УДК 681.3-181.4

М54

Рецензенты:

Кандидат физико-технических наук, генеральный директор

ЗАО «ИНФОРМТЕК»

**М.В. Матвеичев**

Печатается по решению редакционно-издательского совета Муромского института

М54 **ПРОГРАММИРОВАНИЕ АППАРАТНЫХ СРЕДСТВ IBM PC**

**И учебного стенда SDK -1.1**

: ” Персональные ЭВМ и Спец. ЭВМ " / Сост.: М.Н. Кулигин – Муром: Изд.- полиграфический центр МИ ВлГУ, 20011.– … с. - Библиогр.: 19 назв.

Лабораторный практикум предназначен для студентов четвёртого курса, обучающихся по специальностям 230101 ”Вычислительные машины, комплексы, системы и сети”, 230105 ”Программное обеспечение вычислительной техники и автоматизированных систем ”. Он содержат варианты заданий для 5-и лабораторных работ и контрольные вопросы к ним, требования к оформлению отчётов, необходимый теоретический материал

и примеры программ на языке ассемблера для домашней подготовки. В приложении приведено описание системных функций, необходимых для выполнения лабораторных работ.

УДК 681.3-181.4

© Муромский институт (филиал)

Владимирского государственного

университета, 2011

**ОГЛАВЛЕНИЕ**

[Введение 5](#_Toc186388258)

[1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 7](#_Toc186388259)

[1.1. Режимы работы, системы команд и дополнительные способов адресации процессоров семейства IA-32 7](#_Toc186388260)

[1.2. Программирование контроллера прерываний 8259А и организация прерываний в IBM PC 36](#_Toc186388261)

[1.3. Программирование таймера 8254 и генерация звука 53](#_Toc186388262)

[1.4.Возможности последовательного интерфейса RS-232 и программирование его контроллера. 59](#_Toc186388263)

[2. ОБЩИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ 75](#_Toc186388264)

[3. ЛАБОРАТОРНАЯ РАБОТА № 1 76](#_Toc186388265)

[Изучение режимов работы, системы команд и дополнительных способов адресации процессоров семейства IA-32. 76](#_Toc186388266)

[3.1. Домашняя подготовка 76](#_Toc186388267)

[3.2. Лабораторное задание 76](#_Toc186388268)

[3.3. Варианты задания 77](#_Toc186388269)

[3.4. Контрольные вопросы 83](#_Toc186388270)

[4. ЛАБОРАТОРНАЯ РАБОТА №2 84](#_Toc186388271)

[Программирование контроллера прерываний 8259А и организация прерываний в IBM PC 84](#_Toc186388272)

[4.1. Домашняя подготовка 84](#_Toc186388273)

[4.2. Лабораторное задание 84](#_Toc186388274)

[4.3. Варианты задания 85](#_Toc186388275)

[4.4. Контрольные вопросы 87](#_Toc186388276)

[5. ЛАБОРАТОРНАЯ РАБОТА № 3 89](#_Toc186388277)

[Программирование таймера 8254 и генерация звука. 89](#_Toc186388278)

[5.1. Домашняя подготовка 89](#_Toc186388279)

[5.2. Лабораторное задание 90](#_Toc186388280)

[5.3. Варианты задания 90](#_Toc186388281)

[5.4. Контрольные вопросы 92](#_Toc186388282)

[6. ЛАБОРАТОРНАЯ РАБОТА № 4 93](#_Toc186388283)

[Организация доступа к дискам в режиме MS DOS и анализ системных ошибок……………………………………………………………………………..93](#_Toc186388284)

[6.1. Домашняя подготовка 93](#_Toc186388285)

[6.2. Лабораторное задание 93](#_Toc186388286)

[6.3. Варианты заданий 93](#_Toc186388287)

[6.4. Контрольные вопросы 95](#_Toc186388288)

[6.5. Теоретическая часть 96](#_Toc186388289)

[7. ЛАБОРАТОРНАЯ РАБОТА № 5 101](#_Toc186388290)

[Изучение возможностей последовательного интерфейса RS-232 и программирование его контроллера. 101](#_Toc186388291)

[7.1. Домашняя подготовка 101](#_Toc186388292)

[7.2. Лабораторное задание 101](#_Toc186388293)

[7.3. Варианты задания 101](#_Toc186388294)

[Библиографический список 104](#_Toc186388295)

[*Приложения* 106](#_Toc186388296)

[*Приложение 1* 106](#_Toc186388297)

[Терминальные функции Win32 106](#_Toc186388298)

[*Приложение 2* 112](#_Toc186388299)

[Функции MS DOS для выполнения лабораторных работ 112](#_Toc186388300)

## Введение

Персональные ЭВМ (персональные компьютеры) и спец. ЭВМ, построенные на базе современных микропроцессоров, представляют собой комплекс аппаратных и программных средств, для изучения которых лучше всего использовать “родной” язык микропроцессора (МП) – язык ассемблера.

Он является “естественным” языком низкого уровня, на котором “разговаривает” программист с компьютером. Специализация ЭВМ достигается путем подключения к ней нестандартного оборудования с целью автоматизации управления этим оборудованием, поэтому при решении подобных задач язык ассемблера незаменим – с его помощью программист получает прямой доступ к аппаратным ресурсам компьютера.

Для подключения к компьютеру нестандартных периферийных устройств (ПУ) необходимо овладеть методикой работы с этими устройствами на уровне регистров (портов) и прерываний. Поэтому лабораторные работы, описание которых представлено в данном пособии, рассчитаны как на изучение архитектуры базового микропроцессора IBM PC – совместимого компьютера, так и на приобретение навыков написания программ, работающих с аппаратурой компьютера через порты, прерывания и функции драйверов.

Целью данного учебного пособия является методическое обеспечение по выполнению курса лабораторных работ. В описание каждой лабораторной работы включен необходимый теоретический материал, рассчитанный на самостоятельную домашнюю подготовку.

**Структура учебного пособия.**

Пособие содержит описания 5 лабораторных работ.

* Лабораторная работа №1 – изучение режимов работы, системы команд и дополнительных способов адресации процессоров семейства IA-32.
* Лабораторная работа №2 – организация прерываний в IBM PC и программирование контроллера прерыва­ний 8259А.
* Лабораторная работа №3 – программирование таймера 8254 и генерация звука в IBM PC.

- Лабораторная работа №4 – проектирование программного обеспечения систем управления на базе учебного лабораторного комплекса SDK -1.1.

* Лабораторная работа №5 – изучение возможностей последовательного интерфейса RS-232 и программирование его контроллера.

Описание лабораторной работы включает задание для домашней подготовки, контрольные вопросы, лабораторное задание и необходимый теоретический материал для домашней подготовки.

**Основные термины и определения**

**Интерфейс** -это совокупность унифицированных аппаратных, программных и конструктивных средств, необходимых для реализации взаимодействия различных функциональных эле­ментов в системах при условиях, предписанных стандартом и направленных на обеспечение информационной, электри­ческой и конструктивной совместимости указанных элемен­тов.

Устройства сопряжения и интерфейсные устройства предназначены для управления вводом и выводом информации и согласования интерфейсов компьютера с одной стороны и внешних устройств и объекта управления с другой стороны. Конструктивно эти устройства могут выполняться в виде одной БИС, нескольких БИС или в виде отдельной платы. Устройства сопряжения и интерфейсные устройства могут иметь достаточно сложную структуру Выполняемые ими функции могут программироваться, в ряде случаев в состав таких устройств входит микропроцессор. Устройства сопряжения и интерфейсные устройства со сложной структурой иногда называют контроллерами.

**Процессор** – основная часть ЭВМ, непосредственно осуществляющая процесс обработки данных и управляющая этим процессом.

**Микропроцессор (МП)** – программно управляемое устройство, осуществляющее процесс обработки цифровой информации и управления им построенное на одной или нескольких интегральных микросхемах. Иными словами микропроцессор это процессор, реализованный на одной или нескольких СБИС.

**Оперативная память** – память, предназначенная для временного хранения данных и команд, необходимых процессору для выполнения им операций.

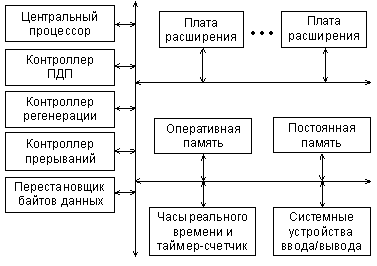
**Постоянная** **память** - это память, используемая для хранения информации, которая не должна изменяться в ходе работы компьютера.

**Прерывание** – это аппаратная функция, вызывающая приостановку операций процессора, запоминание его состояния и выполнение специальной программы, которая называется программой обработки прерывания или обработчиком прерывания (ISR).

1. **Общие сведения об**  **архитектуре персонального компьютера и**

**архитектуре стенда SDK 1.1**

Персональный компьютер типа IBM PC имеет традиционную архитектуру микропроцессорной системы и содержит все обычные функциональные узлы: процессор, постоянную и оперативную память, устройства ввода/вывода, системную шину, источник питания (рис.1.1). Основные особенности архитектуры персональных компьютеров сводятся к принципам компоновки аппаратуры, а также к выбранному набору системных аппаратных средств.

  
Рис. 1.1. Архитектура персонального компьютера типа IBM PC

Функции основных узлов компьютера следующие:

* Центральный процессор - это микропроцессор со всеми необходимыми вспомогательными микросхемами, включая внешнюю кэш-память и контроллер системной шины. В большинстве случаев именно центральный процессор осуществляет обмен по системной шине.
* Оперативная память может занимать почти все адресуемое пространство памяти процессора. Однако чаще всего ее объем гораздо меньше. В современных персональных компьютерах стандартный объем системной памяти составляет, как правило, от 512 Мбайт до 2 Гбайт. Оперативная память компьютера выполняется на микросхемах динамической памяти и поэтому требует регенерации.
* Постоянная память (ROM BIOS — Base Input/Output System) имеет небольшой объем (до 128 Кбайт), содержит программу начального запуска, описание конфигурации системы, а также драйверы (программы нижнего уровня) для взаимодействия с системными устройствами.
* Контроллер прерываний (описание функций этого контроллера приведено в разделе 2.2). Все режимы функционирования контроллера прерываний задаются программно процессором перед началом работы.
* Контроллер прямого доступа к памяти принимает запрос на ПДП из системной магистрали, передает его процессору, а после предоставления процессором магистрали производит пересылку данных между памятью и устройством ввода/вывода. Все режимы функционирования контроллера ПДП задаются программно процессором перед началом работы. Использование встроенных в компьютер контроллеров прерываний и ПДП позволяет существенно упростить аппаратуру применяемых плат расширения.
* Контроллер регенерации осуществляет периодическое обновление информации в динамической оперативной памяти путем проведения по шине специальных циклов регенерации. На время циклов регенерации он становится хозяином (задатчиком) шины.
* Часы реального времени и таймер-счетчик - это устройства для внутреннего контроля времени и даты, а также для программной выдержки временных интервалов, программного задания частоты и т.д.
* Системные устройства ввода/вывода - это те устройства, которые необходимы для работы компьютера и взаимодействия со стандартными внешними устройствами по параллельному и последовательному интерфейсам. Они могут быть выполнены на материнской плате, а могут располагаться на платах расширения.
* Платы расширения устанавливаются в слоты (разъемы) системной магистрали и могут содержать оперативную память и устройства ввода-вывода. Они могут обмениваться данными с другими устройствами на шине в режиме программного обмена, в режиме прерываний и в режиме ПДП. Предусмотрена также возможность захвата шины, то есть полного отключения от шины всех системных устройств на некоторое время.

