поэтому проверяется и обслуживается только после завершения выполнения очередной команды;
* запрос аппаратного прерывания, обслуживается специальными аппаратными средствами без участия программы процессора и не требует сохранения его состояния, поэтому может поступать и обслуживаться после завершения любого этапа цикла процессора, прерывая выполнение текущей команды.

Подробнее о прерываниях будет сказано позже в разделе 7.

# *3.3. Организация процессора и памяти в архитекуре Intel X86.*

В процессоре Intel 8086 длина слова составляет 16 бит или 2 байта. Минимально адресуемой и обрабатываемой единицей информации является байт, при этом адрес слова совпадает с адресом младшего байта и является четным. При разрядности адреса 16 бит максимальная емкость прямо адресуемой памяти составляет   
216 = 64 Кбайт. Для расширения адресуемого пространства памяти используется ее разбиение на блоки – сегменты, называемое сегментированием памяти. Каждый сегмент имеет произвольную длину, не превышающую  
64 Кбайт. Адрес байта в сегментированной памяти задается двумя составляющими: сегментная часть (16 бит), определяющая адрес начала сегмента, и смещение (16 бит) байта в пределах сегмента. Обычно адрес представляется в виде пары Сегмент (Segment) : Смещение (Offset). Начало размещения сегмента выравнивается на границу блока памяти из 16 байт, называемого «параграфом». Соответственно, физический адрес памяти получается путем суммирования сегмента, сдвинутого на 4 бита влево, со значением смещения. Результатом такого суммирования является 20-битный физический адрес, чем обеспечивается адресация 1 Мбайт памяти.

***Пример.*** Пусть Segment = DCBA, Offset = 5678.

Физический адрес Adr будет определяться суммой: DCBA0 + 5678 = E2218.

# 3.3.1. Программно доступные регистры процессора

Начиная с 80386, процессоры Intel предоставляют 16 основных регист­ров для пользовательских программ плюс еще 11 регистров для работы с числами с плавающей запятой (FPU/NPX) и мультимедийными при­ложениями (ММХ). Все команды так или иначе изменяют значения ре­гистров, и всегда быстрее и удобнее обращаться к регистру, чем к памяти.

*Регистры общего назначения*

16-битные регистры АХ (аккумулятор), ВХ (база), СХ (счетчик), DX (регистр данных) могут использоваться без ограничений для любых це­лей – временного хранения данных, аргументов или результатов различ­ных операций. На самом деле, начиная с процессора 80386, все эти регистры имеют размер 32 бита и называ­ются они ЕАХ, EВХ, EСХ, EDX. Кроме этого, отдельные байты в 16-бит­ных регистрах АХ - DX тоже имеют свои имена и могут использоваться как 8-битные регистры. Старшие байты этих регистров называются АН, ВН, СН, DH, а младшие - AL, DL, CL, DL.

Другие четыре регистра общего назначения (РОН) – SI (индекс источни­ка), DI (индекс приемника), BP (указатель базы), SP (указатель сте­ка) – имеют более конкретное назначение и могут применяться для хра­нения всевозможных временных переменных, только когда они не используются по назначению. Регистры SI и DI используются в стро­ковых операциях, ВР и SP используются при работе со стеком. Так же как и для регистров АХ - DX, начиная с процессора 80386, эти четыре регистра являются 32-битными, называются ESI, EDI, EВР и ESP соответственно и могут использоваться как универсальные РОН.

*Сегментные регистры*

При использовании памяти для формирования любого адреса применяются два числа – адрес начала сегмента и смещение искомого байта относительно этого начала. Для хранения адреса начала сегмента в процессорах Intel8086 предусмотрены следующие сегментные регистры – CS (сегмент кода), DS (сегмент данных), ES (дополнительный сегмент данных), SS(сегмент стека). Начиная с процессора 80386, появились дополнительные регистры FS и GS.

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

*Регистр флагов*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Х | Х | Х | Х | OF | DF | IF | TF | SF | ZF | 0 | AF | 0 | PF | 1 | CF |

0

7

8

15

CF - флаг переноса (CARRY);

PF - флаг четности (PARITY);

AF - дополнительный флаг переноса (AUXILARY);

ZF - флаг нуля (ZERO);

SF - знаковый флаг (SIGN);

TF - флаг слежения, ловушка (TRAP);

IF - флаг прерываний (INTERRUPTION);

DF - флаг направления (DIRECTION);

OF - флаг переполнения (OVERFLOW).

# 3.3.2. Организация стека в архитектуре Intel Х86

Стек – специальная область памяти, достуная для записи (заполнения) и выборки (выталкивания) данных только с одного конца, называемого верхушкой стека. В процессорах Intel Х86 на верхушку стека указывает РОН указатель стека (SP – stack pointer), стек может работать только со словами, заполнение стека происходит в сторону уменьшения адресов с помощью команды PUSH, а освобождение стека (выталкивание данных из стека) – в сторону увеличения адресов командой РОР.

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

PUSH AX: POP AX:

(SP)-2🡪SP [SP]🡪AX

(AX)🡪[SP] (SP)+2🡪SP

При выполнении программ в процессорах Intel Х86 стек используется в случаях:

* промежуточного хранения содержимого регистра или группы регистров;
* обмена содержимого регистров;
* сохранения адресов возврата при вызове подпрограмм;
* передачи параметров между вызываемой и вызывающей программами;
* сохранения адресов возврата и регистра флагов при обработке прерываний.

# 3.3.3. Организация выполняемых программ в MS DOS

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

* \*.bat – задают порядок выполнения некоторой последовательности программ;
* \*.com – файлы исполняемых программ типа com;
* \*.exe – файлы исполняемых программ типа exe.

Файлы типа \*.com содержат только исполняемый код без дополнительной информации о программе, формируются в загрузочном виде и не требуют настройки. Весь код, данные и стек такой программы располагаются в одном сегменте, имеют длину не более 64 Кб и служат для организации простых модулей, ориентированных на модели памяти tiny и small (приложение 1).

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

Структура размещения в памяти файла типа COM показана на рис. 3.3.

|  |  |  |
| --- | --- | --- |
| 64 Кб | ↓ Стек    ↑ Данные и код программы | ←SS:SP |
|  |  |
|  | ←100h 🡨 CS:IP ( директива ORG 100h) |
|  | PSP |  |
|  |  |
| 0 | ← DS, CS, ES, SS |

Рис. 3.3

В начальной части com-файла программы (а также и exe-файла) MS DOS размещает специальный блок – префикс сегмента программы ПСП (PSP - рrogram segment prefix), который содержит информацию для доступа программы к параметрам командной строки, к среде окружения, для реакции программы на критические ошибки и управляющие команды типа Ctrl+C. В начальной части PSP размещена команда вызова обработчика прерывания для завершения программы и возврата в DOS. Так как после загрузки все сегментные регистры, включая CS, указывают на начало PSP, а IP = 0, то программа не может исполняться, начиная с этого адреса, и первой командой делают ORG 100h , устанавливающуую CS:IP на конец PSP.

Структура размещения в памяти файла типа EXE показана на рис. 3.4ю мых файлов. сразу за данной.., а состав ряда полей ПСП – в табл. 3.1.

SS:SP

Стек

SS:00

Данные

DS (сюда его необходимо установить)

Код

100h CS:00

PSP

DS (начальное положение), ES

Рис.3.4

Таблица 3.1. Состав основных полей префикса сегмента программы

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

# 3.3.4. Режимы адресации памяти в архитектуре Intel Х86

Большинство команд процессора Intel Х86 выполняются с аргументами, которые принято называть *операндами.* Операнды в программе могут задаваться следующим образом:

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

Для указания места расположения операнда используются 8 режимов адресации, использование которых иллюстрируется в таблице 3.2.

**1. Регистровая адресация**

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

**2. Непосредственная адресация**

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

**3. Прямая адресация**

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

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

**4. Косвенная адресация**

Адрес операнда в памяти можно не указывать непосредственно, а хранить в любом регистре. До процессоров i80386 для этого можно было использовать только регистры ВХ, SI, DI и ВР, но потом эти ограничения были сняты и адрес операнда разрешили считывать также и из ЕАХ, ЕВХ, ЕСХ, EDX, ESI, EDI**,** ЕВР и ESP (но не из AX, CX, DX или SP напрямую – надо исполь­зовать ЕАХ, ЕСХ, EDX, ESP соответственно или предварительно скопи­ровать смещение в ВХ, S1, DI или ВР). Как и в случае прямой адресации, DS используется по умолчанию, но не во всех случаях: если смещение берут из регистров ESP, ЕВР или ВР, то в качестве сегментного регистра используется SS. В реальном режиме можно свободно пользоваться всеми 32-битными регистрами, надо толь­ко следить, чтобы их содержимое не превышало границ 16-битного слова.

**5-6. Базовая или индексная адресация.**

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

До процессора i80386 в качестве базового регистра можно было использовать только регистры ВХ, ВР, SI или DI и сдвиг мог быть только байтом или словом (со знаком). Начиная с процессоров i80386 и старше, можно допол­нительно использовать ЕАХ, ЕВХ, ЕСХ, EDX, ЕВР, ESP, ESI и EDI, так же как и для обычной косвенной адресации. С помощью этого метода можно организовывать доступ к одномерным массивам байт: смещение соответствует адресу начала массива, а число в регистре – индексу эле­мента массива, который надо использовать. Очевидно, что если массив состоит не из байт, а из слов, придется умножать базовый регистр на два, а если из двойных слов – на четыре. Для этого предусмотрен следующий специальный метод адресации.

**7. Индексная адресация с масштабированием**

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

mov ax, [esi\*2]+2

Множитель, который может быть равен 1, 2, 4 или 8, соответствует размеру элемента массива – байту, слову, двойному слову, учетверенно­му слову соответственно. Из регистров в этом варианте адресации мож­но использовать только ЕАХ, ЕВХ, ЕСХ, EDX, ESI, EDI, ЕВР, ESP, но не **SI,** DI, ВР или SP, которые можно было использовать в предыдущих ва­риантах.

**8. Адресация по базе с индексированием и масштабированием**

Это самая полная возможная схема адресации, в которую входят все случаи, рассмотренные ранее, как частные. Смещение может быть байтом, словом или двойным словом. Если ESP или ЕВР используются в роли базового регистра, селектор сегмента операн­да берется по умолчанию из регистра SS, во всех остальных случаях – из DS.

Таблица 3.2. Использование режимов адресации в архитектуре Intel Х86

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Режим адресации | | Описание в Ассемблере | Регистр сегмента (по умолчанию) | Пример использования |
| Код | Название |
| 0 | Регистровая | EAX/AX/AL | − | MOV EAX, EDX ; DEC CX  MOV AL, CH ; PUSH DS  IN AL, DX |
| 1 | Непосредственная | Данное в виде константы (номер порта ввода-вывода) | − | MOV AX, 1000h; ADD AL, -30h AND EAX, 0000FFFFh  OUT 21h, AL |
| 2 | Прямая | Имя (метка) сме- щения в памяти | DS | MOV BL, Mem\_B1  MOV AX, OFFSET Table |
| 3 | Косвенно-регистровая | [BX], [SI],  [BP],  [DI],  [EAX] и др. | DS  SS  DS(ES)  DS | MOV AL, [SI]  MOV CX, [BP]  MOV AX, ES:[BX]  ADD EAX, [EDX] |
| 4 | Базовая | [BX] + смещение,  [BP] + смещение | DS  SS | MOV AX, CS:[BX] +4  MOV CX, PAR\_TAB[BP] |
| 5 | Индексная | [SI] + смещение,  [DI] + смещение | DS  DS(ES) | CMP ‘A’, STR1[SI]  MOV AL, BYT\_TAB[DI] |
| 6 | Базово-индексная | BX] [SI] + смещ.,  BX] [DI] + смещ.,  [BP] [SI] + смещ.,  [BP] [DI] + смещ. | DS  DS  SS  SS | MOV AX, ES:VALUE[BX][DI] |
| 7 | Индексная с масштабированием | [ERg\* m] + смещ. | DS  SS – для EBP,ESP | MOV CX, WORD\_TAB[ESI\*2] |
| 8 | Базово-индексная с масштабированием | [ERg1] [ERg2\* m] + смещение | DS  SS – для EBP,ESP | MOV BX, [EDX][ECX\*2] + 8 |

**Примечания:**

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

# 3.3.5. Краткая характеристика системы команд процессоров Intel Х86

***Формат команды***  представлен в табл. 3.3.

Таблица 3.3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Префиксы | Переопределе-ние сегмента | Код операции (КОП) | Режим адресации | МИБ | Смещение | Данные |
| 0/1/2\*)/3\*) | 0/1 | 1/2\*) | 0/1 | 0/1\*) | 0/1/2/4\*) | 0/1/2/4\*) |

В первой строке табл. 3.3 указывается имя поля команды, во второй – длина поля в байтах. Знаком \*)  помечены значения длины поля, используемые, начиная с процессора i80386. Максимальная длина команды для процессора i8086 равна 6 байт.

Поле префиксов в процессоре i8086 состоит из 1 байта и может задавать префикс повторения команды, который в ассемблере обозначается как REP (повторять), REPE (повторять поравну), REPNE (повторять не поравно) или префикс LOCK – запрет доступа к шине на время выполнения команды. Более подробно префиксы поясняются в прил. 1 при описании команд строковой обработки. Начиная с процессора i80386, также добавляются однобайтовые префикс размера адреса и префикс размера операнда.

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

Поле КОП – единственное поле, которое не может быть пустым, оно определяет код (вид) операции, которая должна быть выполнена процессором и тип обрабатываемых данных (для процессора i8086 – байт или слово). Для процессора i8086 поле является однобайтовым, а начиная с процессора i80386 – двухбайтовым.

Поле режима адресации служит для задания режима адресации, места размещения операнда: в регистре или в памяти конкретного используемого регистра.

Поле МИБ (масштаб, индекс, база) является расширением поля режима адресации, используемым начиная с процессора i80386 и задающим при формировании исполнительного адреса операнда масштабный коэффициент, индексный регистр и регистр базы.

Поле смещение содержит смещение адреса памяти при использовании прямой адресации.

Поле данные содержит значение операнда при непосредственной адресации.

Все команды процессора i8086 можно разделить на следующие группы:

1. команды передачи данных;
2. команды арифметических операций над целыми числами;
3. логические команды;
4. сдвиговые команды;
5. команды обработки строк;
6. команды передачи управления;
7. команды прерываний;
8. команды управления флагами;
9. команды управления состоянием процессора;
10. команды плавающей арифметики;
11. команды мультимедийных расширений (ММХ – MultiMedia eXtension);
12. команды потокового расширения (SSE – Streaming SIMD Extension).

# 3.3.6 .Арифметическая обработка чисел с использованием математического сопроцессора

Арифметическая обработка числовых данных обычно выполняется с использованием математического сопроцессора (FPU – Floating Point Unit, NPR – Numeric PRocessor), который сначала выполнялся в виде отдельной микросхемы (i8087 – i80387), а начиная с процессора i80486DX встраивается в состав основного процессора. Сопроцессор называется так потому, что может работать параллельно с процессором после инициализации. Языки высокого уровня используют его непосредственно, а ассемблер порождает ESC команды.

Независимо от наличия сопроцессора, выполнение команд FPU может быть осуществлено тремя способами:

1. С использованием библиотеки альтернативной математики: считается, что сопроцессор отсутствует и выполнение команд FPU моделируется на уровне операций основного процессора.
2. С использованием библиотеки эмулятора: автоматически определяется наличие сопроцессора, и если он есть, то выполняет команды FPU, а иначе их выполнение эмулируется (моделируется на уровне микро-операций) основным процессором.
3. При отсутствии математического сопроцессора возникает прерывание и выполнение программы прекращается.

Первый способ – самый медленный и сейчас практически не применяется. Второй способ не всегда совместим с некоторыми резидентными программами (одни и те же прерывания используются для эмулятора и резидентных программ). Третий способ самый быстрый и сейчас является основным.

***Назначение математического сопроцессора***

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

***Типы данных*** ***математического сопроцессора***

Сопроцессор поддерживает 7 типов данных: три целых (16 бит – Word Integer, 32 бит – Short Integer и 64 бит– Long Integer), 80-битные двоично-десятичные целые (Packed Decimal) и три формата с плавающей запятой, представленые в табл. 3.4.

Таблица 3.4

|  |  |  |
| --- | --- | --- |
| Разрядность (порядка-p/мантиссы-m) | Точность в десятичных разрядах | Диапазон изменения |
| 32 (8/24) Короткое вещественное | 6 / 7 | 10-38 - 1037 |
| 64 (11/53) Длинное вещественное | 15 / 16 | 10-308 - 10307 |
| 80 (15/65) Расширенное вещественное | 18 / 19 | 10-4932 - 104931 |

Сопроцессор выполняет все вычисленя в 80-битном раширенном формате, а 32-битный и 64-битный форматы используются для обмена данными с основным процессором и памятью.

Кроме обычных чисел сопроцессор использует несколько специальных данных, получаемых при выполнении операций:

p m

Положительная бесконечность 0 | 1 … 1 00 … 0

Отрицательная бесконечность 1 | 1 … 1 00 … 0

Неопределенность 1 | 1 … 1 10 … 0

Не число 0 | 1 … 1 1Х…Х , где Х…Х ≠ 0

***Регистры*** ***математического сопроцессора***

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

Кроме того, сопроцессор использует пять вспомогательных регистров:

1. Регистр управления CR – задает режим обработки данных: маскирование ошибок (некорректная операция, деление на 0, переполнение и т.д.), точность обработки (расширенная, двойная, одинарная), способы округления (к ближайшему числу, к нулю, к +∞ или к -∞).
2. Регистр состояния SR – содержит: флаги особых случаев, возникающих в результате выполнения операций (IE - некорректная операция, DE - денормализованный операнд, ZE – деление на ноль, OE – переполнение, UE – антипереполнение, PE – неточный результат); флаги условий, возникающие при операциях сравнения; поле указателя вершины стека ST или TOP; бит B занятости сопроцессора.
3. Регистр тегов TW – содержит двухбитовое поле для каждого из восьми числовых регистров сопроцессора

TW(i) ST(i)

00 вещественное число, не равное нулю

01 вещественное число, равное нулю

10 не число

11 не инициализировано

1. Регистр указателя команды FIP – содержит адрес последней выполненной команды.
2. Регистр указателя операнда DIP – содержит адрес операнда последней выполненной команды.

Два последних регистра используются обработчиком исключений при анализе вызвавшей его команды.

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

Примеры:

1. f:= (a + b) \* pi – c преобразуется к виду ab + pi \* c – f :=
2. f:= atan ( sqrt (x2 / (x2 - 1) ) преобразуется к виду xx \* xx \* 1 - / sqrt atan f :=

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

Data Segment

a DQ 2.5

b DQ 6.0

c DQ 3.5

f DQ ?

Data EndS

Code Segment

Assume cs : Code, ds : Data

Public f

Expression Proc Far

Fld a ; a 🡪 st(0)

Fld b ; b 🡪 st(0), a 🡪 st(1)

Fadd ; st(0) + st(1) 🡪 st(0) или (a+b) 🡪 st(0)

Fld pi ; pi 🡪 st(0), (a+b) 🡪 st(1)

Fmnl ; st(0) \* st(1) 🡪 st(0) или (a+b)\*pi 🡪 st(0)

Fld c ; c 🡪 st(0), (a+b)\*pi 🡪 st(1)

Fsub ; st(0) – st(1) 🡪 st(0) или (a+b)\*pi - c 🡪 st(0)

Fstp f ; f:= st(0); st(1) 🡪 st(0)

Ret

Expression EndP

Code EndS

Краткие сведения по программированию на языке Ассемблера IBM PC, используемом здесь и далее для иллюстрации управления обработкой данных в ЭВМ, приведены в приложении 2.

# Организация прерываний в процессорах Intel X86

В архитектуре процессоров Intel X86 предусмотрены особые случаи, когда процессор прерывает выполнение текущей программы и передает управление программе, обрабатывающей возникшую ситуацию. Такие особые ситуации назыаются прерываниями и используются как механизм асинхронного взаимодействия процессов, одновременно выполняемых в вычислительной системе, и как еще один способ вызова подпрограмм. Весь программный интерфейс прикладных программ с функциями операционной системы DOS и сервисами BIOS реализуется именно на основе прерываний.

***Прерывания*** бывают двух видов:

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

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

***Исключения*** делятся на три типа: ошибки, ловушки и остановы. ***Ошибка*** происходит до выполнения команды, например, если такой команды не существует, или выполняется обращение к некоторой привилегированной команде или запрещенной для доступа области данных. Адрес возврата в этом случае указывает на ошибочную команду. ***Ловушка***– прерывание, возникающее после выполнения команды (например, для организации режима пошагового выполнения программы под управлением отладчика). Адрес возврата указывает на следующую команду. ***Останов*** – ситуация с неопределенным результатом. Возврат может вообще не происходить.

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

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

*1. Прерывания низкого уровня.*

Номера веторов прерываний: NВП= 0…1Fh :

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

- прерывания от контроллера прерываний (микросхема 8259) 8…0Fh (8 – системный таймер;  
9 – клавиатура; 0Ah – обслуживание видеоадаптера; 0Eh – обслуживание жесткого диска).

- прерывания BIOS 0Ah…1Fh (0Ah – обмен данными с дисплеем; 0Bh – возвращение объема памяти; 0Ch – обмен данными с диском; 0Dh – последовательный порт ввода вывода и т.д.).

*2. Прерывания среднего уровня.*

NВП=20h…5Fh

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

Прерывания пользователей.

NВП=60h…7Fh (прерывания от 70h используются для различных устройств).

*3. Прерывания языков высокого уровня.*

**Уровни приоритета**

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

IRQ0 (INT 8) – прерывания от системного таймера;

IRQ1 (INT 9) – прерывания от клавиатуры;

IRQ2 – размножитель приоритетных уровней;

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

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

IRQ10- IRQ12 – резерв для дополнительных устройств;

IRQ13 (INT 2) – прерывания по ошибке FPU;

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

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

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

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

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

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

**Маскируемые прерывания**

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

Способы реализации.

1. Общее маскирование.

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

2) Выборочное маскирование.

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

IRR – регистр запроса прерывания;

ISR – регистр обслуживания прерывания (порт 20h);

IMR – регистр маскирования прерывания (порт 21h.

К регистру IRR подключены все линии запросов IRQ0..IRQ15. Регистр ISR хранит приоритет текущего обслуживаемого прерывания. Если запрос прерывания, поступивший в регистр IRR не замаскирован, то происходит сравнение приоритетов PRIRR и PRISR .Если PRIRR>PRISR,то поступивший запрос принимается на обслуживание.

Например, запрет прерываний от жесткого диска обеспечивается командами

mov al, 01000000b; маскируются запросы прерывания от жесткого диска

out 21h

------

mov al, 0

out 21h

***Разработка собственных прерываний***

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

Существует два способа заполнения вектора прерывания адресом своего обработчика:

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

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

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

Data Segment

old\_cs dw 0; буфер для хранения

old\_ip dw 0; старого вектора прерывания (ВП)

---------

old\_vp dd 0;

Data ENDS

Code Segment

;сохранение адреса старого обработчика

mov ax, 3560h

int 21; старый\_ВП\_cs🡪es, старый\_ВП\_ip🡪bx

mov old\_cs, es

mov old\_ip, bx

;задание адреса нового обработчика в ВП 60h

push ds

mov dx, offset New\_sub

mov ax, seg New\_sub

mov ds, ax

mov ax, 2560h

int 21h

pop ds

;новый обработчик прерывания 60h

New\_sub proc far

push ax

; тело нового обработчика

pop ax

mov al, 20 ; загрузка в регистр ISR кода 20

out 20h ; для разрешения прерываний более низкого уровня

iret

---------

Final EQU $

---------

New\_sub ENDP

;восстановление старого ВП

lds dx, DWORD PTR OLD\_CS

mov ax, 2560h

int 21h

Возможные проблемы:

* если данные передаются через память, нужно тщательно следить за содержимым регистра DS; лучше данные передавать через регистры или стек;
* если возможно прерывание обработчика через Ctrl+Break, необходимо предусмотреть восстановление адреса старого обработчика;
* требуется минимизировать код обработчика прерывания, так как на время его выполнения может быть запрещено выполнение других прерываний.

***Перекрытие обработчика прерываний***

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

* создать новый обработчик прерываний (реализующий дополнительные функции), который вызывает старый (системный), размещенный по новому неиспользованному ранее вектору в диапазоне 60h…70h;
* перенести адрес старого обработчика прерываний в новый вектор прерывания;
* изменить вектор прерывания с системным номером таким образом, чтобы он указывал на новый обработчик прерывания;
* завершить программу установки нового обработчика и оставить ее резидентной в памяти.

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

push f

call old\_handler;

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

jmp cs:old\_handler;

если дополнительные функции реализуются до выполнения старого обработчика.

***Разработка резидентных обработчиков прерываний***

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

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

* наличие свободного вектора прерывания и указание метки конца обработчика для int 27h (определение длины обработчика +100PSP);
* для int 31h длина обработчика задается в параграфах.

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

Cod\_s Segment

Begin: jmp short set\_up; переход на инициализацию

Rezid\_h: proc far

push ds

;тело процедуры обработчика прерывания

pop ds

iret

Final EQU $; текущее значение счетчика размещения

Rezid\_h ENDP

Set\_up: mov dx, offset Rezid\_h

mov ax, 2568h; задание свободного ВП с номером 68h

int 21h

;завершение с оставлением в памяти

lea dx, Final

int 27h

ret

Cods ENDS

# Эволюция микроархитектуры Intel X86

Первый однокристальный четырехбитный микропроцессор 4004 фирма Intel разработала в 1971 году с целью использования в качестве универсального контроллера для калькуляторов. Он содержал 2300 транзисторов на кристалле кремния площадью 12 мм2, был изготовлен по 10 мкм технологии и работал с тактовой частотой 750 кГц. Успех процессора 4004 вдохновил Intel на создание 8-битного микропроцессора 8008, а затем и 8080, который, в свою очередь, превратился в 16-битные процессоры 8086 (1978 год) и 80286 (1982 год). В 1985 году Intel представила процессор 80386, который расширил архитектуру 8086 и превратил ее в 32-битную, ознаменовав появление архитектуры Intel X86.

В таблице 3.5показана эволюция микропроцессоров Intel X86. За 40 лет после создания 4004, линейные размеры транзисторов уменьшились в 160 раз, число транзисторов на кристалле увеличилось на пять порядков, а рабочая частота выросла почти на четыре порядка. Ни в одной другой области техники никогда не было такого удивительного прогресса в столь короткий промежуток времени.

Таблица 3.5**Процессор**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Процессор | Год | Технология (мкм) | Число транз-ов (млн) | Частота (Мгц) | Микроархитектура |
| 80386 | 1985 | 1.5–1.0 | 0,275 | 16–25 | Многотактная |
| 80486 | 1989 | 1.0–0.6 | 1,2 | 25–100 | Конвейерная |
| Pentium | 1993 | 0.8–0.35 | 3,2–4,5 | 60–300 | Суперскалярная |
| Pentium II | 1997 | 0.35–0.25 | 7,5 | 233–450 | Внеочередная версия |
| Pentium III | 1999 | 0.25–0.18 | 9,5–28 | 450–1400 | Внеочередная версия |
| Pentium 4 | 2001 | 0.18–0.09 | 42–178 | 1400–3730 | Внеочередная версия |
| Pentium М | 2003 | 0.13–0.09 | 77–140 | 900–2130 | Внеочередная версия |
| Core Duo | 2005 | 0.065 | 152 | 1500–2160 | Двухядерная |
| Core 2 Duo | 2006 | 0.065–0.045 | 167–410 | 1800–3300 | Двухядерная |
| Core i3-i7 | 2009 | 0.045–0.032 | 382–731 | 2530–3460 | Многоядерная |

Intel 80386 был многотактный процессор. Программируемая логическая матрица (ПЛА), содержащая *микрокод*, использовалась устройством управления для того, чтобы определить порядок перехода между состояниями управляющего автомата. Блок управления памятью управлял доступом к внешней памяти.

У процессора Intel 80486 производительность значительно улучшилась благодаря использованию конвейерной обработки. По-прежнему выделяются тракт данных, устройство управления и ПЛА с микрокодом.

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

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

В Pentium III 32-битный тракт данных назван целочисленным функциональным блоком (ФБ). Тракт данных для операций над числами с плавающей запятой (ПЗ) назван блоком вычислений с ПЗ. В процессоре также имеется ФБ SIMD для пакетной арифметики с целыми числами и числами с ПЗ. Площадь кристалла для организации внеочередного запуска команд на выполнение, оказалась больше, чем площадь для самого выполнения команд. Кэши команд и данных были увеличены до 16 Кб каждый и на кристалл также была добавлена более большая, но и более медленная кэш-память второго уровня размером 256 Кб.

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

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

# Управление выполнением команд в компьютерах.

Пример: Пусть требуется описать этапы цикла процессора при реализации команды Аdd ax, Mem1, которая размещается в памяти в двух 16-битных словах в виде

|  |  |  |
| --- | --- | --- |
| КОП  Add | АХ, режим адресации 2-го операнда | 🡨(K+0 |
| 15 8 7 0  Mem 1 | | 🡨(K+2 |

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

1. Выборка и дешифрация первого слова команды.

ПСч 🡪 РАП ; переслать значение программного счетчика в регистр адреса контроллера памяти

Код “Чт” 🡪 КонтрЗУ ; загрузить код операции «Чтение» в контроллер памяти

“0” 🡪 ГОТЗУ, пуск ЗУ ; сбросить флаг «Готовность ЗУ», запустить операцию в памяти

Проверить условие:

ГОТЗУ = 1

Если условие «ложно», то ожидать завершения операции в памяти, иначе продолжить дальше.

РДП 🡪 РгКом ; переслать содержимое регистра данных контроллера памяти в регистр команд процессора;

(ПСч)+2 🡪 (ПСч) ; подготовить в ПСч адрес следующего слова команды;

РгКом.КОП🡪ДешКОП ; поле КОП (код опреации) регистра команд подать на вход дешифратора операций;

РгКом.РА🡪Деш.РA ; поле РА (режим адресации) регистра команд подать на вход дешифратора режимов адресации.

1. Выборка второго слова команды.

ПСч 🡪 РАП | Эту последовательность операций

Код “ЧТ” 🡪 КонтрЗУ | обозначим в дальнейшем

“0” 🡪 ГОТЗУ, пуск ЗУ | как “ЧТ”ЗУ

Ожидание ГОТЗУ = 1 |

РДП 🡪 БуфРг ; переслать содержимое регистра данных контроллера памяти в БуфРг (буферный регистр процессора)

ПСч + 2 🡪 ПСч ; подготовить в ПСч адрес следующей команды.

3. Выборка операнда из Mem1.

БуфРг 🡪 РАП

“ЧТ”ЗУ

РДП 🡪 РгBх АЛУ ; переслать содержимое регистра данных контроллера памяти в РгBх – входной регистр АЛУ.

4. Исполнение операции.

РгAX +РгBх 🡪 РгАХ ; суммирование аккумулятора РгАХ и РгBх АЛУ

Учет переносов

{Qi} 🡪 РгССП ; установка флажков в регистре Слова Состояния Процессора по результату завершения операции.

Для описания элементарных операций, составляющих процесс выполнения команды, будем использовать язык межрегистровых передач (ЯМРП) [6], любой оператор которого описывается в следующем виде:

<ОП\_ЯМРП> : : = [ метка ] условие : список \_ действий

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

Совокупность микроопераций, выполняемых за один такт, называют микрокомандой.

Последовательность микрокоманд, реализующих всю команду, называют микропрограммой.

Примеры описания на языке МРП элементарных операций, использовавшихся выше при реализации этапов выполнения команды, приведены в табл. 4.1

Таблица 4.1.

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

В табл. 4.1, наряду с использовавшимися выше, применяются обозначения: ВЫБК – этап выборки команды, Тi – i-й такт генератора, Уi – i-й сигнал управления, ША – шина адреса, ШУ – шина управления, ШД – шина данных.

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

1. Аппаратный, реализуемый на основе жесткой логики (Wired Logic).
2. Микропрограммный, реализуемый на основе программируемой логики (Stored /Programmed Logic).

## Аппаратный способ формирования управляющих сигналов

Аппаратный способ формирования управляющих сигналов (УС) основывается на использовании автомата управления и реализации его схемным (аппаратным) образом. Под автоматом понимается абстрактный дискретный процесс, который определяется тремя множествами и двумя функциями. К используемым множествам относятся:

1. Множество состояний процесса S = {S1, S2, . . . , Sk}.

Это множество конечно и поэтому автоматы также называются конечными.

1. Множество входных воздействий X = {X1, X2, . . . , Xn} автомата, поступающих в дискретные моменты времени ti. В качестве входных воздействий автомата для управления выполнением команд ВМ рассматриваются: коды операций, режимы адресации, тактовые сигналы Ti, состояния устройств Qi и т.п.
2. Множество выходных сигналов Y = {Y1, Y2, . . . , Ym} автомата, также формируемых в дискретные моменты времени ti.

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

Функциями, описывающими поведение автомата являются:

1. Функция переходов δ, определяющая правила перехода автомата из одного состояния в другое:

T: S x X 🡪 S | S (t + Δ) = δ[s(t),x(t)], где Δ > 0 – длительность такта.