Важная особенность подобной архитектуры - ее открытость, то есть возможность включения в компьютер дополнительных устройств, причем как системных устройств, так и разнообразных плат расширения. Открытость предполагает также возможность простого встраивания программ пользователя на любом уровне программного обеспечения компьютера.

**Учебные микропроцессорные комплексы (стенды)** на базе микроконтроллеров предназначены для изучения принципов организации и работы микропроцессорной элементной базы, вспомогательных элементов (память, контроллеры ввода-вывода и др.),получения навыков проектирования и программирования микропроцессорных систем различного назначения.

Внимания заслуживает опыт ООО «ЛМТ» (Санкт-Петербург), которое разработало и последовательно развивает семейство микропроцессорных стендов инструментального и учебного назначения - SDK.

Основу лабораторного комплекса составляет контроллер-конструктор (микропроцессорный стенд) SDK-1.1на базе ОКЭВМ фирмы Analog Devices ADuC812. Сам лабораторный комплекс представляет собой совокупность контроллера-конструктора, подключенного к персональному компьютеру, и программного обеспечения для ПК и SDK-.1.1.

Подключение осуществляется к СОМ-порту ПК через кабель RS232, комплекс инструментальных программ

обеспечивает весь процесс программирования SDK-1.1: компиляцию, доставку

и запуск программ в SDK-1.1.

Главной областью применения микропроцессорного стенда SDK-1.1, безусловно, является обучение различным аспектам встраиваемой вычислительной техники. Студенты имеют возможность ознакомиться на практике с проектированием, программированием, отладкой и использованием создаваемой ими из "конструктора" SDK-1.1. Разнообразные устройства, входящие в состав стенда, позволяют изучить круг вопросов, связанных с организацией взаимодействия

с ними через типичные интерфейсы, применяемые во встраиваемых вычислительных системах.

Микропроцессорный стенд SDK-1.1 построен на базе однокристальной микро-ЭВМ фирмы Analog Device типа ADuC812 (вычислительное ядро MCS-51) и имеет в своем составе разнообразные устройства, предназначенные для ввода, обработки и вывода информации в цифровом и аналоговом виде:

* 8-ми канальный аналого-цифровой преобразователь:
* 2-х канальный цифроаналоговый преобразователь;
* устройства ввода и отображения информации (матричная клавиатура 4x4 клавиши, текстовый жидкокристаллический дисплей, пьезокерамический  
  акустический излучатель, набор сигнальных светодиодов и переключателей);
* параллельный 16..20-ти разрядный порт:
* часы/календарь реального времени:
* электрически стираемое программируемое ПЗУ (EEPROM) на кристалле ADuC812 (640 байт):
* внешняя память - EEPROM емкостью 128 байт.

Стенд SDK-1.1 может работать полностью автономно от ПК.

Лабораторный комплекс предназначен для освоения студентами архитектуры и методов проектирования:

* систем на базе микропроцессоров и однокристальных микро-ЭВМ;
* встраиваемых контроллеров и систем сбора данных:
* периферийных блоков вычислительных систем:
* подсистем ввода-вывода встраиваемых вычислительных систем.

Подготовки программ для микроконтроллера ADuC812 осуществляется на языке программирования Си или Ассемблера (рис. 1.2) на ПК в обычном текстовом редакторе (или средах программирования, IDE, предназначенных для разработки программ под ядро MCS-51), далее программа компилируется в исполняемый модуль, доставляемый в стенд через СОМ-порт ПК с помощью специального программного обеспечения, входящего в состав стенда.

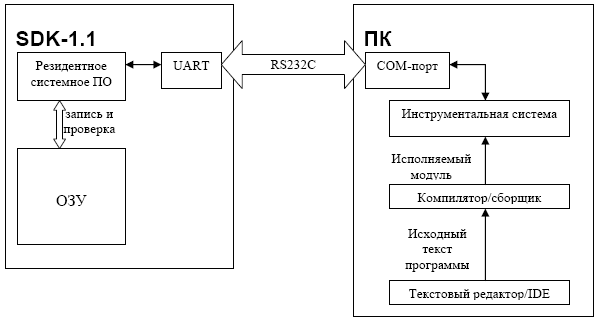


Рисунок 1.2 - Процесс написания программ для SDK-1.1

Программы для стенда располагаются во внешней памяти программ/данных – внешнем ОЗУ объемом 512 Кб. Из этих 512 Кб как память программ (особенности MCS-51) могут использоваться лишь 56 Кб (в стенде первые 8 Кб памяти программ заняты ПЗУ, в котором располагается системное программное обеспечение, отсюда 64 Кб - 8 Кб = 56 Кб). Однако, как показывает практика, программы такого размера для стенда подготавливать не требуется.

В процессе обучения с использованием SDK-1.1 студенты могут на практике ознакомиться с управлением периферийными устройствами, взаимодействующими с вычислителем посредством различных интерфейсов, освоить некоторые специфические аспекты программирования встраиваемых вычислительных систем, эффективного управления ресурсами. В стенде предусмотрена возможность симулировать некоторые внешние сигналы без использования дополнительного оборудования: сигналы внешних прерываний, счетные импульсы таймеров, аналоговые сигналы на входах АЦП. Интересно отметить возможность программного инициирования прерываний, не предусмотренную в MCS-51. однако реализованную в стенде через механизм программного управления состоянием входа внешнего прерывания INT0 ADuC812.

ПЗУ с резидентным программным обеспечением реализовано на кристалле ОЭВМ ADuC812 по технологии FLASH/ЕЕ и может быть перепрограммировано через интерфейс RS232C с обычного ПК. Новые версии резидентного ПО могут доставляться в стенд без использования специальных программаторов, а тем более новых микросхем ПЗУ - достаточно иметь лишь образ доставляемой программы в виде файла и специальную утилиту на ПК.

Некоторые устройства стенда подключены к вычислителю через периферийный расширитель, реализованный на базе ПЛИС небольшой емкости, перепрограммируемой через имеемый в SDK-1.1 JTAG-порт, что дает возможность при желании изменять механизмы работы с этими устройствами.

## 2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ по тематике лабораторных работ

## 2.1. Режимы работы, системы команд и дополнительные способов адресации процессоров семейства IA-32

Операционная система Windows поддерживает два типа приложений.

**Оконное приложение** строится на базе специального набора функций API, составляющих графический интерфейс пользователя (Graphic User Interface, GUI). Оконное приложение представляет собой программу, которая весь вывод на экран производит в графическом виде. Первым результатом работы оконного приложения является отображение на экране специального объекта — окна.

**Терминальное приложение**, также называемое **консольным**, представляет собой программу, работающую в текстовом режиме. На первый взгляд 32-разрядные консольные приложения Windows очень похожи на 16-разрядные программы для MS DOS. Оба типа программ выполняют чтение данных со стандартного устройства ввода и записывают данные в стандартное устройство вывода, могут вывести на экран текстовые данные в цвете. Однако при более детальном рассмотрении 32-разрядные терминальные программы для Windows существенно отличаются от 16-разрядных программ для MS-DOS. Они используют 32-разрядный защищённый режим работы процессора, тогда как программы для MS-DOS работают в реальном режиме адресации. В программах для MS-DOS используются системные функции BIOS и DOS, вызываемые посредством программных прерываний. Консольные приложения, написанные для Win32, вызывают функции из той же библиотеки, что и графические приложения Windows.

В системе Windows для обращения к функциям используется стандартный ***интерфейс прикладных программ***(API), который представляет собой набор определений структур, констант и функций, использующихся во всех приложениях Windows. Благодаря этому обеспечивается возможность прямого манипулирования любым объектом системы посредством обычных вызовов системных функций. Интерфейс прикладных программ (API) Win32 подробно описан в документе, озаглавленном Platform SDK, изданном фирмой Microsoft. Аббревиатура SDK расшифровывается как Software Development Kit, или набор инструментальных средств разработки программного обеспечения. Он состоит из различных утилит, библиотек, примеров программного кода и документации, которая помогает программистам создавать приложения для системы Windows.

**Windows API** - это набор системных функций, входящих в состав ОС Windows и готовых для использования их прикладными программами. Эти функции сгруппированы по областям применения в динамических библиотеках (DLL), таких как kernel32.dll, user32.dll и gdi32.dll и др. Kernel32.dll содержит API функции, взаимодействующие с памятью и управляющие процессами. User32.dll контролиpуют пользовательский интерфейс. Набор функций Gdi32.dll ответственен за графические операции. Динамические библиотеки хранятся на диске в файлах с расширением .dll. Загрузка той или иной библиотеки в оперативную память компьютера происходит только при обращении к ней прикладной программы, то есть, код API функций не включается в исполняемый файл.

Вызов системных функций API win32 из программы на ассемблере подчиняется набору соглашений stdcall, ведущему свою родословную в части именования функций - от языка C, а в части передачи аргументов - от языка Pascal. С точки зрения прикладного программиста и с учетом специфики Windows и MASM эти соглашения заключаются в следующем:

- аргументы передаются вызываемой функции через стек. Если аргумент укладывается в 32-битное значение и не подлежит модификации вызываемой функцией, он обычно записывается в стек непосредственно. В остальных случаях программист должен разместить значение аргумента в памяти, а в стек записать 32-битный указатель на него (см. раздел « Передача параметров процедурам»). Таким образом, все передаваемые функции API параметры представляются 32-битными величинами, и количество байт, занимаемых в стеке для передачи аргументов, кратно четырем;

- вызывающая программа загружает аргументы в стек последовательно, начиная с последнего аргумента, указанного в описании функции, и кончая первым. После загрузки всех аргументов программа вызывает функцию командой call;

- за возвращение стека в исходное состояние после возврата из функции API отвечает сама эта вызываемая функция. Программисту заботиться о восстановлении указателя стека esp нет необходимости;

- вызываемая функция API гарантированно сохраняет регистры общего назначения ebp, esi, edi. Регистр eax, как правило, содержит возвращаемое значение. Состояние остальных регистров после возврата из функции API следует считать неопределенным. Полный набор соглашений stdcall регламентирует также сохранение системных регистров ds и ss. Однако для flat-модели памяти, используемой в win32, эти регистры значения не имеют.

На системном уровне этот набор соглашений добавляется вот еще чем. Компилятор при формировании из исходного текста объектного файла добавляет к началу имени функции символ подчеркивания, а к концу - выражение вида @n, где n - десятичное число, равное количеству байт, занятому в стеке под аргументы функции. Так формируется технологическое имя, позволяющее осуществить связывание не только по имени вызываемой функции, но и по количеству ее аргументов. Благодаря этому имени при сборке обнаруживаются ошибки программиста в случае, когда он задал для вызываемой функции неправильное число аргументов. Конечно, этому сервису далеко до строгого контроля типов C++, но от огромного количества трудноустранимых ошибок он все-таки оберегает. Соблюдение перечисленных соглашений обеспечивается компилятором автоматически. Для этого необходимо включить в начало исходного файла комбинацию директив:

.386  
.model flat, stdcall

Для всех приложений win32 она должна иметь именно такой вид, как показано здесь (см. также раздел Директива INVOKE).

**2.1.1 Наборы символов и функции Windows API.**

В системе Windows предусмотрены два типа наборов символов, которые можно использовать при вызове функций Win32 API: 8-разрядные символьные наборы стандарта ASCII/ANSI и расширенные 16-разрядные символьные наборы стандарта Unicode, применяемые в системах Windows NT, 2000 и ХР. Поэтому для работы с текстовыми данными существует два набора одинаковых функций Windows API, отличающихся в названии только последней буквой. Функции, оканчи­вающиеся на букву "А", обрабатывают 8-разрядные ASCII-строки, а если название функции заканчивается на букву "W" (от английского слова wide, или расширенный), они обрабатывают 16-разрядные расширенные наборы символов, включая стандарт Unicode. Один из примеров - функция WriteConsole:

- WriteConsoleA;

- WriteConsoleW.

Функции, название которых оканчивается на букву "W", не поддерживаются в систе­мах Windows 95 и 98. С другой стороны, в системах Windows NT, 2000 и ХР стандартным набором символов считается Unicode. Поэтому при вызове в них функций, таких как WriteConsoleA, операционная система сначала конвертирует текстовую строку из формата ANSI в формат Unicode, а затем вызывает функцию WriteConsoleW.

**2.1.2. Создание консольных Windows-приложений**

Язык ассемблера — язык системных программистов, исследователей принципов работы операционных систем, программ и аппаратных средств. Здесь не всегда нужны красивые графические оболочки, а наоборот, велика потребность в удобных средствах для работы с текстовой информацией. Операционная система Windows обеспечивает встроенную поддержку консолей, которые, по определению, являются интерфейсами ввода-вывода для приложений, работающих в текстовом режиме. Понятие «консоль» существует в вычислительной технике давно. В общем случае под «консолью» подразумевают текстовый терминал для управления компьютером. Видимая часть такого терминала — клавиатура (для ввода управляющих воздействий) и монитор (как средство отображения - реакции вычислительной системы). В Windows консоль представляет собой приложение, которое позволяет взаимодействовать с операционной системой посредством ввода текстовых команд. Такой способ управления компьютером позволяет решать в основном административные задачи. Приложение, с помощью которого поддерживается этот режим, называется консольным. Видимая часть консольных приложений называется *окном консольного приложения*.

Написание консольных приложений на ассемблере — задача более актуальная, чем написание оконных. Причина простая — малыми затратами нам становятся доступны практически все возможности Win32 API. Программист может запустить одновременно нескольких консольных приложений и при этом работать с мышью и клавиатурой в стиле Windows. Далее рассмотрим порядок действий для запуска консольного Windows-приложения и организацию обмена данными с ним.

Консоль состоит из одного входного и нескольких экранных буферов. Входной буфер представляет собой очередь, каждая запись которой содержит информацию относительно отдельного входного события консоли. Экранный буфер — двухмерный массив, содержащий символы, выводимые в окно консоли, и данные об их цвете.

Очередь входного буфера содержит информацию о следующих событиях:

- нажатии и отпускании клавиш;

- манипуляциях мышью — движение, нажатие и отпускание кнопок;

- изменении размера активного экранного буфера, состоянии прокрутки.

При запуске любого приложения в системе Windows для него создаётся либо текстовое (консольное), либо графическое окно. Для трансляции консольных приложений можно использовать MASM32 6.14 и TASM32 5.0.

Для MASM:

ml /с /coff cons1.asm  
link /subsystem:console cons1.obj

Для TASM32:

TASM32 /ml cons1.asm  
tlink32 /ap cons1.obj

Обычно при разработке программ на любом языке программирования всегда существуют правила, по которым оформляется файл исходного кода.

Ассемблер для Windows не является исключением. Ниже приводится шаблон, который можно всегда использовать при написании новой программы. Подробное описание директив шаблона см. в разделе «Директива INVOKE».

.386

.MODEL Flat, STDCALL

option casemap: none

includelib\masm32\lib\kernel32.lib

includelib \masm32\lib\user32.lib

includelib \masm32\lib\gdi32.lib

include \masm32\include\kernel32.inc

include \masm32\include\user32.inc

include \masm32\include\gdi32.inc

include \masm32\include\windows.inc

.DATA

;<Ваши инициализируемые данные>

......

.DATA?

;<Ваши не инициализируемые данные>

......

.CONST

;<Ваши константы>

......

.CODE

<метка>

;<Ваш код>

…..

......

end <метка>

.DATA

.DATA?

.CONST

.CODE

Все четыре директивы описывают секции в образе исполняемого файла. В адресном пространстве программы Win32 нет специфических сегментов, но существует возмож­ность поделить его на логические секции. Начало следующей секции отмечает конец предыдущей. Возможны две группы секций: данных и кода.

.DATA - эта секция содержит инициализированные во время трансляции программы данные (переменные, строки, массивы).

.DATA? - эта секция содержит неинициализированные данные программы. Преимущество неинициализированных данных в том, что они описываются (резервируются), но не включаются в образ исполняемого файла. Вы всего лишь сообщаете транс­лятору, сколько места вам нужно выделить в момент загрузки программы в памяти для размещения статических неинициализированных переменных, получающих свои значения в процессе работы программы.

.CONST - эта секция содержит объявления константных ячеек, используемыхпрограммой в режиме только чтение.

При написании программ не обязательно задействовать все три секции данных. Объявляются только те, которые действительно будут востребованы.

Для размещения исполняемого кода программы возможно наличие, и только в единственном экземпляре, секции .CODE.

Чтобы использовать функции из Win32 API, вы должны импортировать соответствующею dll. Это делается библиотеками импорта (.lib). Эти библиотеки необходимы, потому что они позволяют системе (windows) загружать dll динамически. Пакет win32asm ([win32asm.cjb.net](javascript:if(confirm('http://win32asm.cjb.net/%20%20\n\nThis%20file%20was%20not%20retrieved%20by%20Teleport%20Pro,%20because%20it%20is%20addressed%20on%20a%20domain%20or%20path%20outside%20the%20boundaries%20set%20for%20its%20Starting%20Address.%20%20\n\nDo%20you%20want%20to%20open%20it%20from%20the%20server?'))window.location='http://win32asm.cjb.net/')) снабжен библиотеками для большинства стандартных dll. Вы можете подключить библиотеку директивой includelib.

Includelib C:\masm32\lib\kernel32.lib

Этот код подключает библиотеку импорта kernel32.lib. В приведенной выше команде мы указали библиотеку kernel32.lib. Зачем она нужна? Файл этой библиотеки входит в набор инструментальных средств разработки программного обеспечения для платформы Microsoft Windows (Software Development Kit, или SDK). В нем содержится информация, позволяющая связать пользовательскую программу с функциями операционной системы, которые находятся в еще одном файле под именем kernel32.dll. Последний является неотъемлемой частью операционной системы Microsoft Windows и называется динамиче­ски загружаемой библиотекой (Dynamic Link Library, или DLL). В нем находится испол­няемый код функций, с помощью которых осуществляются операции посимвольного ввода-вывода. На самом деле, файл kernel32 .lib является как бы "переходником" для файла kernel32.dll.

Но подключение библиотека импорта это еще не все, что вы должны сделать. Файлы include (.inc) также необходимы. Файлы include подключаются так: include \masm32\include\kernel32.inc.

Внутри include файлы, содержат определение прототипов функций dll, так что вы можете использовать директиву invoke (см. раздел Директива **PROTO)**.

**include-файл 'windows'**

Есть еще специальный include-файл, названный windows.inc, который содержит все константы и структуры для Win32 API.

В шаблоне используется include-файл из пакета masm:

include \masm32\include\windows.inc.

Директива option casemap:none говорит ассемблеру сделать метки чувствительными к регистрам, то есть MessageBox и messagebox - это различные имена. Для правильной работы файла windows.inc, это значение должно быть 'none'.

**2.1.3. Использование процедур при разработке программ**

Любая программа, кроме самой простой, состоит из некоторого количества команд, выполняющих различные задачи, обусловленные алгоритмом. Можно, конечно, напи­сать одну большую программу, весь код которой состоит из одной процедуры. Однако через некоторое время вы поймете, что разобраться в ней, а тем более отладить код, ста­новится все труднее и труднее. Интуиция нам подсказывает, что одну большую програм­му нужно разделить на части, каждая из которых будет решать какую-то отдельную зада­чу, и оформить эти фрагменты кода в виде отдельных процедур. Причем эти процедуры могут находиться как в одном, так и в нескольких исходных файлах, содержащих про­граммный код.

Перед тем как приступить к написанию программы, желательно сначала составить спецификацию этой программы, т.е. перечень требований, которым должна отвечать программа, и список выполняемых ею действий. Обычно спецификация составляется в результате тщательного анализа поставленной перед вами задачи. Готовую специфика­цию можно взять за основу при разработке программы.

Как было уже сказано выше, при использовании стандартного подхода к проектиро­ванию программ, сложная задача разбивается на ряд более простых, решение каждой из которых оформляется в виде отдельной процедуры. Процесс разбиения сложной задачи на ряд простых, называют функциональной декомпозицией, а такой подход к проектированию - нисходящим. Ниже приведены несколько предпосылок, положенных в основу нисходящего подхода к проектированию.

• Одну сложную задачу можно легко разделить на ряд простых подзадач.

• Программу гораздо легче написать, отладить и сопровождать, если каждую проце­дуру можно протестировать независимо от других.

• Использование нисходящего подхода к проектированию позволяет увидеть суще­ствующие взаимосвязи между процедурами.

• Когда создана общая структура проекта, вы легко можете сосредоточиться на решении конкретных задач, а также написании кода, реализующего каждую про­цедуру.

*При программировании под Win32 необходимо помнить следующее: Windows использует esi, edi, ebp и ebx для своих целей и ожидает корректной работы программы по сохранению их значений. Если же вы используете какой-либо из этих четырёх регистров между вызовами системных функций, то не забывайте сохранять и восстанавливать их значения в процессе работы.*

*В MASM метки, стоящие в процедуре, автоматически считаются локальными и, следовательно, имена меток в процедурах могут дублироваться. В TASM все метки по умолчанию считаются глобальными. Чтобы сделать метки, стоящие в процедуре локальными, они должны иметь префикс @@, а в начале программы следует указать директиву LOCALS*.

**Документирование процедур**

Хорошим стилем программирования считается использование в программе коротких и понятных комментариев, которые позволяют программисту быстро разобраться в ее су­ти. Ниже приведены несколько рекомендаций по поводу информации, которая должна быть размешена в начале каждой процедуры.

• Описание всех функций, выполняемых процедурой.

• Список входных параметров и описание их значений. Если какой-либо из пара­метров имеет особый тип, его нужно также указать. Обычно входные параметры указываются после ключевого слова **Передается**  (Receives).

• Список возвращаемых процедурой значений, указанных после ключевого слова **Возвращается** (Returns).

• Перечень особых требований (если таковые имеются), которые должны быть удовлетворены перед вызовом процедуры. Они называются входными условиями и указываются после ключевого слова **Требуется**  (Requires). Например, для про­цедуры, которая чертит на экране прямую линию, одним из входных условий является работа видеоадаптера в графическом режиме.

Учтя приведенные замечания, задокументируем процедуру с именем Sum:

Sum PROC

; Вычисляет и возвращает сумму трех 32-разрядных целых чисел.