1. Функция выходов λ, определяющая правила формирования выходных сигналов автомата:

| λ[s(t), x(t)], эта функция соответствует поведению

R: S x X 🡪 Y | Y (t + Δ) = < автомата Мили (Mealy).

| λ[S (t + Δ)], эта функция соответствует поведению

автомата Мура (Moore).

Вид функции выходов зависит от типа автомата управления – для автомата Мили сигнал управления вырабатывается в момент перехода автомата из одного состояния в другое и является импульсным, для автомата Мура – сигнал управления вырабатывается после перехода автомата в некоторое состояния и является потенциальным. Структурная схема синхронного автомата управления представлена на рис. 4.1.

Входы х1 Комбинационная y1 Выходы

схема

хn ym

Текущее Si Элементы памяти S’iСледующее

состояние (задержка на Δ) состояние

Рис. 4.1

Комбинационная схема – это логическая схема, у которой выходные сигналы определяются входными сигналами в текущий момент времени. Элементы памяти обеспечивают задержку на один такт Δ. Поэтому переменные следующего состояния S’iбудут определять текущее состояние Siв следующий момент времени t + Δ. Отметим основные преимущества и недостатки аппаратного способа формирования УС:

1. Основным достоинством аппаратного способа формирования УС является максимально достижимое быстродействие управления выполнением команд процессор.
2. Основной недостаток аппаратного способа формирования УС является жесткость структуры автомата управления, требующая его полного переконструирования даже при небольшом изменении состава входных или выходных сигналов, функций переходов или выходов.
3. Синтез и реализация автомата с большим количеством состояний является сложной научно-технической проблемой.
4. Нерегулярность структуры автомата затрудняет его реализацию с помощью серийно выпускаемых интегральных схем (ИС) и требует их выполнения в заказном варианте.

## Микропрограммный способ формирования управляющих сигналов

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

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

Структурная схема устройства микропрограммного управления представлена на рис. 4.2.

КОП

РгКом

РгАМК

n

МПЗУ Схема ГТ

синхронизации

m

РгМК

ОпЧМК(m`) (m``)АЧМК

ФАСлМК

ФормУС

{Уi} {Qi} из АЛУ и РгСостВУ

Рис. 4.2.

На рисунке приняты следующие обозначения: МПЗУ – микропрограммное запоминающее устройство (память микропрограмм), РгКом и КОП – соответственно регистр команд (машинных) и его поле кода операции, РгАМК – регистр адреса текущей микрокоманды длиной n бит, РгМК – регистр текущей микрокоманды длиной m бит, ОпЧМК и АЧМК – соответственно операционная (длиной m’) и адресная (длиной m”) части микрокоманды, ФАСлМК – формирователь адреса следующей микрокоманды, ФормУС – формирователь управляющих сигналов Уi, ГТ – генератор тактов, РгСостВУ – регистр состояния внешнего устройства.

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

1. Регулярная структура устройства управления позволяет *реализовать* *его на основе серийных БИС.*
2. *Большая гибкость* в плане изменения или специализации набора команд ВМ путем замены (в случае ПЗУ) или перезаписи (в случае ППЗУ) микропрограммной памяти.
3. Возможность *рационального распределения аппаратного и микропрограммного управления* порождает семейство машин с одним набором команд, но разной производительностью.
4. Микропрограммное управление предоставляет возможность *эмуляции* на микропрограммном уровне новых архитектур машин на имеющейся инструментальной машине.
5. Микропрограммная реализация системных программ (компиляторы, файловые системы) значительно повышает производительность ВМ и широко используется в суперкомпьютерах.

Основные недостатки микропрограммного управления:

1. *Снижение производительности* процессора из-за замедления выполнения каждой микрокоманды, связанного с обращением к МПЗУ.
2. *Увеличение площади кристалла*, необходимой для размещения устройства МПУ.

**Способы кодирования микрокоманд**

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

1. *Горизонтальное или унитарное кодирование* (по существу, отсутствие какого-либо кодирования).

Как видно из рис. 4.3, здесь каждый бит ОпЧМК определяет выполнение только одной микрооперации (МОп). Это позволяет в одной МК задавать любые сочетания МОп и соответствующих им УС Уi , обеспечивая их параллельное выполнение. Кроме того, отсутствует необходимость в декодировании ОпЧМК. В то же время длина ОпЧМК определяется максимально возможным числом микроопераций

LОпЧМК = (NМОп)max

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

ОпЧМК

У1 У2 Уm`

Рис. 4.3

Достоинствами горизонтального кодирования являются:

1. простота и высокая скорость формирования управляющих сигналов;
2. максимальный параллелизм выполнения МОп.
3. *Вертикальное кодирование микрокоманд* .

В случае вертикального кодирования каждой микрооперации присваивается отдельная двоичная комбинация, например, соответствующая двоичному коду ее номера в общем списке возможных МОп. Рарядность этого двоичного кода будет определять длину ОпЧМК в соответствии с выражением

LОпЧМК = Int [log2 (NМОп)max] +1, где Int(Х) – целая часть числа Х .

При этом ОпЧМК будет иметь минимальную длину и это приведет к уменьшению затрат памяти на хранение микропрограмм. С другой стороны, для распознавания кода микрооперации необходим дешифратор МК, преобразующий код МОп в соответствующий УС и требующий на это некоторое время. Кроме того, в каждой МК можно задать только 1 МОп, что увеличивает длину микропрограммы и время ее реализации.

1. *Смешанное кодирование микрокоманд.*

В этом случае весь набор микроопераций делится на группы, связанные с управлением конкретными устройствами (АЛУ, память и т.п.), как показано на рис.4.4. В каждую группу включаются взаимно несовместные МОп, которые не встречаются вместе в одной МК. При этом УС, формируемые в одном и том же такте, оказываются в разных группах и могут выполняться параллельно.

Управление регистрами Управление АЛУ Управление памятью

\_\_\_

ОпЧМК

⇓ n1 ⇓n2 ⇓n3

ДШ 1 ДШ 2 ДШ 3

↓ … ↓ ↓ … ↓ ↓ … ↓

{Yi} {Yi} {Yi}

2 n1 2 n2  2 n3 (число

выходов ДШ)

Рис. 4.4

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

При использовании К независимых групп длина ОпЧМК определяется соотношением

К

LОпЧМК = ∑ ni , где ni = Int[Log2(NМОп)i] + 1

i=1

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

**Способы адресации микрокоманд***.*

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

# Компьютеры с сокращенным набором команд

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

Система команд первого и частично второго поколений машин содержали не более пятидесяти команд. Основная проблема, по которой набор команд не расширялся, – это цена аппаратуры (и управляющая, и обрабатывающая части процессора реализовывались аппаратно), а также необходимость программирования в кодах (программист не мог запомнить большое количество команд). Период доминирования аппаратного управления 50-е – начало 60-х годов, можно назвать «эрой аппаратчиков».

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

1. Минимизация длины кода программы
2. Упрощение реализации компиляторов за счет снижения семантического разрыва между ЯВУ и машинными командами.

Это вызвало рост набора команд компьютеров за счет увеличения их сложности и увеличения числа форматов от ~50 до ~300 команд (рекордсменом был Vaх11/780, у него было 303 команды). Компьютеры с большим набором команд и разнообразием их форматов получили название CISC-компьютеров (Complex Instruction Set Computer – машина со сложным набором команд). Для них характерно увеличение сложности и соответственно размеров микропрограммного устройства управления, которое интерпретировало выполнение этих команд. К тому времени благодаря технологическим достижениям тактовая частота процессоров стала достигать 100 Мгц и повышение производительности требовало размещения всех частей процессора на одном кристалле для сокращения длины соединений его элементов. В то же время микропрограммное управление из-за своей сложности стало занимать до шестидесяти процентов площади кристалла, что либо не допускало использования эффективных средств арифметической обработки данных, либо требовало размещения частей процессора на разных кристаллах. Все это приводило к существенному ограничению производительности, увеличивало сроки разработки и снижало выход годных кристаллов.

В 80-х годах рядом исследователей было замечено, что при выполнении большинства программ наиболее активно используется около 30% сравнительно простых команд арифметики и управления. Постепенно стало формироваться направление развития архитектуры компьютеров, требующее чтобы система команд процессора содержала минимальный набор наиболее часто используемых и наиболее простых команд (возврат к примерно 50 командам). Это направление получает название компьютеров с сокращенным набором команд или RISC-компьютеров и имеет лозунг: «Меньше команд, выше скорость выполнения!».

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

1. Основной набор команд не должен интерпретироваться микрокомандами, а должен выполняться аппаратным обеспечением.
2. Все команды должны иметь одинаковую длину и минимальное число форматов (обычно не более 2–3), это упрощает логику управления при выборе и при исполнении команды.
3. Любая команда основного набора должна выполняться за один машинный цикл, обратно пропорциональный тактовой частоте процессора (стандартом является команда сложения регистра с регистром, занимающая от 3–10 нс); это достигается одновременным (параллельным) выполнением максимально возможного числа команд путем конвейеризации или использования нескольких обрабатывающих узлов
4. Обращение к памяти производиться только по специально выделенным командам работы с памятью типа: Load – загрузка и Store – сохранение, а вся обработка данных должна вестись в регистровом формате; при этом количество регистров должно быть велико (100 и более).
5. Система команд должна обеспечивать поддержку компиляции с конкретного языка программирования (компиляторы для RISC на порядок сложнее, чем компиляторы для CISC).

Немного истории: Идея создания RISC-компьютера впервые была реализована в машинах Cray–1. В 1975 г. сотрудник фирмы IBM John Cook впервые попытался применить RISC-архитектуру при создании модели IBM7095, производительность при этом повысилось в два – три раза. Само название RISC появилось в середине восьмидесятых годах в университете Беркли, где под руководством Дэвида Паттерсона и Карло Секвина была создана машина RISC–1, а затем последовал RISC–2, позже принятый за основу машин семейства SPARC фирмы Sun. Почти одновременно в Стэнфордском университете был разработан процессор MIPS, положивший начало серийному выпуску машин R4000 – R10000 фирмы MIPS.

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

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

*Формат команд и структура процессора RISC–1*

В большинстве RISC-процессоров команды являются трехадресными (рис. 5.1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 7 | 1 | 5 | 5 | 1 | 13 |
| КОП | Усл | Dest | SRC1 | IMM | SRC2 |
| 31 25 | 24 | 23 19 | 18 14 | 13 | 12 0 |

Рис. 5.1

КОП – код операции.

Усл – бит условия (для команд переходов).

Dest – номер регистра назначения (длина пять бит – NPOH = 32).

SRC1 – номер регистра-источника 1.

SRC2 – номер регистра или непосредственного значения источника 2:

если IMM = 1 ⇒ SRC2 – непосредственное данное, если IMM = 0 ⇒ SRC2 – регистр.

В процессоре RISC-1 используется два вида формата команды.

1. Rd ⇐ Rs1 oper S2– для выпонения операций обработки;
2. Rd ⇔ M((Rs1) + S2) – для выпонения обмена с памятью.

Шина процессор-память

Кэш команд

АЛУ

Буфер предвыборки

и логика управления Блок Блок

Рг1 Рг2

Логика Регистр Логика

выборки резуль- выборки

Рг1 тата Рг2

Структура процессора RISC -1 показана на рис. 5.2.

Рис. 5.2

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

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

1. Двухадресная обработка команд в CISC-компьютерах с использованием ячеек памяти.

Add b, a

Add d, c

Mull d, b

Store e, d

║ ВК | ДШ | ОА | ВО | ИСП | ЗР ║ ВК | | |

║ ВК | ДШ | ОА | ВО | ИСП | …… | ЗР ║ | |

║ ВК | ДШ | ОА | …… | …… | …… | ВО | | |

║ …… | …… | …… | …… | …… | …… | ВК | | |

ВК – выборка команды, ДШ – дешифрация, ОА – обработка адресов, ВО – выборка операндов,

ИСП – исполнение операции, ЗР – запись результата, **…….** – такт ожидания.

1. Трехадресная обработка команд в RISC-компьютерах с использованием регистров.

S1 S2 d

Add R1, R2, R6

Add R3, R4, R7

Mull R6, R7, R5

║ ВК | ДШ | ИСП ║

║ ВК | ДШ | ИСП ║

║ ВК | ДШ | ИСП ║

Основная проблема: при большом количестве используемых регистров: надо сохранять их в памяти при переключениях с подпрограммы на подпрограмму. Решение проблемы сохранения и восстановления набора регистров при вызове подпрограмм предлагается реализовать путем организации так называемых “Окон перекрытия регистров” – MORS – Multiple Overlapping Register Set (Rolodex в RISC–1).

Сто тридцать восемь регистров в RISC-1 распределялись между 8 программами (по 32 регистра) как показано на рис. 5.3.

Физические регистры Логические регистры

Проц. А

R137 R31A

HIGH A

R132 R26A

R131 R25A

Окно А LOCAL A

R122 R16A Проц. B

R121 R15A R31B

LOW A / HIGH B

R116 R10A R26B

R115 R25B

Окно В LOCAL B

R106 R16B Проц. С

R105 R15B R31C

LOW B / HIGH C

R100 R10B R26C

R99 R25C

Окно С LOCAL C

R90 R16C

R89 R15C

LOW C

R84 R10C

**. . .**

R9

Global R

R0

Рис. 5.3

Передача параметров (до 6 штук) между процедурами реализуется без затрат времени просто за счет

перекрытия логических регистров (физические регистры совпадают).

Недостатки:

1. Ограниченное число параметров при передаче через регистры.
2. Ограниченное число подпрограмм, которым могут предоставляться регистровые окна.

*Арифметические особенности RISC-процессоров*

В RISC-процессорах имеется возможность увеличения аппаратной поддержки арифметических операций благодаря уменьшению места на кристалле для размещения управляющей части процессора.

Все RISC-процессоры по размерам обрабатываемых данных удовлетворяют стандарту ANSI:

* с фиксированной запятой – 32 бит
* с плавающей запятой – 64 бит

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

Рассмотрим пример сравнения использования команд CISC- и RISC-процессоров на примере компьютеров VAX и SPARC.

|  |  |  |
| --- | --- | --- |
|  | VAX  (CISC) | SPARC  (RISC) |
| **Статика** |  |  |
| Число строк | 1 | 1.5 |
| Операции обращения к памяти | 1 | 1.5 |
| Фиксированная арифметика | 1.5 | 1 |
| Плавающая арифметика | 1 | 3…4 |
| Переходы | 1 | 2.5…3 |
| **Динамика** |  |  |
| Число выполненных строк | 1 | 3…4 |
| Обращения к памяти | 1 | 3…5 |
| Фиксированная арифметика | 2…2.5 | 1 |
| Плавающая арифметика | 1 | 2…2.5 |
| Переходы | 1 | 1…1.5 |

Время выполнения программы упрощенно можно оценить по следующему выражению

Tпрограммы = Nкоманды x Cтакт/ком x Tтакт,

где Tтакт – длительность такта, и для параметров CISC- и RISC-процессоров справедливы соотношения:

NкомRISC = (1.5 - 2.0) \* NкомCISC,

Cтакт/ком. CISC= (5 - 10) \* Cтакт/ком. RISC,

TCISC = (3 - 4) \* TRISC.

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

Тпрогр.CISC = (10 - 20)Тпрогр.RISC.

Основные ***достоинства RISC- процессоров:***

1. Повышение производительности обработки программ вычислительных задач.
2. Благодаря использованию простых команд и минимума их форматов сокращается время разработки RISC-процессора.
3. Улучшение технологичности RISC-процессоров благодаря большей свободе в размещении их элементов на кристалле интегральной схемы и повышение вероятности выхода годных схем.

***Недостатки RISC- процессоров:***

1. Нарушение основных принципов программирования:

а) Минимум длины исполняемого кода программы.

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

1. Сложность построения компилятора, поскольку программа с языка высокого уровня должна транслироваться в микрокод с оптимизацией использования регистров.
2. Высокие требования к быстродействию памяти.

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

# Организация памяти в компьютере

# 6.1. Назначение и основные характеристики памяти

Память используется для хранения следующих объектов:

1. Компьютерные программы.
2. Состояния всех устройств.
3. Данные (постоянные или переменные).

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

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

1. МАЕП – минимально адресуемая единица памяти.

В зависимости от вида данных:

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

1 байт (арифметические данные, команды).

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

Основные характеристики памяти:

1. **Емкость** (обозначается С) с диапазоном: 1 байт (регистр памяти) – n\*100 Г байт (винчестер, оптический диск).
2. **Быстродействие** (обозначается Т) с диапазоном: n\* 1нс (регистровая память) – n\* 10 с (магнитная лента, оптический диск).

Чем больше емкость памяти, тем обычно меньше ее быстродействие. Для преодоления противоречия емкости и быстродействия используется иерархическая организация памяти.

Основные параметры, характеризующие быстродействие памяти:

а) t ВЫБОРКИ – время от запуска памяти для считывания данного и до появления его в буферном регистре (не включает установку и дешифрацию адреса).

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

Как правило, выполняется соотношение tОБР ≥ 2t ВЫБ.

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

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

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

1. **Плотность записи** (бит /см2), зависит от типа среды хранения информации, наиболее высокая плотность у оптических накопителей.
2. **Стоимость хранения** одного бита – важна для пользователя с финансовой точки зрения.

# 6.2. Основные среды хранения информации

1. Магнитная среда.

Исторически самые первые запоминающие устройства использовали магнитную среду, где в качестве носителя информации использовались магнитные материалы, в настоящее время применяются только в устройствах внешней памяти из-за низкого быстродействия.

1. Среда с накоплением зарядов.

В данном случае в качестве элемента памяти используются конденсатор и транзистор, позволяющие хранить один бит информации. В зависимости от вида материалов, различают: биполярную полупро-водниковую память (более быстрая) и память на МОП-структурах (металл – окисел – полупроводник), более медленную, но дешевую. Из-за разряда конденсатора информация хранится краковременно и ее надо регенерировать каждые 5-10 мсек.

1. Память на активных элементах с усилительными свойствами.

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

1. Оптические запоминающие устройства.

Запись информации осуществляется лазерным лучом, а представление информации определяется либо различными свойствами прохождения луча через среду, либо поляризацией материала среды (достаточно высокая плотность записи и малая цена хранения одного бита информации).

# 6.3. Методы доступа к данным.

Запоминающие устройства (ЗУ) отличаются способом доступа к данным.

1. Последовательный доступ. Ориентирован на хранение последовательности блоков данных (записей) и для доступа к нужному элементу (байту или слову) надо прочитать все предшествующие ему данные. Время доступа зависит от положения записи среди других и положения элемента в записи. Примером может служить ЗУ на магнитной ленте.
2. Прямой доступ. Доступ к записи осуществляется адресно, а к элементу внутри записи - последовательно. Соответственно, время доступа - переменное в небольших пределах. Примером может служить ЗУ на магнитных дисках.
3. Произвольный доступ. Каждая ячейка памяти имеет свой физический адрес. Обращение к любой ячейке производится в произвольной очередности и занимает одинаковое время. Примеры – основная память типа RAM (Random Access Memory) и постоянные ЗУ – память типа ROM (Read Only Memory).
4. Ассоциативный доступ (АЗУ) – доступ к ячейкам памяти осуществляется по их содержимому, а не по адресу путем параллельного сравнения содержимого всех ячеек с заданным образцом.

# 6.4. Память с произвольным доступом (ППД)

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

Шина адреса

ш

Рг А и

н

а

Дш А у

п

Контроллер р

Накопитель управления а

чтением/ в записью л

БРГД е

н

и

Шина данных я

я

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

Рис. 6.1

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

В случае реализации ЗЭ на основе конденсатора и запирающего ключа ППД называется динамической (DRAM – Dynamic RAM) и требует регенерации, а в случае реализации ЗЭ на основе триггера ППД называется статической (SRAM – Static RAM) и сохраняет информацию, пока подается питание. Динамическая ППД - более дешевая и медленная, но имеет большую емкость, а статическая ППД - более быстрая, дорогая и ввиду малой емкости применяется только как быстрая буферная память (память типа Кэш).

*Режимы работы ППД*

Для повышения быстродействия обращения к памяти применяются различные режимы работы ИМС памяти, позволяющие: исключить повторные указания адреса строки, выполнять передачу данных по обоим фронтам импульса спхронизации и выполнять групповую (пакетную) передачу данных, требующую задания адреса столбца только для 1-го данного пакета. В качестве примеров ускоренных режимов обращения к памяти можно привести:

1. Режим быстрого страничного доступа FPM (Fast Page Mode) - режим, при котором при многократном последовательном обращении к одной и той же строке номер строки задается только первый раз, а кроме того адрес столбца меняется по заднему фронту сигнала RAS (системная шина меньше двадцати пяти мегагерц).
2. Режим удвоенной скорости DDR (Double Data Rate) – передача данных по обоим фронтам импульса синхронизации.
3. EDO (Extended Data Out) - режим при котором адресация нового столбца осуществляется до завершения предыдущего, производительность повышается примерно вдвое (системная шина от пятидесяти до шестидесяти мегагерц).
4. SDRAM (Synchronous Dynamic RAM), ориентирована на обработку пакетов из четырех 32-битных или 64-битных слов, отличается общей синхронизацией управляющих сигналов от общего сигнала (системная шина выше семидесяти пяти мегагерц).
5. RDRAM (Rambus Direct RAM) – режим, сочетающий DDR с асинхронным блочно-ориентированным протоколом передачи, при котором передача данных между контроллером и памятью передается пакетами.

Более подробные сведения по различным режимам обращения к ППД можно получить в [1,2].

## 6.5. Блочная организация основной памяти.

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

1. Старшие разряды адреса А определяют номер банка b, а младшие – адрес w ячейки в банке; такая организация называется чисто блочной;
2. Номер банка определяется по правилу b = A mod B, а адрес ячейки w = A div B; такая блочная организация называется циклической и для нее младшие разряды адреса А определяют номер банка b, а старшие – адрес w ячейки в банке.

Примеры структур блочной организации ОП для памяти емкостью 512 слов, состоящей из 4-х банков, показаны для чисто блочной схемы - на рис. 6.2, а для циклической блочная схемы - на рис. 6.3.

А w- адрес ОП (0-511)

Дешифратор номера банка памяти

b - номер банка (0-3)

|  |
| --- |
| Банк 0 |
| 0 |
| … |
| 127 |

|  |
| --- |
| Банк 1 |
| 128 |
| … |
| 255 |

А8 А7 А6 … А0

|  |
| --- |
| Банк 2 |
| 256 |
| … |
| 383 |

|  |
| --- |
| Банк 3 |
| 384 |
| … |
| 511 |

РА Памяти

Блок управления

Запрос А8..А0

Мультиплексор

ШУ ША ШД

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

Рис. 6.2 Чисто блочная организация ОП

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

Дешифратор номера банка памяти

А1, А0

А8..А2

РАП-3

РАП-2

РАП-1

РАП-0

|  |
| --- |
| Банк 0 |
| 0 |
| … |
| 508 |

|  |
| --- |
| Банк 1 |
| 1 |
| … |
| 509 |

Блок упра- вле-ния

|  |
| --- |
| Банк 2 |
| 2 |
| … |
| 510 |

|  |
| --- |
| Банк 3 |
| 3 |
| … |
| 511 |

РгД-3

РгД-2

РгД-1

РгД-0

Зап А8..А0

Мультиплексор

рос

ШУ ША

ШД

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

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

Циклическая блочная организация использует принцип расслоения памяти или чередование адресов (interleaving) для параллельного доступа к смежным ячейкам памяти, размещаемым в различных банках. Поскольку в каждом такте на шине адреса задается адрес только одной ячейки, квазипараллеьное обращение производится со сдвигом на 1 такт. Адрес ячейки запоминается в индивидуальном регистре адреса (РАП) и дальнейшие операции по доступу к ячейкам каждого банка протекают независимо. Среднее время доступа к ОП может сокращаться почти в В раз. Поэтому в функциональном отношении такая память может рассматриваться как единое ЗУ, емкость которого равна суммарной емкости банков, а быстродействие соответствует быстродействию одного банка, помноженному на число банков.

## 6.6. Постоянные запоминающие устройства (ПЗУ - ROM)

Используется для хранения фиксированных микропрограмм, подпрограмм и констант (BIOS). По технологии изготовления и способу использования различают:

1. МПЗУ (масочное ПЗУ) – устройство, в котором запись информации осуществляется фирмой-изготовителем путем выжигания связей (участков) в процессе изготовления модуля памяти.
2. ППЗУ (программируемое ПЗУ (PROM)) – устройство, которое в исходном виде поставляяется пользователю, и он сам прошивает необходимые связи для записи требуемой информации с помощью специального прибора – программатора). После такой процедуры ППЗУ не может больше перепрошиваться.
3. ПППЗУ (перепрограммируемое ПЗУ (EPROM)) – это устройство аналогично ППЗУ, только с возможностью стирания информации ультрафиолетом в течение длительного времени (более 10 минут) и затем записи новой информации.

## 6.7. Ассоциативные запоминающие устройства (АЗУ)

В отличие от других видов памяти доступ к информации в АЗУ осуществляется не по адресу размещения данного, а по содержимому – значению самого данного или его части. При этом в АЗУ хранимая информация сравнивается с некоторым образцом и проверяется их соответствие. Образец, по которому производится поиск информации, называется ассоциативным признаком. На рис. 6.4 показан вариант построения АЗУ, включающий:

1. память данных Х для хранения N m-разрядных слов;
2. регистр ассоциативного признака (разрядностью k ≤ m) для создания образца поиска;
3. регистр маски, позволяющий исключить из признака поиска определенные биты;
4. схемы совпадения, реализующие параллельное сравнение каждого битавсех слов памяти данных с соответствующим битомпризнака поиска и выработки сигналов совпадения;
5. регистр совпадений, каждый бит которого соответствует одной из ячеек памяти и в него заносится 1, если все разряды этой ячейки совпали с одноименными разрядами признака поиска;
6. комбинационная схема, формирующая сигналы, характеризующие результаты поиска.

Регистр ассоциативного

признака key [1..k], k ≤ m

Адрес считывания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1  2  **.**  **.**  **.**  N | 1 2 . . . . . . m | | | |  |
|  |  | **…** |  |
|  |  | **…** |  |
| **. . . . .** | | | |
|  |  | **…** |  |
|  | | | |

Р с

е о

г в

и п

с а

т д

р е

н

и

й

Шифратор

Схемы

cовпадения

i = 1..N, j=1..k

Регистр маски Rm

Комбин схема

конт-роля

ассо-

циации

а0

а1 1..N 1..N

а2

Память данных Х Xi[j] &Rm[j] = key[j] &Rm[j] N-разрядное

(N m-разрядных слов) слово

Рис.6.4. Структура ассоциативного ЗУ

Поскольку результаты поиска могут быть неоднозначными, комбинационная схема контроля ассоциации по содержимому регистра совпадений формирует признаки ai , показывающие, что:

1. а0 - слово не найдено, 2) а1 - найдено только 1слово, 3) а2 - найдено много слов

При считывании данных производится контроль ассоциации и при а0 =1, считывание отменяется,

при а1 =1 считывается слово, на которое указывает 1 в регитсре совпадений, а при а2 = 1 сбрасывается самая старшая 1 в регистре совпадений и извлекается соответствующее ей слово памяти данных. Повторяя последнюю операцию можно считать все слова.

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

# 6.8. Иерархическая система памяти

Для достижения компромисса между емкостью и быстродействием памяти, а также для снижения влияния разрыва в быстродействии процессора и пмяти, в современных коиьютерах память реализуется в виде иерархической системы, показанной на рис. 6.5. Более высокий уровень ЗУ харктеризуется большим быстродействием, меньшей емкостью и большей стоимостью хранения 1 бита информации.

Иерахическая система памяти состоит из следующих уровней:

1. Сверхоперативная память (СОЗУ), которая реализуется на регистрах процессора, а также Кэш – память 1 уровня размещаются на одном кристалле с процессором и являются наиболее быстрыми видами памяти.
2. Буферная память, создаваемая на основе Кэш – памяти 2 и 3 уровней, вместе с Кэшем 1 уровня служит для согласования скорости процессора и основной памяти. Образует с основной памятью систему буферизованной памяти. Для программиста эта память является прозрачной (скрытой), поэтому и называется кэш-памятью, она реализуется на биполярных элементах и размещается на одной плате с процессором. Для Гарвардской архитектуры буферная память может разделяться на кэш команд и кэш данных.
3. Основная память - блочно организованная память прямого доступа, обеспечивающая хранение системных и пользовательских программ и данных, выполняемых в текущем интервале времени.
4. Дисковый Кэш – буферная память, используемая для согласования быстродействия основной и дисковой памяти, реализуется на полупроводниковых элементах и может быть отдельным ЗУ, включаемым в состав НМЖД, или фрагментом основной памяти.
5. Дисковая магнитная память – медленная (по сравнению с ОП) память большой емкости, размещаемая на жестком диске (винчестере), и служащая либо для хранения программ и данных, не используемых в настоящее время, либо для расширения доступного программисту адресного пространства путем организации совместно с основной памятью системы виртуальной памяти.
6. Архивная память – многотомные накопители информации большого объема, реализуемые на магнитных лентах или оптических дисках. Имеют самое низкое быстродействие и предназначены для долговременного хранения данных без разрушения.

Процессор

(СОЗУ, Кэш1ур.)

Буферная память Система буферизации ОП

(Кэш 2ур.,Кэш 3ур.)

Основная память Система виртуальной памяти

Дисковый Кэш Система буферизации диска

Дисковая

магнитная

память

Оптические

диски

Р

Х

Магнитные И   
 ленты В

А

Рис.6.5. Иерархическая система памяти

В целом следует отметить, что иерархическая организация памяти ведет к уменьшению общей стоимости хранения и доступа к информации при заданном уровне производитльности.

## Организация кэш-памяти.

Использование кэш-памяти основано на свойстве ***локальности*** программ по обращению, имющем три составляющих:

1. Пространственная локальность программ, заключающаяся в том, что при выборке команд из памяти адрес очередной команды либо следует непосредственно за адресом предыдущей, либо находится вблизи него;
2. Пространственная локальность данных, связанная с тем, что за счет широкого использования структурированных данных при обращении к данным, как правило, используются последовательные ячейки памяти;
3. Временная локальность программ, заключающаяся в том, что в течение достаточно длительных интервалов времени исполняемая программа использует сравнительно небольшой диапазон адресов команд и\или небольшой диапазон адресов данных. Это обусловлено наличием в программе циклов и подпрограмм, а также необходимостью длительной обработки массивов данных.

Идея использования кэша как буферной памяти заключается в наличии двух видов памяти:

быстрой памяти малой емкости М1 (n1, tобр1) и медленной памяти большой емкости М2(n2, tобр2), параметры которых: число ячеек - ni и время обращения tобрiхарактеризуются неравенствами

n1 << n2 и t обр1 << t обр2.

Если данные имеются в быстрой кэш-памяти, то они выбираются за время t обр1, а если отсутствуют, то за время t обр1 + t обр2 данные выбираются из основной памяти и одновременно подгружаются в кэш-память.

Если благодаря свойству локальности удается исполняемый фрагмент программы и обрабатываемый массив данных разместить в кэш-памяти, то вероятность 0 < α << 1 отсутствия команд или данных в кэш - памяти удается сделать достаточно малой - обычно α ≈ 0.02…0.05. Тогда среднее время обращения к такой системе буферизованной памяти будет

М {t обр} = (1 - α) \* t обр1 + α \* (t обр1 + t обр2) = t обр1 +α\* t обр2 ≈ t обр1

Поскольку емкость кэш-памяти значительно меньше емкости основной памяти (ОП), то эпизодически происходит обмен информацией между КЭШем и ОП, чтобы в КЭШе оказывалась актуальная на данное время информация. При этом обмен происходит *блоками* одинакового размера, которые в кэш-памяти принято называть *строками*. Когда процессор пытается выбрать слово из памяти, то сначала он ищет это слово в КЭШе и если находит, то выбирает его из КЭШа. Такая ситуация называется *попаданием* (hit), а при отсутствии слова в КЭШе возникает ситуация *промах* ( miss) и слово выбирается из ОП. Одновременно из ОП в кэш-память пересылается блок данных, содержащий это слово.

Так как число блоков ОП больше числа строк КЭШа, строка КЭШа не может быть выделена постоянно одному блоку ОП. Поэтому каждой строке КЭШа соответствует признак (*тег*), показывающий, копия какого блока ОП хранится в ней в данное время. В качестве тега обычно используется часть адреса ОП. При этом для разных типов (уровней) КЭШа применяются различные способы отображения ОП на кэш-память.

Рассмотрим различные способы отображения блоков ОП на кэш-память на следующем примере.

Соп = 256 Кслов=218  слов, Скэш  = 2 Кслов=211  слов, Сблока = 16 слов (С - емкость);

Моп = Nбл.оп. = 256К/16 = 16K = 214 блоков., Мкэш = 2К/16 = 128 = 27 блоков (М – число блоков);

Lадр.оп = 18бит = 14бит (адрес блока ОП) { 7бит (тег) +7бит (строка кэша)} + 4бит (слово в блоке)

Lадр.кэш = 11бит= 7бит (адрес строки кэша) + 4бит (слово в блоке) (L – длина адреса в битах)

**Прямое отображение блоков ОП на кэш-память**

При *прямом отображении* адрес строки i кэш-памяти, на которую может быть отображен блок j ОП, однозначно определяется выражением

**i = j mod Mкэш**

В нашем примере i = j mod 128, где i = [0,127], j = [0,16383] и на строку КЭШа с номером i отображается каждый 128 блок ОП, начиная с блока i , что поясняется на рис. 6.6 . Здесь 14-битный адрес блока ОП разбивается на два поля: 7-битный тег (7 старших разрядов адреса) и 7-битный номер строки кэш-памяти, на которую может быть отображен этот блок ОП. При этом поле тега определяет, какой из списка блоков ОП, закрепленных за данной строкой кэша, сейчас адресуется. Когда блок ОП фактически заносится в соответствующую строку кэш-памяти, в поле тегов этой строки нужно записать тег именно этого блока, в качестве тега служат 7 старших разрядов адреса блока.