; Перелается: три числа в регистрах ЕАХ, ЕВХ, ЕСХ.

; Числа могут быть как со знаком, так и без него.

; Возвращается: сумма в регистре ЕАХ, а также флаги состояния (переноса, переполнения и др.)

add eax, ebx

add еах, есх

геt

Sum ENDP

**Передача параметров процедурам**

Существует два основных типа параметров процедуры — регистровые и стековые. Далее рассмотрим способы объявления и использования стековых пара­метров.

Значения, которые передаются в процедуру перед ее вызовом, называются **аргументами.** Переменные процедуры, вместо которых подставляются переданные в процедуру значения, называются **параметрами**. Регистровые параметры используются при выполнении оптимизации скорости рабо­ты программы. Однако часто это приводит к излишнему загромождению кода вызываю­щей программы. Кроме того, обычно при загрузке в регистры значений аргументов при­ходится сохранять в стеке их текущее состояние, как, например, при вызове процедуры DumpMem:

Pushad ; сохраним регистры

mov esi, OFFSET array ; начальный адрес массива

mov ecx, LENGTHOF array ; размер массива в блоках

mov ebx. TYPE array ; определим формат вывода

call DumpMem ; отобразим содержимое памяти

popad ; восстановим регистры

Альтернативой регистровым являются стековые параметры. При этом перед вызовом процедуры нужные параметры сначала нужно поместить в стек. Например, если бы в процедуре DumpMem использовались стековые параметры, приведенный выше фрагмент кода выглядел бы так:

push TYPE array

push LENGTHOF array

push OFFSET array

call DumpMem

Для удобства программиста в компиляторе MASM предусмотрена специальная ди­ректива INVOKE, которая позволяет с помощью одного оператора поместить в стек аргу­менты и вызвать процедуру. Поэтому вместо приведённых выше четырех строк кода можно написать только одну:

INVOKE DumpMem, OFFSET array, LENGTHOF array, TYPE array

Кроме того, есть еще одна причина, по которой нужно освоить стековый способ передачи параметров: он используется практически во всех компиляторах языков высо­кого уровня. Поэтому, если вы хотите, например, вызвать одну из библиотечных функ­ций системы Windows, вы должны передать ей аргументы через стек (см. также раздел Директива INVOKE).

**Передача параметров процедурам через регистры**

При написании процедуры, которая выполняет одно из стандартных действий, напри­мер, такое как суммирование элементов целочисленного массива, не имеет смысла исполь­зовать в ней конкретные имена переменных. Дело в том, что тогда данная процедура смо­жет обрабатывать только элементы одного массива с указанным именем. Гораздо лучше при вызове процедуры передать ей в качестве параметров адрес массива и количество его элементов. Будем называть эти параметры аргументами или входными параметрами. В языке ассемблера для передачи параметров процедурам часто используются регистры общего назначения.

Для примера рассмотрим процедуру Sum, которая вычисляет сумму трех чисел, находящихся в регистрах еах, ebx и есх. Перед вызовом этой процедуры из процедуры main нам нужно загрузить соответствующие значения в регистры ЕАХ, ЕВХ и ЕСХ:

.data

Summa DWORD ?

. code

main PROC

mov eax, 10000h ; Первый аргумент

mov ebx, 20000h ; Второй аргумент

mov ecx, 30000h ; Третий аргумент

call Sum

mov Summa, eax ; Сохраним сумму в переменной Sum

После вызова процедуры Sum с помощью команды CALL в регистре EAX будет на­ходиться искомая сумма трех чисел, которую мы можем сохранить в переменной для дальнейшего использования.

**Передача параметров по значению и по ссылке**

*Передача по значению*. Если во время вызова процедуры ей в качестве аргументов пе­редаются копии значений переменных, то в таком случае говорят, что параметры пере­даются по значению. Программисты передают аргументы по значению в случае, если нужно защитить их значения от случайного изменения в процедуре. В этом случае значение переменной помешается в стек перед вызовом процедуры. В самой же вызываемой процедуре происходит выборка значения параметра из стека и его после­дующее использование. И даже если значение параметра будет изменено в процедуре, значение соответствующей переменной вызывающей программы, которая передана ей в качестве аргумента, останется без изменения. Дело в том, что при передаче аргументов по значению, из вызываемой процедуры нельзя получить доступ к переменным вызываю­щей программы.

*Передача по ссылке*. Если во время вызова процедуры ей в качестве аргументов пере­лаются адреса переменных, то в таком случае говорят, что параметры передаются по ссылке. При этом программист получает возможность изменить значение исходной пере­менной из вызываемой процедуры, воспользовавшись переданным адресом. Существует хорошее практическое правило, которое гласит, что параметры нужно передавать по ссылке только в том случае, если в процедуре их значение должно быть изменено.

**Оператор ADDR**

Оператор ADDR используется в директиве INVOKE для того, чтобы передать в процедуру указатель на переменную (т.е. адрес переменной), а не значение самой переменной. Такой способ передачи аргументов называется передачей параметров по ссылке. Например, в приведенном ниже фрагменте в процедуру Array передается адрес массива myArray:

INVOKE Array, ADDR myArray

Оператор addr возвращает значение ближнего (near) или дальнего (far) указателя следующей за ним переменной в зависимости от выбранной программистом модели памяти программы. В защищенном режиме обычно используется линейная (flat) модель памяти, поэтому операторы ADDR и OFFSET возвращают одинаковые значения - 32-разрядное смешение переменной относительно начала сегмента памяти (т.е. адреса 00000000h).

**Директива INVOKE.**  Директива INVOKE (*можно использовать только* ***с к****омпилятором MASM*) является очень гибким средством вызова процедур и по сути заме­няет команду call процессоров Intel. Она позволяет передать в процедуру несколько аргументов. Аргументы, передаваемые процедуре в директиве INVOKE, перечисляются через запя­тую и могут отсутствовать вовсе.

Нетрудно заметить основную разницу между директивой INVOKE и командой CALL: - у последней нет списка аргументов.

*В директиве invoke можно указать произвольное число аргументов, причем их спи­сок может занимать несколько строчек исходного кода.*

**Директива PROTO**  создает прототип существующей процедуры. В прототипе описыва­ется имя процедуры и список ее параметров. Прототипы позволяют вызывать процедуру до того, как она будет формально определена. Тем, кому приходилось программировать на C++, понятие прототипа должно быть уже знакомо, поскольку ни одно объявление клас­сов не обходится без использования прототипов функций.

*Компилятор MASM* *требует*, чтобы для каждой процедуры, вызываемой с помощью оператора **invoke**, был описан прототип. Директива PROTO должна предшествовать в исходном файле оператору INVOKE. Другими словами, стандартный порядок этих дирек­тив и операторов должен быть такой:

My\_Sub PROTO ; Прототип процедуры

INVOKE My\_Sub ; Вызов процедуры

My\_Sub PROC

…….. ; Тело процедуры

My\_Sub ENDP

Возможен и другой вариант, когда тело процедуры находится в программе перед опе­ратором invoke, который ее вызывает. В этом случае объявлением прототипа процедуры считается директива PROC:

My\_Sub PROC

…. ; Тело процедуры

My\_Sub ENDP

INVOKE My\_Sub ; Вызов процедуры

*Для того чтобы использовать директиву invoke для вызова функций из Win32 API, определение прототипов функций dll разместили в файлы с расширением .inc (include файлы).*

В качестве примера использования директив **INVOKE** и **PROTO** рассмотрим программу сложения двух чисел.

.386

.model flat, stdcall

Includelib \myasm\lib\kernel32.lib

ExitProcess proto :DWORD

.code

start:

mov eax, 2

add eax, 3

Invoke ExitProcess, 0

end start

В этой программе инструкции процессора mov eax,2 и add eax,3 окружены директивами - специальными командами, которые должен выпол­нить не процессор, а сама программа-ассемблер (компилятор). Первая директива нашей пер­вой программы .386 начинается с точки и показывает, для какого процессора предназначена программа. В нашем случае это процессор Intel 80386 и более поздние модели, ведь семейство процессоров Intel совместимо снизу вверх и то, что умеет процессор 80386, под силу и процессорам 80486, Pentium, Pentium III, Pentium 4 и т. д.

Вторая директива .model flat, stdcall показывает, под управлением какой О.С. будет работать программа. Директива .model flat указывает компилятору, что нужно генерировать код для защищённого режима работы процессора, а параметр stdcall позволяет вызывать в программе функции системы MS Windows (точнее, параметр stdcall определяет порядок передачи параметров вызываемым процедурам).

Две следующие директивы тесно связаны с вызовом системной процедуры ExitProcess:

includelib \myasm\lib\kerne132.lib

ExitProcess proto :DWORD

Строка includelib \myasm\lib\kernel32.lib подключает к ассемблерному тексту файл библиотеки kernel32.fib, содержащий сведения о процедурах операционной системы, необходимые для их правиль­ного вызова. Есть там и описание процедуры ExitProcess. Символы \myasm\lib\ указывают компилятору путь к файлу библиотеки.

Директива proto, стоящая в строке ExitProcess proto :DWORD кратко описывает параметры процедуры ExitProcess. Ассемблер переведет программу на язык процессора, когда параметры, описан­ные директивой proto, соответствуют описанию процедуры в библиотеке, а так­же параметрам, указанным при вызове процедуры директивой invoke.

Следующая директива .code помечает сами инструкции. Процессор начнет вы­полнять первой ту команду, перед которой стоит метка, указанная в самом конце программы ди­рективой end <метка>. Эта директива, а также сама метка никак не переводятся в инструкции ассемблера, а лишь помогают компилятору получить программу, которую спо­собен выполнить процессор. В нашем случае директива end указывает ассемблеру, что перед первой инструк­цией программы стоит метка start:. Значит, процессор выполнит вначале инст­рукцию mov eax,2, затем перейдет к следующей (у нас это add еах,З) и т.д.

*Процедура ExitProcess вызывается в каждой ассемблерной программе для передачи управления Windows (ExitProcess - это функция системы Windows, которая завершает выполнение текущей программы, называемой процессом).* Вызов процедуры ExitProcess выглядит так:

invoke ExitProcess, 0

Единственный передаваемый ей параметр - число 0 отделено от имени про­цедуры запятой. *При выполнении программы под управлением MS DOS для её завершения используют функцию 4Ch int21h.*

Далее рассмотрим пример программы нахождения простых чисел. Она состоит из двух циклов: внешний перебирает проверяемые числа, а внутренний - осуществляет саму проверку. Вывод простых чисел на экран в программе не предусмотрен.

.386

.model flat, stdcall

option casemap:none

include \myasm\include\kernel32.inc

includelib \myasm\lib\kernel32.lib

SSIZE equ 1000

.data?

PrimeNumbers DWORD SSIZE dup(?)

.code

start:

mov ebx, 3 ;первое проверяемое число = 3

mov edi, 0 ;нулевой элемент массива

mov ebp, 0 ;счетчик простых чисел = 0

nxtdig:

mov edx, 0 ;готовим число edx:eax

mov eax, ebx ;к проверке

mov ecx, ebx ;число проверок меньше

sub ecx, 2 ;проверяемого числа на 2

mov esi, 2 ;первый делитель = 2

nxtpr:

div esi ;делим число edx:eax на esi

cmp edx, 0 ;остаток = 0 ?

jz skip ;да - идем к след. проверке

mov edx, 0 ;нет

mov eax, ebx ;восстанавливаем edx:eax

inc esi ;и делим на следующее число

loop nxtpr ;есть на что делить - продолжим

mov PrimeNumbers[edi], ebx ; нет - число простое

inc ebp ;увел. счетчик прост, чисел

cmp ebp, SSIZE ;все простые числа найдены?

jz done ;да - уходим

add edi.4 ;нет - след. элемент массива

skip:

inc ebx ; проверяем

jmp nxtdig ; след. число

done:

invoke ExitProcess, 0

end start

В тексте этой программы есть новые директивы.

Директива option casemap:none приказывает компилятору различать строчные и прописные буквы в именах вызываемых процедур. Описание директивы .data? - см. раздел «Создание консольных Windows-приложений». В программе не объявлен прототип процедуры ExitProcess, на самом же деле все прототипы собраны в подключаемом файле kernel32.inc (см. описание шаблона программы).

**Терминальные функции Win32 API**

***В приложении приведен полный список терминальных функций Win32 и их краткое описание.*** Практически во всех терминальных функциях системы Win32 в качестве первою па­раметра нужно указывать некоторый дескриптор. Дескриптор (handle) - это 32-разрядное целое число без знака, которое уникальным образом идентифицирует некото­рый объект в системе, например, растровое изображение, пишущий узел или любое уст­ройство ввода-вывода. По умолчанию стандартный дескриптор ввода связан с клавиатурой, стандартный дескриптор вывода - с экраном. Получить стандартный дескриптор ввода-вывода можно с помощью функции GetStdHandle. На вход функции GetStdHandle должно быть подано одно из следующих значений:

STD\_INPUT\_HANDLE EQU -10 ; Стандартное устройство ввода

STD\_OUTPUT\_HANDLE EQU -11 ; Стандартное устройство вывода

STD\_ERROR\_ HANDLE EQU -12 ; Стандартное устройство для вывода сообщений об ошибках

Если в программе, написанной для Win32, планируется что-то выводить на экран, сначала нужно вызвать функцию Windows GetStdHandle и определить с ее помощью дескриптор (целое число) стандартного устройства вывода, как показано ниже:

.data

consoleHandle DWORD ?

.code

INVOKE GetStdHandle, STD\_OUTPUT\_HANDLE

Mov consoleHandle, eax

Для установки курсора в нужную позицию на экране нужно вызвать функцию систе­мы Windows SetConsoleCursorPosition. В качестве параметров ей передаются деск­риптор стандартного устройства вывода и адрес структурной переменной типа COORD, содержащей значения координат X и Y символа на экране, где будет нахо­диться курсор:

.data

XYPos COORD <10,5>

.code

INVOKE SetConsoleCursorPosition, consoleHandle, XYPos

***Необходимо знать***, что при запуске консольных Windows-приложений из командной строки, например из Far'a, предполагаемое в программе сообщение будет выводиться в консоль Far'a. При запуске программы как Windows-приложения консольное окно появляется лишь на секунду. Это связано с тем, что консольные приложения могут создать свою консоль. В этом случае весь ввод-вывод будет производиться в эту консоль. Если же приложение консоль не создает, то здесь может возникнуть двоякая ситуация: либо наследуется консоль, в которой программа была запущена, либо Windows создает для приложения свою консоль. Работать с чужой консолью не всегда удобно, а для того чтобы создать свою консоль, используется функция AllocConsole. Для вызова функции AllocConsole не требуется никаких параметров. По завершении программы все выделенные консоли автоматически освобождаются. Однако это можно сделать и принудительно, используя функцию FreeConsole. Следует отметить, что при запуске программы в "чужой" консоли она наследует эту консоль, поэтому, пока мы не выполним функцию FreeConsole, новой консоли не создать - чужой консоли эта функция закрыть не может.

**Чтение данных с терминала**

***Входной буфер терминала****.* В системе Win32 каждому терминалу назначается входной буфер, реализованный в виде массива записей, каждая из которых содержит информа­цию об одном событии, поступившем от какого-либо устройства ввода. При наступлении события ввода, такого как нажатие на клавишу, перемещение мыши или щелчке кнопкой мыши, во входном буфере терминала создается соответствующая запись. При использо­вании функций высокого уровня, таких как ReadConsole, эти записи отфильтровыва­ются и вызвавшей программе возвращается только поток символов.

**Функция ReadConsole**

Эта функция представляет собой довольно удобное средство для чтения введенного с терминала текста и помещения его в буфер. Вот ее прототип:

ReadConsole PROTO,

handle:DWORD, ; Дескриптор устройства ввода

pBuffer:PTR BYTE, ; Адрес буфера

maxBytes:DWORD, ; Максимальное число вводимых символов

pBytesRead:PTR DWORD, ; Адрес переменной, в которую

; помещается реальное количество

; прочитанных байтов.

notUsed:DWORD ; Зарезервировано

Вместо параметра handle в функцию нужно передать один из действующих дескрип­торов устройств ввода с терминала, возвращаемых функцией GetStdHandle. Вместо па­раметра pBuffer подставляется адрес массива символов. Параметр maxBytes является 32-разрядным целым числом и определяет максимальное число вводимых символов. Вместо параметра pBytesRead подставляется адрес 32-разрядной переменной, в кото­рую функция записывает реальное количество символов, помешенных в буфер. Послед­ний параметр функции не используется, должен быть равен нулю.

**Вывод на терминал**

В ряде терминальных функций Win32 используются заранее определенные структуры данных, такие как COORD или SMALL\_RECT. С помощью структуры COORD в терминаль­ные функции передаются Х- и Y-координаты позиции символа на экране, которые по умолчанию находятся в пределах 0-79 и 0-24 соответственно:

COORD STRUCT

X WORD ?

Y WORD ?

COORD ENDS

Структура SMALL\_RECT позволяет определить на экране терминала прямоугольный символьный блок, координаты которого задаются в позициях символов:

SMALL\_RECT STRUCT

Left WORD ?

Top WORD ?

Right WORD ?

Bottom WORD ?

SMALL\_RECT ENDS

**Функция WriteConsole**

Эта функция позволяет вывести строку символов на экран терминала, которая опре­деляется указанным дескриптором устройства стандартного вывода. Она проста в ис­пользовании и обрабатывает стандартные управляющие ASCII-символы, такие как *табу­ляция, возврат каретки и перевод строки*. Ниже приведен прототип функции:

WriteConsole PROTO,

handle:DWORD, ; Дескриптор устройства стандартного вывода

pBuffer:PTR BYTE, ; Адрес буфера

bufsize:DWORD, ; Размер буфера

pCount:PTR DWORD, ; Адрес числа выведенных байтов

lpReserved:DWORD ; Зарезервировано

Параметры функции WriteConsole (слева направо) имеют следующий смысл.

1-й параметр - дескриптор буфера вывода консоли, который может быть получен при помощи функции GetStdHandle. 2-й параметр - указатель на буфер, где находится выводимый текст. 3-й параметр - количество выводимых символов. 4-й параметр - указывает на переменную DWORD, куда будет помещено количество действительно выведенных символов. 5-й параметр - резервный параметр, должен быть равен нулю.

Установить цвет выводимых букв можно с помощью функции SetConsoleTextAttribute. Первым параметром этой функции является дескриптор выходного буфера консоли, а вторым - цвет букв и фона. Цвет получается путем комбинации (сумма или операция "ИЛИ") двух или более из представленных ниже констант. Причем возможна "смесь" не только цвета и интенсивности, но и цветов.

FOREGROUND\_BLUE equ 1h ; синий цвет букв

FOREGROUND\_GREEN equ 2h ; зеленый цвет букв

FOREGROUND\_RED equ 4h ; красный цвет букв

FOREGROUND\_INTENSITY equ 8h ; повышенная интенсивность

BACKGROUND\_BLUE equ 10h ; синий свет фона

BACKGROUND\_GREEN equ 20h ; зеленый цвет фона

BACKGROUND\_RED equ 40h ; красный цвет фона

BACKGROUND\_INTENSITY equ 80h ; повышенная интенсивность

Для определения заголовка окна консоли используется функция SetConsoleTitle, единственным параметром которой является адрес строки с нулем на конце. Здесь следует оговорить следующее: если для вывода в само окно консоли требуется DOS-кодировка, то для установки заголовка требуется Windows-кодировка. Эту проблему можно решить средствами Windows. Существует специальная функция CharToOem. Первым параметром этой функции является указатель на строку, которую следует перекодировать, а вторым параметром - на строку, куда следует поместить результат. Причем поместить результат можно и в строку, которую перекодируем.

*Следует заметить, что для большинства консольных функций характерно то, что при правильном их завершении возвращается ненулевое значение. В случае ошибки в EAX помещается ноль.*

Пример минимальной программы консольного Windows-приложения (*директива invoke не используется).*

.486

.model flat, STDCALL

include WindowConA.inc

extrn AllocConsole: PROC

extrn SetConsoleTitleA: PROC

extrn ExitProcess: PROC

.data

TitleText db 'Win32-console application',0

.code

start proc near ; точка входа в программу:

;запрос консоли

call AllocConsole

test eax.eax ; проверить успех запроса консоли

jz exit ; неудача

;выведем заголовок окна консоли SetConsoleTitie:

push offset TitleText

call SetConsoleTitleA

test eax.eax ;проверить успех вывода заголовка

jz exit ;неудача

;работаем ...

exit: ; выход из приложения, готовим вызов ExitProcess

push 0

call ExitProcess

start endp

end start

Если убрать комментарии, то кода будет совсем немного. В нем представлены вызовы трех функций: AllocConsole, SetConsoleTitle, ExitProcess.

Первой функцией консольного приложения должна быть функция запроса консоли AllocConsole: BOOL AllocConsole (VOID). Для вызова функции AllocConsole не требуется никаких параметров. В случае успеха функция AllocConsole возвращает ненулевое значение, при неудаче - нуль. Выделенная консоль представляет собой типичное для Windows окно.

Процесс в конкретный момент времени может использовать одну консоль. Если ему нужно запустить еще одну консоль, то прежняя должна быть закрыта или освобождена с помощью функции FreeConsole: BOOL FreeConsole(VOID). В случае успеха функция FreeConsole возвращает ненулевое значение, при неудаче - нуль. При завершении процесса выделенная процессу консоль освобождается автоматически.

В нашем случае использован именно этот вариант закрытия консоли - функцией ExitProcess: VOID ExitProcess (UINT uExitCode). Функции ExitProcess передается код завершения процесса и всех завершаемых цепочек в этом процессе. Проанализировать этот код можно с помощью функций GetExitCodeProcess и GetExitCodeThread. В общем случае в различных ветвях кода может быть несколько точек выхода с вызовом функции ExitProcess. Задавая различные значения кода завершения, можно идентифицировать причину завершения процесса.

Окно консоли может иметь заголовок, для отображения которого предназначена функция SetConsoleTitle: BOOL SetConsoleTitle (UPCTSTR IpConsoleTitie). У функции SetConsoleTitle один параметр - указатель на строку с заголовком консоли, заканчивающуюся нулем.