Память Память Тег

тегов данных 0 1 2 . . . . . . 127 Строка

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | **0** |  | Блок 0 |  | Блок  0 | Блок 128 | Блок 256 | **….…** | Блок 16256 | 0 |
| 1 | **2** | Блок 257 | Блок  1 | Блок 129 | Блок 257 | Блок 16257 | 1 |
| 2 | **1** | Блок 130 | Блок  2 | Блок 130 | Блок 258 | Блок 16258 | 2 |
| …. | **……** | ………. | ……….……….……….………. | | | ………. | …. |
| 126 | **2** | Блок 382 | Блок  126 | Блок 254 | Блок 382 | Блок 16382 | 126 |
| 127 | **127** | Блок 16383 | Блок  127 | Блок 255 | Блок 383 | Блок 16383 | 127 |

7 бит 4 бит

**К э ш п а м я т ь О с н о в н а я п а м я т ь**

Рис.6.6

При несомненной простоте прямого отображения его существенным недостатком является жесткое закрепление строки кэша за определенными блоками ОП. Поэтому при поочередном обращении программы к словам из двух блоков, отображаемых на одну и ту же строку кэша, постоянно будет происходить обновление этой строки, резко снижающее скорость доступа к памяти. Кэш с прямым отображением ввиду своей экономичности используется для построения кэшей 2-го и 3-го уровней сравнительно большого объема.

А**ссоциативное отображение блоков ОП на Кэш-память**

Такой способ отображения, иллюстрируемый рис.6.7, позволяет производить загрузку любого блока ОП в любую строку кэш-памяти. В кэш-памяти используется поле тегов, содержащее 2 тега: тег1 (14 бит), соответствющий адресу блока ОП, и тег2 (7 бит), определяющий адрес блока в КЭШе, а также поле данных, определяющее адрес слова в блоке. Контроллер кэш-памяти выделяет в адресе ОП два поля: поле тегов и поле слова. Поле тега совпадает с 14-битным адресом блока ОП. Для проверки наличия копии блока ОП в кэш-памяти логика управления контроллера кэша должна одновременно сравнить теги 1 всех строк КЭШа на совпадение с полем тега адреса ОП. Это обеспечивается использованием ассоциативной памяти для хранения тегов КЭШа. После нахождения строки КЭШа, тег1 которой совпал с адресом искомого блока ОП, по тегу 2 определяется размещение искомого блока в КЭШе.

Ассоциативное отображение обеспечивает гибкость при выборе строки кэш-памяти для размещения блока ОП, но требует использования дорогостоящей ассоциативной памяти, поэтому применяется только для построения КЭШей 1 уровня (процессорных).

**Наборно-ассоциативное отображение блоков ОП на Кэш-память**

Такой способ отображения сочетает достоинства прямого и ассоциативного способов и позволяет каждому блоку ОП претендовать на одну из нескольких строк кэш-памяти, объединенных в набор (множество). Можно считать, что в этом случае используется несколько параллельно и согласованно работающих каналов прямого отображения: каждому набору строк КЭШа соответствует жестко заданная группа блоков ОП, а в пределах набора контроллеру кэша по ассоциативному способу приходится выбирать, в какую из строк набора помещать очередной блок данных из ОП. Кэш с наборно-ассоциативным отображением может применяться для построения как первичных, так и вторичных кэшей.

Адрес блока Адрес блока

в ОП (14б) в КЭШе (7б)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | **20** | **2** | 0  1  2  126  127 | Блок 258 |  | Блок 20 |
| ….. |
| 1 | **258** | **0** | Блок 32 | Блок 32 |
| ….. |
| 2 | **3200** | **126** | Блок 20 | Блок 258 |
| ….. |
| … | **…..** | **…** | ………. | Блок 1600 |
| ….. |
| 126 | **32** | **1** | Блок 3200 | Блок 3200 |
| ….. |
| 127 | **1600** | **127** | Блок 1600 | ….. |
| ….. |

Тег1(14 бит) Тег2(7 бит) (4 бит)

**Т е г и Данные**

**К э ш - п а м я т ь О с н о в н а я п а м я т ь**

Рис.6.7

При использовании кэш-памяти возникает две важных группы проблем:

1. Замещение блока в строке кэш-памяти на новый блок ОП при необходимости загрузки последнего в полностью занятую кэш-память. Эта проблема возникает только для ассоциативного и наборно-ассоциативного способов отбражения блоков ОП на строки кэш-памяти. В настоящее время в основном используется два способа замещения блоков ОП в КЭШе:

1. LRU (Least-Recently-Used) – производится замена в кэш-памяти блока ОП, к которому дольше всего не обращались; этот способ требует поддерживать историю обращения к блокам (с помощью счетчиков).
2. FIFO (Fist-in, Fist-out) или обычная очередь – производится замена в кэш-памяти блока ОП, который загружался раньше всех других; это более дешевая, но не столь эффективная стратегия замены блоков в КЭШе по сравнению с LRU-стратегией.

2. Согласование данных в КЭШе и основной памяти.

1. Store through (Write through) – сквозная запись (одновременная запись данного в блок кэша и в блок основной памяти), при этом происходит трата ресурсов и времени на обновление основной памяти. Достоинство: надежный способ согласования данных.
2. Store In (Write back) – обратная запись, обновление блока ОП происходит только при выгрузке строки из КЭШа в ОП. Достоинство - более высокая производительность. Недостаток - временное различие данных в ОП и в КЭШе может привести к конфликтам (проявляется в многопроцессорных компьютерах с общей памятью).

# 6.10. Организация виртуальной памяти

Виртуальная память – система основной и дисковой памяти, организуемая для расширения адресного пространства, доступного программам пользователей. Для виртуальной памяти различают:

1. ФАП (физическое адресное пространство) – совокупность адресов, соответствующих реально адресуемым физическим ячейкам памяти.
2. ЛАП (ВАП) (логическое или виртуальное адресное пространство) – это совокупность адресов, которая может использоваться для доступа к данным.

Очевидно, что емкость ВАП должна значительно превышать емкость ФАП.

Для обеспечения доступа к ВАП из программ пользователей необходимо установить строгое соответствие между данными, размещаемыми в ОП, и данными, размещенными на диске, отражаемое в специальных таблицах соответствия адресов. Для сокращения длины таблицы соответствия адресов информация, хранящаяся в основной памяти и на диске, должна быть разбита на одинаковые (в обоих видах памяти) блоки, для которых и устанавливается соответствие.

Разбиение происходит по двум способам :

1. Страничный;
2. Сегментный

При первом вся память (основная и дисковая) разбивается на одинаковые по размерам блоки, называемые страницами; разбиение происходит независимо от типа и характера размещаемых данных. Говорят, что страничное разбиение ориентируется на физическую память.

При втором разбиение основной и дисковой памяти происходит на сегменты, как правило, соответствующие логически завершенным объектам задачи (программа, массив данных и т.д.); вследствие этого длина каждого сегмента может быть произвольной. Говорят, что сегментное разбиение имеет логическую ориентацию. Первый способ имеет более простые организацию и стратегию подгрузки и замещения страниц в основной памяти. Для второго способа достоинством является удобство организации защиты информации и коллективного доступа к данным.

*Страничную организацию виртуальной памяти* можно пояснить с помощью схемы, показанной на рис. 6.8.

Виртуальный (логический) адрес (ВА)

Базовый адрес Номер Смещение в Регистр

таблицы страниц страницы странице адреса

(в регистре) + Pва d

Рб

Диск

P

m / d р’

Замещение

**3’** страниц

Таблица соответствия **4’**

адресов (таблица страниц)

d

Основная

память (ФА)

Основная

память (ФА)

**1**

**2’**

Page\_fault Карта диска

Page\_hit

**2,5’**

Страничный

кадр

(фрейм)

**3,6’**

Рис.6.8

Сущность преобразования адресов: ВА p**.**d --> ФА p’**.**d

Признак m/d задает «бит присутствия»: 1 – страница есть в ОП, 0 – страница имеется только на диске.

При замещении страниц в основной памяти используются следующие стратегии:

1. FIFO – заменяется страница ОП, которая была загружена первой из имеющихся в ОП страниц.
2. LRU – заменяется страница ОП, к которой дольше всего не было обращений.
3. WS (work set) – рабочее множество: для конкретных интервалов времени создается совокупность страниц ОП, которые активно используются и не замещаются, остальные можно удалить из памяти.

Приведенная выше схема соответствует принципу прямого соответствия адресов, при котором для любого логического адреса есть свой физический адрес.

Для сокращения длины таблицы страниц и времени доступа к ней используется принцип ассоциативного отображения страниц. В этом случае для хранения адресов наиболее часто используемых страниц используется память типа кеш и обращение к ней происходит по следующей схеме (рис. 3.6)

Такая структура называется буфер страничной трансляции (БСТ).

***Сегментная организация виртуальной памяти***

При сегментной организации ВП строка таблицы сегментов должна содержать специальные атрибуты, задающие не только факт присутствия сегмента в ОП, но также его длину и способ доступа к сегменту.

S

Сущность преобразования адресов:

Атрибуты S’ ВА S**.**d --> ФА S’**.**d,

где операция ‘**.**’ означает конкатенацию

Формат строки таблицы сегментов

m/d L R W E АВП Здесь признак m/d – «бит присутствия»

1 nI 1 1 1 nII

L – длина сегмента; R,W,E – доступность по чтению, записи, выполнению; АВП – адрес во внешней памяти.

Одной из проблем виртуальной памяти является фрагментация виртуальной памяти. Фрагментацию обычно различают внутреннюю и внешнюю. Внутренняя возникает от неполного использованния страницы сегмента и образованию участков памяти, доступ к которым затруднен. Внешняя возникает только при сегментной организации виртуальной памяти из-за того, что удаляемый и размещаемый на его место сегменты имеют разные размеры. Для преодоления фрагментации ОС запускает процедуру, называемую «сборка мусора» для объединения свободных участков памяти в сплошные блоки. Для устранения внешней фрагментации и сокращения длины таблицы страниц применяется сегментно-страничная организация (рис. 6.9).

S P d

Номер Номер Смещение

сегмента страницы в странице

+ ФА1

БА Сегмента Таблица стр.1

+

БА ТС1 +

i

ФАi

БА ТС i +

Таблица сегментов Таблица стр. i

Рис.6.9

Сегмент рассматривается не как целый, а разбивается на страницы, именно части страниц будут теряться при фрагментации. Таблицу сегментов также не рассматривают как указатель на сплошное адресное пространство, а как совокупность указателей на различные, не обязательно смежные страницы. Для различных задач могут быть разные таблицы сегментов и разные связи с таблицами страниц. В результате обеспечивается на логическом уровне – сегментная организация, а на физическом – страничная.

# Организация виртуальной памяти в Intel 80386 и более старших моделях.

Эти процессоры могут работать в трех режимах:

1. Реальный режим – работает как быстрый процессор i8086, особенно с 32-битными данными, и имеет адресное пространство памяти до 4Гб.
2. Защищенный режим – использует виртуальную организацию памяти, многозадачную работу, развитую систему защиты памяти (программ и данных).
3. Реализация виртуальной машины i8086 – используется для совместимости с процессором i8086, при этом процессор эффективно реализует программы, написанные для процессора i8086

В защищенном режиме размер виртуальной памяти C ВАП =246б = 64Гб. Это достигается благодаря разбиению ВАП на 16К сегментов, максимальный размер каждого из которых равен 4 Гб.

При организации виртуальной памяти используется три вида адресов.

1. Логический адрес ЛА ∈ ВАП.
2. Физический адрес ФА ∈ ФАП.
3. Линейный адрес – представляет собой объединение базового адреса сегмента и смещения в пределах сегмента.

Любое описание сегмента состоит из двух частей: ***программно-доступной,*** *называемой* ***селектором*** ***сегмента*** и размещаемой в одном из сегментных регистров, ***и скрытой,*** *называемой* ***дескриптором сегмента*** и находящейся в одной из специальных структур в ОП, называемых таблицами дескрипторов. Считается, что дескриптор после выборки размещается в скрытой части регистра сегментов, находящейся в памяти (рис. 6.10).

Доступная (селектор) Скрытая (дескриптор)

15 0 63 0

CS БАС Граница Атрибуты - это дескриптор сегмента

(БАС – базовый адрес сегмента)

DS

GS

Сегментный

регистр Индекс TI RPL (Поля селектора)

Селектор 15 … 3 2 1 0 (Номера битов)

Рис. 6.10

Индекс задает смещение строки с описанием (дескриптором) данного сегмента в таблице дескрипторов.

Поле TI определяет таблицу дескрипторов, на которую ссылается селектор (0 – глобальная, 1 – локальная ).

Поле RPL (0 - 3) - запрашиваемый уровень привелегий (инициатор запроса – программа, которая хочет получить доступ к ресурсам).

Содержание селекторов сегментов формируется операционной системой при загрузке задачи.

*Структура дескриптора сегмента* (рис. 6.11)

31 …24 23 22 21 20 19 … 16 15 14..13 12 11.. 9 8 7 … 0

База G D 0 AVL Граница P DPL S тип А База

База сегмента [0-15] Граница сегмента [0 -15]

Рис 6.11

Основные поля сегмента размещены в различных частях дескриптора: 32-битная база сегмента размещена в трех частях и определяет его место внутри 4-Гбайтного линейного адресного пространства, 20-битная граница сегмента размещена в двух частях и определяет его длину, остальные биты дескриптора образуют поле атрибутов. Длина сегмента, в зависимости от бита дробности (G), задается либо в байтах, либо в страницах: при G=0 – в байтах, а при G= 1 – в страницах (С = 4Кб) и объем будет 220 \* 215 = 232 (4Гб).

Бит D задает тип данных, размещенных в сегменте:

D = 0 – 16-битные данные, D = 1 – 32-битные данные.

AVL – бит, предоставляемый в распоряжение пользователя (available).

Бит P – бит присутствия: 0 – на диске, 1 – в оперативной памяти, биты DPL – уровень привилегий дескриптора (Descriptor Privilege Level), обеспечивающий защиту доступа к сегменту..

При доступе к сегменту запрашиваемый уровень привилегий RPL или текущий уровень привилегий CPL (соответствует уровню привилегий дескриптора кода выполняемой сейчас программы) должны быть больше DPL (в логическом смысле; для сравнения их числовых значений отношение должно быть меньше).

Бит S – бит режима сегмента: пользовательский или системный (User / Supervisor).

Биты типа говорят, является сегмент программой или данными и определяют режим доступа:

|  |  |  |  |
| --- | --- | --- | --- |
| Биты | 11 | 10 | 9 |
| Данные | 1 | E | W |
| Код | 0 | C | R |

R – только чтение; W – только запись; E – бит расширения при загрузке сегмента (1 – при увеличении адресов, 0 – при уменьшении адресов); бит С – признак согласования сегментов (уровень привилегий, с которыми обращается к данному сегменту, всегда приравнивается к уровню данного сегмента – в моделях процессоров выше i386 он существует только для поддержки).

Бит А – бит обращения к сегменту, устанавливается при обращении к сегменту и через некоторое время сбрасывается операционной системой (обеспечивает дисциплины замещения сегментов).

*Алгоритм преобразования виртуального адреса в физический адрес* (рис. 6.12).

База Виртуальный

или

логический

+ Смещение адрес

(ЛА)

+ \* Индекс

Коэффициент

масштабирования (1,2,4,8)

ЛА селектор ЛА смещения

Сегментная трансляция

Линейный адрес

0

PG&PE

1

DIR PAGE offset

Страничная трансляция

Физический адрес ФА

Рис. 6.12

*Схема выполнения сегментной трансляции* (рис. 6.13)

GDT LDT

& Дескриптор & Дескриптор

0

T I 1

Регистр

Индекс TI RPL сегмента База сегмента Граница Атрибуты

15 3 2 1 0 (32) (20) (12)

+ Смещение

Рис.6.13 Линейный адрес (32)

I этап

***Сегментная трансляция*** – это преобразование логического адреса (селектор сегмента и смещение сегмента из команды программы) в линейный адрес. На основании селектора происходит выборка дескриптора из локальной или глобальной таблицы дескрипторов и запись его в скрытую часть сегментного регистра. В результате формируется линейный адрес сегмента (ЛАС) по правилу ЛАС = БАС + смещение.

II этап

***Страничная трансляция*** (Рис.6.14) – это преобразование линейного адреса в физический адрес. Если используется реальный режим или нет разбиения сегмента на страницы, то этот этап пропускается и ФА = ЛАС.

Каталог Страница Смещение Линейный адрес

(10) (10) (12)

PG PE TC1 TC2

CR0

+ + + ФА

dir page

CR3 База адреса каталога

Рис.6.14

CRi – i-й ( i =0..3) регистр управления, PG – бит признака страничной организации сегмента, PE (Protect Enable) – признак установки защищенного режима; (CR0. PG)&(CR0. PE) = 1 - признак страничной трансляции. ТС1 – таблица каталогов, ТС2 – таблица страниц

Размер ФАП = 1К каталогов\* 1К страниц \* 4К смещение = 1k x 1k x 4k= 4Gb.

Формат строки таблицы страниц:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 12 | 11 | 10 7 | 6 | 5 | 4 3 | 2 | 1 | 0 |
| Адрес страни-чного кадра | AVL | 0 | D | A | 00 | U/S | R/W | P |

AVL – бит, предоставляемый в распоряжение пользователя, бит P – бит присутствия страницы, бит D – бит мусора, бит А – бит обращения к странице, бит U/S (User / Supervisor) – бит режима использования страницы: пользовательский или системный, бит R/W (Read/Write) – бит доступа (по чтению/записи).

# Защита памяти в процессоре Intel 80386

Обеспечивается только при CR0.PE =1(защищенный режим). Защита может осуществляться на уровне страниц и на уровне сегментов.

Защита на уровне сегментов и на уровне страниц обеспечивается либо по атрибутам дескриптора сегмента, либо по атрибутам строки таблицы страниц. При защите на уровне сегментов влияет соотношение уровней привилегий:

RPL – запрашиваемый уровень (при помещении селектора в сегментный регистр);

DPL – уровень привилегий дескриптора, фиксируется при первом обращении к данным дескриптора и является минимальным уровнем, при котором данный сегмент доступен.

CPL – текущий уровень привилегий, который соответствует уровню кодового сегмента текущей программы.

***Виды защиты на уровне сегментов:***

1. Контроль типа не зависит от уровня привилегий, определяется дескриптором
2. Контроль границ
3. Ограничение адресуемой области памяти.
4. Ограничение точек входа в процедуру зависят от RPL, DPL, CPL
5. Ограничение набора команд.

*Контроль типа выполняется:*

1. На этапе загрузки селектора в регистр сегмента (проверка типа сегмента, для которого осуществляется загрузка селектора)
2. На этапе обращения к сегменту.

Примеры: В CS может быть загружен только селектор дескриптора программного сегмента; в DS не может быть загружен селектор дескриптора, который не доступен для чтения; в SS не может быть загружен селектор дескриптора, который не доступен для записи.

При обращении:

1. Нельзя записать в кодовый сегмент.
2. Нельзя записать в сегмент данных, если бит записи равен нулю.
3. Нельзя читать кодовый сегмент, если бит чтения равен нулю.

*Контроль границ.* Предотвращает выход за пределы сегмента.

*Ограничение адресуемой области памяти.*  Доступ к памяти возможен при условии

{RPL, CPL} ≤ DPL (численно).

*Ограничение точек входа в процедуру*.

Если CPL = DPL, то возможен вход в процедуру по Call, Jump.

Если CPL ≥ DPL (численно), то возможен только доступ по Call через специальный дескриптор, называемый «вентиль вызова».

*Ограничение набора команд.* Все команды по уровню доступа на выполнение делятся на 3 группы:

1. Свободные команды (непривилегированные) – выполняются в любом режиме.
2. Привилегированные команды – выполняются на уровне ядра.
3. Чувствительные команды, зависят от уровня привилегий, но могут выполняться в любом режиме супервизора (от 0 до 2). В основном это команды ввода-вывода.

***Виды защиты на уровне страниц:***

1. Контроль типа.
2. Ограничение адресной области памяти.

*На уровне страниц* защита зависит только от битов режима U/S и доступа R/W.

# *Организация работы с внешней памятью*

## 7.1. Типы, виды, свойства дисковых накопителей информации.

Выпускаемые накопители информации представляют собой гамму запоминающих устройств (ЗУ) с различным принципом действия, физическими и эксплуатационными характеристиками. ЗУ принято делить на виды и категории в связи с их принципами функционирования, эксплуатационными, физическими и программными характеристиками. Так, например, по принципам функционирования различают следующие виды устройств:

* электронные,
* магнитные,
* оптические
* смешанные – магнитооптические.

Каждый тип устройств организован на основе соответствующей технологии хранения/воспроизведения/записи цифровой информации. Рассмотрим дисковые магнитные накопители – накопители на жестких и гибких магнитных дисках (НЖМД и НГМД).

## 7.2. Магнитные дисковые накопители.

Магнитные ЗУ состоят из собственно устройств чтения/записи информации и магнитного носителя, на который осуществляется запись и с которого считывается информация. Общая технология магнитных ЗУ состоит в намагничивании переменным магнитным полем участков носителя при записи информации и считывания информации, закодированной как области переменной намагниченности. Запись производится в цифровом коде. Дисковые устройства делят на два типа:

* гибкие (*Floppy Disk*)
* и жесткие (*Hard Disk*).

Основным свойством дисковых магнитных ЗУ является запись информации на концентрические замкнутые дорожки носителя с использованием физического и логического цифрового кодирования информации. Плоский дисковый носитель вращается в процессе чтения/записи, обеспечивая обслуживание всей концентрической дорожки, чтение и запись осуществляется при помощи магнитных головок чтения/записи, которые позиционируют по радиусу носителя с одной дорожки на другую. Дисковые устройства, как правило, используют метод записи, называемый методом без возвращения к нулю с инверсией (*Not Return Zero – NRZ*). Запись по методу *NRZ* осуществляется путем изменения направления тока в обмотках головок чтения/записи, вызывающее изменение полярности намагниченности магнитных головок и попеременное намагничивание участков носителя вдоль концентрических дорожек с течением времени и продвижением по окружности носителя.

Накопители на ЖМД объединяют в одном корпусе носители и устройство чтения/записи, а также интерфейсную часть, называемую собственно контроллером жесткого диска. Обычно, рядом с корпусом носителей и головок располагаются схемы управления головками, дисками и интерфейсная часть и/или контроллер.

Информация заносится на концентрические дорожки, равномерно распределенные по всему носителю. В случае большего, чем один диск, числа носителей все дорожки, находящиеся одна под другой, называются цилиндром. Операции чтения/записи производятся подряд над всеми дорожками цилиндра, после чего головки перемещаются на новую позицию.

Диски вращаются постоянно, а скорость вращения носителей довольно высокая (от 4500 до 10000 об/мин и выше), что обеспечивает высокую скорость чтения/записи. По величине диаметра носителя чаще других производятся 5.25, 3.14, 2.3 дюймовые диски.

*Позиционирование головок.*

В настоящее время, для позиционирования головок чтения/записи, наиболее часто, применяются шаговые и линейные двигатели механизмов позиционирования и механизмы перемещения головок в целом. В системах с шаговым механизмом и двигателем головки перемещаются на определенную величину, соответствующую расстоянию между дорожками, дискретность шагов зависит либо от характеристик шагового двигателя, либо задается серво-метками на диске. В системах с линейным приводом головки перемещаются электромагнитом, а для определения необходимого положения служат специальные сервисные сигналы, записанные на носитель при его производстве и считываемые при позиционировании головок. Линейные приводы перемещают головки значительно быстрее, чем шаговые, кроме того они обеспечивают положение головки, наилучшее для считывания с каждой дорожки и этим повышают достоверность считываемых данных. Также все устройства с линейным приводом имеют автоматический механизм парковки головок чтения/записи при отключении питания устройства.

*Парковкой головок* называют процесс их перемещения в безопасное положение. Это - так называемое "парковочное" положение головок в той области дисков, где ложатся головки. Там, обычно, не записано никакой информации, кроме серво - это специальная "посадочная зона" (Landing Zone). Для фиксации привода головок в этом положении в большинстве ЖД используется маленький постоянный магнит, когда головки принимают парковочное положение - этот магнит соприкасается с основанием корпуса и удерживает позиционер головок от ненужных колебаний.

*Плата электроники* современного накопителя на ЖМД представляет собой самостоятельный микрокомпьютер с собственным процессором, памятью, устройствами ввода/вывода и прочими традиционными атрибутами присущими компьютеру.

## 7.3. Основные физические и логические параметры ЖМД

Все накопители соответствуют стандартам, определяемым либо независимыми комитетами, либо самими производителями. Среди множества технических характеристик, отличающих одну модель от другой, можно выделить наиболее важные с точки зрения пользователей и производителей, которые используются при сравнении накопителей различных производителей и выборе устройства:

* Диаметр дисков (disk diameter)
* Число поверхностей (sides number)
* Число цилиндров (cylinders number)
* Число секторов (sectors count)
* Число секторов на дорожке (sectors per track)
* Частота вращения шпинделя (spindle speed)
* Время перехода от одной дорожки к другой (track-to-track seek time)
* Среднее время установки или поиска (average seek time)
* Время ожидания (latency)
* Время доступа (access time)
* Среднее время доступа к данным (average access time)
* Скорость передачи данных (data transfer rate)
* Размер кеш-буфера контроллера (internal cash size).
* Средняя потребляемая мощность (capacity).
* Уровень шума (noise level)
* Среднее время наработки на отказ (MTBF)
* Физический и логический объем накопителей.

## 7.4. Контроллеры жестких дисков

Контроллер НЖМД физически расположен на плате электроники и предназначен для обеспечения операций преобразования и пересылке информации от головок чтения/записи к интерфейсу накопителя. Это сложнейшее устройство - микрокомпьютер, со своим процессором, ОЗУ и ПЗУ, схемами и системой ввода/вывода и т.п.. Однако, в большинстве случаев, производители размещают их в одном или двух микрочипах. Многие производители создают устройства, которые записывают различный объем информации на внутренние и внешние дорожки за счет размещения на них разного числа секторов. Это возможно, благодаря аппаратному скрытию от программ и пользователя физических характеристик устройства на уровне его контроллера и/или интерфейса (устройства с IDE, EIDE и SCSI интерфейсами). Поэтому, накопители, как правило, имеют различное физическое и логическое число цилиндров.

Режимы работы контроллеров НЖМД

Большинство современных накопителей поддерживают следующие режимы работы контроллеров:

* PIO
* DMA2
* Ultra DMA.

PIO (Programmed Input/Output - программный ввод/вывод), при котором все пересылки выполняет непосредственно центральный процессор. DMA - Direct Memory Access - прямой доступ к памяти - режим взаимодействия контроллера накопителя и интерфейса ПК, при котором обмен данными осуществляется без участия центрального процессора ПК. Режим DMA позволяет заметно разгрузить процессор по сравнению с режимом PIO. Все современные накопители могут работать в режиме DMA2, если это поддерживается ОС, а скорость обмена при этом может достигать16.6 Мб/с.

Накопители и системы с поддержкой режима Ultra DMA, при использовании соответствующего драйвера, могут передавать и принимать информацию со скоростью 33.3 Мб/с для Ultra DMA-33, 66 Мб/с для Ultra DMA-66 и 100 Мб/с для Ultra DMA-100. В настоящее время все современные модели винчестеров поддерживают стандарты Ultra DMA-66 и Ultra DMA 100.

## Логическое хранение и кодирование информации

Для обеспечения оптимальной производительности НЖМД, а также для улучшения программного интерфейса, в них на основе физически присутствующих структур - дорожек и секторов, используется логическая структура хранения и доступа к информации. Она называется файловой системой, а ее тип и характеристики зависят от используемой ОС. В настоящее время имеется много типов различных файловых систем, но они основывают свои логические структуры данных на нескольких первичных логических структурах. Рассмотрим их подробнее.

*Master Boot Record (MBR)*

Первый сектор жесткого диска содержит хозяйственную загрузочную запись - **Master Boot Record** (MBR), которая, в свою очередь, содержит загрузочную запись - Boot Record (BR), выполняющуюся в процессе загрузки ОС. Загрузочная запись жестких дисков является объектом атаки компьютерных вирусов, заражающих MBR. За загрузчиком расположена таблица разделов - Partition Table (PT), содержащая 4 записи - элементы логических разделов - Partitions. Завершается MBR специальной сигнатурой - последовательностью из 2-х байт с шестнадцатиричными значениями 55H и ААH, указывающая на то, что данный раздел, после которого расположена сигнатура, является последним разделом в таблице.

Структура MBR.

|  |  |
| --- | --- |
| Название записи в MBR | Длина, байт |
| Загрузочная запись – Boot Record | 446 |
| Элемент таблицы разделов 1 – Partition 1 | 16 |
| Элемент таблицы разделов 2 – Partition 2 | 16 |
| Элемент таблицы разделов 3 – Partition 3 | 16 |
| Элемент таблицы разделов 4 – Partition 4 | 16 |
| Сигнатура окончания Partition Table | 2 |

Каждый элемент таблицы разделов содержит информацию о логическом разделе. Первым байтом в элементе раздела идет флаг активности раздела (0 - не активен, 128 (80H) - активен). Он служит для определения, является ли раздел системным и можно ли производить с него загрузку ОС при старте компьютера. Активным может быть только один раздел. Небольшие программы, называемые менеджерами загрузки (Boot Manager), могут располагаться в первых секторах диска. Они интерактивно запрашивают пользователя, с какого раздела производить загрузку, и соответственно корректируют флаги активности разделов. За флагом активности раздела следует байт номера головки, с которой начинается раздел. За ним следует два байта, означающие соответственно номер сектора и номер цилиндра загрузочного сектора, где располагается первый сектор загрузчика ОС. Загрузчик ОС представляет собой маленькую программу, осуществляющую считывание в память начального кода ОС во время ее старта. Затем следует байт – кодовый идентификатор ОС, расположенной в разделе, за ним расположен байт номера головки конца раздела, за которым идут два байта – номер сектора и номер цилиндра последнего сектора раздела.

*Формат таблицы разделов*

| **Название записи элемента Partition Table** | **Длина, байт** |
| --- | --- |
| Флаг а**к**тивности раздела | 1 |
| Номер головки начала раздела | 1 |
| Номер сектора и номер цилиндра загрузочного сектора раздела | 2 |
| Кодовый идентификатор операционной системы | 1 |
| Номер головки конца раздела | 1 |
| Номер сектора и цилиндра последнего сектора раздела | 2 |
| Младшее и старшее двухбайтовое слово относительного номера начального сектора | 4 |
| Младшее и старшее двухбайтовое слово размера раздела в секторах | 4 |

Завершают элемент раздела младшее и старшее двухбайтовое слово относительного номера первого сектора раздела и размер раздела в секторах соответственно.

Номера сектора и номер цилиндра секторов в разделах занимают 6 и 10 бит соответственно. Ниже представлен формат записи, содержащей номера сектора и цилиндра.

Биты номера цилиндра Биты номера сектора

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Для жестких дисков типичной является ситуация, когда имеется четыре записи в таблице разделов и соответственно четыре раздела. ОС MS-DOS использует только два из них, остальные резервируются для параллельного использования других ОС. Благодаря наличию такой структуры как MBR, на одном физическом жестком носителе может располагаться несколько файловых систем различного типа различных операционных систем. Структуры MBR представляют собой важную информацию, повреждение которой приводит к частичной или полной потере доступа к данным логических устройств жесткого диска и к невозможности загрузки ОС с поврежденного носителя.

***Логические разделы.***

Логические разделы тоже имеют иерархическую структуру в зависимости от типа ОС и ее файловой системы.

Так, первый раздел жесткого диска в MS-DOS называется главным разделом (**Primary Partition**), а второй расширенным (**Extended Partition**). Главный раздел всегда должен присутствовать на диске, с него происходит загрузка MS-DOS. Расширенного раздела может не быть, он создается лишь в случае, когда необходимо получить более одного логического устройства на физическом диске. Логический раздел размещает в себе такие структуры файловой системы как логические диски, устройства или тома (оформленные как подразделы), загрузчик операционной системы, таблицы распределения файлов, области пользовательских данных в которых размещаются записи о каталогах и файлах и данные файлов. Число логических подразделов может быть более четырех, а последний элемент каждого показывает, является ли он последним логическим подразделом раздела.

Основной единицей хранения информации в MS-DOS и других ОС с похожей логической структурой разделов является кластер (cluster) - группа секторов. В таких ОС для распределения минимального дискового пространства в один байт выделяется целый кластер, содержащий много секторов и еще больше байт (килобайты), что приводит к нерациональному использованию пространства ЖД для мелких файлов. Для доступа к каждому кластеру создается таблица соответствия номеров кластеров файлам логического раздела - таблица распределения файлов (File Allocation Table - FAT). Поэтому, файловые систем такого типа называют FAT-системы. Это не самый оптимальный, но довольно быстрый способ организации информации на разделах, поэтому он "дожил" до наших дней с давних времен начала развития ПК, где использовался исключительно для накопителей на гибких магнитных дисков. Все остальные логические структуры - файлы или каталоги связаны локализацией с FAT.