В литературе по платформе Win32 часто встречаются такие понятия, как процесс и поток. Процесс (приложение) представляет собой экземпляр программы, загруженной в память для выполнения. Процесс инертен, он просто владеет пространством памяти в 4 Гбайт. В этом пространстве содержатся код и данные, другие ресурсы, загружаемые в адресное пространство процесса. В качестве ресурсов могут быть, в свою очередь, исполняемые файлы или библиотеки DLL. Для того чтобы процесс исполнялся, в нем должен быть создан поток, который, собственно, и отвечает за исполнение кода, содержащегося в адресном пространстве процесса.

В приведенной ниже программе продемонстрирована работа с функциями GetStdHandle, ExitProcess и WriteConsole на примере вывода строки символов на экран терминала.

В этой программе вызываются следующие терминальные функции Win32:

GetStdHandle, ExitProcess, WriteConsole

INCLUDE Irvine32.inc ; подключить файлы с прототипами и библиотеками

.data

endl EQU <0dh,0ah> ; Признак конца строки

message \

BYTE "------------------- Consolel.asm --------------“

BYTE endl,endl

BYTE "Это простая демонстрационная программа,

которая выводит ",endl

BYTE "текст на терминал с помощью прямого вызова

функций GetStdHandle и ",endl

BYTE "WriteConsole системы Win32.",endl

BYTE "------------------------------------------------“

BYTE endl,endl,endl

messageSize =($-message)

consoleHandle DWORD 0 ; Дескриптор стандартного устройства вывода

bytesWritten DWORD ? ; Количество реально записанных байтов

.code

main PROC

;Определим дескриптор стандартного устройства вывода

INVOKE GetStdHandle, STD\_OUTPUT\_HANDLE

mov consoleHandle,eax

;Выведем строку на терминал

INVOKE WriteConsole,

consoleHandle, ; Дескриптор вывода на терминал

ADDR message, ; Адрес строки

messageSize, ; Длина строки

ADDR bytesWritten, 0 ; Адрес переменной, содержащей количество реально символов зарезервировано

; Завершим программу

INVOKE ExitProcess,0

main ENDP

END main

Программа выведет на экран следующий текст:

-------------------------------- Consolel.asm ---------------------------------------

Это простая демонстрационная программа, которая выводит текст на терминал с помощью прямого вызова функций GetStdHandle и WriteConsole системы Win32.

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

**Примечание**.

* + - 1. Для компиляции необходимы библиотеки и компилятор из файла Masm615 [17].

1. Программа не создаёт своё окно.

***В заключении следует напомнить***, что на сегодняшний день доступно достаточно много средств, позволяющих разрабаты­вать приложения под Windows на языке ассемблер. Существуют специализированные интегрированные системы разработки, автоматизирующие этот процесс. Кроме, непосредственно, автоматического получения испол­няемого модуля, подобные системы обладают рядом полезных свойств, таких, как настраи­ваемая подсветка синтаксиса, поддержка различных паке­тов ассемблера, возможность свертки процедур и других блочных конструкций при навигации по исходному коду, редактирование ресурсов. Вот некоторые из извест­ных интегрированных сред разработки, поддерживающие разработку программ на языке ассемблер: RadASM, WinASM Studio, Negatory Assembly Studio, Visual SlickEdit, Chrome.

Для системы WinAsm Studio есть описание на русском языке.

## 2.2. Программирование контроллера прерываний 8259А и организация прерываний в IBM PC

**2.2.1 Механизм прерываний**

Прерывание – это аппаратная функция, вызывающая приостановку операций ЦП, запоминание его состояния и выполнение специальной программы, которая называется программой обработки прерывания или обработчиком прерывания (ISR).

В настоящее время сложно представить себе вычислительную машину, не обладающую механизмом прерываний, на основе которого организуется од­новременная работа с несколькими периферийными устройствами, а также мультипрограммный режим работы. Персональный компьютер не является исключением. Нормальное выполнение программы в микропроцессорах Intel 80XX86 может быть прервано для обработки внешних событий или особых случаев. Обычно говорят, что при выполнении программы могут иметь место внешние и внутренние прерывания, подразумевая тем самым, что внешние прерывания информируют о внешних событиях по отношению к процессору (асинхронных событиях), а внутренние прерывания обнаруживаются самим процессором при выполнении команд.

Существуют два источника внешних прерываний и два источника внут­ренних прерываний. ***Внешние прерывания*** могут иметь место в следующих случаях:

* на входную линию INTR процессора поступает запрос на обработку прерывания от внешнего устройства (маскируемые внешние прерывания);
* обнаружена серьезная ошибка или ситуация, требующая немедленного вмешательства процессора, о чем он извещается по входной линии NMI (немаскируемое прерывание).

***Внутренние прерывания*** возникают либо в случае обнаружения процессо­ром некорректных или особых условий выполнения команд (например, деление на ноль или выполнение очередной команды в пошаговом режиме), либо при выполнении специальных инструкций «программных прерываний» (INT, INTO, Bound).

С помощью программных прерываний осуществляется доступ к системным средствам обслуживания аппаратуры компьютера и вычислительного процесса; аппаратные прерывания от периферийного оборудования компьютера позволяют процессору мгновенно реагировать на такие события, как срабатывание таймера, перемещение мыши или нажатие клавиши на клавиатуре; внутренние прерывания помогают обрабатывать ошибки вроде деления на нуль или отказа питания.

При возникновении прерывания, независимо от того, происходит оно в самом микропроцессоре, вырабатывается контроллером прерываний или командой int, выполняются одни и те же действия, называемые последовательностью прерывания. Если данное прерывание разрешено, то процессор завершает текущую команду, помещает в стек содержимое регистров флагов, CS и IP, после чего сбрасывает флаги IF и TF (запрещает аппаратные прерывания и режим трассировки) и вычисляет адрес соответствующего вектора прерывания. Векторы прерываний образуют таблицу размером 1К, расположенную в начале ОЗУ. Каждый вектор представляет собой два 16-разрядных слова, размещенных в соседних ячейках памяти, причем в ячейке с меньшим адресом (четным) содержится смещение, а в следующей ячейке – сегментный адрес обработчика прерывания. Таким образом, вектор прерывания с номером N находится по адресу 0000:4\*N. Первое слово вектора прерывания помещается в IP, второе – в CS, при этом управление передается программе, адрес которой содержится в векторе прерывания. По завершении обработки прерывания из стека в обратном порядке извлекается содержимое регистров IP, CS и флагов, и управление возвращается прерванной программе.

В тех же случаях, когда в прикладной программе необходимо обрабатывать прерыва­ния от аппаратуры или программные прерывания пользователя, программисту прихо­дится создавать собственные обработчики прерываний. Общие принципы построения обработчиков аппаратных и программных прерываний одинаковы, однако обработка аппаратных прерываний требует учета целого ряда дополнительных соображений [2,8,9].

Обработчик прерывания похож на обычную программную процедуру с учетом следующих особенностей:

1. обработчик всегда имеет атрибут far;
2. при вызове обработчика сохраняется не только адрес возврата, но и регистр флагов;
3. возврат в прерванную программу осуществляется инструкцией iret, а не ret, как в случае обычных процедур. Если не нужно восстанавливать регистр флагов (например, при возврате значений в этом регистре), то возможно использование команды ret 2.

«Перехватить» прерывание – значит изменить его вектор так, чтобы он указывал на ваш собственный обработчик. Перед этим следует прочитать и сохранить старый вектор на тот случай, если потребуется обратиться по адресу исходного обработчика или восстановить прежнее значение вектора.

Перед началом чтения или изменения вектора необходимо запретить аппаратные прерывания, чтобы исключить возможность модификации вектора в процессе чтения или возможность передачи управления по частично измененному вектору. Функции 25H и 35H обеспечивают это автоматически.

**… Контроллер прерываний и его взаимодействие с процессором**

Для управления внешними прерываниями в компьютере используется программируемый контроллер прерываний типа Intel 8259A. Контроллер выполняет функцию арбитра на основе приоритетов одновременных прерываний, сигнализирует процессору о внешнем прерывании по входной линии INTR и передает в процессор номер внешнего прерывания.

На рисунке 1 приведена структурная схема контроллера прерываний Intel 8259A. Ниже описано назначение основных элементов контроллера прерываний [11].

*Регистр запроса прерываний* (Interrupt Request Register -IRR). На этот регистр поступают прерывания по линиям запроса прерывания IR (Interrupt Request). При этом каждой линии IR0-IR7 соответствует один бит. Регистр предназначен для хранения всех поступивших запросов, так как в общем случае одновременно может быть запрошено более одного прерывания.

*Блок выбора приоритета* (Priority Resolver). Определяет приоритеты за­просов, проставленных в регистре IRR, и во время импульса INТА перено­сит бит, соответствующий прерыванию с максимальным приоритетом, в ре­гистр ISR.

*Регистр маски прерывания* (Interrupt Mask Register - IMR). Данный регистр управляет регистром IRR. Он позволяет запретить не все, а лишь отдельные системные прерывания записью единицы в соответствующий разряд.