Для других ОС, например, UNIX - использование разделов происходит иначе. Как правило, их может быть более четырех, все они равноправны и одинаково могут быть загрузочными, содержат собственные файловые системы на основе i-узлов. Такие файловые системы являются теговыми и не имеют таблиц распределения порций информации. Дисковое пространство распределяется посекторно, что дает максимально возможное использование пространства раздела, но несколько снижает производительность. Весь раздел разбивается на иерархически связанную цепочку узлов разного уровня, которым соответствует некоторое количество секторов. На основе узлов строится понятие файлов и каталогов, и в таких системах файлы и каталоги действительно не различаются, т.к. каталог является файлом, содержащим структуру узлов. Один раздел отводится для дискового свопа и имеет упрощенную структуру, т.к. никогда не содержит файлов и каталогов.

Все разделы могут содержать **загрузчик операционной системы**, который располагается, как правило, в первом секторе и занимает один сектор. В этом секторе располагаются структуры - записи, имеющие отношение лишь к конкретной ОС и следовательно они могут отличаться для разных разделов и версий ОС. Многие специализированные программы (например, защиты данных, по борьбе и профилактике вирусов и др.) могут изменять структуру или отдельные части загрузчика операционных систем. Загрузчик большинства персональных однопользовательских операционных систем является объектом заражения вирусами, которые заражают загрузочные сектора жестких дисков.

## Интерфейсы жестких дисков

Интерфейсом накопителей называется набор электроники, обеспечивающий обмен информацией между контроллером устройства (кеш-буфером) и компьютером. В настоящее время в настольных ПК, чаще других, используются две разновидности интерфейсов ATAPI - AT Attachment Packet Interface (Integrated Drive Electronics - IDE, Enhanced Integrated Drive Electronics - EIDE) и SCSI (Small Computers System Interface).

*Интерфейс IDE*

Разрабатывался как недорогая и производительная альтернатива высокоскоростным интерфейсам ESDI и SCSI. Интерфейс, предназначен для подключения двух дисковых устройств. Отличительной особенностью дисковых устройств, работающих с интерфейсом IDE состоит в том, что собственно контроллер НЖМД располагается на плате самого накопителя вместе со встроенным внутренним кэш-буфером. Такая конструкция существенно упрощает устройство интерфейсной карты и дает возможность размещать ее не только на отдельной плате адаптера, вставляемой в разъем системной шины, но и интегрировать непосредственно на материнской плате компьютера. Интерфейс характеризуется простотой, высоким быстродействием, малыми размерами и относительной дешевизной.

Сегодня на смену интерфейсу IDE пришло детище фирмы Western Digital - Enhanced IDE, или сокращенно EIDE. Сейчас это лучший вариант для подавляющего большинства настольных систем. Жесткие диски EIDE заметно дешевле аналогичных по емкости SCSI-дисков (см. ниже) и в однопользовательских системах не уступают им по производительности, а большинство материнских плат имеют интегрированный двухканальный контроллер для подключения четырех устройств.

Можно выделить шесть основных отличий расширенного стандарта:

1. большая емкость дисков; если IDE не поддерживал диски свыше 528 мегабайт, то EIDE преодолевает это ограничение;
2. к нему подключается больше устройств - четыре вместо двух. Вместо одного канала контроллера для подключения двух IDE устройств используются два: основной - на высокоскоростной локальной шине и вспомогательный;
3. появилась спецификация ATAPI, дающая возможность подключения к этому интерфейсу не только жестких дисков, но и других устройств - стриммеров и дисководов CD-ROM;
4. повысилась производительность: НЖМД с интерфейсом IDE имели максимальную скорость передачи данных на уровне 3 мегабайт в секунду, а диски EIDE поддерживают несколько новых режимов обмена, которые обеспечивают скорость передачи данных 11.1 и 16.6 мегабайт в секунду соответственно.
5. поддерживается режим прямого доступа к памяти - Multiword Mode 1 DMA (Direct Memory Access) или Multiword Mode 2 DMA и Ultra DMA, которые поддерживают обмен данными в монопольном режиме, при котором канал ввода-вывода в течение некоторого времени обслуживает только одно устройство;
6. расширена система команд управления устройством, передачи данных и диагностики, увеличен кеш-буфер обмена данными и существенно доработана механика.

*Интеллектуальный многофункциональный интерфейс SCSI*

Был разработан в конце 70-х годов в качестве устройства сопряжения компьютера и интеллектуального контроллера дискового накопителя. Интерфейс SCSI является универсальным и определяет обмен данными между центральным процессором и несколькими внешними устройствами, имеющими свой контроллер. Помимо электрических и физических параметров, определяются также команды, при помощи которых, устройства, подключенные к шине, осуществляют связь между собой. Интерфейс SCSI поддерживает значительно более широкую гамму периферийных устройств и стандартизован ANSI (X3.131-1986).

Сегодня применяются в основном три стандарта:

* SCSI-2
* Ultra SCSI.
* SCSI-3

В режиме Fast SCSI-2 скорость передачи данных доходит до 10 мегабайт в секунду при использовании 8-разрядной шины и до 20 мегабайт при 16-разрядной шине Fast Wide SCSI-2.

Появившийся позднее стандарт Ultra SCSI отличается еще большей производительностью - 20 мегабайт в секунду для

8-разрядной шины и 40 мегабайт для 16-разрядной.

В новейшем SCSI-3 увеличен набор команд, но быстродействие осталось на том же уровне.

Все применяющиеся сегодня стандарты совместимы с предыдущими версиями "сверху - вниз", то есть к адаптерам SCSI-2 и Ultra SCSI можно подключить старые SCSI-устройства. Интерфейс SCSI-Wide, SCSI-2, SCSI-3 - стандарты модификации интерфейса SCSI, разработаны комитетом ANSI. Общая концепция усовершенствований направлена на увеличение ширины шины до 32-х, с увеличением длины соединительного кабеля и максимальной скорости передачи данных с сохранением совместимости с SCSI. Это наиболее гибкий и стандартизованный тип интерфейсов, применяющийся для подключения 7 и более периферийных устройств, снабженных контроллером интерфейса SCSI. Интерфейс SCSI остается достаточно дорогим и самым высокопроизводительным из семейства интерфейсов периферийных устройств персональных компьютеров.

## Работа накопителя

Теперь рассмотрим процесс работы накопителя от запуска до остановки. При подаче питающих напряжений начинает работать микропроцессор контроллера. Вначале он, как и компьютер, выполняет самотестирование и в случае его успеха запускает схему управления двигателем вращения шпинделя. Диски начинают раскручиваться, увлекая за собой прилегающие к поверхностям слои воздуха, и при достижении некоторой скорости давление набегающего на головки потока воздуха преодолевает силу пружин, прижимающих их к дискам, и головки "всплывают", поднимаясь над дисками на доли микрона. С этого момента, вплоть до остановки дисков, головки не касаются дисков, поэтому ни диски, ни сами головки практически не изнашиваются. Двигатель шпинделя продолжает раскручивать поверхности и его скорость приближается к номинальной (тысячи оборотов в минуту). В это время накопитель потребляет максимум питающего напряжения 12 вольт. Поскольку в любой зоне дисков присутствует серворазметка, то сервоимпульсы начинают поступать с головок сразу же после начала вращения, и по их частоте контроллер судит о скорости вращения дисков. После достижения шпинделем номинальной скорости вращения освобождается фиксатор позиционера головок чтения/записи, и система его управления проверяет способность поворачиваться и удерживаться на выбранной дорожке путем выборочного произвольного позиционирования. При этом делается серия быстрых поворотов в разные стороны, что на слух выглядит как характерное "тарахтение", слышимое через несколько секунд после старта. Во время перемещения позиционера головок происходит слежение за поступающими с головок серво-импульсами, и система управления всегда "знает", над сколькими дорожками прошли головки. Аналогично происходит и удержание головок над выбранной дорожкой - при отклонении от центра дорожки изменяется во времени величина и форма серво-импульсов. Система управления может ликвидировать отклонение, изменяя ток в обмотках двигателя позиционера головок.

Микрокомпьютер НЖМД имеет ПЗУ, в котором записана BIOS накопителя - набор программ для начального запуска и управления во время работы, и ОЗУ, в которое после раскрутки механической системы загружаются остальные части управляющих программ. Также в ОЗУ загружается так называемая карта переназначения дефектных секторов, в которой отмечены дефектные секторы, выявленные при заводской разметке дисков. Эти секторы исключаются из работы и иногда подменяются резервными, которые имеются на каждой дорожке или в специальных резервных зонах каждого диска. На каждом диске имеется некоторый запас резервных секторов, которыми можно подменить и появляющиеся впоследствии дефекты либо под управлением специальных программ, либо автоматически в процессе работы.

После начальной настройки электроники и механики микрокомпьютер ЖД переходит в режим ожидания команд контроллера, расположенного на системной плате или интерфейсной карте, который в свою очередь программируется процедурами собственной BIOS или BIOS компьютера под управлением ОС. Получив команду, он позиционирует на нужный цилиндр, по сервоимпульсам отыскивает нужную дорожку, дожидается, пока до головки дойдет нужный сектор, и выполняет считывание или запись информации. Если контроллер запросил чтение/запись не одного сектора, а нескольких - накопитель может работать в блочном режиме, используя ОЗУ в качестве буфера и совмещая чтение/запись нескольких секторов с передачей информации к контроллеру или от него.

При выключении питания двигатель шпинделя работает в режиме генератора, обеспечивая питание плат электроники на время, необходимое для корректного завершения работы. Прежде всего, блокируется подача тока записи в магнитные головки, чтобы они не испортили информацию на поверхностях, а остаток энергии подается в обмотки привода головок, толкая их к центру дисков для автопарковки. Любой исправный накопитель всегда запаркует головки, как бы внезапно не было выключено питание, но если в этот момент происходила запись информации, то последствия могут быть плохими из-за недописанных областей данных или управляющих структур файловой системы ПК, независимо от типа и вида установленной ОС.

## Внешняя память на CD и DVD дисках.

*Устройство CD-диска*

Стандартный компакт-диск (CD) состоит из трех слоев:

* основы,
* отражающего
* и защитного.

Основа выполнена из прозрачного поликарбоната, на котором методом прессования сформирован информационный рельеф. Поверх рельефа напыляется металлический отражающий слой (алюминий, золото, серебро, другие металлы и сплавы). Отражающий слой покрывается сверху защитным слоем поликарбоната или нейтрального лака - так, чтобы вся металлическая поверхность была защищена от контакта с внешней средой.

Информационный рельеф диска представляет собой непрерывную спиральную дорожку, начинающуюся от центра и состоящую из последовательности углублений - питов (pits). Промежутки между питами носят название lands. Чередованием питов и промежутков различной длины на диске записывается закодированный цифровой сигнал: переход от промежутка к питу и наоборот обозначает единицу, а длина пита или промежутка - длину серии нулей. Расстояние между витками дорожки выбирается от 1.4 до 2 мкм, стандарт определяет расстояние в 1.6 мкм.

*Представление звукового сигнала*

Исходный стереофонический звуковой сигнал подвергается оцифровке в 16-разрядные отсчеты (квантование) с частотой дискретизации 44.1 кГц. В начале диска располагается вводная зона, содержащая информацию о формате диска, структуре звуковых программ, адресах фрагментов, названиях произведений и т.п. В конце записывается выводная зона (дорожка с номером AA), выполняющая роль границы записанной области диска.

Общая длительность записи на компакт-диске - 74 минуты. Записи состоят из фреймов, по 588 отстчетов для левого и для правового каналов. В каждой секунде звукозаписи содержится по 75 фреймов соответственно. Можно перемножить 588\*75=44100.

*Способы записи и изготовления*

Основной способ изготовления дисков - прессование с матрицы. Оригинал формируется с исходной цифровой мастер-ленты, содержащей уже подготовленный и закодированный цифровой сигнал, специальным высокоточным станком на стеклянном диске, покрытом слоем фоторезиста - материала, изменяющего свою растворимость под воздействием лазерного луча. При обработке записанного оригинала растворителем на стекле возникает требуемый рельеф, который методом гальванопластики переносится на никелевый оригинал (негатив), который может служить матрицей при мелкосерийном производстве, либо основой для снятия позитивных копий, с которых, в свою очередь, снимаются негативы для массового тиражирования. Штамповка выполняется методом литья под давлением: с негативной матрицы прессуется поликарбонатная подложка с рельефом, сверху напыляется отражающий слой, который покрывается лаком.

*Воспроизведение звука*

При воспроизведении звуковой компакт-диск вращается с постоянной линейной скоростью относительно воспроизводящей головки (приблизительно 1.25 м/с). Система стабилизации скорости вращения обеспечивает скорость считан-ного цифрового потока равной 4.3218 Мбит/с, поэтому в зависимости от длины питов и промежутков действительная скорость может изменяться. Угловая скорость диска при этом изменяется от 500 об/мин при чтении самых внутренних участков дорожки до 200 об/м на самых внешних.

Для считывания информации с диска используется полупроводниковый лазер с длиной волны около 780 нм (инфракрасный диапазон). Луч лазера, проходя через фокусирующую линзу, падает на отражающий слой, отраженный луч попадает в фотоприемник, где происходит определение питов и промежутков, а также проверка качества фокусировки пятна на дорожке и его ориентации по центру дорожки.

*CD-R и CD-RW*

Система однократной (CD-Recordable - записываемый CD) и многократной (CD-ReWritable - перезаписываемый CD) записи компакт-дисков. Терминами CD-R, CD-E и CD-RW обозначаются как устройства для записи, так и сами диски.

Для однократной записи используются так называемые "болванки", представляющие собой обычный компакт-диск, в котором отражающий слой выполнен преимущественно из золотой или серебряной пленки, а между ним и поликарбонатной основой расположен регистрирующий слой из органического материала, темнеющего при нагревании. В процессе записи лазерный луч нагревает выбранные точки слоя, которые темнеют и перестают пропускать свет к отражающему слою.

В перезаписываемых дисках используется промежуточный слой из органической пленки, изменяющей под воздействием луча свое фазовое состояние с аморфного на кристаллическое и обратно, в результате чего меняется прозрачность слоя. Фиксация изменений состояния происходит благодаря тому, что материал регистрирующего слоя при нагреве свыше критической температуры переходит в аморфное состояние и остается в нем после остывания, а при охлаждении до температуры значительно ниже критической восстанавливает кристаллическое состояние. Существующие диски выдерживают от тысяч до десятков тысяч циклов перезаписи. Однако их отражающая спосо-бность существенно ниже штампованных и однократных CD, что затрудняет их считывание в обычных приводах. Для чтения CD-RW формально необходим привод с автоматической регулировкой усиления фотоприемника

Запись дисков CD-R выполняется при помощи специальных программ - Easy CD, CD Creator, CD Publisher, Direct CD и т.п. Процесс записи одной дорожки представляет собой единую операцию, которая не может быть прервана, иначе диск будет испорчен. Для обеспечения равномерности поступления записываемой информации на лазер все приводы имеют буфер, исчерпание данных в котором приводит к аварийному прерыванию записи.

*Приводы CD-ROM*

Типовой привод состоит из:

* платы электроники,
* шпиндельного двигателя,
* системы оптической считывающей головки
* и системы загрузки диска.

На плате электроники размещены все управляющие схемы привода, интерфейс с контроллером компьютера, разъемы интерфейса и выхода звукового сигнала.

Шпиндельный двигатель служит для приведения диска во вращение с постоянной или переменной линейной скоростью.

Система оптической головки состоит из самой головки и системы ее перемещения. В головке размещены лазерный излучатель, на основе инфракрасного лазерного светодиода, система фокусировки, фотоприемник и предварительный усилитель. Изменение напряженности магнитного поля вызывает перемещение линзы и пеpефокусиpовку лазерного луча. Система перемещения головки имеет собственный приводной двигатель, приводящий в движение каретку с оптической головкой.

Система загрузки диска выполняется в двух вариантах: с использованием специального футляра для диска, вставляемого в приемное отверстие привода, и с использованием выдвижного лотка, на который кладется сам диск.

При стандартной скорости вращения скорость передачи данных составляет около 150 Кб/с. В двух- и более скоростных CD-ROM диск вращается с пропорционально большей скоростью, и пропорционально повышается скорость передачи, например, 1200 Кб/с для 8-скоpостного диска.

В настоящее время типичные скорости 40-50.

*Интерфейсы приводов CD-ROM*

В настоящее время CD-ROM выпускаются только с интерфейсами SCSI и IDE. Диски подключается непосредственно к магистрали SCSI или IDE (ATA) с заданием номеpа устройства для SCSI или Master/Slave - для IDE.

*DVD*

Первоначально сокращение DVD расшифровывалось как digital video disc - оптические диски с большой емкостью. Эти диски используются для хранения компьютерных программ и приложений, а так же полнометражных фильмов и высококачественного звука. Появившаяся позже расшифровка аббревиатуры DVD, как digital versatile disc, т.е. универсальный цифровой диск - более логична.

Хотя DVD выглядят как обычные диски CD-ROM, они могут хранить в 26 раз больше данных. Однослойный, односторонний диск DVD может хранить 4.7GB данных, двухслойный, односторонний - до 8.5GB, двухсторонний - до 17GB.

Как и CD-ROM, диски DVD хранят данные за счет насечек , расположенных вдоль спиральных треков на отражающей металлической поверхности, покрытой пластиком. Используемый в устройствах чтения DVD дисков лазер скользит вдоль треков по насечкам, а отраженный луч интерпретируется приемным устройством в виде единиц или нулей. Благодаря разработке более высокочастотного полупроводникового лазера с меньшей длиной волны, стало возможным использовать насечки меньшего размера.

Другие новшества:

* новый формат секторов,
* более надежный код коррекции ошибок
* улучшенная модуляция каналов.

Все эти улучшения увеличивают плотность записи данных в полтора раза. Жесткие производственные требования и незначительно большая поверхность записи стали препятствием при разработке DVD, ограничивающим емкость размещаемых на диске данных до 4.7Gb. Для записи видео и звука на DVD применяется компрессии данных, носящая название MPEG-2 и MPEG-4. Это увеличивает временную емкость записываемых данных.

## Принципы организации RAID массивов

***Назначение.***

В переводе с английского “RAID” (Redundant Array of Independent Disks) означает “избыточный массив независимых дисков”. Впервые термин RAID появился в 1987 году, когда исследователям из Калифорнийского Университета в Беркли удалось создать массив из нескольких жестких дисков.

Предназначение RAID - создание на базе нескольких жестких дисков сравнительно небольшой емкости одного логического диска c:

* большой емкостью;
* увеличенной скоростью доступа;
* увеличенной надежностью хранения;
* возможностью восстановления данных в случае отказа части оборудования.

Именно эти обстоятельства сделали RAID-массивы столь востребованными бизнесом и военными. Впрочем, за объем, скорость и надежность пришлось платить повышением стоимости и сложности систем хранения данных. Со временем оборудование для построения RAID массивов стало более доступным, особенно с появлением дешевых решений для IDE/ATA и SATA дисков.

Впрочем, найти оптимальное решение одновременно по надежности, скорости, емкости и цене дисковой памяти не просто. Надо быть готовым к тому, что придется купить не один, а несколько жестких дисков, и емкость как минимум одного из них не будет использоваться. Кроме того, потребуется специальный корпус с отдельным или двумя блоками питания, платой контроллера и соответствующее программное обеспечение.

## Основные принципы построения RAID массивов

В основе теории RAID лежат пять основных принципов:

* массив (Array);
* зеркалирование (Mirroring) или дублирование;
* чередование полос (Striping);
* контроль четности (Parity).

***Массивом*** называют несколько накопителей, которые централизованно настраиваются, форматируются и управляются. Логический массив – это уже более высокий уровень представления, на котором не учитываются физические характеристики системы. Соответственно, логические диски могут по количеству не совпадать с физическими. Но лучше все-таки соблюдать соответствие: физический диск – логический диск. Наконец, для операционной системы вообще весь массив является одним большим логическим диском.

***Зеркалирование*** – технология, позволяющая повысить надежность системы. В RAID массиве с зеркалированием все данные одновременно пишутся не на один, а на два жестких диска. То есть создается «зеркало» данных. При выходе из строя одного из дисков вся информация остается сохраненной на втором.

За такую стопроцентную защиту приходится дорого платить: считайте, что один винчестер у вас работает просто так, не увеличивая доступную емкость ни на Мегабайт. При этом нет никакого выигрыша в производительности. Столь дорогое решение используется только во внешних RAID-массивах, предназначенных для ответственных приложений.

***Чередование полос*** – отличная возможность повысить быстродействие системы. Очевидно, если чтение и запись вести параллельно на нескольких жестких дисках, можно получить выигрыш в скорости. Как это делается? Записываемый файл разбивается на части определенного размера (полосы - strip) и одновременно размещается на всех имеющихся накопителях в последовательном порядке. В таком фрагментированном виде файл и хранится. Считывание и запись соседних полос выполняется параллельно с разных дисков.

Размер «кусочка» может быть минимальным – 1 байт, но чаще используют более крупное дробление – по 512 байт (размер сектора).

***Контроль четности***  является альтернативным решением, соединяющим в себе достоинства зеркалирования (высокая надежность) и чередования (высокая скорость работы).

Используется тот же принцип, что и в контроле четности оперативной памяти. Если имеется i блоков данных и на их основе вычисляется еще один дополнительный экстраблок, из получившихся (i+1) блоков всегда можно восстановить информацию даже при повреждении одного из них. Соответственно, для создания нормального RAID-массива в этом случае требуется (i+1) жесткий диск.

Распределение блоков по дискам точно такое же, как при чередовании. Экстраблок может записываться на отдельный накопитель, либо раскидываться по дискам.

Что же хранится в экстраблоке? Каждый бит экстраблока равен результату выполнения логической операции XOR над соответствующими битами всех i блоков. Многие помнят со школы, что XOR – удивительный оператор, при его повторном наложении мы можем получить первоначальный результат. То есть (A XOR B) XOR B = A. Это правило распространяется на любое количество операндов.

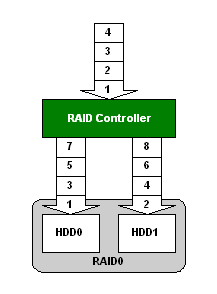
Плюсы четности очевидны. За счет использования чередования повышается скорость работы. Повышение надежности за счет зеркалирования здесь делается так, что при этом «нерабочий» объем массива заметно уменьшается, он одинаков при любом количестве дисков и составляет емкость одного диска, то есть при 5 дисках в массиве пропадает всего 20% емкости.

Но у четности есть весомый минус. Для формирования экстраблоков требуются вычисления! Их надо делать на лету, причем с миллионами бит! Если это дело поручить центральному процессору, мы получим очень медленную систему. Необходимо использовать довольно дорогие платы с RAID-контроллерами, которые «берут все вычисления на себя». В случае выхода из строя одного из дисков, процесс восстановления будет не столь быстрым, как при зеркалировании.

## Одиночные уровни RAID

Исследователи из Беркли в 1988 году придумали такую классификацию, не учитывая, что обычно уровни четко связаны с иерархией. В RAID уровнях нет преемственности. RAID 4 не есть улучшенная модификация RAID 3, так же как RAID 5 не лучше RAID 1. Просто они разные. Следует отметить, что есть простые (single) и составные (multiple) RAID массивы. Составные являются сочетанием двух простых. Сначала рассмотрим простые 7 уровней.

***RAID 0***

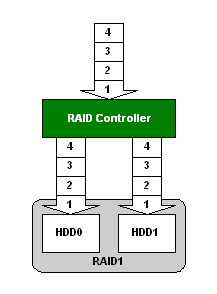


Структура RAID 0

Простейший массив, использующий чередование полос без четности. Вся входящая информация разбивается на блоки фиксированной длины (например, 16 кбайт) и раскидывается на все имеющиеся диски.

При наличии двух-четырех дисков RAID 0 дает ощутимый выигрыш в скорости передачи данных, но совершенно не обеспечивает надежность. Для его построения подойдет любой дешевый и даже программный RAID-контроллер. Подходит для тех, кому нужно выжать максимум производительности от файловой системы при минимальных затратах.

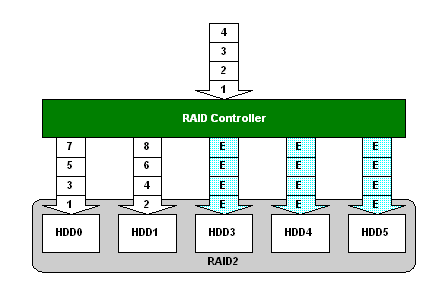
***RAID 1***



Структура RAID 1

Этот уровень является обычным зеркалированием. На два жестких диска пишутся две одинаковые копии данных. При этом можно использовать дешевый RAID контроллер или даже его программную реализацию. RAID 1 позволяет надежно защитить данные и обеспечить работу системы даже при поломке одного из дисков. Вот почему он получил широкое распространение среди пользователей, желающих защитить от потери личные данные. Выигрыша в скорости при использовании RAID1 нет.

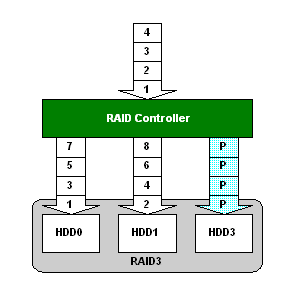
***RAID 2***



Структура RAID 2

Второй уровень RAID умер, так и не родившись. Исследователи из Беркли предложили использовать одновременно две технологии – побитовое чередование и код Хэмминга для восстановления ошибок. В теории это должен быть неплохой по надежности и рабочей емкости массив, построенный из 14 или 39 дисков (!). При этом часть дисков (10 или 32) используется для хранения данных с чередованием, остальные – для хранения вычисленных контрольных сумм. Реализация таких систем требовала специальных дорогостоящих контроллеров, которые так и не прижились на рынке. В итоге RAID 2 сейчас не используется.

***RAID 3***

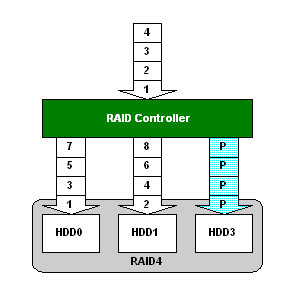


Структура RAID 3

Третий уровень использует чередование полос и выделенный диск для контроля четности. Блоки данных обычно имеют длину меньше 1024 байт. Информация распределяется на несколько дисков, а вычисленные значение контроля по четности сохраняются на отдельном диске.

Все скоростные преимущества чередования сводятся на нет необходимостью записывать контрольную сумму на выделенный диск, а больше всех страдает скорость случайной записи. К достоинствам следует отнести возможность работы массива при отказе одного из дисков.

***RAID 4***

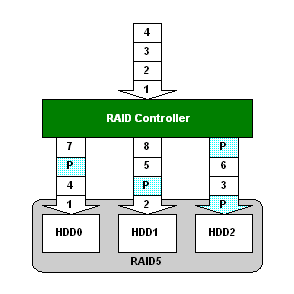


Структура RAID 4

Отличается от RAID 3 только размером блока данных при чередовании (значительно больше – не менее 2-х физических блоков диска). Это несколько улучшает работу массива при случайном чтении, но запись все равно довольно медленная. Диск с контрольными суммами является ярко выраженным «узким местом» в системе.

RAID 4 является компромиссным вариантом между RAID 3 и RAID 5, не нашел своего места на рынке и редко используется. Это держит цены на соответствующие контроллеры на высоком уровне.

***RAID 5***



Структура RAID 5

Наиболее распространенный в системах хранения данных – пятый уровень. Он характеризуется применением чередования полос и контроля четности. В отличие от RAID 3, контрольные суммы не хранятся на одном диске, а распределяются по всем, что позволяет значительно поднять скорость записи. Главный принцип распределения экстраблоков - они не должны располагаться на том же диске, который учитывался привычислении паритета.

Надежность и скорость работы такой системы оказываются очень высокими. При восстановлении информации всю работу на себя берет RAID контроллер, так что операция проходит довольно быстро.

***RAID 6***

Для некоторых особо критичных приложений требуется повышенная надежность. Например, чтобы при выходе из строя даже двух дисков массив сохранил данные и остался работоспособным. Это обеспечивается на уровне RAID 6. При этом используются все те же технологии чередования полос и контроля четности, но в отличие от RAID 5 контрольная сумма вычисляется два раза и копируется на два разных диска. В итоге данные окажутся потерянными только в случае выхода из строя сразу двух жестких дисков. По сравнению с RAID 5 это более дорогое и медленное решение: увеличивается время на вычисление запись паритетной информации требуется дополнительное дисковое пространство. На практике RAID 6 почти не используется, так как выход из строя сразу двух дисков – слишком редкий случай, а повысить надежность можно другими способами.

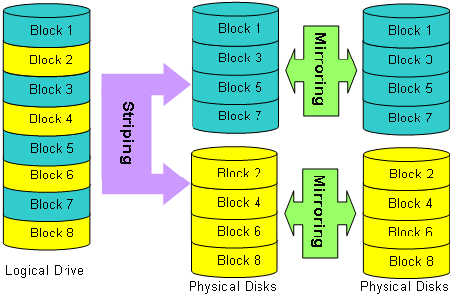
## Составные уровни RAID массивов

У основных уровней RAID есть свои достоинства и недостатки. И вполне понятно, почему инженеры стали мечтать о таком RAID, который бы объединял достоинства нескольких уровней. Составной RAID массив – это чаще всего сочетание быстрого RAID 0 с надежным RAID 1, 3 или 5. Итоговый массив действительно обладает улучшенными характеристиками, но и платить за это приходится повышением стоимости и сложностью решения.

Составной RAID строится так: сначала диски разделяются на наборы (set). Затем на основе каждого из наборов строятся простые массивы. А завершается все объединением этих массивов в один мегамассив. Запись типа X+Y означает, что сначала диски объединены в RAID уровня X, а затем несколько RAID X массивов объединены в RAID уровня Y.

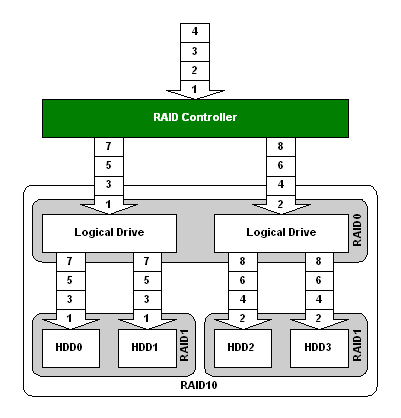
***RAID 0+1 (01) и 1+0 (10)***

RAID 0+1 часто называют «зеркалом страйпов», а RAID 1+0 – «страйпом зеркал» (нормальное русское «чередование» практически не используется, сменившись англоязычным). В обоих случаях используются две технологии – чередование и зеркалирование, но результаты разные.



Структура RAID 0 + 1

RAID 0+1 обладает высокой скоростью работы и повышенной надежностью, поддерживается даже дешевыми RAID контроллерами и является недорогим решением. Но по надежности несколько лучше RAID 1+0. Так, массив из 10 дисков (5 по 2) может остаться работоспособным при отказе до 5 жестких дисков! Основной недостаток массивов 1+0 – низкий процент использования емкости накопителей – всего 50%. Но для домашних систем именно RAID 01 или 10 может оказаться оптимальным решением.



Структура RAID 1 + 0

***RAID 0+3 (03) и 3+0 (30)***

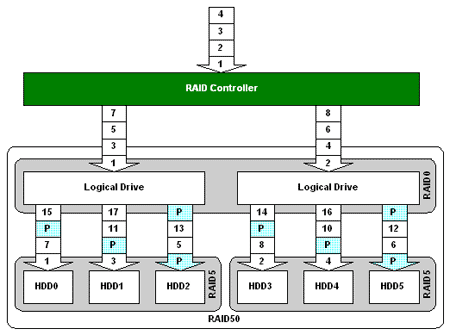
По идее сочетание чередования и RAID 3 дает выигрыш в скорости, но он довольно мал. Зато система заметно усложняется. Наиболее простой уровень 3+0. Из двух массивов RAID 3 строится страйп, и минимальное количество требуемых дисков – 6. Получившийся RAID 3+0 с точки зрения надежности лучше, чем 0+3.

Достоинства этих комбинаций в довольно высоком проценте использования емкости дисков и высокой скорости чтения данных. Недостатки – высокая цена, сложность системы.

***RAID 0+5 (05) и 5+0 (50)***

Что будет, если объединить чередование с распределенной четностью с обыкновенным чередованием ? Получится быстрая и надежная система. RAID 0+5 представляет собой набор страйпов, на основе которых построен RAID 5. Такая комбинация используется редко, так как практически не дает выигрыша ни в чем. Широкое распространение получил составной RAID массив 5+0.

Чаще всего это два массива RAID 5, объединенных в страйп. Такая конфигурация позволяет получить высокую производительность при работе с файлами малого размера. Типичный пример – использование в качестве WEB-сервера.



Структура RAID 5 + 0

***RAID 1+5 (15) и 5+1 (51)***

Этот уровень построен на сочетании зеркалирования и чередования с распределенной четностью. Основная цель RAID 15 и 51 – значительное повышение надежности. Массив 1+5 продолжает работать при отказе трех накопителей, а 5+1 - даже при потере пяти из восьми жестких дисков! Платить приходится большим количеством неиспользуемой емкости дисков и общим удорожанием системы.

Чаще всего для построения RAID 5+1 используют два контроллера RAID 5, которые зеркалируют на программном уровне, что позволяет снизить затраты.

# *Сравнительные результаты*

*Распространенные single RAID массивы*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **RAID 0** | **RAID 1** | **RAID 3** | **RAID 5** | **RAID 6** |
| **Технология** | Чередование | Зеркали- рование | Чередование, четность | Чередование, четность | Чередование, четность |
| **Контроллер** | Все | Все | Аппаратный | Аппаратный Hi-End | Специали- зированный |
| **Кол-во жест. диск.** | 2, 4 | 2 | 3 и больше | 3 и больше | 3 и больше |
| **Доступное рабочее пространство, %** | 100 | 50 | 66 для 3, 75 для 4 | 66 для 3, 75 для 4 | 33 для 3 50 для 4 60 для 5 |
| **Стойкость при отказе диска** | Нет | Высокая | Высокая | Высокая | Очень высокая |
| **Восстановление данных** | Нет | Быстрое | Быстрое | Быстрое | Очень быстрое |
| **Скорость случайного чтения** | Высокая | Хорошая | Хорошая | Очень хорошая | Очень хорошая |
| **Скорость случайной записи** | Высокая | Хорошая | Плохая | Нормальная | Плохая |
| **Скорость линейного чтения** | Высокая | Хорошая | Очень хорошая | Очень хорошая | Хорошая |
| **Скорость линейной записи** | Высокая | Хорошая | Хорошая | Хорошая | Средняя |
| **Цена** | Самая низкая | Низкая | Средняя | Средняя | Высокая |

*Распространенные multi-RAID массивы*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **RAID 0+1** | **RAID 1+0** | **RAID 5+0** | **RAID 5+1** |
| **Технология** | Чередование, зеркалирование | Чередование, зеркалирование | Чередование, четность | Чередование, четность, зеркалирование |
| **Контроллер** | Почти все | Почти все | Специали- зированный | Специали- зированный |
| **Кол-во жест. диск** | 4 min | 4 min | 6 min | 6 min |
| **Доступное рабочее пространство, %** | 50 | 50 | 66 для 2 страй-пов по 3 диска | 33-40 |
| **Стойкость при отказе диска** | Оч. хорошая | Отличная | Хорошая | Отличная |
| **Восстановление данных** | Быстрое | Очень быстрое | Среднее | Быстрое |
| **Скорость случайного чтения** | Очень хорошая | Очень хорошая | Очень хорошая | Очень хорошая |
| **Скорость случайной записи** | Хорошая | Хорошая | Хорошая | Хорошая |
| **Скорость линейного чтения** | Очень хорошая | Очень хорошая | Очень хорошая | Очень хорошая |
| **Скорость линейной записи** | Хорошая | Хорошая | Хорошая | Хорошая |
| **Цена** | Относительно высокая | Относительно высокая | Высокая | Очень высокая |

# Системные и локальные шины

# 9.1. Общие положения и требования к шинам

При работе компьютера, состоящего из множества подсистем, необходим механизм для их взаимодействия. Эти подсистемы должны быстро и эффективно обмениваться данными. Например, процессор, с одной стороны, должен быть связан с памятью; с другой стороны, необходима связь памяти и процессора с устройствами ввода/вывода. Связь между различными подсистемами компьютера обеспечивается с помощью наборов линий, называемых шинами. Шины можно разделить на группы в соответствии с выполняемыми ими функциями. Шины могут быть внутренними по отношению к процессору и служить для передачи данных между АЛУ и регистрами, а могут быть внешними по отношению к процессору и связывать процессор с памятью или устройствами ввода/вывода.

Операции на шине назавыются **транзакциями**. Основные виды – транзакции чтения и транзакции записи или транзакции ввода и транзакции вывода. Шинная транзакция состоит из двух частей: посылка адреса и прием (или посылка) данных.

Связанные с шиной устройства должны подчиняться определённым правилам, которые называют протоколами шины. Некоторые устройства, связанные с шиной, являются активными и могут инициировать передачу информации по шине, а другие – пассивными и ждут запросов. Активное устройство называют задающим (bus master), пассивное – подчинённым (bus slave). Процессор является задающим устройством, если он требует от контроллера считать или записать информацию. В этом случае контроллер является пассивным устройством. Контроллер становится задающим устройством, если он командует приёмом слов в память, ранее считанных им с диска.

Механизм, обеспечивающий как связь, так и взаимодействие устройств компьютера, реализуется с помощью унифицированной совокупности средств связи – интерфейсов, которые требуют стандартизации, распространяемой на форматы передаваемых данных, команды, наборы шин, алгоритмы, сигналы и т. д. Интерфейс – это совокупность унифицированных шин для передачи информации, электрических схем, управляющих прохождением сигналов по шинам и алгоритмов, управляющих обменом информации.

Интерфейсы подразделяют на односвязные и многосвязные. Односвязные интерфейсы используют единственную центральную шину – магистраль, к которой подсоединяются все устройства на основе принципа разделения времени. Так как несколько устройств могут одновременно стать ведущими и пытаться захватить шину, то магистраль должна использоваться в режиме разделения времени. Ведущие устройства снабжаются приоритетом использования магистрали.

Подобная организация имеет два основных преимущества:

* низкая стоимость,
* универсальность.

Поскольку такая шина является единственным местом подсоединения для разных устройств, новые устройства могут быть легко добавлены в систему. Стоимость такой организации получается достаточно низкой, поскольку для реализации множества путей передачи информации используется единственный набор линий шины, разделяемый множеством устройств. Главным недостатком организации с единственной шиной является то, что **шина создает узкое место**, ограничивая максимальную пропускную способность ввода/вывода.

В коммерческих системах, где ввод/вывод осуществляется очень часто, а также в суперкомпьютерах, где необходимые скорости ввода/вывода очень высоки из-за высокой производительности процессора, одним из главных вопросов связи устройств является создание системы нескольких шин, способной удовлетворить все запросы. Использование для связи устройств в компьютере нескольких независимых систем шин характерно для многосвязных интерфейсов.

Для всех видов передачи информации используется стандартная совокупность сигналов и общие временные диаграммы. Так как устройства имеют разные характеристики по быстродействию, то они подключаются к шинам через соответствующие контроллеры. Характеристики интерфейса со стороны периферийного устройства должны быть согласованы с характеристиками этого устройства. В соответствии с типом устройства надо использовать специализированный интерфейсный кристалл, на котором реализованы аппаратные средства интерфейса. Поскольку внешние сигналы периферийных устройств могут отличаться, то для унификации интерфейсного кристалла эти отличия реализуются программным путем. Такой интерфейс принято называть программируемым. Отличия определяются с помощью записи значений соответствующих битовых комбинаций в предусмотренные для этого регистры интерфейсного кристалла.

## 9.2. Основные виды, характеристики и параметры шин

Одна из причин трудностей, возникающих при разработке шин, заключается в том, что максимальная скорость шины главным образом лимитируется физическими факторами:

* длиной шины,
* количеством подсоединяемых устройств.

Эти физические ограничения не позволяют произвольно ускорять шины. Требования малой задержки, высокой пропускной способности и подключения разных устройств являются противоречивыми. В современных крупных системах используется целый комплекс взаимосвязанных шин, каждая из которых обеспечивает упрощение взаимодействия различных подсистем, высокую пропускную способность, избыточность (для увеличения отказоустойчивости) и эффективность.

Традиционно шины делятся:

* на шины, обеспечивающие организацию связи процессора с памятью,
* шины ввода/вывода.

Шины ввода/вывода могут иметь большую протяженность, поддерживать подсоединение многих типов устройств и обычно следуют одному из шинных стандартов.

Шины процессор–память, с другой стороны, сравнительно короткие, обычно высокоскоростные и соответствуют организации системы памяти для обеспечения максимальной пропускной способности канала память–процессор. На этапе разработки системы для шины процессор–память заранее известны все типы и параметры устройств, которые должны соединяться между собой, в то время как разработчик шины ввода/вывода должен иметь дело с устройствами, различающимися по задержке и пропускной способности.

Как уже было отмечено, с целью снижения стоимости некоторые компьютеры имеют единственную шину для памяти и устройств ввода/вывода. Такая шина часто называется системной. Персональные компьютеры, как правило, строятся на основе одной системной шины в стандартах ISA или PCI. Необходимость сохранения баланса производительности по мере роста быстродействия микропроцессоров привела к двухуровневой организации шин в персональных компьютерах на основе системной и локальной шин. Локальной шиной называется шина, электрически выходящая непосредственно на контакты микропроцессора. Она обычно объединяет процессор, память, схемы буферизации для системной шины и ее контроллер, а также некоторые вспомогательные схемы. Типичным примером локальной шины является VL-Bus.

Разработка системы шин связана с обеспечением ряда функциональных возможностей, характеризующихся определенными параметрами (табл. 9.1).

Таблица 9.1

| **Возможность** | **Высокая**  **производительность** | **Низкая стоимость** |
| --- | --- | --- |
| Общая разрядность шины | Отдельные линии  адреса и данных | Мультиплексирование  линий адреса и данных |
| Ширина (разрядность)  данных | Чем шире, тем быстрее  (например, 32 бит) | Чем уже, тем дешевле  (например, 8 бит) |
| Размер пересылки | Пересылка нескольких слов имеет меньшие накладные расходы | Пересылка одного слова дешевле |
| Режим и способ  передачи данных | Дуплексный,  параллельный | Симплексный, полудуплекс-  ный, последовательный |
| Главные устройства  шины | Несколько  (требуется арбитраж) | Одно  (арбитраж не нужен) |
| Наличие расщепленных  транзакции? | Да – отдельные пакеты Запроса и Ответа дают большую полосу пропускания (но нужно несколько главных устройств) | Нет – продолжающееся соединение дешевле и имеет меньшую задержку |
| Тип синхронизации | Синхронные | Асинхронные |

Решение о выборе той или иной возможности зависит от целевых параметров стоимости и производительности. Первые три возможности являются очевидными:

* раздельные линии адреса и данных,
* более широкие (имеющие большую разрядность) шины данных,
* режим групповых пересылок (пересылки нескольких слов).

Они дают увеличение производительности за счет увеличения стоимости. Далее на скорость работы шины оказывают влияние режим и способ передачи данных. Существуют следующие режимы передачи данных: симплексный, полудуплексный и полнодуплексный (или просто дуплексный) и способы передачи: параллельный и последовательный.

При симплексном режиме данные передаются только в одном направлении. Используя транспортную аналогию, симплексную передачу можно представить как однонаправленную однополосную дорогу. Сейчас она редко используется па практике.

Полудуплексный режим является самым распространенным. Он похож на однополосную дорогу, по которой движение может осуществляться в обоих направлениях, но не одновременно, а последовательно.

Режим полного дуплекса похож на двухполосную, двунаправленную дорогу. Данные могут передаваться в обоих направлениях одновременно.

Параллельная передача характеризуется тем, что группа битов передается одновременно по нескольким проводникам. Каждый бит передается по собственному проводу. Например, все внутренние коммуникации компьютера с его устройствами осуществляются через параллельную передачу. Это быстрый способ передачи. Однако при больших расстояниях он становится экономически невыгодным не только из-за того, что требует значительно больше кабеля, но и по причине взаимных помех этих проводников.

При последовательной передаче группа битов передается последовательно, один за другим по одному проводнику. Она медленнее, но экономически более выгодна при передаче на большие расстояния.

Также важной характеристикой шины является количество ее главных (задающих) устройств (bus master). Главное устройство шины – это устройство, которое может инициировать транзакцию (передачу) записи или чтения. ЦП, например, всегда является главным устройством шины. Шина может иметь несколько главных устройств, если имеется несколько ЦП или когда устройства ввода/вывода могут инициировать транзакции на шине. Если вналичии несколько таких устройств, то требуется схема арбитража, чтобы решить, кто следующий захватит шину. Арбитраж часто основан либо на схеме с фиксированным приоритетом, либо на более «справедливой» схеме, которая случайным образом выбирает, какое главное устройство захватит шину.

В настоящее время используются два типа шин, отличающиеся способом коммутации:

* шины с коммутацией цепей (circuit-switched bus),
* шины с коммутацией пакетов (packet-switched bus).

Они получили свои названия по аналогии со способами коммутации в сетях передачи данных.

Шина с коммутацией пакетов при наличии нескольких главных устройств шины обеспечивает значительно большую пропускную способность по сравнению с шиной с коммутацией цепей за счет разделения транзакции (передачи) на две логические части: запроса шины и ответа. Такая методика получила название «расщепления» транзакций (split transaction). Транзакция чтения разбивается на транзакцию запроса чтения, которая содержит адрес, и транзакцию ответа памяти, которая содержит данные. Каждая транзакция теперь должна быть помечена (тегирована) соответствующим образом, чтобы ЦП и память могли сообщить, что есть что.

Шина с коммутацией цепей не делает расщепления транзакций, любая транзакция на ней есть неделимая операция. Главное устройство запрашивает шину, после арбитража помещает на нее адрес и блокирует шину до окончания обслуживания запроса. Большая часть этого времени обслуживания при этом тратится не на выполнение операций на шине (например, на задержку выборки из памяти). Таким образом, в шинах с коммутацией цепей это время просто теряется. Расщепленные транзакции делают шину доступной для других главных устройств, пока память читает слово по запрошенному адресу. Это, правда, также означает, что ЦП должен бороться за шину для посылки данных, а память должна бороться за шину, чтобы вернуть данные. Таким образом, шина с расщеплением транзакций имеет более высокую пропускную способность, но обычно она имеет и большую задержку, чем шина, которая захватывается на все время выполнения транзакции. Транзакция называется расщепленной, поскольку произвольное количество других пакетов или транзакций могут использовать шину между запросом и ответом.

Последний вопрос связан с выбором типа синхронизации и определяет, является ли шина синхронной или асинхронной. Если шина синхронная, то она включает сигналы синхронизации, которые передаются по линиям управления шины, и фиксированный протокол, определяющий расположение сигналов адреса и данных относительно сигналов синхронизации. Поскольку практически никакой дополнительной логики не требуется для того, чтобы решить, что делать в следующий момент времени, эти шины могут быть и быстрыми, и дешевыми. Однако они имеют два главных недостатка. Все на шине должно происходить с одной и той же частотой синхронизации, поэтому из-за проблемы перекоса синхросигналов синхронные шины не могут быть длинными. Обычно шины процессор–память синхронные.

Асинхронная шина, с другой стороны, не тактируется. Вместо этого обычно используется старт-стопный режим передачи и протокол «рукопожатия» (handshaking) между источником и приемником данных на шине. Данные передаются как последовательность нулей и единиц, поэтому приемник должен уметь выделять байт в этом потоке данных. При асинхронной передаче каждый байт обрамляется стартовым и стоповым битом, с помощью которых приемник может их разделить. Эта схема позволяет гораздо проще приспособить широкое разнообразие устройств и удлинить шину без беспокойства о перекосе сигналов синхронизации и о системе синхронизации. В целом асинхронная передача является относительно недорогой, потому что не требует дорогостоящего оборудования.

Если может использоваться синхронная шина, то она обычно быстрее, чем асинхронная, из-за отсутствия накладных расходов на синхронизацию шины для каждой транзакции. Выбор типа шины (синхронной или асинхронной) определяет не только пропускную способность, но также непосредственно влияет на емкость системы ввода/вывода в терминах физического расстояния и количества устройств, которые могут быть подсоединены к шине. Асинхронные шины по мере изменения технологии лучше масштабируются. Шины ввода/вывода обычно асинхронные.

# 9.3. Стандарты шин

Обычно количество и типы устройств ввода/вывода в вычислительных системах не фиксируются, что позволяет пользователю самому подобрать необходимую конфигурацию. Шина ввода/вывода компьютера может рассматриваться как шина расширения, обеспечивающая постепенное наращивание устройств ввода/вывода. Поэтому стандарты играют огромную роль, позволяя разработчикам компьютеров и устройств ввода/вывода работать независимо. Появление стандартов определяется разными обстоятельствами.

Иногда широкое распространение и популярность конкретных машин становятся причиной того, что их шина ввода/вывода становится стандартом де-факто. Примерами таких шин могут служить PDP-11 Unibus и IBM PC-AT Bus. Иногда стандарты появляются в результате определенных достижений по стандартизации в некотором секторе рынка устройств ввода/вывода. Интеллектуальный периферийный интерфейс (IPI – Intelligent Peripheral Interface) и Ethernet являются примерами стандартов, появившихся в результате кооперации производителей. Успех того или иного стандарта в значительной степени определяется его принятием такими организациями, как ANSI (Национальный институт по стандартизации США) или IEEE (Институт инженеров по электротехнике и радиоэлектронике). Иногда стандарт шины может быть прямо разработан одним из комитетов по стандартизации: примером такого стандарта шины является FutureBus.

В табл. 9.2 представлены характеристики нескольких стандартных шин.

Таблица 9.2.

| **Параметр** | **VME bus** | **FutureBus** | **Multibus II** | **IPI** | **SCSI** |
| --- | --- | --- | --- | --- | --- |
| Ширина шины  (количество сигналов) | 128 | 96 | 96 | 16 | 8 |
| Мультиплексирование  адреса/данных | Нет | Да | Да | Да | Да |
| Разрядность данных | 16/32 бит | 32 бит | бит 32 | 16 бит | бит |
| Количество главных  устройств шины | Несколько | Несколько | Несколько | Одно | Несколько |
| Максимальная  полоса пропускания | 27.9 Мб/c | 95.2 Мб/c | 40.0 Мб/c | 25.0 Мб/c | 5.0 Мб/c |
| Максимальное  количество устройств | 21 | 20 | 21 | 8 | 7 |
| Максимальная длина шины | 0.5 м | 0.5 м | 0.5 м | 50 м | 25 м |

*Системная шина IBM PC/XT*

Одной из популярных шин персональных компьютеров была системная шина IBM PC/XT, обеспечивавшая передачу 8 бит данных. Кроме того, эта шина включала 20 адресных линий, которые ограничивали адресное пространство пределом в 1 Мбайт. Для работы с внешними устройствами в этой шине были предусмотрены также 4 линии аппаратных прерываний (IRQ) и 4 линии для требования внешними устройствами прямого доступа к памяти (DMA). Для подключения плат расширения использовались специальные 62-контактные разъемы. При этом системная шина и микропроцессор синхронизировались от одного тактового генератора с частотой 4.77 МГц. Теоретическая скорость передачи данных могла достигать немногим более 4 Мбайт/с.

*Системная шина VME*

Шина VME приобрела большую популярность как шина ввода/вывода в рабочих станциях и серверах на базе RISC-процессоров. Эта шина высоко стандартизована, имеется несколько версий этого стандарта. В частности, VME32 – 32-битовая шина с производительностью 30 Мбайт/с, а VME64 – 64-битовая шина с производительностью 160 Мбайт/с.

*Шина ввода/вывода SCSI*

Одной из наиболее популярных шин ввода-вывода является шина SCSI. Под термином SCSI –Small Computer System Interface (Интерфейс малых вычислительных систем) обычно понимается набор стандартов, разработанных Национальным институтом стандартов США (ANSI) и определяющих механизм реализации магистрали передачи данных между системной шиной компьютера и периферийными устройствами. На сегодняшний день приняты два стандарта (SCSI-1 и SCSI-2). Стандарт SCSI-3 находится в процессе доработки.

Начальный стандарт 1986 г., известный теперь под названием SCSI-1, определял рабочие спецификации протокола шины, набор команд и электрические параметры. В 1992 г. этот стандарт был пересмотрен с целью устранения недостатков первоначальной спецификации (особенно в части синхронного режима передачи данных) и добавления новых возможностей повышения производительности, таких, как «быстрый режим» (fast mode), «широкий режим» (wide mode) и помеченные очереди. Этот пересмотренный стандарт получил название SCSI-2 и в настоящее время используется большинством поставщиков вычислительных систем.

Первоначально SCSI предназначался для использования в небольших дешевых системах и поэтому был ориентирован на достижение хороших результатов при низкой стоимости. Характерной его чертой является простота, особенно в части обеспечения гибкости конфигурирования периферийных устройств без изменения организации основного процессора. Главной особенностью подсистемы SCSI является размещение в периферийном оборудовании интеллектуального контроллера.

Для достижения требуемого высокого уровня независимости от типов периферийных устройств в операционной системе основной машины, устройства SCSI представляются имеющими очень простую архитектуру. Например, геометрия дискового накопителя представляется в виде линейной последовательности одинаковых блоков, хотя в действительности любой диск имеет более сложную многомерную геометрию, содержащую поверхности, цилиндры, дорожки, характеристики плотности, таблицу дефектных блоков и множество других деталей. В этом случае само устройство или его контроллер несут ответственность за преобразование упрощенной SCSI-модели в данные для реального устройства.

Стандарт SCSI-2 определяет, в частности, различные режимы: Wide SCSI, Fast SCSI и Fast-and-Wide SCSI. Стандарт SCSI-1 определяет построение периферийной шины на основе 50-жильного экранированного кабеля, описывает методы адресации и электрические характеристики сигналов. Шина данных SCSI-1 имеет разрядность 8 бит, а максимальная скорость передачи составляет 5 Мбайт/с. Fast SCSI сохраняет 8-битовую шину данных и тем самым может использовать те же самые физические кабели, что и SCSI-1. Он отличается только тем, что допускает передачи со скоростью 10 Мбайт/с в синхронном режиме. Wide SCSI удваивает либо учетверяет разрядность шины данных (либо 16, либо 32 бит), допуская соответственно передачи со скоростью либо 10, либо 20 Мбайт/с. В комбинации Fast-and-Wide SCSI возможно достижение скоростей передачи 20 и 40 Мбайт/сек соответственно. Однако поскольку в обычном 50-жильном кабеле просто не хватает жил, комитет SCSI решил расширить спецификацию вторым 66-жильным кабелем (так называемый B-кабель). B-кабель имеет дополнительные линии данных и ряд других сигнальных линий, позволяющих реализовать режим Fast-and-Wide.

*Системные шины ISA и EISA*

Системная шина *ISA (Industry Standard Architecture)* впервые стала применяться в персональных компьютерах IBM PC/AT на базе процессора i286. Эта системная шина отличалась наличием второго, 36-контактного дополнительного разъема для соответствующих плат расширения. За счет этого количество адресных линий было увеличено на 4, а данных – на 8, что позволило передавать параллельно 16 бит данных и обращаться к 16 Мбайт системной памяти. Количество линий аппаратных прерываний в этой шине было увеличено до 15, а каналов прямого доступа – до 7. Системная шина ISA полностью включала в себя возможности старой 8-разрядной шины. Шина ISA позволяет синхронизировать работу процессора и шины с разными тактовыми частотами. Она работает на частоте 8 МГц, что соответствует максимальной скорости передачи 16 Мбайт/с.

С появлением процессоров i386, i486 и Pentium шина ISA стала узким местом персональных компьютеров на их основе. Новая системная шина *EISA (Extended Industry Standard Architecture),* появившаяся в конце 1988 г., обеспечивает адресное пространство в 4 Гбайт, 32-битовую передачу данных (в том числе и в режиме DMA), улучшенную систему прерываний и арбитраж DMA, автоматическую конфигурацию системы и плат расширения. Устройства шины ISA могут работать на шине EISA. Шина EISA предусматривает централизованное управление доступом к шине за счет наличия специального устройства – арбитра шины. Поэтому к ней может подключаться несколько главных устройств шины. Улучшенная система прерываний позволяет подключать к каждой физической линии запроса на прерывание несколько устройств, что снимает проблему количества линий прерывания. Шина EISA тактируется частотой около 8 МГц и имеет максимальную теоретическую скорость передачи данных 33 Мбайт/с.

В настоящее время широко не используется, вытеснена шиной PCI.

*Системная шина PCI*

Первая версия шины PCI (Peripheral Component Interconnect) поддерживает 32-битовый канал передачи данных между процессором и периферийными устройствами, работает на тактовой частоте 33 МГц и имеет пропускную способность 120 Мбайт/с. Важной чертой шины является ее процессорная независимость. Ee легко подключить к различным ЦП: Pentium, Alpha, R4400 и PowerPC. В связи с переходом на 64-разрядный канал данных, она вытеснила шины стандарта EISA с рынка ПК. Хотя обычно для связи устройств компьютера для эффективного соотношения стоимость/производительность используется несколько типов шин. Пример организации компьютера фирмы Intel с несколькими типами шин показан на рис. 9.1.

Шина Локальная Шина

Кэш- шина памяти

Кэш

память

второго

уровня

Мост

PCI

Процессор

Основная память

SCSI

USB

Мост

ISA

Диск IDE

Графический адаптер

Монитор

Принтер

Звуковая карта

Клавиатура

памяти

Свободное

гнездо PCI

Шина PCI

Шина ISA

Модем

Свободное

гнездо ISA

Рис. 9.1

Центральный процессор, память и шина PCI связаны при помощи моста PCI. Процессор обменивается с памятью по локальной шине процессора и специальной шине памяти. Шина ISA, образующая интерфейс расширения для низкоскоростных устройств, связана с шиной PCI при помощи мостов. Так, мост ISA связывает шину PCI с шиной ISA и поддерживает диск IDE. Имеется свободное гнездо PCI для подключения дополнительных высокоскоростных периферийных устройств и свободное гнездо ISA для дополнительного подключения низкоскоростных периферийных устройств.

Разработка шины PCI началась весной 1991 г. как внутренний проект корпорации Intel. Специалисты компании поставили перед собой цель разработать недорогое решение, которое бы позволило полностью реализовать возможности нового поколения процессоров 486/Pentium/P6. В результате шина PCI появилась в июне 1992 г. (Release 1.0). Разработчики Intel отказались от использования шины процессора и ввели еще одну «антресольную» (mezzanine) шину.

Благодаря такому решению шина получилась, во-первых, процессорно-независимой (в отличие от VLbus), а во-вторых, могла работать параллельно с шиной процессора, не обращаясь к ней за запросами. Например, процессор работает с кэшем или с системной памятью, а в это время по сети на винчестер пишется информация. На самом деле не все получается так идеально, но загрузка шины процессора сильно снижается. Кроме того, стандарт шины был объявлен открытым и передан PCI Special Interest Group, которая продолжила работу по совершенствованию шины (в настоящее время доступен R2.1).

*Основные возможности шины PCI следующие:*

* Синхронный 32- или 64- разрядный обмен данными. При этом для уменьшения числа контактов (и стоимости) используется мультиплексирование, т. е. адрес и данные передаются по одним и тем же линиям.
* Поддержка 5V и 3.3V логики. Разъемы для 5 и 3.3V плат различаются расположением ключей; существуют и универсальные платы, поддерживающие оба напряжения; заметим, что частота 66 MHz поддерживается только 3.3V логикой.
* Частота работы шины 33 MHz или 66 MHz (в версии 2.1) позволяет обеспечить широкий диапазон пропускных способностей (с использованием пакетного режима): 32 МВ/с при 32-bit/33 MHz; 264 MB/с при 32-bit/66 MHz; 264 MB/с при 64-bit/ 33MHz; 528 МВ/с при 64-bit/ 66MHz.

При этом для работы шины на частоте 66MHz необходимо, чтобы все периферийные устройства работали на этой частоте.

* Полная поддержка multiply bus master (например, несколько контроллеров жестких дисков могут одновременно работать на шине).
* Поддержка write-back и write-through кэша.
* Автоматическое конфигурирование карт расширения при включении питания.
* Спецификация шины позволяет комбинировать до восьми функций на одной карте (например, видео + звук и т.д.).
* Шина позволяет устанавливать до 4 слотов расширения, однако возможно использование моста PCI-PCI для увеличения количества карт расширения.
* PCI-устройства оборудованы таймером, который используется для определения максимального промежутка времени, в течение которого устройство может занимать шину.

При разработке шины в ее архитектуру были заложены передовые технические решения, позволяющие повысить пропускную способность:

Шина поддерживает метод передачи данных, называемый «linear burst» (метод линейных пакетов). Этот метод предполагает, что пакет информации считывается (или записывается) «одним куском», т. е. адрес автоматически увеличивается для следующего байта. Естественным образом при этом увеличивается скорость передачи собственно данных за счет уменьшения числа передаваемых адресов.

*Шина AGP*

Эта шина имеет следующие существенные отличия от шины PCI:

* шина способна передавать два блока данных за один 66 MHz цикл (AGP 2x);
* устранена мультиплексированность линий адреса и данных (напомню, что в PCI для удешевления конструкции адрес и данные передавались по одним и тем же линиям);
* дальнейшая конвейеризация операций чтения/записи, по мнению разработчиков, позволяет устранить влияние задержек в модулях памяти на скорость выполнения этих операций.

В результате пропускная способность шины была оценена в 500 МВ/с, и предназначена она для того, чтобы видеокарты хранили текстуры в системной памяти, соответственно имели меньше памяти на плате, и, соответственно, дешевели. Парадокс в том, что видеокарты все-таки предпочитают иметь БОЛЬШЕ памяти, и ПОЧТИ НИКТО не хранит текстуры в системной памяти, поскольку текстур такого объема пока практически нет. При этом в силу удешевления памяти вообще карты особенно и не дорожают.

Шина имеет два основных режима работы: Execute и DMA. В режиме DMA основной памятью является память карты. Текстуры хранятся в системной памяти, но перед использованием (тот самый execute) копируются в локальную память карты. Таким образом, AGP действует в качестве «тыловой структуры», обеспечивающей своевременную «доставку патронов» (текстур) на передний край (в локальную память). Обмен ведется большими последовательными пакетами. В режиме Execute локальная и системная память для видеокарты логически равноправны. Текстуры не копируются в локальную память, а выбираются непосредственно из системной. Таким образом, приходится выбирать из памяти относительно малые случайно расположенные куски. Поскольку системная память выделяется динамически, блоками по 4 К, в этом режиме для обеспечения приемлемого быстродействия необходимо предусмотреть механизм, отображающий последовательные адреса на реальные адреса 4 килобайтных блоков в системной памяти. Эта нелегкая задача выполняется с использованием специальной таблицы (Graphic Address Re-mapping Table или GART), расположенной в памяти.

При этом адреса, не попадающие в диапазон GART (GART range), не изменяются и непосредственно отображаются на системную память или область памяти устройства (device specific range). На рис. 5 в качестве такой области показан локальный фрейм-буфер карты (Local Frame Buffer или LFB). Точный вид и функционирование GART не определены и зависят от управляющей логики карты.

Шина AGP полностью поддерживает операции шины PCI, поэтому AGP-траффик может представлять собой смесь чередующихся AGP и PCI операций чтения/записи. Операции шины AGP являются раздельными (split). Это означает, что запрос на проведение операции отделен от собственно пересылки данных.

Такой подход позволяет AGP-устройству генерировать очередь запросов, не дожидаясь завершения текущей операции, что также повышает быстродействие шины.

В 1998 г. спецификация шины AGP получила дальнейшее развитие – вышел Revision 2.0. В результате использования новых низковольтных электрических спецификаций появилась возможность осуществлять 4 транзакции (пересылки блока данных) за один 66-мегагерцовый такт (AGP 4x), что означает пропускную способность шины в 1GB/с.

Однако потребности и запросы в области обработки видеосигналов все возрастают, и Intel готовит новую спецификацию – AGP Pro (в настоящее время доступен Revision 0.9), направленную на удовлетворение потребностей высокопроизводительных графических станций. Новый стандарт не видоизменяет шину AGP. Основное направление – увеличение энергоснабжения графических карт. С этой целью в разъем AGP Pro добавлены новые линии питания.

# Организация системы ввода-вывода в ВМ

# 10.1. Назначение и основные требования к системе ввода-вывода ВМ

***Назначение системы ввода-вывода*** – это обеспечение взаимодействия центральной части машины с внешней средой (пользователи, устройства, процессы), которое реализуется периферийными (или внешними) устройствами (ПУ или ВУ).

Связь процессора с периферийными устройствами осуществляется:

1. Через аккумулятор (регистр общего назначения).
2. Через оперативную память.

Оперативная память может непосредственно взаимодействовать с внешним устройством (ВУ) в режиме прямого доступа в память (ПДП).

***Основные проблемы ввода-вывода.***

1. Существует большое количество ВУ с различными параметрами, существенно отличающимися:
2. По скорости передачи данных.
3. Формату передачи данных (устройства последовательного, параллельного представления информации и т.д.).
4. Размерам передаваемых данных (биты, байты, слова, блоки, секторы).
5. Количеству выполняемых функций (чтение, запись, перемотка, подсчет, измерение).
6. Различные скорости работы центральной части машины и ВУ (требуется многоуровневая буферная память).
7. Асинхронность работы центральной части машины и ВУ (согласование операций в устройствах).

***Требования к системе ввода-вывода***

1. Гарантировать эффективное согласование центральной части машины и ВУ с целью достижения максимальной производительности.
2. Обеспечение распределения ВУ между одновременно выполняемыми задачами в системе (большинство машин работают в многозадачных режимах).
3. Обеспечение управления каждым конкретным внешним устройством.
4. Обеспечение дружественного интерфейса с пользователем.

# 10.2. Архитектура систем ввода-вывода

Существует два основных способа организации системы ввода вывода.

1. Прямой ввод-вывод (рис. 10.1).
2. Косвенный (канальный) ввод-вывод (рис.10.2).