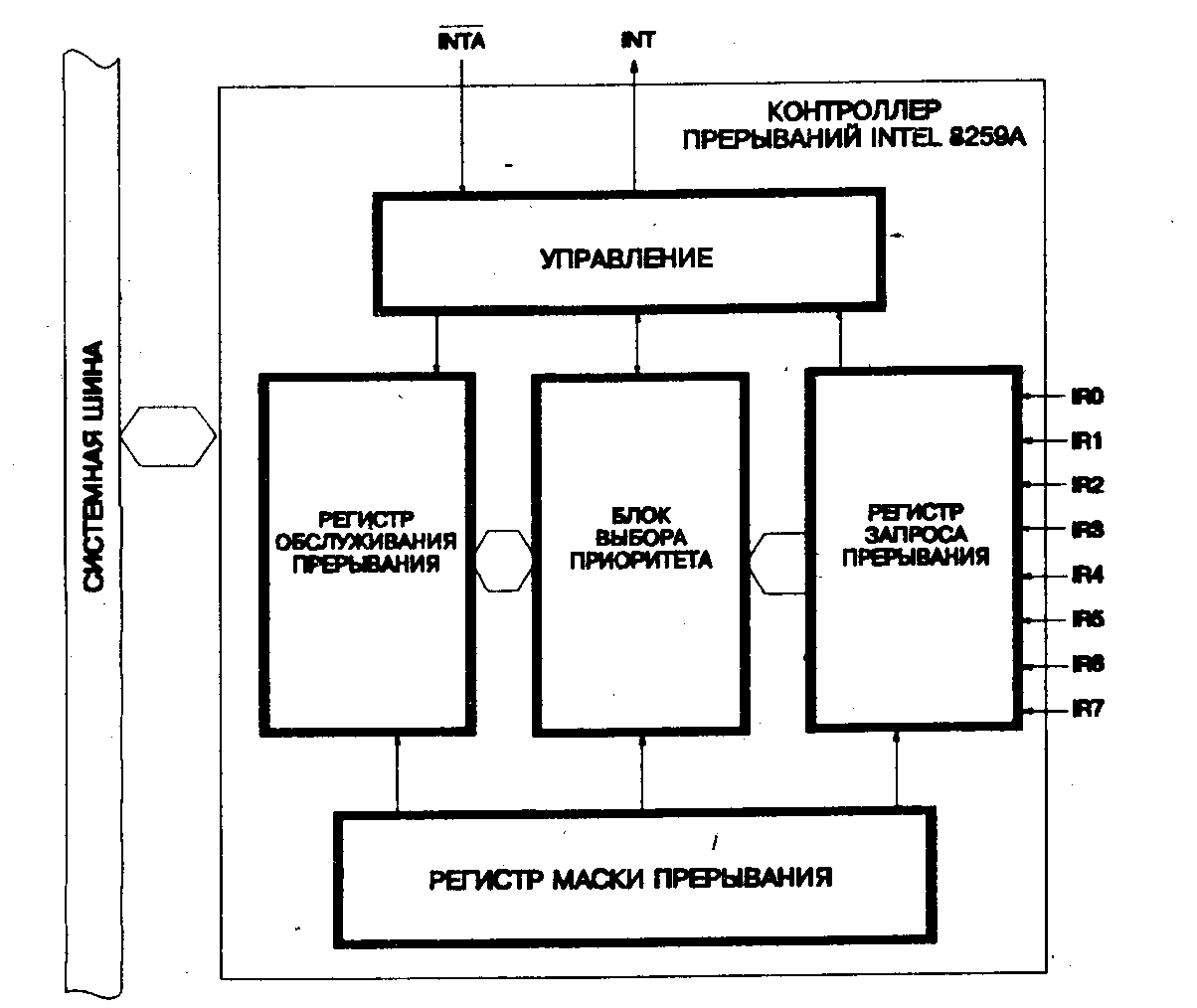


Рисунок 1 - Структурная схема контроллера прерываний Intel 8259A

*Регистр обслуживаемых запросов прерывания*  (In-Service Register -ISR). Этот регистр предназначен для хранения тех прерываний, которые обрабатываются контроллером. Регистр подсоединен каскадно к регистру IRR, при этом каждый бит 8-ми битового регистра соответствует 8 линиям прерываний.

*Сигнал запроса прерывания* (Interrupt - INT) поступает прямо на линию микропроцессора INTR и собственно запрашивает обработку прерывания у процессора.

*Сигнал подтверждения прерывания* (Interrupt Acknowledge - INTA) является подтверждением со стороны процессора получения им запроса INT и разрешает контроллеру передачу номера прерывания на шину данных.

Ниже приведена последовательность событий, возникающих при обработке прерывания от внешнего устройства.

На одной или нескольких линиях запроса прерывания IR появляется единичный сигнал, и в регистре IRR устанавливаются в 1 соответствующие биты;

1. контроллер принимает запрос, посылая в процессор сигнал INT;
2. процессор подтверждает получение запроса INT с помощью сигнала INTA; при получении сигнала INTA от процессора в регистре ISR устанавливается бит, соответствующий самому приоритетному биту, установленному в регистре IRR на шаге 1. После установки в ISR соответствующий бит в IRR обнуляется;
3. процессор генерирует второй сигнал INTA. Пока данный сигнал активен, контроллер прерываний выставляет на шину данных 8-ми битовое значение номера прерывания, которое задается и хранится в контроллере;
4. цикл прерывания завершается. В режиме AEOI взведенный в ISR бит сбрасывается по окончанию второго импульса INTA. Иначе, бит остается взведенным до тех пор, пока не будет послана команда EOI из обработчика прерывания.

В состав компьютеров IBM PC и IBM PC/XT входит один контроллер прерываний, однако контроллеры Intel 8259A в общем случае допускают каскадное подключение, т.е. сигнал запроса прерывания INT одного кон­троллера прерываний может поступать не в процессор, а на входную линию другого контроллера. При этом первый контроллер называется ведомым (slave), а второй - ведущим (master) или главным. Таким образом, к каждой входной линии ведущего контроллера может быть подключен один ведомый контроллер, при этом общее число входных линий увеличивается до 64.

Стандартная схема подключения внешних устройств к входным линиям ве­домого и ведущего контроллера для персональных компьютеров IBM PC/AT приводится в таблице 1 [11].

**.. Программирование контроллера прерываний.**

Рассмотрим программирование контроллера прерываний в том объеме, в котором это может оказаться необходимым для написания программ для компьютеров фирмы IBM. Описание контроллера дается с некоторыми ограничениями. Причиной этих ограничений служит то, что в современных компьютерах уже не используется микросхема Intel 8259A в чистом виде, а применяются интегрированные шинные комплекты, в состав которых входит и контроллер прерываний. При этом многие функции контроллера Intel 8259A (например, обеспечение микропроцессорных комплектов Intel 8080 и Intel 8085) в них не реализованы.

При программировании контроллеров прерывании Intel 8259А используются порты ввода-вывода с адресами 20h; 21h – для ведущего и А0h; А1h – для ведомого.

Запись определенных значений в регистры этих портов называется выдачей команд контроллера прерываний. Существуют два типа команд контроллера прерываний Intel 8259A:

* команды инициализации (Initialization Command Words -ICW),
* команды операций (Operation Command Words - OCW).

Команды инициализации служат для определения режима работы контроллера: ведущий/ведомый (master/slave) для тех конфигураций, в которых используются два контроллера прерываний и для задания начального базового вектора прерываний. Команды операций используются для изменения состояния контроллера прерываний и для получения информации о его состоянии.

**Команды инициализации контроллера прерываний**

Для контроллера прерываний Intel 8259A определены команды инициализации ICW1, ICW2, ICW3 и ICW4. При операции записи в регистр 20h байта со значением бита 4, равным 1, контроллер переходит в состояние инициализации и интерпретирует этот байт как ICW1. Порядок инициализации контроллера прерываний приведен на рисунке 2. После перехода в состояние инициализации контроллер находится в этом состоянии до тех пор, пока не будут выданы все необходимые команды инициализации. По завершении процесса инициализации регистр маски прерывания будет обнулен и установлен режим чтения регистра IRR.

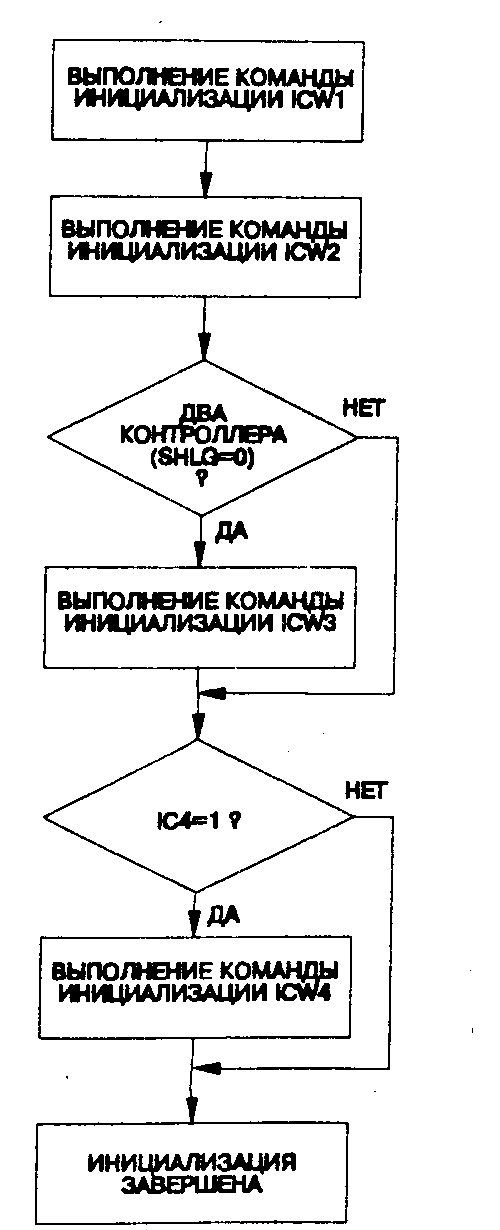


Рисунок 2 - Инициализация контроллера прерываний

Таблица 1

**Стандартное подключение внешних устройств**

| **Уровень** | **№ прерывания** | **Устройство** |
| --- | --- | --- |
| IR0 | 08h | Канал 0 таймера |
| IR1 | 09h | Клавиатура |
| IR2 | 0Ah | Вход ведомого контроллера |
|  |  | Линии IR ведомого контроллера прерываний |
| IR8  IR9  IR10  IR11  IR12  IR13  IR14  IR15 | 70h  71h  72h  73h  74h  75h  76 h  77 h | Часы реального времени  Перенаправление линии IR2  Зарезервировано  Зарезервировано  Мышь (PS/2); зарезервировано (IBM PC/AT)  Внутреннее прерывание сопроцессора  Контроллер винчестера  Зарезервировано |
| IR3 | 0Bh | Последовательный порт 2 |
| IR4 | 0Ch | Последовательный порт 1 |
| IR5 | 0Dh | Параллельный порт 2 |
| IR6 | 0Eh | Контроллер НГМД |
| IR7 | 0Fh | Параллельный порт 1 |

**Примеры обработки аппаратных прерываний**

В ПК таймер используется как источник перио­дических прерываний. Напомним, что при настройке по умолчанию на выходе нулевого канала таймера возникают сигналы с частотой 18,2 Гц. Эти сигналы возбуждают прерывания на линии IRQ0 с вектором 08h, которые в нормальном режиме работы компьютера обрабатываются программой BIOS, осуществляющей отсчет текущего времени. С помощью прерываний от таймера мож­но периодически выводить на экран некоторую информацию, отсчитывать интервалы времени, управлять в реальном времени аппаратурой, подключаемой к компьютеру, и т. д. Рассмотрим простой обработчик прерываний от системного таймера.

Задачей обработчика прерываний от таймера является отсчет за­данного интервала времени (конкретно - 3с) и по истечении этого интервала пере­ключение основной программы на другую ветвь ее выполнения. В результате програм­ма в течение 3с выполняет одну работу, а по истечении 3с переключается на выпол­нение других действий. В рассматриваемом примере программа в течение 3с просто крутится в цикле, а через 3с выходит из этого цикла и, выведя на экран предупреж­дающее сообщение, завершается. В реальном случае программа могла бы, включив не­которую измерительную аппаратуру, в течение заданного интервала времени выводить на экран информацию, получаемую из этой аппаратуры, а по истечении интервала времени выключить аппаратуру и приступить к обработке накопленных данных.

Измерение интервала времени осуществляется путем подсчета числа поступивших прерываний. Интервал 3с соответствует приблизительно 54 прерываниям; соответственно на первые 53 прерывания обработчик прерываний просто завершается, а на 54 выполняет переключение ветвей основной программы.