1 ОП

Системная шина

ЦП

2

ОП Контроллер ВУ Контроллер ВУ

ВУ ВУ1 ВУ2 ВУ3

Механизм

Носитель  
 информации

Рис.10.1

При прямом вводе-выводе работа внешних устройств и памяти управляется центральным процессором, и все они подключаются к одной системной шине, при этом в зависимости от варианта подключения памяти может быть: в случае 1 – одно пространство адресов памяти и портов внешних устройств, а в случае 2 – отдельные адреса обращения к памяти и внешним устройствам. Процесс взаимодействия внешних устройств и центральной части машины определяется интерфейсом ввода вывода, под которым понимается совокупность сигналов, линий связи и алгоритмов управления, обеспечивающих заданный протокол взаимодействия внешних устройств и процессора. Под протоколом понимается последовательность формирования прямых и квитирующих сигналов взаимодействия (ответный сигнал называется квитирующим).

ВУ1 ВУ2 ВУ3 ВУ4

Контроллер 1 Контроллер 2

Канал 1

(Мультиплексный)

ЦП ОП

Канал 2 Контроллер ВУ (Селекторный)

Рис.10.2

На рис10.2 показан процесс ввода-вывода по способу, называемому канальным.

Каналы делятся на следующие.

1. Мультиплексные (обслуживают много ВУ, но медленных).
2. Селекторные (обслуживают мало ВУ, но быстродействующих).

Отличительная особенность канального ввода-вывода - процессор освобождается от управления внешними устройствами, функция процессора заключается в инициализации запуска канальных программ и завершении операции ввода-вывода, выполняемой с помощью канала, по соответствующему признаку из канала (по существу это многопроцессорная система).

***Состав и сравнение функций контроллеров и каналов***

*Контроллер*

1. В текущий момент времени он выполняет одну команду ввода-вывода, получаемую от процессора или канала, и одновременно обслуживает одно внешнее устройство.

В его функции входит:

1. Опознание своего адреса выборки.
2. Подтверждение готовности внешних устройств.
3. Управление операцией во внешнем устройстве.
4. Согласование форматов данных.
5. Согласование скоростей передачи (буферизация).
6. Фиксация момента и характера операции ввода-вывода.

Контроллер должен содержать:

1. Селектор адреса (логическая схема, выдающая разрешающий сигнал на один адрес).
2. Регистры управления (содержит команду) и состояния. Состояния характеризуются следующими битами: DONE, BUSY, ERROR, часто применяются биты приоритета.
3. Буферные регистры данных, которые служат для согласования форматов и скоростей передачи.

*Канал*

1. Выполняет целую канальную программу из многих команд.
2. Допускает одновременное управление несколькими внешними устройствами.

В его функции входит:

1. Опознание своего адреса и подтверждения готовности.
2. Прием команд процессора, инициализирующих работу канала, и нахождения в памяти своей канальной программы.
3. Поиск контроллера и внешнего устройства, участвующего в операции, и проверки их готовности.
4. Запуск канальной программы и управление обменом.
5. Сообщение центрального процессора о завершении операции и всей программы.

Канал представляет собой специализированный процессор с расширенными управлениями, и ограниченный арифметическими возможностями.

# 10.3. Способы выполнения операции передачи данных

Классификация основных способов выполнения передачи данных показан на рис. 10.3, а на рис.10.4-17.7 представлены схемы выполнения каждого из способов передачи данных.

Передача данных

Синхронная Асинхронная

Несовмещенный Вв/выв по программному

вв/выв прерыванию

Вв/выв по аппаратному

прерыванию

Рис 10.3

***Синхронная передача данных***

|  |  |
| --- | --- |
| ↓ | ← Команда вв/выв |
| Передача 1 |  |
| ↓ | ← Команда вв/выв |
| Передача 2 |  |
| ↓ |  |

Рис. 10.4. Синхронная передача данных - самый быстрый, но ненадежный способ передачи данных.

***Несовмещенный ввод-вывод***

Команда

Вв/выв

Фоновая

программа Нет ВУ

готово?

Да

Передача

Рис.10.5

……

В данном случае центральный процессор после выдачи команды ввода-вывода ожидает готовности ВУ, проверяя установку флажка готовности (говорят, что процессор «висит» на флажке готовности). Этот бит представляет собой своеобразный семафор, который управляет доступом к данным либо от процессора, либо от ВУ. Такой способ обмена удобен для внешнего устройства и не эффективен для процессора. Так как ВУ диктует условия, то со стороны ВУ скорость передачи данных – максимально для него возможная. Иногда удается в цикле ожидания вставить выполнение процессором фоновой программы.

***Ввод-вывод по программному прерыванию***

Основная программа(ОП)

Запрос ВУ

Подпрограмма

обслуживания

ВУ (передача)

…….

(команда Вв/вывода)

Продолжение ОП

Рис. 10.6

В данном случае ВУ для осуществления передачи устанавливает запрос на прерывание работы процессора и, если прерывание возможно, то подпрограмма обслуживания прерывания выполняет передачу данных, по завершении которой происходит возврат на продолжение основной программы.

Данный способ обмена удобен для процессора и неэффективен для ВУ, в таком режиме к процессору подключаются медленные устройства с произвольными моментами готовности к передаче данных.

***Ввод-вывод по аппаратному прерыванию (прямой доступ к памяти)***

Основная программа

(Инициализация ПДП)

Запрос 1

на передачу

цикл памяти

(передача 1)

Запрос 2

на передачу цикл памяти

(передача 2)

Рис. 10.7

Процесс передачи осуществляется без участия центрального процессора под управлением котроллера прямого доступа в память (ПДП). Процессор только производит начальную настройку контроллера ПДП, а затем выполняет собственную программу. В моменты готовности данных к передаче у процессора только занимается один цикл памяти, необходимый для выполнения передачи. В таком режиме обычно выполняется передача блоков данных для уменьшения накладных расходов, связанных с инициализацией контроллера ПДП перед операцией ввода-вывода.

Этот режим обеспечивает параллельную работу процессора и выполнение операций ввода-вывода, он используется для подключения быстрых ВУ, так как передача идет между памятью и ВУ, и управление передачей происходит аппаратно.

# 10.4. Структуры контроллеров ВУ для различных режимов передачи данных

*Контроллер несовмещенного ввода-вывода (КНВВ)*

С ШУ (состояние) & РгУС

и

с Done Err Выв/Вв т ША

е Селектор

м адреса

н

а ШУ(команды) & ВУ

я

ш & РгД

и ЩД (Выв) Вв

н

а &

ЩД (Вв)

Выв

Рис.10.8

На рис. 10.8 показан контроллер несовмещенного ввода-вывода.

Основы узла контроллера несовмещенного ввода-вывода (КНВВ):

* Селектор адреса, который выдает на выходе единицу (разрешающий сигнал), только для одного адреса (своего адреса);
* РгУС – регистр управления и состояния, который принимает от шины управления команду ввода-вывода, а на выход передает состояние ВУ до и после выполнения команды.
* БРгД – буферный регистр данных осуществляет промежуточное хранение данного для согласования форматов и выравнивания скоростей ВУ и процессора.

*Контроллер обмена по программному прерыванию(ОПП)*

Контроллер ОПП состоит из двух частей:

1. Индивидуальный контроллер управления ВУ (аналогичен КНВВ).
2. Общий или выделенный контроллер обслуживания прерываний.

С НВП

И IRR

С (ША)

Т

Е

М INT IRQ N &

Н (ШУ) ISR

А

Я

(ШУ)

Ш

И IMR

Н

А

IM

PRIOR

(ШД) EI DONE PRIOR … РгУс

Контроллер прерываний КНВВ

Рис. 10.9

INTA

INTA

На схеме рис. 10.9 используются обозначения:

IRR – регистр приема запросов, ISR – регистр обслуживания запросов,

IMR – регистр маскирования прерываний, PRIOR – арбитр, схема разрешения приоритетов,

IM – маска прерывания (выдается по шине данных ШД),

Int – линия шины управления ШУ для выдачи запроса прерывания в процессор,

Inta – линия ШУ для подтверждения разрешения на прерывание,

НВП – Номер Вектора Прерывания (выдается по шине адреса ША),

EI – бит разрешения прерывания в РГУС КНВВ,

Done – бит завершения операции ввода-вывода в РГУС КНВВ.

Если сигнал INTA = ‘разрешено', то тогда контроллер прерываний выставляет НВП на ША.

Далее идет обычная передача данных через КНВВ с той разницей, что процессор не висит на флаге ожидания. Удобно для процессора, но неудобно для ВУ, так как устройство должно ожидать, пока контроллер получит разрешение от процессора на прерывание.

*Контроллер передачи данных в режиме аппаратного прерывания  
 (или прямого доступа к памяти – КПДП)*

(ЦП) КПДП

BACK DACK

Контр. CAR CWR Mode R КВУ

шины (Рг ТА) (Сч Слов) (Рг Реж)

BRK DRQ

Внутренняя шина

Контр. BAR BCR Управ- ША

прерыв. (Рг БА) (Рг БС) ление ШД

ШУ

ШУ ША ШД ШУ

С и с т е м н а я ш и н а

РиР

Рис. 17.3

Основные компоненты КПДП

CAR – регистр текущего адреса – работает в режиме автоинкрементирования; увеличивается на 1 или 2 после каждой передачи байта или слова данных.

CWR – счетчик слов (или байт), который работает в режиме автодекрементирования: после каждой передачи содержимое регистра уменьшается на 1(байт) или 2 (слово).

ModeR – регистр режима передачи (определяет характер и направление передачи).

BAR – регистр базового адреса, содержит начальный адрес памяти и в процессе передачи не изменяется (служит для средств контроля передачи).

BCR – регистр базового счетчика, содержит первоначальное количество передаваемых слов (служит для средств контроля передачи).

Управление – управление организацией передачи по внутренней шине КПДП.

По ШУ в КПДП передаются следующие сигналы:

MemR - чтение памяти

IOR - чтение устройства

MemW - запись памяти

IOW - запись устройства

В режиме прямого доступа к памяти выполняется блочная передача данных между памятью и быстрыми ВУ (HDD, Sound card и т. д.). Управление передачей происходит без участия процессора (процессор должен только инициализировать регистры контроллера).

В обеспечении передачи задействованы все три вида контроллеров:

1. Контроллер ПДП.
2. Контроллер прерываний, который в данном случае формирует сигнал в процессор о завершении передачи в режиме прямого доступа в память.
3. Индивидуальный контроллер КНВВ, управляющий операцией Вв/выв в конкретном ВУ.

*Алгоритм передачи данных в режиме ПДП*

1. После подготовки устройством данного (или готовности к приему очередного данного) оно выставляет запрос DRQ на передачу в режиме прямого доступа.
2. По его получении КПДП выставляет запрос BRK на возможность захвата шины.
3. Вырабатывается подтверждающий сигнал BACK от контроллера памяти.
4. Вырабатывается подтверждающий сигнал DACK от КПДП к внешнему устройству.
5. Далее выдается пара команд: либо {MemR и IOW}, либо {MemW и IOR }, реализующие собственно передачу данного между ВУ и памятью.
6. После этого сбрасывается сигнал запроса шины и разрешения доступа к памяти: BRQ ← 0, DACK ← 0.
7. Изменяется содержимое регистра РгТА: TA ++ и счетчика слов: CC -- .

Пункты с 1 по 7 называются «занятие цикла памяти» и выполняются для передачи одного данного. Они повторяются до тех пор, пока не выполнится условие СС = 0, по которому формируется запрос в контроллер прерываний, в результате обслуживания которого процессор узнает о завершении передачи блока данных в режиме ПДП.

В компьютерах iХ86 в качестве КПДП используются микросхемы:

1. 8237A – XT (содержит четыре канала с номерами 0–3, работает с однобайтовыми портами).
2. 8237A – 5 – AT и выше (содержит две группы по четыре канала, первая – как у XT (для поддержки), а вторая – для работы с двухбайтовыми портами).

***Состав регистров и адресация 8237A:***

00h – 07h – номера регистров по каждому из каналов.

0Ah регистр маски разрешает и запрещает работу по каждому из каналов.

номера битов:

0, 1 – номер канала

2 – разрешить, запретить

3…7 – не используются

0Bh – регистр режима

номера битов:

0, 1 – номер канала

2, 3 – тип цикл DMA

00 – цикл проверки

01 – цикл занятости

10 – цикл чтения

11 – запрещенная комбинация

4 – режим инициализации

5 – направление приращения адреса

0 – увеличение

1 – уменьшение

6, 7 – режим обслуживания

00 – по требованию

01 – одиночная передача

10 – блочная передача

11 – каскадирование (использование двухуровневых контроллеров)

0Ch – сброс триггеров байта

0Eh – сброс регистра маски

81h…8Fh – регистры страниц, текущий адрес равен 16 бит, шина адреса и шина данных (прямого доступа к памяти) равны 20 бит, регистры страниц задают адрес страницы (старшие четыре бита адреса – работают с 1 Мб), одна страница – 64 Кб.

Для 8237A-5-AT используются 3-байтовые регистры страницы (24 бит) и появляется возможность адресовать до 16 Мб оперативной памяти.

# Программные средства управления вводом-выводом (ПС УВВ)

# Состав ПС УВВ

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | ПСУВВ | | | | | |  | | | | |
|  | | | | | | | | | |  | | | | | | | |
| Управление данными | | | | | | | |  | | | | Управление устройствами | | | | | | |
| ↓ |  | ↓ |  | ↓ |  | ↓ | | |  | | ↓ | | |  | ↓ |  | ↓ |
| Блочно–ориент. I/O | Записе- ориен. I/O | FCB доступ | FH доступ | | | Распред. ВУ между задачами | | | Подсистема буфери-зации | Драйверы внешних устройств |

Рис. 11.1

На рис.11.1 показаны две основных части ПС УВВ: управление данными и управление устройствами.

Подсистема управления данными обеспечивает:

- блочно-ориентированный ввод-вывод для файлов последовательного доступа;

- записе-ориентированный ввод-вывод для файлов прямого доступа.

Система управления устройствами содержит три подсистемы:

Диспетчеризация устройств между задачами.

Подсистема буферизации загрузки и выгрузки данных

Набор драйверов для управления операциями ввода-вывода в конкретных ВУ.

# 11.2. Основные компоненты процедуры управления ввода-вывода общего вида

1. Подготовительная (инициализирующая) часть:

Формирование областей для хранения данных и создание указателей на эти области.

Указания типа и логического номера внешнего устройства.

Задание режима передачи данных и направления передачи.

Этот этап обычно реализует процессор, который осуществляет подготовку.

1. Планирование, проверка готовности и запуск устройства.

Выбор канала.

Выбор внешнего устройства.

Замена логических номеров на физические адреса.

Проверка готовности для выполнения операции.

Запуск устройства для работы.

1. Выполнение устройством операции ввода-вывода

Для реализации используются драйверы и программы управления данными

1. Завершение операции ввода-вывода; следовательно освобождаются занимаемые области памяти и устройства, участвовавшие в операции.

# 11.3 Состав и реализация устанавливаемого драйвера символьного типа

Возможны четыре варианта использования драйвера:

1. Непосредственно написание драйвера на уровне битов портов ввода-вывода и включение его в собственную программу; используется в случае редко используемых драйверов и специализированных машин (малоперспективно).
2. Подключение драйвера в оверлейном режиме (сокращение занимаемой области памяти).
3. Написание драйвера как резидентного обработчика прерываний. Достоинство: доступен всем программам. Недостаток: как правило, доступен только из ассемблера.
4. Создание устанавливаемого в операционной системе драйвера, который включается в файл конфигурации config.sys с помощью команды типа device = path\my\_driver

Хорош тем, что драйвер полностью поддерживается средствами операционной системы. Соответственно к нему применимы команды операционной системы вида:

copy a:\ my\_driver c:\drv

Во-вторых, с ним можно работать из языков высокого уровня с помощью операций работы с файлами.

Устанавливаемые драйверы бывают двух типов: блочные и символьные.

Первые используют файловую организацию и передачу данных блоками (обычно применяются для работы с дисковыми накопителями). Вторые используют посимвольную передачу данных (она проще) и применимы к любым внешним устройствам.

Символьный драйвер состоит из следующих элементов:

1. Заголовок драйвера.
2. Процедура стратегии.
3. Буфер запроса.
4. Обработчик прерываний подключает таблицу функций с набором операций, которые могут выполняться данным устройством.

1.  Заголовок драйвера (18 байт)

1. Адрес следующего драйвера (4 байт).
2. Атрибуты (2 байт).
3. Смещение процедуры стратегии (2 байт).
4. Смещение обработчика прерываний (Обр\_прер).
5. Имя устройства (8 байт).

Заголовок драйвера Буфер запроса

1)Адрес след.драйвера 1)Длина буф. запроса

2)Атрибуты 2)Код ВУ

3)Смещение (3) 3)Код команды Вв/выв

4)Смещ. обр-ка прер. (2) 4)Статус

5)Имя устройства 5)Резерв

Процедура стратегии (4) Данные

(5)

Обработчик прерываний (не найден)

| инициализация

ТаблФ | ввод

| вывод

(6) (7)

ПП инициализации

ПП ввода

(функция выполняемой команды)

ПП вывода

(8)

RET статус завершения

возврат

в прикладную

программу

Драйвер обычно записывается как самостоятельный модуль, но без PSP, поэтому не может запускаться самостоятельно.

1. Атрибуты:

15 бит : 1- символьный драйвер

0 – блочный драйвер

14 бит : поддержка IOCTL

13 бит : формат блоков

1 – IBM

0 – любой другой

………

3 бит : 1 – часы

0 – не часы

2 бит : 1 – null

0 – не null

1 бит : 1 – STDOUT (стандарт вывода)

0 - STDIN (стандарт ввода)

2. Процедура стратегии.

Процедура стратегии выполняются только один раз на этапе загрузки и служит для запоминания длинного указателя на буфер запроса, создаваемый для драйвера самой операционной системой.

DEV\_STRAT:

mov cs: SEG\_PQBF, es ;

mov cs: OFF\_PQBF, es ;

ret

SEG\_PQBF DW ?

OFF\_PQBF DW ?

3. Буфер запроса.

Структура данных, через которые прикладная программа связывается с драйвером (прикладная программа задает вид операции ввода-вывода и место расположения данных, а драйвер возвращает ей свой статус (все идет через буфер запроса)).

Буфер состоит их стандартной части 13 байт, называемой заголовок и последующей части, называемой данные, содержащей различную информацию в зависимости от вида драйвера и исполнительной операции ввода-вывода.

Заголовок запроса.

+0 : длина буфера запроса

+1 : код внешнего устройства

+2 : код команды ввода-вывода

+3 : статус

+5 : резерв

+13d : данные

4. Обработчик прерывания.

Это только процедура, которая таблично вызывает функцию, реализующую команду ввода-вывода; она завершается командой RET и выдачей статуса.

Таблицы А.Ф. Ком 1

Функции . . . . . . . . . . .

. . . . . . . . . . .

А.Ф. Ком N

Ф. Ком 1

Функции . . . . . . . . .

. . . . . . . . .

Ф. Ком N

**Код обработчика прерываний**

dev\_intpr: push A

mov ax, SEG\_PQBF ;

mov es, ax ;

mov bx, OFF\_PQBF ;

mov al, es:[bx+2] ;

shl al, 1 ;

xor ah, ah ; Подготовка и загрузка

lea di, FUN\_TAB ; смещения функции

add di, ax ; в таблицу функции

jmp word ptr[DI] ;

FUN\_TAB LABEL WORD;

⎧ DW INITIALIZE

⎪ DW CHECK\_MEDIA

⎪ . . . . . . .

⎪ DW INPUT\_DATA

Тринадцать ⎨ . . . . . . .

функций ⎪ DW OUPUT\_DATA

⎪ . . . . . . .

⎪ DW OUTPUT\_STATUS

⎪ . . . . . . .

⎩ DW IOCTL\_OUT

Определенная функция находится в определенной строке таблицы.

Предположим что драйвер поддерживает только две функции init и out, тогда:

CHECK\_MEDIA:

. . . . . .

INPUT\_DATA:

. . . . . .

IOCTL\_OUT:

or es.word ptr[bx]+3, 8103h

JMP QUIT

INITIALIZE:

lea ax, E\_O\_P

mov es, word ptr[bx]+14, ax

mov es, word ptr[bx]+16, cs

JMP QUIT

OUTPUT\_DATA

QUIT:

OR es:word ptr[bx]+3, 100h

POPA

RET

В поле статус, бит

15 – ошибка

9 – драйвер занят

8 – функция завершена

с 0 – 7 – код ошибки

Пример: 8103h означает 8 – ошибка , 1 – операция завершена, 03 – неизвестная ошибка.

После разработки драйвера нужно вставить его в config.sys

Надо учесть, что драйвер не прощает ошибок. Так как он скрыт от пользователя, поэтому нельзя пользоваться отладчиком.

# Список литературы

1. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. СПб.: Питер, 2014.

2. Паттерсон Д., Хеннесси Дж. Архитектура компьютера и проектирование компьютерных систем. 4-е изд. СПб.: Питер, 2012.

3. Жмакин А.П. Архитектура ЭВМ. 2-е изд. СПб.: БХВ-Петербург, 2010.

4. Орлов С.А., Цилькер Б.Я. Организация ЭВМ и систем. 2-е изд. СПб.: Питер, 2011.

5. Гук М. Аппаратные средства IBM PC. Энциклопедия 2-е изд. СПб.: Питер, 2002.

6. Зубков С.В. Assembler. Для DOS, Windows и Unix. М.: ДМК, 1999.

7. Касперски К. Техника оптимизации программ. Эффективное использование памяти. СПб.: БХВ-Петербург, 2003.

Дополнительная

1. Компьютеры на СБИС: В 2 кн. Кн.1 / Пер. с япон.; Мотоока Т., Томита С., Танака Х. и др. М.: Мир, 1988.

2. Королев Л.Н. Микропроцессоры, микро- и мини-ЭВМ. – М.: Изд-во МГУ, 1988

3. Рудометов Е., Рудометов В. Архитектура ПК, комплектующие, мультимедиа. СПб.: Питер, 2000.

4. Рудаков П.И., Финогенов К.Г. Программируем на языке ассемблера IBM PC. Обнинск: Принтер, 1999.

5. Фрир Дж. Построение вычислительных систем на базе перспективных микропроцессоров / Пер. с англ. М.: Мир, 1990.

6. Фролов А.В., Фролов Г.В. Аппаратное обеспечение персонального компьютера. М.: Диалог-МИФИ, 1997 (Библиотека системного программиста; т.33).

7. Джордейн Р. Справочник программиста ПК типа IBM PC XT и AT / Пер. с англ. М.: Финансы и статистика, 1991.

# ПРИЛОЖЕНИЯ

**П1. Система команд процессора iX86**

***П1.1.  Команды передачи данных***

a)  **MOV** приемник, источник; 8086, пересылка данных.

mov d, s ; (s)🡪d ; d – destination (приемник); s – source (источник).

Команда MOV действует аналогично операторам присваивания в языках высокого уровня. Оба операнда должны быть одного и того же раз­мера – байт, слово или двойное слово. Нельзя выполнять пересылку данных с помощью MOV из одной пере­меной (ячейки памяти) в другую, из одного сегментного регистра в другой, и нельзя поме­щать в сегментный регистр непосредственный операнд. Для пересылок из одного сегментного регистра в другой используют две команды MOV (из сегментного регистра в обычный и уже из него в другой сегментный) или пару команд PUSH/POP (более длинный по времени способ).

b) **XCHG** операнд1, операнд2; 8086, обмен операндов между собой;

автоматически устанавливает режим LOCK

xchg al, ah ;

xchg bx, MEM ;

Содержимое операнда 2 копируется в операнд 1, а старое содержимое операнда 1 –

на место операнда 2.

c) **LEA** приемник, источник; 8086, вычисление эффективного (исполнительного) адреса.

NEAR: lea bx, MEM ; эта команда аналогична команде mov bx, offset MEM.

Вычисляет эффективный адрес источника (переменная) по отношению к началу сегмента (по умолчанию DS) и помещает его в приемник (регистр).

d) **LDS** приемник, источник; 8086, загрузить исполнительный адрес, используя DS.

**LES** приемник, источник; 8086, загрузить исполнительный адрес, используя ES.

FAR: lds bx, MEM ; аналогична трем командам:

{ mov bx, offset MEM ; mov ax, seg MEM ; mov ds, ax }

Операнд-источник для этих команд – всегда переменная в памяти. Смещение адреса этой переменной загружается в РОН – приемник, а сегментная часть адреса – в сегментный регистр, заданный в операции.

e) **PUSH** источник; 8086, поместить данные в стек.

**POP** приемник; 8086, считать данные из стека.

Для пересылок между регистрами обычно применяются в виде пары команд:

**push s**

**pop d**

PUSH помещает содержимое источника в стек. Источником может быть ре­гистр, сегментный регистр, непосредственный операнд или переменная. Фактически эта команда копирует содержимое источника в память по адресу SS:[SP] и уменьшает SP на размер источника в байтах. Начиная с 80286, команда PUSH SP помещает в стек зна­чение SP до того, как эта же команда его уменьшит, в то время как на 8086 SP помещался в стек уже уменьшенным на два.

POP помещает в приемник слово или двойное слово, находящееся в вер­шине стека, увеличивая SP на два. POP выполняет действие, полностью обратное PUSH. Приемником может быть регистр общего назначения, сегментный регистр, кроме CS, или переменная. Если в роли приемника выступает операнд, использующий ESP для косвен­ной адресации, команда POP вычисляет адрес операнда уже после того, как она увеличивает ESP.

Начиная с i386, появились команды **pusha** и **popa** – загрузка и выгрузка 8 регистров общего назначения.

f) **IN** приемник, источник; 8086, считать данные из порта.

Копирует число из порта ввода-вывода, номер которого указан в ис­точнике, в приемник. Приемником может быть только AL, АХ или ЕАХ. Источник – или непосредственный операнд (можно указывать номера портов не больше 255), или DX (можно указывать номера портов до 65535).

g) **OUT** приемник, источник; 8086, записать данные в порт.

Копирует число из источника (AL, АХ или ЕАХ) в порт ввода-выво­да, номер которого указан в приемнике. Приемник может быть либо не­посредственным номером порта, либо регистром DX.

На командах IN и OUT строится все общение процессора с устройствами ввода-выво­да.

***П1.2. Команды арифметических операций над целыми числами***

a) **ADD** приемник, источник; 8086, сложение.

add d, s; s + d 🡪 d

Команда выполняет арифметическое сложение приемника и источ­ника и помещает сумму в приемник. Приемник может быть регистром или переменной, источник может быть числом, регистром или переменной, но нельзя использовать пере­менную одновременно и для источника, и для приемника. Команда ADD никак не различает числа со знаком и без знака, но, употребляя значе­ния флагов CF (перенос при сложении чисел без знака), OF (перенос при сложении чисел со знаком) и SF (знак результата), можно исполь­зовать ее и для тех, и для других.

b) **ADC** приемник, источник; 8086, сложение с переносом

add d, s; s + d + CF 🡪 d

Эта команда во всем аналогична ADD, кроме того, что она выполняет арифметическое сложение приемника, источника и флага CF. Пара ко­манд ADD/ADC используется для сложения чисел повышенной точно­сти.

c) **SUB** приемник, источник; 8086, вычитание.

sub d, s; d – s 🡪 d

Вычитает источник из приемника и помещает разность в приемник. Приемник может быть регистром или переменной, источник может быть числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника, и для приемника. Точно так же, как и ко­манда ADD, SUB не делает различий между числами со знаком и без зна­ка, но флаги позволяют использовать ее и для тех, и для других.

d) **SBB** приемник, источник; 8086, вычитание с заемом.

sbb d, s; d – s – CF 🡪 d

Эта команда во всем аналогична SUB, кроме того, что она вычитает из приемника значение источника и дополнительно вычитает значение флага CF.

e) **СМР** приемник, источник; 8086, сравнение.

cmp d, s; d – s

Сравнивает приемник и источник и устанавливает флаги. Сравнение осуществляется путем вычитания источника (число, регистр или пере­менная) из приемника (регистр или переменная; приемник и источник не могут быть переменными одновременно), причем результат вычита­ния никуда не записывается, единственным результатом работы этой команды оказывается изменение флагов CF, OF, SF, ZF, AF и PF. Обыч­но команду СМР используют вместе с командами условного перехода (Jcc).

f) **MUL** источник; 8086, умножение чисел без знака.

mul bl; bl \* al 🡪 ax

Выполняет умножение содержимого источника (регистр или пере­менная) и регистра AL, АХ, ЕАХ (в зависимости от размера источника) и помещает результат в АХ, DX:AX, EDX:EAX соответственно. Если старшая половина результата (АН, DX, EDX) содержит только нули (ре­зультат целиком поместился в младшую половину), флаги CF и OF ус­танавливаются в 0, иначе – в 1. Значение остальных флагов (SF, ZF, AF и PF) не определено.

g) **IMUL** источник; 8086, 80386, умножение чисел со знаком.

**IMUL** приемник, источник;

**IMUL** приемник, источник1, источник2;

Эта команда имеет три формы, различающиеся числом операндов:

1. IMUL источник: источник (регистр или переменная) умножается на AL, АХ или ЕАХ (в зависимости от размера операнда), и результат располагается в AX, DX:AX или EDX:EAX соответственно.

2. IMUL приемник, источник: источник (число, регистр или переменная) умножается на приемник (регистр), и результат заносится в приемник.

3. IMUL приемник, источник 1, источник 2: источник 1 (регистр или пе­ременная) умножается на источник 2 (число), и результат заносится в приемник (регистр).

Во всех трех вариантах считается, что результат может занимать в два раза больше места, чем размер источника. В первом случае приемник автоматически оказывается достаточно большим, но во втором и третьем случаях могут произойти переполнение и потеря старших бит результата. Флаги OF и CF будут равны единице, если это произошло, и нулю, если ре­зультат умножения поместился целиком в приемник (во втором и третьем случаях) или в младшую половину приемника (в первом случае).

Значения флагов SF, ZF, AF и PF после команды IMUL не определены.

h) **DIV** источник; 8086, целочисленное деление без знака.

div bl; ax / bl 🡪 al, ah частное в al, остаток в ah

Выполняет целочисленное деление без знака АХ или ЕАХ (в за­висимости от размера источника) на источник (регистр или перемен­ная) и помещает результат в AL, АХ или ЕАХ, а остаток – в АН, DX или EDX соответственно. Результат всегда округляется в сторону нуля, аб­солютное значение остатка всегда меньше абсолютного значения дели­теля. Значения флагов CF, OF, SF, ZF, AF и PF после этой команды не определены, а переполнение или деление на ноль вызывает исключение #DE (ошибка при делении) в защищенном режиме и прерывание 0 – в реальном.

i) **IDIV** источник; 8086, целочисленное деление со знаком.

Выполняет целочисленное деление со знаком AL, АХ или ЕАХ (в за­висимости от размера источника) на источник и помещает результат в AL, АХ или ЕАХ, а остаток – в АН, DX или EDX соответственно. Результат всегда округляется в сторону нуля, знак остатка всегда совпадает со знаком делимого, абсолютное значение ос­татка всегда меньше абсолютного значения делителя. Значения флагов CF, OF, SF, ZF, AF и PF после этой команды не определены, а перепол­нение или деление на ноль вызывает исключение #DE (ошибка при де­лении) в защищенном режиме и прерывание 0 – в реальном.

k) **INC** приемник; 8086, инкремент.

inc d; (d) + 1 🡪 d

Увеличивает приемник на 1. Единственное отличие этой команды от ADD состоит в том, что флаг CF не затрагивается. Остальные арифметические флаги (OF, SF, ZF, AF, PF) устанавливаются в соответствии с результатом сложения.

l) **DEC** приемник; 8086, декремент.

dec d; (d) – 1 🡪 d

Уменьшает приемник на 1. Единственное отличие этой команды от SUB состоит в том, что флаг CF не затрагивается. Остальные арифметические флаги (OF, SF, ZF, AF, PF) устанавливаются в соответствии с результатом вычитания.

m) **NEG** приемник; 8086, изменение знака.

neg s; -(s) 🡪 s

Выполняет над числом, содержащимся в приемнике, операцию дополнения до двух. Эта операция эквивалентна обращению знака операнда, если рассматривать его как число со знаком. Если приемник равен нулю, флаг CF устанавливается в 0, иначе – в 1. Остальные флаги (OF, SF, ZF, AF, PF) устанавливаются в соответствии с результатом операции.

***П1.3. Логические команды***

a) **AND** приемник, источник; 8086, логическое И.

and d, s; s & d 🡪 d

Команда выполняет побитовое «логическое И» над приемником и источником (источник и приемник не могут быть переменными одновременно) и помещает результат в приемник. Любой бит результата равен 1, толь­ко если соответствующие биты обоих операндов были равны 1, и равен 0 в остальных случаях.

b) **OR** приемник, источник; 8086, логическое ИЛИ.

or d, s; s ! d 🡪 d

Выполняет побитовое «логическое ИЛИ» над приемником и источником (источ­ник и приемник не могут быть переменными одновременно) и помеща­ет результат в приемник. Любой бит результата равен 0, только если со­ответствующие биты обоих операндов были равны 0, и равен 1 в остальных случаях.

c) **XOR** приемник, источник; 8086, логическое исключающее ИЛИ.

xor d, s; s xor d 🡪 d

Выполняет побитовое «логическое исключающее ИЛИ» над прием­ником и источником (источник и приемник не могут быть переменными одновре­менно) и помещает результат в приемник. Любой бит результата равен 1, если соответствующие биты операндов различны, и нулю, если оди­наковы.

d) **NOT** приемник; 8086, инверсия

not s; ¬(s) 🡪 s

Каждый бит приемника, равный нулю, ус­танавливается в 1, и каждый бит, равный 1, сбрасывается в 0. Флаги не затрагиваются.

e) **TEST** приемник, источник; 8086, логическое сравнение.

and d, s; s & d

Вычисляет результат действия побитового «логического И» над при­емником и источником (источник и приемник не могут быть переменными одновре­менно) и устанавливает флаги SF, ZF и PF в соответствии с полученным результатом, не сохраняя результат (флаги OF и CF обнуляются, значе­ние AF не определено). TEST, так же как и СМР, используется в основ­ном в сочетании с командами условного перехода *(Jcc)*.

***П1.4. Сдвиговые команды***

a) **SAR** приемник, счетчик; 8086, арифметический сдвиг вправо.

**SAL** приемник, счетчик; 8086, арифметический сдвиг влево.

**SHR** приемник, счетчик; 8086, логический сдвиг вправо.

**SHL** приемник, счетчик; 8086, логический сдвиг влево.

Эти четыре команды выполняют двоичный сдвиг приемника вправо или влево на значение счетчика (число или регистр CL, из которого учитываются только младшие пять бит, которые могут принимать значения от 0 до 31). Операция сдвига на 1 эквивалентна ум­ножению (сдвиг влево) или делению (сдвиг вправо) на 2. Так, число 0010b (2) после сдвига на 1 влево превращается в 0100b (4). Команды SAL и SHL выполняют одну и ту же операцию (на самом деле это одна и та же команда) – на каждый шаг сдвига старший бит заносится в CF, **все** биты сдвигаются влево на одну позицию, и младший бит обнуляет­ся. Команда SHR выполняет прямо противоположную операцию: млад­ший бит заносится в CF, все биты сдвигаются на 1 вправо, старший бит обнуляется. Эта команда эквивалентна беззнаковому целочисленному делению на 2. Команда SAR действует по аналогии с SHR, только стар­ший бит не обнуляется, а сохраняет предыдущее значение, так что, например, число 11111100b (-4) перейдет в 11111110b (-2). SAR, таким образом, эквивалентна знаковому делению на 2, но, в отличие от IDIV, округление происходит не в сторону нуля, а в сторону отрицательной бесконечности. Так, если разделить -9 на 4 с помощью IDIV, результат будет -2 (и остаток -1), а если выполнить арифметический сдвиг впра­во числа -9 на 2, результат будет -3. Сдвиги больше, чем на 1, эквива­лентны соответствующим сдвигам на 1, выполненным последователь­но. Схема всех сдвиговых операций приведена на рис. 2.1.

Сдвиги на 1 изменяют значение флага OF: SAL/SHL, устанавливают его в 1, если после сдвига старший бит изменился (т. е. старшие два бита исходного числа не были одинаковыми), и в 0, если старший бит остался тем же. SAR устанавливает OF в 0, и SHR устанавливает OF в значение старшего бита исходного числа. Для сдвигов на несколько бит значение OF не определено. Флаги SF, ZF, PF устанавливаются все­ми сдвигами в соответствии с результатом, значение AF не определено (кроме случая, если счетчик сдвига равен нулю, в котором ничего не происходит и флаги не изменяются).

В процессорах 8086 непосредственно можно задавать в каче­стве второго операнда только число 1 и при использовании CL учиты­вать все биты, а не только младшие 5.

b) **ROR** приемник, счетчик; 8086, циклический сдвиг вправо.

**ROL** приемник, счетчик; 8086, циклический сдвиг влево.

**RCR** приемник, счетчик; 8086, циклический сдвиг вправо через флаг переноса.

**RCL** приемник, счетчик; 8086, циклический сдвиг влево через флаг переноса.

Эти команды осуществляют циклический сдвиг приемника на число бит, указанное в счетчике (число или регистр CL, из которого учитываются только младшие пять бит, принимающие значения от 0 до 31). При выполнении циклического сдвига на 1 коман­ды ROR (ROL) сдвигают каждый бит приемника вправо (влево) на одну позицию, за исключением самого младшего (старшего), который записывается в позицию самого старшего (младшего) бита. Команды RCR, RCL выполняют аналогичное действие, но включают флаг CF в цикл, как если бы он был дополнительным битом в приемнике.

***П1.5. Команды передачи управления***

В состав группы команд передачи управления можно включить:

- команды безусловного и условных переходов;

- команды зацикливания;

- команды вызова подпрограмм и команды возврата из подпрограммы;

- команды вызова прерываний и возврата из прерывания.

**a) Команда безусловного перехода**

**JMP** операнд; 8086, безусловный переход

JMP передает управление в другую точку программы. Операндом может быть непосред­ственный адрес для перехода, а также регистр или переменная, содержащая адрес.

В зависимости от типа перехода различают:

* переход типа *short* (короткий переход) – если адрес перехода нахо­дится в пределах от -127 до +128 байт от команды JMP;
* переход типа *near* (ближний переход) – если адрес перехода находит­ся в том же сегменте памяти, что и команда JMP;
* переход типа *far* (дальний переход) – если адрес перехода находится в другом сегменте.

**b) Команды условного перехода**

**Jcc** операнд; 8086, переход по адресу, задаваемому операндом, если условие перехода сс истинно.

Команда Описание

JA Переход, если выше (CF = 0 и ZF = 0)

JAE Переход, если выше или равно (CF = 0)

JB Переход, если ниже (CF = 1)

JBE Переход, если ниже или равно (CF = 1 или ZF = 1)

JC Переход, если перенос (CF = 1)

JCXZ Переход, если регистр CX равен 0

JE Переход, если равно (ZF = 1)

JZ Переход, если 0 (ZF = 1)

JG Переход, если больше (ZF = 0 и SF = OF)

JGE Переход, если больше или равно (SF = OF)

JL Переход, если меньше (SF <> OF)

JLE Переход, если меньше или равно (ZF=1 или SF <> OF)

JNA Переход, если не выше (CF = 1 и ZF = 1)

JNAE Переход, если не выше или равно (CF = 1)

JNB Переход, если не ниже (CF = 0)

JNBE Переход, если не ниже или равно (CF=0 и ZF=0)

JNC Переход, если нет переноса (CF = 0)

JNE Переход, если не равно (ZF = 0)

JNG Переход, если не больше (ZF = 1 или SF <> OF)

JNGE Переход, если не больше или равно (SF <> OF)

JNL Переход, если не меньше (SF = OF)

JNLE Переход, если не меньше или равно (ZF=0 и SF=OF)

JNO Переход, если нет переполнения (OF=0)

JNP Переход, если нет контроля четности (PF = 0)

JNS Переход, если нет знака (SF = 0)

JNZ Переход, если нет нуля (ZF = 0)

JO Переход, если переполнение (OF = 1)

JP Переход, если контроль четности (PF = 1)

JPE Переход, если контроль на проверку четности (PF = 1)

JPO Переход, если контроль на проверку нечетности (PF = 0)

JS Переход, если знак отрицательный (SF = 1)

JZ Переход, если ноль (ZF = 1)

Команды условного перехода (за исключением команды JCXZ) проверяют флаги, которые были установлены предыдущей командой. Условия для каждой мнемонической формы команды даны в круглых скобках выше после каждого описания. Термины «меньше» и «больше» используются при сравнении целых со знаком; «выше» и «ниже» – для целых без знака. Если условие выполняется, то происходит переход по адресу, задаваемому операндом, иначе – выполняется команда, следующая за командой условного перехода. Использование данной команды наиболее эффективно, когда цель условного перехода находится в текущем кодовом сегменте и в пределах от -128 до +127 байт относительно первого байта следующей команды. Если цель команды условного перехода находится за пределами -128 до +127 байт относительно первого байта следующей команды, используйте команду с противоположным условием перехода в сочетании с командой безусловного перехода.

Команда JCXZ отличается от других команд условного перехода тем, что она проверяет не флаги, а содержимое регистра CX на равенство 0. Команда JCXZ полезна в начале условного цикла, который заканчивается командой условного перехода к началу цикла (например, LOOPNE метка цели). Команда JCXZ предотвращает нахождение в цикл при регистре CX, равном нулю, что может привести к выполнению цикла 64К раз вместо нуля раз.

**с) Команды зацикливания**

**LOOP** метка; 8086, повторить цикл (декремент CX и возврат на метку), если (CX) ≠ 0.

Команда LOOP уменьшает содержимое регистра СХ на 1 и выполняет переход типа short (на расстояние [-128, +127] байт от команды Loop). Эта команда используется для организации циклов с регистром СХ в качестве счетчика и эквивалентна паре команд: { dec CX ; jnz метка }, но не меняет значения флагов. Для принудительного выхода из цикла до исчерпания заданного числа повторений используются команды **LOOPcc,** где сс – дополнительное условие перехода, комбинируемое по «И» с условием (CX) ≠ 0.

**LOOPE** метка; 8086, повторить цикл, пока равно ( если (CX) ≠ 0 & ZF=1)

**LOOPZ** метка; 8086, повторить цикл, пока ноль ( если (CX) ≠ 0 & ZF=1)

**LOOPNE** метка; 8086, повторить цикл, пока не равно ( если (CX) ≠ 0 & ZF=0)

**LOOPNZ** метка; 8086, повторить цикл, пока не ноль ( если (CX) ≠ 0 & ZF=0)

**d) Команды вызова подпрограмм и команды возврата из подпрограммы**

**CALL** операнд; 8086, переход на подпрограмму с возвратом.

Сохраняет адрес следующей команды в стеке и передает управление по адресу, указанному в операнде. Операндом может быть непосредственное значение (метка в ассемблере), регистр или переменная, содержащие адрес перехода. Если подпрограмма размещена в том же сегменте, что и команда CALL, то выполняется ближний вызов подпрограммы: прямой (на метку) или косвенный (через регистр или переменную). При этом в стеке сохраняется текущее значение IP, а в IP загружается смещение адреса подпрограммы. Если подпрограмма размещена в другом сегменте, нежели команда CALL, то в стеке сохраняются текущие значения CS и IP, а в CS и IP загружаются сегмент и смещение адреса подпрограммы.

**RET (RETN, RETF)** [число]; 8086, возврат из подпрограммы

Cчитывает из стека слово (RETN) или два слова (RETF) в зависимости от описания подпрограммы (ближнего или дальнего вызова) и загружает их соответственно в IP или в IP и CS. Операнд-число для RET не обязателен, но, если он присутствует, после считывания адреса возврата из стека будет удалено заданное этим числом количество байт.

**e) Команды вызова прерываний и возврата из прерывания**

**INT** операнд; 8086, вызов процедуры обработчика прерывания

**|INTO** ; 8086, прерывание 4 – если флаг переполнения равен 1

Команда INT n генерирует вызов «обработчика прерывания». Непосредственный операнд – целое число от 0 до 0FFh, задает номер индекса в таблице векторов прерываний (в старших моделях – дескрипторов прерываний) для вызываемой подпрограммы «обработчика прерывания». Таблица векторов прерываний представляет собой массив из указателей дальнего вызова длиной в четыре байта, начинающийся по адресу 0000h : 0000h. Первые 32 прерывания резервированы Intel для системных целей. Некоторые из этих прерываний используются для внутренних исключений.

Команда INT n помещает в стек содержимое регистра флагов, регистров CS и IP в указанной последовательности и затем выполняет переход к дальнему указателю, индексируемому номером прерывания.

Условная команда INTO идентична команде прерывания INT n, за исключением того, что номер прерывания неявно устанавливается равным 4, и прерывание происходит только при установленном флаге переполнения процессора.

**IRET** ; 8086, возврат из прерывания

|Команда IRET извлекает из стека указатель команд IP, регистр CS и флаговый регистр, после чего возобновляет выполнение прерванной программы.

***П1.6. Команды обработки строк***

Все команды работы со строками считают, что строка-источник находится по адресу DS:SI, а строка-приемник – по адресу ES:DI. Кроме того, все строковые команды за один раз выполняют операцию только с одним элементом строки (байтом, словом или двойным словом). Для выполнения операций над всеми элементами строки необходимо задать один из префиксов повторения операций: REP (Повторять), REPE (Повторять, пока равно), REPZ (Повторять, пока ноль), REPNE (Повторять, пока не равно), REPNZ (Повторять, пока не ноль). По аналогии с командами LOOP префиксы используют регистр СХ в качестве счетчика повторений, уменьшая его при каждом выполнении на 1, и комбинируют по «И» проверку условия (CX) = 0 с дополнительным условием, определяемым названием префикса: (ZF = 1) – для REPE, REPZ и (ZF = 0) – для REPNE, REPNZ. Обычно префикс REP используется с командами MOVS, LODS, STOS, INS и OUTS, а остальные префиксы – с командами CMPS и SCAS.

**MOVS/ MOVSB/MOVSW/MOVSD** приемник, источник; 8086, копирование строки

Команда копирует элемент строки из памяти по адресу DS:SI в память по адресу ES:DI.

Эта и все последующие строковые команды имеют разновидности вида **MOVSB, MOVSW** и **MOVSD** (только для i386 и выше) для задания операции над байтом, словом иди двойным словом. Если используется форма MOVS,то тип операнда определяет сам ассемблер.После того, как копирование выполнено, происходит автоматическое продвижение регистров SI и DI. Если флаг DF равен 0 (т.е. была использована команда CLD), происходит инкремент этих регистров; если же флаг DF равен 1 (была выполнена команда STD), то происходит декремент регистра. При копировании байтов выполняется инкремент или декремент на 1, при копировании слов на 2. Команде MOVS может предшествовать префикс REP для блочного сравнения с использованием CX байтов или слов.

**LODS/LODSB/LODSW/LODSD**  источник; 8086, загрузка строки

Команда LODS загружает регистр AL, AX или EAX байтом, словом или двойным словом памяти из адреса, на который указываеи регистр SI. После выполнения загрузки регистр SI автоматически продвигается на 1 при загрузке байта, на 2 при загрузке слова и на 4 при загрузке двойного слова. Команде LODS может предшествовать префикс REP; однако команда LODS чаще используется в конструкции цикла LOOP, поскольку далее обычно следует обработка данных, загруженных в регистры AL, AX или EAX.

**STOS/STOSB/STOSW/STOSD** приемник; 8086, сохранение строки

Команда STOS сохраняет содержимое регистра AL, AX или EAX в байт, слово или двойное слово памяти, заданное адресом, на который указывает регистр DI (EDI) относительно сегмента в регистре ES. Переопреде-ление сегмента не допускается. Адрес назначения определяется содержимым регистра DI, а не явно заданным операндом команды STOS. Этот операнд используется только для удостоверения адресуемости сегмента ES и определения типа данных. Перед выполнением команды STOS следует загрузить в регистр DI или EDI правильное значение индекса. После сохранения элемента строки регистр DI автоматически продвигается на 1 при сохранении байта, на 2 при сохранении слова и на 4 при сохранении двойного слова. Команде STOS может предшествовать префикс REP для заполнения блока из CX или ECX байтов, слов или двойных слов.

**INS/ INSB/INSW/INSD** приемник, источник; 8086, ввод строки

**OUTS/OUTSB/OUTSW/OUTSD** приемник, источник; 8086, вывод строки

**CMPS/CMPSB/CMPSW/CMPSD** приемник, источник; 8086, сравнение строковых данных

Команда CMPS сравнивает байт, слово или двойное слово, на которое указывает регистр SI, с байтом, словом или двойным словом, на которое указывает регистр DI. Сравнение выполняется посредством вычитания операнда, индексированного регистром DI, из операнда, индексированного регистром SI. Отметим, что направление вычитания для команды CMPS: [SI] – [DI] противоположно направлению, принятому по обычным соглашениям Intel, где левый операнд является назначением, а правый – источником. Результат вычитания не записывается; он отражается только изменением флагов. После того, как сравнение выполнено, происходит автоматическое продвижение в регистрах SI и DI. Команде CMPS может предшествовать префикс REPE или REPNE для блочного сравнения с использованием CX-байтов или слов.

**SCAS/SCASB/SCASW/SCASD** приемник; 8086, сравнение строковых данных

Команда SCAS вычитает байт или слово памяти, задаваемое регистром DI, из регистров AL, AX или EAX. Результат ***отбрасывается***; происходит только установка флагов. Операнд должен адресоваться относительно сегмента в регистре ES; переопределение сегмента не допускается. После того, как сравнение выполнено, регистр назначения автоматически обновляется. Команде SCAS может предшествовать префикс REPE или REPNE для поиска в блоке CX или ECX байтов или слов.

***П1.7. Команды работы с флагами***

**CLC** ; 8086, Очистить флаг переноса

Команда CLC очищает флаг CF (CF <-0). На другие флаги или регистры она не влияет.

**CLD** ; 8086, Очистить флаг направления

Команда CLD очищает флаг направления DF (DF <-0). На другие флаги или регистры она не влияет. После выполнения команды CLD строковые команды будут инкрементировать используемые ими индексные регистры (SI/DI).

**CLI** ; 8086, Очистить флаг прерывания

Команда CLI очищает флаг прерывания IF (IF <-0), если текущий уровень привилегированности как минимум равен IOPL. На другие флаги она не влияет. Внешние прерывания не распознаются в конце команды CLI и, начиная с этого момента, до установки флага IF.

**CMC** ; 8086, Дополнение флага переноса

Команда CMC изменяет на противоположное значение флага CF (CF <- NOT CF) . На другие флаги влияния не оказывает.

**STC** ; 8086, Установка флага переноса

Команда STC устанавливает флаг CF (CF <- 1)

**STD** ; 8086, Установка флага направления

Команда STD устанавливает флаг направления DF (DF <- 1), в результате чего все последующие строковые операции будут выполнять декремент индексных регистров (E)SI или (E)DI, с которыми они работают.

**STI**  ; 8086, Установка флага прерывания

Команда STI устанавливает флаг IF (IF <- 1). После выполнения следующей команды процессор может реагировать на внешние прерывания, если эта следующая команда оставляет флаг IF в состоянии, разрешающем прерывания. Если внешние прерывания запрещены, и после команды RET использована команда RET (например, в конце подпрограммы), то команда RET будет выполнена до того, как начнут распознаваться внешние прерывания. Также, если внешние прерывания запрещены и использована команда STI, а за ней команда CLI, то внешние прерывания не будут распознаваться, поскольку команда CLI очищает флаг IF во время своего выполнения.

**PUSHF** ; 8086, Помещение в стек флаговых регистров

Команда PUSHF декрементирует указатель стека на 2 и копирует регистр FLAGS в новую вершину стека.

**POPF** ; 8086, Извлечение из стека регистров FLAGS

Команда POPA снимает с вершины стека слово и помещает его во флаговый регистр.

***П1.8. Команды математического сопроцессора FPU***

**a) команды пересылки данных**

**FLD** ; Загрузка действительного значения в стек

Работа команды: Декремент указателя TOP вершины стека FPU; ST(0) <- SRC;

Команда загружает действительное число в стек сопроцессора из памяти (или из другого регистра). Загружается регистр FPU, расположенный на вершине стека (определяется указателем TOP), обозначаемый здесь и далее ST(0) или просто ST. Если исходное число задано в регистре стека, то используется его номер, который был до декремента указателя вершины стека. В частности, FLD ST(0) дублирует вершину стека.

**FILD** ; Загрузка целочисленного значения в стек

Работа команды: Декремент указателя вершины стека FPU; ST(0) <- SRC;

Команда FILD преобразует исходный целочисленный операнд со знаком, хранящийся в памяти (или в другом регистре), в формат расширенного действительного и помещает его в стек.

**FLD1/FLDL2T/FLDL2E/FLDPI/FLDLG2/FLDLN2/FLDZ** ; Загрузка константы в стек

Работа команды: Декремент указателя вершины стека FPU; ST(0) <- CONSTANT;

Описание:

FLD1 – Поместить +1.0 в стек

FLDL2T – Поместить log 210 в стек

FLDL2E – Поместить log 2 *e* в стек

FLDPI – Поместить число π в стек

FLDLG2 – Поместить lg 2 в стек

FLDLN2 – Поместить ln 2 в стек

FLDZ – Поместить +0.0 в стек

Каждая команда загрузки константы помещает в стек FPU одну из указанных выше общепринятых констант (в расширенном действительном формате).

**FLDCW** ; Загрузка управляющего слова

Команда FLDCW заменяет текущее значение управляющего слова FPU значением, находящимся в заданном слове памяти.

**FST/FSTP** ; Сохранить действительное число

Работа команды: DEST <- ST(0); IF команда = FSTP THEN извлечение из стека ST; FI;

Команда FST копирует текущее значение регистра ST в операнд назначения (другой регистр, либо память действительного формата одинарной или двойной точности); при этом аппаратный стек не меняется. FSTP (копирование с очисткой стека) выполняет сначала копирование, а затем извлечение ST из стека. Если источник является регистром, то используется номер регистра, который был до извлечения из стека.

**FXCH** ; Поменять местами содержимое регистров

Работа команды: врем <- ST; ST <- DEST; DEST <- врем;

Команда меняет местами содержимое регистра назначения и вершины стека. Если назначение явно не задано, то используется ST (1). Многие числовые константы работают только с вершиной стека; FXCH обеспечивает простое средство использования этих команд по отношению к нижним элементам стека. Например, следующая последовательность берет квадратный корень из содержимого третьего регистра сверху (предполагая, что ST непустой):

FXCH ST(3)

FSQRT

FXCH ST(3)

**b) команды арифметической обработки**

Выполняют четыре арифметических действия над всеми типами данных. Даже обработка целых чисел в сопроцессоре идет по принципу вещественных чисел.

**FADD/FADDP/FIADD** ; Сложение

Работа команды: DEST <- DEST + SRC; IF команда = FADDP THEN извлечение из стека ST FI;

Команды сложения складывают операнды источника и назначения и возвращают сумму в операнд назначения. Операнд в вершине стека может быть удвоен командой FADD ST, ST(0)

Здесь и далее команда с окончанием Р выталкивает ST(0) из стека: помечает ST(0) как пустой и увеличивает TOP на 1.

**FSUB/FSUBP/FISUB** ; Вычитание

Работа команды: DEST <- ST - Другой операнд; IF команда = FSUBP THEN извлечение из стека ST; FI;

Команды вычитания вычитают из вершины стека другой операнд и возвращают разность в регистр назначения.

**FSUBR/FSUBPR/FISUBR** ; Обратное вычитание

Работа команды: DEST <- Другой операнд - ST; IF команда = FSUBRP THEN извлечение из стека ST; FI;

Команды обратного вычитания вычитают вершину стека из другого операнда и возвращают разность в регистр назначения.

**FCOM/FCOMP/FCOMPP** ; Сравнение действительных чисел

Команды сравнивают вершину стека с источником, который может являться регистром или операндом памяти, являющимся действительным числом одинарной или двойной точности. Если операнд не задан, то ST сравнива-ется с ST(1). После выполнения команды коды условий отражают отношение между ST и исходным операндом.

**FICOM/FICOMP** ; Сравнение целочисленных значений

Команды сравнивают вершину стека с исходным операндом. После выполнения команды коды условий отражают соотношение между ST и исходным операндом.

**FCHS** ; Изменение знака

Команда меняет на противоположный знак ST (знаковый бит ST <- NOT (знаковый бит в ST) ). Эта операция заменяет положительное значение на отрицательное при той же абсолютной величине, и наоборот.

**FMUL/FMULP/FIMUL ;**  Умножение

Работа команды: DEST <- DEST x SCR; IF команда = FMULP THEN извлечение из стека ST FI;

Команды умножения умножают операнд назначения на исходный операнд (источник) и возвращают произведение в операнд назначения.

**FDIV/FDIVP/FIDIV** ; Деление

Команды деления делят вершину стека на прочие операнды и возвращают частное в операнд назначения.

При делении нормального числа на 0 происходит исключение деления на 0 и результат будет бесконечность соответствующего знака. При делении бесконечности на ноль (или любое число) результат – бесконечность, при делении нуля на бесконечность (или любое число) результат – ноль.

**FDIVR/FDIVPR/FIDIVR** ; Деление в обратном порядке

Команды делят другой операнд на вершину стека и возвращают частное в операнд назначения. В остальном эти команды аналогичны предыдущим.

**FTST** ; Тестирование

Работа команды:

CASE (отношение операндов) OF

Не сравнимы: C3, C2, C0 <- 111;

ST > SRC: C3, C2, C0 <- 000;

ST < SRC: C3, C2, C0 <- 001;

ST = SRC: C3, C2, C0 <- 100;

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

Флаги FPU | EFlags

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

C0 CF

C1 отсутствует

C2 PF

C3 ZF

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

Команда тестирования сравнивает вершину стека с 0.0. После выполнения команды коды условий (флаги) отражают результат сравнения.

**FRNDINT** ; Округление к целому

Работа команды: ST <- округленный ST;

Команда округления к целому округляет значение в ST к целому в соответствии с полем RC управляющего слова FPU.

**c) команды вычисления элементарных функций**

**FABS** ; Абсолютное значение

Команда абсолютного значения FABS очищает знаковый бит ST (ST <- 0). Операция оставляет положительное значение без изменений либо заменяет отрицательное значение положительным, равным по абсолютной величине.

**FSCALE** ; Умножение на масштабный коэффициент

Работа команды: ST <- ST x 2\*\*ST(1);

Команда масштабирования округляет значение в ST(1) до целого в сторону нуля, умножает ST(0) на 2 в степени ST(1) и записывает результат в ST(0). Таким образом, FSCALE обеспечивает быстрое умножение или деление на целочисленные степени 2.

Примечание. Команда FSCALE может использоваться как команда, обратная по отношению к FXTRACT. Поскольку FSCALE не извлекает из стека экспонентную часть, за FSCALE должна следовать команда FSTP ST(1), чтобы полностью отменить действие предыдущей команды FXTRACT.

**FXTRACT** ; Выделение экспоненты и мантиссы

Работа команды: врем <- мантисса ST; ST <- экспонента ST;

Декремент указателя вершины стека FPU; ST <- врем;

Команда разделяет значение в ST(0) на экспоненту и мантиссу, экспонента заменяет исходный операнд в ST(0), и затем TOP уменьшается на 1, и в стек помещается мантисса. После выполнения команды ST(0) (новая вершина стека) содержит значение исходной мантиссы, выраженное действительным числом со знаком исходного операнда, а ST(1) содержит значение истинной (несмещенной) экспоненты исходного операнда, выраженное действительным числом.

**FSQRT** ; Квадратный корень

Команда FSQRT заменяет значение в ST(0) на его квадратный корень.

**FSIN** ; Синус

Работа команды:

IF операнд в допустимом диапазоне

THEN

C2 <- 0;

ST <- sin(ST);

ELSE

C2 <- 1;

FI;

Команда FSIN заменяет содержимое ST на sin (ST). Значение ST, выраженное в радианах, должно лежать в диапазоне | O | < 2\*\*63.

**FCOS** ; Косинус

Команда FCOS заменяет содержимое ST на cos (ST). Значение ST, выраженное в радианах, должно лежать в диапазоне | O | < 2\*\*63. Если операнд находится вне допустимого диапазона, то флаг C2 устанавливается, а ST остается неизмененным. Программист сам ответственен за то, чтобы уменьшить операнд до абсолютного значения, меньшего, чем 2\*\*63, вычитая соответствующее число, кратное 2π .

**FPATAN ;**  Частичный арктангенс

Работа команды: ST(1) <- arctan(ST(1)/ST); извлечение из стека ST;

Частичный арктангенс вычисляет арктангенс от ST(1)/ST(0) и возвращает вычисленное значение в радианах в ST(1). Затем выполняется извлечение из стека ST(0). Результат имеет тот же знак, что и операнд из ST(1), и по величине меньше числа π.

**FPTAN** ; Частичный тангенс

Частичный тангенс заменяет содержимое ST на tg (ST) и затем помещает в стек FPU величину 1.0. Значение ST в радианах должно лежать в диапазоне | O | < 2\*\*63.

**F2XM1** ; Вычисление 2Х - 1

Работа команды: ST <- (2ST – 1);

Команда F2XM1 заменяет содержимое ST на (2ST – 1). ST должен находиться в диапазоне -1 < ST < 1. Если операнд находится вне допустимого диапазона, то результат F2XM1 неопределен.

Значения, не равные 2, могут возводиться в степень по формуле XY = 2(Y \* log2X) .

Команды FLDL2T и FLDL2E загружают константы log2 10 и log2 e, соответственно. Команда FYL2X может быть использована для вычисления y \* log2 x для произвольного положительного x.

**FYL2X** ; Вычисление y \* log2 x

Работа команды: ST(1) <- ST(1) \* log2 ST; извлечение из стека ST;

Команда FYL2X вычисляет логарифм ST по основанию 2, умножает логарифм на ST(1) и возвращает полученное значение в ST(1). Операнд в ST не может быть отрицательным. Если операнд в ST отрицателен, то генерируется исключение неверной операции.

**FYL2XP1** ; Вычисление y \* log2 (x +1)

Работа команды: ST(1 ) <- ST(1) \* log2 (ST +1.0); извлечение из стека ST;

Команда FYL2XР1 вычисляет логарифм (ST+1.0) по основанию 2, умножает логарифм на ST(1) и возвращает полученное значение в ST (1). Операнд в ST должен лежать в диапазоне -(1-(кв.корень из 2/2)) <= ST <= кв.корени из 2 -1

**d) Команды управления**

**FINIT/FNINIT** ; Инициализация модуля FPU операций с плавающей точкой

Команды инициализации устанавливают FPU в иcходное состояние, независимо от действий, выполнявшихся им ранее. Управляющее слово FPU устанавливается в значение 037FY (округление до ближайшего, все исключения маскируются, 64-битовая точность представления). Слово состояния очищается (флаги исключений не установлены, регистр стека R0 = вершине стека). Стековые регистры имеют теги «пусто». Указатели ошибки (как команд, так и данных) очищены.

FINIT проверяет наличие немаскируемых исключений ошибок операций с плавающей точкой и обрабатывает их, прежде чем выполнить инциализацию; FNINIT же этого не делает.

**FWAIT** ; Ожидание

Команда FWAIT заставляет процессор проверить наличие необработанных немаскируемых исключений FPU и обработать их, прежде чем перейти к дальнейшему выполнению программы. Эту команду следует указывать в критических ситуациях после команд FPU, чтобы убедиться, что возможные исключения будут обработаны.

**FNOP** ; Отсутствие операции

Эта команда заменяет место и время, но не выполняет никакого действия. Может использоваться для создания задержек в процессе выполнении программы.

***П1.9. Команды MMX (MultiMedia eXtension) расширения***

Эта группа команд появилась, начиная с процессора i586 (Pentium P54C ), ориентирована на обработку мультимедийных приложений и предназначена для повышения эффективности выполнения программ, работающих с большими потоками данных (большими массивами целых чисел) по несложным алгоритмам (обработка графики и видеоизображений, синтез и обработка звука).

MMX расширение включает в себя дополнительные регистры, типы данных и команды, ориентированные на одновременную обработку нескольких целых чисел.

* 1. **Регистры ММХ**

Расширение ММХ использует восемь 64-битных регистра MM0 – MM7, физически размещающихся в поле мантиссы восьми регистров FPU R0 – R7. При записи числа в ММ i поле экспоненты Ri [64-79] заполняется единицами. Кроме того, поле TOP регистра SR FPU и весь регистр тегов TW обнуляются. Поэтому нельзя одновременно пользоваться командами FPU и командами ММХ. При необходимости этого следует пользоваться командами FSAVE / FRSTOR при переходе от команды FPU к ММХ и обратно для сохранения и восстановления регистров FPU или ММХ соответственно.

* 1. **Типы данных**

Расширение ММХ использует 4 новых типа данных:

1. Учетверенное слово (64-битное число).
2. Упакованные двойные слова (два 32-битных двойных слова, упакованных в 64-битное данное).
3. Упакованные слова (четыре 16-битных слова, упакованных в 64-битное данное).
4. Упакованные байты (восемь байт, упакованных в 64-битное данное).

Отличительные особенности обработки данных:

1. Перемещение данных в память или в регистры осуществляется в упакованном виде, а логическая или арифметическая обработка выполняются над каждым элементом (полем) отдельно.

2. Арифметические операции в ММХ используют специальный способ обработки переполнения, который называется «насыщение». Если результат операции больше (меньше) максимального (минимального) значения соответствующего типа данных, то его полагают равным этому максимальному (минимальному) значению. Так, при операциях с цветом насыщение позволяет сохранять чисто белый цвет при переполнении и чисто черный при антипереполнении, а обычная арифметика привела бы к инверсии цвета.

* 1. **Команды ММХ**

(Кроме пересылок все остальные команды начинаются с буквы P.)

1. *Пересылка*

**MOVD** d, s – пересылка двойных слов.

Если приемник – регистр ММХ, двойное слово записывается в его младшую половину (биты 0 – 31), если источник – регистр ММХ, в приемник записывается младшее двойное слово этого регистра.

**MOVQ** d, s – пересылка учетверенных слов.

1. *Преобразование типов (упаковка со знаковым насыщением).*

**PACKSSWB** d, s – упаковывает и насыщает слова со знаком в байты.

Четыре слова, находящиеся в приемнике (регистре ММХ), копируются в четыре младших байта приемника, а четыре слова источника (регистр ММХ или переменная) копируются в старшие четыре байта приемника. Если значение какого-либо слова больше 127 или меньше -128, в байт помещается число +127 или –128 соответственно.

**PACKSSDW** d, s – упаковывает и насыщает двойные слова со знаком в слова.

Аналогично, два двойных слова из приемника копируются в два младших слова приемника, а два двойных слова из источника копируются в старшие два слова приемника.

1. *Распаковка и объединение старших элементов.*

**PUNPCKHBW**  d, s – распаковка байтов и объединение в слова,

**PUNPCKHWD**  d, s – распаковка слов и объединение в двойные слова,

**PUNPCKHDQ**  d, s – распаковка двойных слов и объединение в учетверенное слово.

Команды распаковывают старшие элементы источника (регистр ММХ или переменная) и приемника (регистр ММХ) и записывают их в приемник через один.

Пример:

7 6 5 4 3 2 1 0

7 6 5 4 3 2 1 0

1. *Арифметические операции.*

**PADDB/W/D** d, s – сложение отдельных байтов/слов/двойных слов без учета переноса

**PADDSB/W** d, s – сложение с насыщением

**PADDUSB/W** d, s – сложение без учета знака с насыщением

Аналогичные команды имеются для операций вычитания (**PSUBB/W/D, PSUBSB/W, PSUBUSB/W**).

**PMULHW** d, s – умножение с фиксацией старших слов результатов

**PMULLW** d, s – умножение с фиксацией младших слов результатов

Эти команды умножают каждое из 4 слов источника на соответствующее слово приемника. Затем старшее (младшее) слово каждого из результатов записывается в соответствующую позицию приемника.

**PMADDWD** d, s – умножение со сложением.

Каждое из четырех слов источника умножается на соответствующее слово приемника. Произведения двух старших пар слов складываются и их сумма записывается в старшее двойное слово приемника, а сумма произведений двух младших пар слов записывается в младшее двойное слово приемника (команда применяется при реализации алгоритмов фильтрации изображений и т.п.).

1. *Сравнения.*

**PCMPEQB/PCMPEQW/PCMPEQD**  d, s – сравнение на равенство

Команды сравнивают отдельные байты/слова/двойные слова источника и приемника и в случае их равенства соответствующий элемент приемника заполняется единицами, а иначе – нулями.

**PCMPGTB/PCMPGTW/PCMPGTD** d, s – сравнение на больше .

Команды аналогичны предыдущим, но заполнение приемника единицами происходит, если элемент приемника больше элемента источника.

1. *Логические команды.*

**PAND** d, s – логическое И

**PANDN** d, s – логическое НЕ-И (штрих Шеффера)

**POR** d, s – логическое ИЛИ

**PXOR** d, s – логическое исключающее ИЛИ

Команды выполняют побитовые логические операции над источником и приемником и сохраняют результирующие биты в приемнике.

***П1.10. Расширение AMD 3DNow!***

В процессорах AMD, начиная с AMD 3D, появилось расширение ММХ-команд для обработки как целых, так и пары упакованных 32-битных вещественных чисел:

- дополнительный тип данных: – упакованные 32-битные вещественные числа;

- дополнительный набор команд над этими данными (начинаются с PF: P – это MMX, а F –- float);

- дополнительный набор команд над обычными ММХ-типами данных (упакованными целыми числами).

Дополнительный набор команд расширения AMD 3DNow!включает:

- команды преобразования упакованных целых чисел в упакованные вещественные и обратно;.

- команды сложения, вычитания, сравнения и умножения упакованных вещественных чисел;

- команды вычисления среднего арифметического для упакованных 8-битных целых чисел без знака;

- команды деления и вычисления квадратного корня по итерационным формулам;

- некоторые другие команды.

***П1.11****.****Команды потокового расширения SSE, SSE2***

Команды потокового расширения SSE (Streaming SIMD Extension) появились в процессорах фирмы Intel, начиная с Pentium III, для дополнения набора групповых операций над упакованными целыми числами и выполнения групповых операций над упакованными 32-битными вещественными числами:

- обработка групп целых чисел, упакованных в 64- и 128-битные слова;

- обработка одной пары вещественных чисел одинарной (32 бит) или двойной (64 бита) точности;

- обработка 4 пар вещественных чисел одинарной или 2 пар – двойной точности.

Команды реализуются на дополнительном блоке XMM из восьми 128-битных регистров, названных ХММ0-ХММ7. При выполнении SSE – команд традиционное оборудование FPU не используется, что позволяет эффективно смешивать их с командами FPU. Дальнейшее развитие технологии SSE в процессорах Pentium 4 вылилось в набор команд SSE2, включающих 271 команду, для выполнения различных арифметических и логических операций с обработкой за 1 такт до 4 32-битных чисел с плавающей запятой, упакованных в 128-битное слово.

**П2.** **Краткое введение в программирование на языке Ассемблера**

Ассемблер – машинно-ориентированный язык, предназначенный для написания программ, наиболее эффективных по времени и потреблению ресурсов, или обеспечивающих расширенные функциональные возможности по использованию ресурсов, недоступные из языков высокого уровня. Обычно он используется для написания относительно коротких программ или фрагментов кода, включаемых в программы на языках высокого уровня.

Особенностями ассемблера по сравнению с языком машинных команд являются:

* символическое наименование операций и операндов;
* отсутствие привязки к конкретным адресам памяти;
* возможность специализации программ с помощью макросредств.

Процесс подготовки, трансляции и выполнения ассемблерной программы можно пояснить с помощью схемы, приведенной на рис. П2.1.

Текстовый редактор

Исходный файл

Prog.asm

Транслятор

MASM (TASM)

Файл листинга Объектный файл Таблица символов

Prog.lst Prog.obj Sym.tab

Библиотека стандартных Компоновщик Файлы \*.obj

процедур и функций Link (Tlink) других модулей

Lib.obj

Карта памяти Исполняемый файл Файл перекрестных

Prog.map Prog.exe (.com) cсылок Prog.crf

Отладчик Загрузчик

AFD, TD

Выполнение программы Выполнение программы

под управлением отладчика в автоматическом режиме

Сообщения отладчика Корректные результаты Run Time Errors

Сообщение отладчика Корректные результаты Run Time Errors

Рис. П2.1

Дальнейшее изложение ориентировано на использование ассемблеров Intel 80X86 – MASM и TASM.

***П2.1 Формат оператора ассемблера***

В ассемблере различают два вида форматов:

1. формат исполняемого оператора имеет вид

*[метка:] операция операнд(ы)] [ ; комментарий]*

1. формат директивы имеет вид

*[имя] директива [аргумент(ы)] [ ; комментарий]*

Указанные поля форматов имеют следующий смысл:

- метка/имя символически задает адрес данной команды в исполняемом файле /адрес директивы в исходном тексте;

- операция символически задает дейстие, выполняемое над операндами при выполнении программы;

- директива символически задает действие, выполняемое над аргументами при трансляции программы и генерации объектного файла.

- операнды – имена, числа, символы, участвующие в операции (может быть 0/1/2 операнда);

- аргументы – имена, числа, символы, используемые в директиве (число аргументов не ограничено);

- комментарий – пояснение к тексту программы, при трансляции не рассматривается.

Директива (иногда называется псевдокоманда) ассемблера выполняется на этапе трансляции исходного текста программы в объектный файл, исполняемых машинных команд не порождает.

Исполняемый оператор в процессе трансляции исходного текста порождает машинные команды, которые выполняются на этапе выполнения программы.

***П2.2. Директивы ассемблера***

**1. Директивы задания данных**

1.1. Директивы определения имен

а) *идентификатор EQU выражение*

Позволяет символически именовать константы в программе.

Например

N EQU 100h

TABLE EQU DS:[BP][SI]

MINS\_DAY EQU 60\*24

b) *идентификатор = выражение*

Позволяет символически именовать переопределяемые в программе константы. Используется только для числовых выражений.

1.2. Директивы выделения памяти

Имеет формат:

*идентификатор D\* список значений*

Здесь D\* - одна из приведенных ниже директив:

DB – выделить байты;

DW – выделить слова;

DD – выделить двойные слова;

DF – выделить блоки по шесть байт;

DQ – выделить учетверенные слова;

DT – выделить блоки по десять байт.

Данная директива позволяет зарезервировать в памяти блоки заданного размера и присвоить им, если требуется, определенные значения, заданные в списке. Например:

text\_string db ‘Hello world’ ; выделяет 11 байт и заполняет их кодами символов

b\_max db 255 ; выделяет 1 байт и записывает в него число 255

b\_min db -128 ; выделяет 1 байт и записывает в него число –128

rez\_w dw ? ; выделяет 1 слово и не заполняет его (обычно ; используется в программе для записи результата)

rez\_tab dw 20dup(?) ; выделяет 20 слов, но не заполняет их данными

b\_tab db 4dup(?),8,5,4dup(1); выделяет 10 байт: 4 не заполняются, затем

; записываются числа 8, 5 и четыре раза по 1

fl\_num dd 5.03E-2 ; выделяется двойное слово и в него записывается

; число с плавающей запятой 5.0Е-2.

**2. Директивы сегментации программы**

Два способа задания сегментов в программе.

Полное описание сегментов

имя\_сегмента SEGMENT атрибуты

тело сегмента

имя\_сегмента ENDS

Пример

dat\_s1 segment byte public ‘data’

a db ?

dat\_s1 ends

Атрибуты:

* ReadOnly – сегмент доступен только для чтения; при попытки записи в этот сегмент MASM выдаст сообщение об ошибке.
* Атрибут выравнивания – указывает ассемблеру и компоновщику, с какого адреса может начинаться сегмент.

BYTE – с любого адреса.

WORD – с четного адреса.

DWORD – с адреса, кратного 4.

PARA – с адреса, кратного 16 (установлен по умолчанию).

PAGE – с адреса, кратного 256.

* Атрибут группирования, комбинирования.

PUBLIC- конкатенация (присоединение частей сегментов друг к другу).

COMMON – размещение сегментов данного класса с одного адреса (для сегментов кода и оверлейных программ).

PRIVATE – сегмент с таким атрибутом не объединяется с другими сегментами (значение по умолчанию).

* Атрибут типа данных.

USE16 – сегмент работает с 16-битными данными.

USE32 – сегмент работает с 32-битными данными.

* Атрибут класса – это любая метка, взятая в одинарные кавычки. Этот атрибут влияет на расположение сегментов в скомпонованной программе.

Связь сегментов с соответствующими сегментным регистром.

ASSUME {регистр\_сегментный: имя\_сегмента,…}

Обычно эта директива идет вслед за сегментом кода.

Пример

assume cs: code\_s, ds: d\_seg,

ss: stack, es: nothing

NOTHING – не устанавливать связь или отменить ее, если она была установлена.

Загрузка начальных адресов сегментов в соответствующие регистры.

mov ax, seg d\_seg ; seg- необязательный оператор

mov ds, ax

**3. Директивы группирования.**

GROUP имя\_сегмента1, имя\_сегмента2,…

Все перечисленные сегменты относятся к одной группе и могут адресоваться относительно одного регистра (обычно в одну группу объединяют сегменты одного назначения, например, data и stack).

Сокращенное описание сегментов.

При таком описании требуется обязательное задание модели памяти, в условиях которой используется данная программа.

.MODEL тип\_модели\_памяти

Эта директива накладывает ограничения на комбинирование сегментов (таблица).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Модель | Тип доступа к коду | Тип доступа к данным | Сегментные регистры | Примечания |
| TINY | Near | Near | (cs)=DGroup  (ds)=(ss)=DGroup | .com |
| SMALL | Near | Near | (cs)=\_Text  (ds)=(ss)=DGroup | .exe |
| MEDIUM | Far | Near | (cs)=<имя\_сегмента>\_Text  (ds)=(ss)=DGroup |  |
| LARGE | Far | Far | (cs)=<имя\_сегмента>\_Text  (ds)=(ss)=DGroup |  |
| HUGE | Far | Far |  |  |

**.CODE** – директива описания сегмента кода; эта запись аналогична

\_TEXT SEGMENT Word Public ‘CODE’

или

<имя\_сегмента> \_TEXT Word Public ‘CODE’

для модели памяти выше MEDIUM

.DATA

\_DATA SEGMENT Word Public ‘DATA’

.STACK

STACK SEGMENT Para Public ‘STACK’

.CONST

CONST SEGMENT Word Public ‘CONST’

.DATA?

\_BSS SEGMENT Word Public ‘BBS’

Отличие от полного описания сегментов заключается в отсутствии директивы ENDS. Таким образом, в результате создаются предопределенные переменные, которые содержат начальные адреса сегментов: @Code, @Data, @Stack, @Const, @BBS. Следовательно, можно написать:

mov ax, @data

mov ds, ax

**4. Порядок размещения сегментов.**

Он важен для того, чтобы уметь определять длину программы, и для возможности работы в отладочном режиме. Обычно компоновщик размещает сегменты в порядке их появления в программе, заданном в главном модуле; для подтверждения такого размещения можно написать:

.SEQ

для размещения в алфавитном порядке:

.ALPHA

размещение сегментов в порядке, принятом в MS DOS:

.DOSSEG

Эти директивы должны располагаться в самом начале программы.

Порядок размещения сегментов, соответствующий .DOSSEG:

* ‘CODE’
* сегменты, не относящиеся к DGROUP (FAR DATE, FAR STACK)
* сегменты DGROUP (‘DATE’, ‘STACK’, ‘BSS’, ‘CONST’)

**5. Директивы ограничения используемых команд.**

По умолчанию используется набор команд процессора i8086 (при попытке исполнить какую-либо другую возникает прерывание). Директивы, определяющие набор допустимых команд:

.ix86 (где x=1,2..6) позволяет использовать команды соответствующих процессоров;

.MXX – возможность применения команд мультимедиа расширения;

.K3D – разрешены команды AMD 3D.

**6. Директива END.**

END [метка старта] – логический конец программы, далее транслятор текст не просматривает. Метка старта – адрес, с которого начинается выполнение программы.

**П2.3. Операции и выражения в ассемблере**

**1. Арифметические операции.**

+, -, \*, /, mod (эти операции выполняются на этапе трансляции)

pi\_int EQU 31416/1000 - целая часть π

pi\_rem EQU 31416 mod 1000 - дробная часть π

SHR\_N - сдвиговые операции (вправо и влево на N двоичных разрядов.

SHL\_R

maska EQU 110010b

maska2 EQU maska SHR\_2

**2. Логические операции.**

Эти операции подразделяются на операции отношения (EQ, NE, LT, LE, GT,GE) и на непосредственно логические (AND, OR, XOR, NOT).

Истина — 0FFFFh

Ложь — 0

mov ax, ((b LT 10) AND 5) OR ((b GE 10) AND 15)

при b=3 предыдущая команда означает: mov ax, 5.

**3. Операции со счетчиком размещения программы (СРП).**

LC – Location Counter

$ – текущее значение СРП, предопределенная переменная.

Message DB ‘Hello!!!’

Mes\_length=$-Message

ORG – директива принудительной установки СРП на константу.

ORG 100h – для \*.com

ORG $+99h – изменение СРП на 99 байт по отношению к текущему значению.

EVEN – задает четное значение СРП (выравнивает СРП на ближайшее четное, большее текущего).

**4. Оператор изменения типа.**

<тип> PTR переменная или метка.

B\_TABLE DB 40DUP

mov ax, B\_TABLE+10

mov ax, word ptd B\_TABLE+10

xword EQU 0FFFCh

xor ax, ax

add al, byte ptr xword ; (AL)=FCh

CALL FAR PTR My\_Sub

**5. Операции выделения сегментной части адреса и смещения.**

SEG – переменная (DS)

OFFSET – метка (СS)

SIZE имя переменной – определяет размер переменной в байтах.

**П2.4. Использование процедур в ассемблере**

Ассемблер относится как к процедурным языкам (Pascal, C, …), так и к непроцедурным. Считается удобным фрагменты текста на ассемблере оформлять в виде процедур, однако CALL far PTR [BX] обращается в произвольное место программы, имя процедуры не используется.

Явное описание процедур:

имя\_процедуры PROC [тип] [язык] [uses regs]

тело процедуры

ret; retf или retn

имя\_процедуры ENDP

тип: far, near (по умолчанию – near);

язык: (по умолчанию – ассемблер);

regs – сохраняются в стеке.

Схема вызова процедуры. При вызове типа NEAR обрабатывание CS не происходит (рис. П2.2)

My\_Proc ENDP

Ret

- - - - - - - - - - - - Стек

My\_Proc PROC IP

CS

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

Call My\_Proc CS IP

Рис. П.2.2

Обеспечивается вложенность процедур (ограничена стеком), могут организовываться рекурсивные вызовы. Этапы выполнения процедуры:

* подготовка параметров для работы с процедурой (типы параметров: значения, ссылки, возвращаемые значения, именования (при макровызовах));
* сохранение адреса возврата;
* передача управления на начало процедуры;
* выполнение тела процедуры, включая сохранение регистров, фиксацию результатов, фиксацию кода завершения, восстановление регистров;
* возвращение в основную программу в место после команды вызова, может быть с очисткой стека.

**Способы передачи параметров.**

1. Передача параметров через РОН.

Достоинства: быстрый и удобный;

Недостатки: ограничен в размерах (не более 5).

Применение: написание операционных систем (печать строки и др.).

2. Передача через общую область памяти. (общие переменные)

Общие переменные описываются следующим образом

В вызывающей процедуре

PUBLIC список имен (имена переменных или меток).

В вызываемой процедуре:

EXTRN {имя параметра, тип}

**Пример:**

;головная программа

DATA Segment

EXTERN A:WORD, R:WORD

DATA ENDS

CODE Segment PUBLIC

ASSUME DS:DATA,CS:CODE

PUBLIC PUB\_PROC FAR

start:

mov AX, DATA

mov DS, AX

CALL far ptr pub\_proc

mov cx, ds:A

sub cx, 2

add cx, ds:R

mov ds:A, cx

ret

CODE ENDP

END start

;Модуль, задающий общие переменные и процедуру.

DATA Segment

PUBLIC A, R

A DW 10

B DW 5

C DW 3

B DW 1

DATA ENDS

CODE Segment PUBLIC

ASSUME DS:DATA,CS:CODE

Public Pub\_Proc

Pub\_Proc Proc Far

mov AX, B

add AX, C

mov R, AX

ret

Pub\_Proc ENDP

CODE ENDS

END

3. Передача параметров через стек.

Такой метод применим не только между ассемблерными модулями, но и с разными языками. STDCALL – стандартная взаимосвязь процедур по параметрам. Основным средством доступа к параметрам является BP, для которого:

* его значение при запуске процедуры сохраняется в стеке;
* доступ к параметрам осуществляется с помощью базовой адресации.

Структура стека на момент начала работы с параметрами вызванной процедуры:

(Этот вариант используется в С.)

|  |  |
| --- | --- |
| BP | 🡨 SP🡪BP |
| CS:IP | 🡨 (BP)+2 |
| Par1 | 🡨 (BP)+4/6 |
| Par2 | 🡨 (BP)+6/8 |
| ……… |  |
| ParN | 🡨 (BP)+(N+4)/(N+6) |
| Стек |  |

Вариант для Pascal такой же, за исключением порядка параметров.

Пример:

;main

CODE Segment

EXTERN ADD3UP

Push AX

Push BX

Push CX

CALL ADD3UP

ADD SP,6

CODE ENDS

END start

;процедура обработки

ADD3UP Proc NEAR

Public ADD3UP

Push BP

mov BP, SP

mov AX, [BP+4]

add AX, [BP+6]

add AX, [BP+8]

pop BP

ret

ADD3UP ENDP

4. Передача параметров через поток кода.

Передаваемые параметры размещаются непосредственно в коде программы, сразу за вызывающей командой.

Пример:

Call Print\_It

db ‘печатная строка’

;продолжение программы.

Плюсы: возможность использования любого числа параметров;

Недостатки: доступ косвенный, следовательно медленный.

5. Передача параметров результата.

Передача параметров результатов может быть реализована через стек, AX (BYTE, WORD), DX:AX (DWORD), ST(0) – вершина стека x87 (FLOAT).

**П2.5. Связь ассемблера с языками высокого уровня**

Существует соглашения для связи ассемблера с каждым из языков. Здесь есть свои особенности.

1. Способ передачи управления (тип вызова). В Pascal, если ассемблеровская процедура описана в основной программе или в implementation, то она вызывается как NEAR; при описании в interface–вызов FAR.
2. Сегментация и модели памяти. Если выбрана модель .LARGE, то стек надо описывать как .FARSTACK и он уже не будет относиться к группе DGROUP.
3. Особенности описания общих и предопределенных переменных. В Pascal ограничений нет, но рекомендуется начинать переменные с заглавной буквы; в С-переменных необходимо начинать с символа подчеркивания «\_»
4. Управление регистрами. DS – нужно следить за ним. РОН нужно сохранять или в головной программе, или в процедуре на ассемблере.
5. Способы передачи параметров. Как правило, параметры передаются через кадр стек, размещение параметров идет в различном порядке.
6. Способ возвращения результата. Возвращение результатов может быть реализовано через стек, AX (BYTE, WORD), DX:AX (DWORD), ST(0) – вершина стека x87 (FLOAT).
7. Способ восстановления стека. Pascal – в вызываемой программе, в С – вызывающей программе (для возможности оптимизации)
8. Способы задания и вызова ассемблерного модуля:

а. Раздельная компиляция. Модули на различных языках компилируются отдельно и в вызывающем модуле указан специальный оператор описания модуля на другом языке и его загрузки в виде объекта файла.

б. Встроенная компиляция. Фрагмент или команда ассемблера указывается как встроенный оператор, и сам компилятор вызывает ассемблерный транслятор автоматически.

**Связь ASSEMBLER и PASCAL.**

**Пример 1:**

Test(i, j, k)- вычисляет f=i+j-k; i, j, k∈Z.

i EQU WORD PTR [bp+8]

j EQU WORD PTR [bp+6]

k EQU WORD PTR [bp+4]

.Model Small

.Code

Public Test

Test Proc

Push bp

mov bp, sp

mov ax, i

add ax, j

sub ax, k

pop bp

ret 6

Test EndP

END

Function Test(i,j,k):Word;

External;

{$L Test.obj}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

F:=Test(i,j,k);

**Пример 2:**

Декодирование символьной строки, которая расположена в области памяти с меткой Buffer длиной Count; кодирование – сдвиг символов на 1 бит влево.

shifr.asm

Code Segment byte Bublic

Assume cs:Code

Public Coder, Decoder

Buffer EQU DWORD PTR [bp+8]

Count EQU DWORD PTR [bp+6]

Cur\_Byte EQU BYTE PTR es:[di]

;FAR вызов Coder(var Buffer, count: word)

Coder Proc

Push bp

mov bp, sp

mov cx, count

jcxz final

les di, Buffer; загрузка смещения с учетом es

cld ; в стороны возрастания адресов

cycle:

moc al, Cur\_Byte

rol al,1

stosb ; переписывает соответствующий символ назад в строку

loop cycle

final:

pop bp

ret 6

Coder EndP

Decoder Proc

Push bp

mov bp, sp

mov cx, count

jcxz final

les di, Buffer; загрузка смещения с учетом es

cld ; в стороны возрастания адресов

cycle:

moc al, Cur\_Byte

ror al,1

stosb ; переписывает соответствующий символ назад в строку

loop cycle

final:

pop bp

ret 6

Decoder EndP

Code EndS

END

Type TMode=(Coder, Decoder);

var Mode:TMode;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{$F+}

Procedure Coder (var Buffer, count: word); External;

Procedure Decoder(var Buffer, count: word); External;

{$F-}

{$L shifr.obj}

Begin

Case Mode of

Code: begin

Coder(str1,80);

write(codfile,str1);

end;

Decode: begin

read(codfile,str1);

Decoder(str1,80);

end;

end;

End.

**Пример 3:** на встроенный ассемблер

Написать функцию, которая формирует слово, выбираемое по адресу 0040:006ch (BIOS счетчик таймера).

Function Get\_Time:Longint;

var time:longunt;

begin

asm

push es

mov ax, 0040h

mov es, ax

mov ax, es:[006ch]

mov time, ax

pop es

end;

Get\_Time:=time;

end;

**Пример 4:**

Использование стандартных функций Паскаля в ассемблерном модуле.

DATA Segment Word Public

EXTERN ch:Byte

DATA ENDS

CODE Segment Byte PUBLIC

ASSUME DS:DATA,CS:CODE

EXTERN Readkey: Far

AsmProc Proc Far

Push bp

mov bp, sp

pusha

Call Readkey

mov ch, al

popa

pop bp

ret 2

AsmProc EndP

CODE ENDS

END

**Связь ASSEMBLER и С**

**Пример 1:**

Test(i, j, k)- вычисляет f=i+j-k; i, j, k∈Z.

Вызов из С скомпилировался в

Push WORD PTR DGROUP: \_k

Push WORD PTR DGROUP: \_j

Push WORD PTR DGROUP: \_i

call Near PTR \_TEST

этот вызов порождает кадр стека

|  |  |
| --- | --- |
|  |  |
| 🡪 | BP |
|  | IP |
|  | I |
|  | J |
|  | K |

.Model Small

.Code

Public \_Test

\_Test Proc

Push bp

mov bp, sp

mov ax, [bp+4]

add ax, [bp+6]

pop bp

sub ax, [bp+8]

ret

\_Test EndP

END

**Пример 2:**

Разработать ассемблерный модуль, который вычисляет

|  |
| --- |
| StartVal+Repeat |
| S=Σ i |
| i=StartVal |

total.asm

.Model Small

.Data

Extern \_Repetitions: word

Public \_StartVal DW 0

Total DW ?; локальная переменная

.Code

Public \_Doloop

\_Doloop Proc

mov cx, \_Repetitions

mov ax, \_StartVal

mov Total, ax

cycle:

inc ax

add Total, ax

Loop cycle

mov ax, Total

ret

\_Doloop EndP

END

Call\_Tot.c

extern “c” int Doloop(void);

extern int Repetitions,

StartVal;

main(){

Repetitions=10;

StartVal=2;

print(“%d\n”,Doloop());

}

**Пример 3:** на встроенный ассемблер в программе написанной на С.

Составить встроенную процедуру для заполнения некоторой области памяти размером count байтов заданным символом.

#pragma inline

void memset(void \*dest, char val, short count){

asm{

push es

push di

les di, dest

mov cx, count

mov al, val

mov Total, ax

rep stosb

pop di

pop es

}

}

int main(){

char buf[0x20];

memset(buf,0xFF, size of (Buf));

return 0

}

**П2.6. Макросредства**

Макросы – это еще один способ однократного описания последовательности действий и затем многократного их выполнения с различными параметрами. Основное отличие от использования процедур заключается в том, что макросы – это директивы ассемблеру (обрабатываются они при помощи препроцессорной обработки).

Достоинства:

* они более универсальны, так как позволяют параметрически управлять не только заданием обрабатываемых объектов, но и действиями над этими объектами;
* их использование не связано с выполнением команд CALL и RETURN, поэтому применение макросов ускоряет выполнение программы;
* поскольку описание макросов воспринимается как обычный текст при подстановке, то легко организуются библиотеки макросов.

Недостатки:

* существенное увеличение длины программы, связанное с подстановкой тела макроса во все точки его вызова.

Использование макросов связано с понятиями макроопределение, макровызов, макрорасширение.

Макроопределение – описание действия, выполняющегося макросом применительно к фиксированным параметрам; состоит из заголовка, тела и концовки.

Заголовок: Nam\_Macro MACRO [список фиктивных параметров]

Тело: последовательность операторов

Концевик: END [Nam\_Macro]

Пример1. Макроопределение установки курсора в левый верхний угол экрана.

Home Macro

mov dh, 0

mov dl, 0

mov al, 2; функция перемещения курсора

int 10h ; обработчик управления указателя

ENDM

Пример 2.

Sr\_Mov Macro R1,R2

push R1

pop R2

ENDM

Пример 3.

Add\_W Macro par1, par2, sum

mov ax, par1

add ax, par2

mov sum, ax

ENDM

Макровызов представляет собой директиву ассемблера, помещаемую в текст программы и состоящую из имени макроса и списка фактических параметров, если они требуются. Если список фактических параметров меньше фиктивных, то оставшимся фиктивным параметрам присваиваются нулевые значения. Макрорасширение – это подстановка тела макроса из макроопределения на место макровызова с заменой фиктивных параметров на фактические. Например, макровызов

Sr\_Mov ds,es

при макрорасширении будет заменен на

push ds

pop es

Пример некорректного задания параметра.

Time\_Msg Macro XXX

TimXXX: db ‘сейчас ХХХ часов$’

ENDM

* параметр макроопределения не может быть частью идентификатора;
* параметр макроопределения непосредственно не может использоваться в строке, так как его трудно распознать.

Правильный вариант.

Time\_Msg Macro XXX

Tim&XXX: db ‘сейчас &ХХХ& часов$’

ENDM

Если в теле макроса используется метка, то она должна быть объявлена локальной, чтобы исключить многократное объявление имен.

Пример 4.

Delay Macro count

local cycle

push cx

mov cx, count

cycle: loop cycle

pop cx

ENDM

В макросредствах широко применяются два вида директив (повторения и условные директивы).

**Использование в макросах директив повторения.**

REPT, IRP, IRPC – это встроенные макросы, следовательно должен быть концевик ENDM.

1. REPT

REPT выражение

тело

ENDM

Выражение задает количество раз, которое должно повториться тело. Пример.

REPT 4

shl ax,1

ENDM

При макрорасширении макровызов будет заменен на четыре повторяющихся команды shl ax,1.

Пример 5.

Alloc Macro tlab, lenght

value=0

tlab EQU this byte

rept lenght

value=value+1

db value

ENDM

ENDM

Вызов:

Date Segment

Alloc tab1, 40

Date ENDS

1. IRP

IRP фиктивный\_аргумент, <список значений>

тело

ENDM

Каждый раз из списка подставляется новое значение вместо фиктивного аргумента.

Пример 6.

Savregad Macro

irp r, <ax, bx, dx>

push r

ENDM

ENDM

1. IRPC

IRPC фиктивный\_аргумент, строка

тело

ENDM

При расширении при каждом повторе тела будет подставляться очередной символ из строки.

Пример 7.

IRPC odd5, 13579

db odd5

ENDM

При макрорасширении получим команды db 1, db 3, db 5, db 7, db 9.

Пример 8.

IRPC char, hello

db char

ENDM

При макрорасширении получим команды db h, db e, db l, db l, db o.

Пример 9.

IRPC char, hello

db char

ENDM

При макрорасширении получим команды db ‘h’, db ‘e’, db ‘l’, db ‘l’, db ‘o’.

**Условные директивы**

Позволяют при трансляции программы просмотреть или пропустить фрагмент программы в зависимости от проверки условия.

1. Директивы условной трансляции.
2. Директивы условной генерации ошибок (позволяют выдать соответствующее сообщение об ошибках на этапе трансляции).

Назначение директивы условной трансляции:

1. Написание отладочных версий программ, в которых после завершения отладки можно протранслировать код без соответствующих операторов

(Например Debug = 1).

1. Написание универсальных макросов, настраиваемых на систему.

IF xxx

Операторы (Тело 1)

[ ELSE

Операторы (Тело 2) ]

ENDIF

xxx – некоторое выражение, проверка которого должна давать булевский результат

xxx = 0 Ложь

xxx ≠ 0 Истина

Допускается 255 уровневое вложение.

Существуют предопределенные операторы IF:

IF1 Его тело транслируется только на первом проходе.

Пример10.

IF1

INCLUDE MYLIB

ENDIF

IFE xxx Транслируется, если выражение (xxx) равно нулю.

ALLOC MACRO LENGHT

VALUE

REPT LENGTH

IFE VALUE – 0FFh

EXITM

ENDIF

VALUE = VALUE + 1

ENDM

ENDM

EXITM – Выход из макроса

Написанный макрос обеспечивает заполнение памяти целыми числами от 1 до 256 байт.

II.

IFDET / IFNDEF имя \_ переменной

Тело выполняется, если определено / не определено имя \_ переменной.

Пример 12. (задает возможность проверки дисплея):

IFDEF Disptest

; часть первой общей проверки дисплея

IFCOLOR

; часть проверки цветного дисплея

ELSE

; часть проверки монохромного дисплея

ENDIF

; часть второй общей проверки

ELSE

; код, выполнения при отсутствие проверки

ENDIF

IFB / IFNB <аргумент> Проверяет, имеется или отсутствует аргумент (IF Blank – если отсутствует)

Применяется:

1. Для выдачи подсказок об отсутствия аргументов.
2. Для разработки рекурсивных макросов с переменным числом в параметрах.

Пример 11.

I способ применения

Make \_ Pasc \_ String Macro Id, Msg

Local MsgLen, Endstr

Ifb <id>

Display ‘Задать переменную id’

ExitM

IFB <Msg>

Display ‘Задать преобразованную строку’

ExitM

Else

Id Db Endstr-id-1,’&Msg&’

Label EndStr Byte

EndIf

EndIf

EndM

II способ применения

PushReg Macro R1, R2, R3, R4, R5, R6

Ifnb <R1>

Push R1

PushReg R2, R3, R4, R5, R6

Endif

EndM

Отличительные признаки рекурсивного вызова макроса:

1. Должен быть вызов заново себя с количеством элементов меньше на единицу.
2. Предусмотрен контроль аргумента с помощью директивы Ifnb.

*Способы использования макросов*

1. Непосредственное размещение макроопределения в тексте программы. (Оно размещается в самом начале, обычно после заголовка.)

Применение:

1. Создание макробиблиотеки с последующим подключением ее к программе. Библиотека строится как обычный файл. В основной файл добавляются следующие команды:

IF1

Include My \_ Maclib

EndIf

Можно исключить некоторые макросы из библиотеки, если они не нужны.

Purge Mac1, Mac2, Mac7