**Пример1**:Обработчик прерываний от таймера.

|  |
| --- |
| main proc |
| mov AX,data ;Инициализация |
| mov DS,AX ;сегментного регистра DS |
| ;Прочитаем и сохраним исходное-содержимое вектора 8 |
| mov AX,3508h |
| int 21h |
| mov word ptr old\_08h,BX |
| mov word ptr old\_08h+2,ES |
| ; Установим наш обработчик прерываний new\_08h |
| mov AX,2508h |
| mov DX,offset new\_08h |
| push DS ;Сохраним на время DS |
| push CS ;Отправим содержимое CS |
| pop DS ; в DS |
| int 21h ;Вызов DOS (функции 25h) |

|  |
| --- |
| pop DS ;Восстановим DS |
| ;Сымитируем действия, выполняемые в течение 3 с, просто зациклив программу |
| stop: jmp short stop |
| ; Вторая ветвь программы, активизируемая по истечении 3 с. |
| fin:mov AH,09h ;Выведем на экран сообщение |
| mov DX,offset msg |
| int 21h |
| mov AX,2508h ;Восстановим вектор 8 |
| lds DX,old\_08h |
| int 21h |
| mov AX,4C00h ; Завершим программу |
| int 21h |
| main endp |

|  |
| --- |
| ;Прикладной обработчик прерываний от таймера, активизируемый 18,2 раза в секунду |
| new\_08h proc far |

|  |  |
| --- | --- |
| push AX | ;Сохраним два используемых |
| push ВР | ;в обработчике регистра |
| dec CS:time | ;Декремент интервала времени |
| jnz outint | ;Пока не 0, выйти из прерывания. |
| ;Содержимое ячейки time уменьшилось до 0, выполнить переключение программы | |
| mov BP,SP | ;ВР=текущая вершина стека |
| mov AX,offset fin | ;Смещение точки перехода |
| mov [ВР+4],АХ | ;Отправим его в стек на место IP |
| mov AX,seg fin | ;Сегмент точки перехода |
| mov [ВР+6],АХ | ;Отправим его в стек на место CS |
| outint: mov AL,20h | ;Команда EOI в контроллер |
| out 20h,AL | ;прерываний |
| pop ВР | ;Восстановим оба |
| pop AX | ;сохраненных регистра |
| iret | ;Выход из прерывания |
| time dw 54 | ;Ячейка для отсчета времени |
| new\_08h endp | ;Поля данных |
| ; В сегменте данных | |
| old\_08h dd 0 | ;Ячейка для хранения исходного вектора |
| msg db 'Временной интервал истек!$' | ;Предупреждающее сообщение |

C начала с помощью функции 35h читается и сохраняется в ячейке old\_08h исходное содержимое вектора 8, затем функцией 25h в вектор заносится полный адрес приклад­ного обработчика new\_08h. После этого программа приступает к действиям, заплани­рованным для выполнения в течение 3-секундного интервала (в данном случае - цикл перехода на метку stop).

Процедура new\_08h обработчика прерываний весьма проста. После сохранения в стеке регистров АХ и ВР (рисунок 3), используемых в программе обработчика, выполняется дек­ремент ячейки time, исходное содержимое которой определяет измеряемый интервал времени. Эта ячейка расположена фактически в процедуре обработчика и обращение к ней требует замены сегмента (DS на CS). Если результат, не достиг нуля, командой jnz осуществляется переход на метку outint и завершение обработки данного прерывания. В процедуру завершения входит генерация команды EOI, посылаемой в ведущий кон­троллер прерываний, восстановление сохраненных ранее регистров и команда выхода из прерывания iret.

**Флаги**

**CS**

**IP**

**BP**

**AX**

SP на момент прерывания

SP при входе в обработчик

[BP+4]=offset fin

[BP+6]=seg fin

Рисунок 3 - Состояние стека в обработчике прерывания от таймера

При обработке 54-го по счёту прерывания команда dec time фиксирует нулевой результат, в результате чего команда условного перехода jnz не срабатывает, и выполняются сле­дующие строки обработчика. Регистр ВР настраивается на текущую вершину стека, и на место находящегося в стеке двухсловного адреса возврата в основную программу записывается двухсловный адрес ячейки fin, на которую мы хотим переключить ос­новную программу по истечении 3 с. Смещение на 4 байта относительно вершины сте­ка понадобилось из-за того, что в начале программы обработчика мы сохранили в сте­ке содержимое двух регистров.

При поступ­лении прерывания от таймера процессор сохраняет в стеке в качестве адреса возврата текущий адрес прерванной программы DOS. Заменив его адресом точки stop нашей программы, мы после выхода из прерывания уже не вернемся в DOS, работа ко­торой по выполнению затребованной функции останется незавершенной. Это приве­дет к выходу системы из строя и необходимости перезагрузки машины. Именно по­этому в примере останов программы в ожидании прихода 54-го прерывания от таймера выполнен с помощью бесконечного цикла **stop: jmp stop**, а не путем вызова, например, функции 01h DOS.

**Второй пример** показывает, как можно обработать прерывание от клавиатуры. Данная программа перехватывает прерывание от клавиатуры, очищает экран и входит в бесконечный цикл, выводя скан – коды нажимаемых клавиш в шестнадцатеричном виде. Для прекращения работы программы нужно нажать кнопку сброса Reset (перезагрузить систему).

*Пример 2: Обработчик прерываний от клавиатуры*

|  |  |
| --- | --- |
| .MODEL TINY |  |
| .CODE |  |
| ORG 100H |  |
| Entry: jmp Set\_handler |  |
| Type\_hex PROC near |  |
| ; Выводит символ из регистра AL на экран в позицию, определяемую регистрами DH и DL, и увеличивает значение DX | |
| push ax |  |
| mov ah,2 |  |
| int 10H |  |
| pop ax |  |
| mov ah,9 |  |
| int 10H |  |
| inc dx |  |
| ret |  |
| Type\_hex ENDP |  |
| Int\_09H PROC far |  |
| push ax | ; Сохранить регистры |
| push bx |  |
| push cx |  |
| push dx |  |
| push di |  |
| push si |  |
| push bp |  |
| in al,60H | ;Ввести скан-код нажатой клавиши |
| mov ah,al | ;Преобразовать в скан-код |
| and ax,0F00FH | ; в два символа ASCII, представляющих |
| mov cl,4 | ; его в шестнадцатеричном виде |
| shr ah,cl |  |
| add ax,3030H |  |
| cmp al,58 |  |
| jb Next\_test |  |
| add al,7 |  |
| Next\_test: |  |
| cmp ah,58 |  |
| jb Test\_position |  |
| add ah,7 |  |
| Test\_position: |  |
| mov dx, word PTR cs:Position | ; Если экран заполнен, то очистить его |
| cmp dh,25 |  |
| jb Type\_code |  |
| push ax |  |
| mov ax,03H |  |
| int 10H |  |
| xor dx,dx |  |
| pop ax |  |
| Type\_code: |  |
| mov bx,0FH | ;Ввести скан-код |
| mov cx,1 |  |
| push ax |  |
| xchg ah,al |  |
| call Type\_hex |  |
| pop ax |  |
| call Type\_hex |  |
| mov al,’H’ |  |
| call Type\_hex |  |
| mov al,’ ’ |  |
| call Type\_hex |  |
| cmp dl,80 | ;Если закончилась строка, то перейти |
| jb Save\_position | ; к следующей |
| inc dh |  |
| xor dl,dl |  |
| Save\_position: |  |
| mov word PTR cs:Position, dx | ;Обработать аппаратное прерывание |
| in al,61H |  |
| push ax |  |
| or al,80H |  |
| out 61H,al |  |
| pop ax |  |
| out 61H,al |  |
| mov al,20H |  |
| out 20H,al |  |
| pop bp | ; Восстановить регистры |
| pop si |  |
| pop di |  |
| pop dx |  |
| pop cx |  |
| pop bx |  |
| pop ax |  |
| iret |  |
| Int\_09H ENDP |  |
| Position dw 0 |  |
| Set\_handler: |  |
| mov dx, OFFSET Int\_09H | ;Перехватить вектор прерывания |
| Mov ax,2509H | ; от клавиатуры |
| int 21H |  |
| Mov ax,03H | ; Очистить экран |
| int 10H |  |
| Wait\_key: |  |
| jmp SHORT Wait\_key |  |
| END Entry |  |

## 

## 2.3. Программирование таймера 8254 и генерация звука

**..Назначение каналов таймера в IBM PC**

Стандартное использование каналов таймера в компьютере поясняется на рисунке 4.

**Канал 0** таймера используется для отсчета текущего времени. При инициализации компьютера он программируется для работы в режиме 3 с двухбайтным начальным значением счетчика равным 0. При таком режиме полный цикл таймера (частота генерации OUT) составляет около 55 миллисекунд. Сигнал GATE всегда установлен в высокий уровень. Сигнал OUT поступает в контроллер прерываний на линию 0 (IRQ0), в результате чего с периодичностью 55 миллисекунд происходит прерывание от таймера.

Стандартный обработчик прерываний от таймера BIOS выполняет следующие функции:

* увеличивает на 1 текущее время, которое содержится в двойном слове по адресу 40h:6Ch; при достижении значения 1800B0h обнуляет это двойное слово и устанавливает байт по адресу 40h:70h в единицу, указывая, что выполнен переход через полночь;
* уменьшает на 1 байт счетчик ожидания выключения мотора дискет по адресу 40h:40h и при достижении нулевого значения выполняет выключение мотора;
* выполняет команду INT 1Ch.

Естественно, нулевой канал таймера может использоваться и для программирования более коротких интервалов времени.

При перепрограммировании таймера на меньший интервал необходимо корректно передавать управление базовому обработчику прерываний таймера, чтобы не нарушить отсчет текущего времени и работу других программ.

**Канал 1** таймера используется как генератор для схемы регенерации памяти. BIOS программирует его в режиме 2 с однобайтовым счетчиком (младший байт), равным 18. Такой счетчик обеспечивает интервал чуть больший 15 микросекунд. Не рекомендуется перепрограммировать канал 1, так как это нарушит цикл регенерации памяти и приведет к разрушению ее содержимого.

На компьютерах класса AT этот канал таймера можно использовать для отсчета интервалов времени. Для этого следует проверять бит 4 управляющего порта 61h, который меняет свое значение на противоположное каждый цикл первого канала таймера (то есть примерно через 15 микросекунд).

**Канал 2** таймера, как правило, используется для генерации звука. Сигнал GATE для второго канала управляется битом 0 порта 61h. Сигнал OUT заведен на динамик. Соединение сигнала OUT с динамиком управляется битом 1 порта 61h.

Для генерации звуковой частоты обычно используется *режим 3* *– режим деления частоты.* *В программе вычисляется требуемое значение счетчика для регистра-защёлки таймера (делитель частоты), для этого входную частоту - 1193000 Гц (см. рис.4) делят на требуемую частоту в герцах.*

В этом режиме на выходе сигнала OUT появляются прямоугольные колебания, частота которых определяется начальным значением *счетчика* таймера. Разумеется, режим 3 не является единственным режимом, используемым для генерации звуков. ***Интересные звуковые эффекты можно получить, программируя канал таймера в других режимах.***

Второй канал таймера может использоваться и для целей отсчета времени. Следует отметить, что средства BIOS используют этот канал таймера только для генерации звука, и он не имеет постоянного режима работы и постоянного начального значения счетчика. Поэтому можно использовать этот канал в любом режиме. Тем не менее, рекомендуется после завершения работы со вторым каналом установить третий режим с двухбайтным счетчиком, так как некоторые BIOS не производят инициализацию этого канала перед выдачей звукового сигнала.

**Генерация тона**

Частоты первой октавы, начиная с ноты C (до) таковы:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Нота | C (до) | D (ре) | E (ми) | F (фа) | G (соль) | A (ля) | B (си) |
| Частота | 523.3 | 587.3 | 659.3 | 698.5 | 784.0 | 880.0 | 987.7 |

Частоты на октаву выше можно получить, удваивая эти значения, на две октавы выше - еще раз удваивая частоты. И наоборот, частоты на октаву ниже равны приблизительно половине этих значений (хорошо настроенное пианино точно не следует арифметическим интервалам).

***Частоты и длительности, заданные в табл.8, можно изменять экспериментально с целью получения оптимального звучания****.*

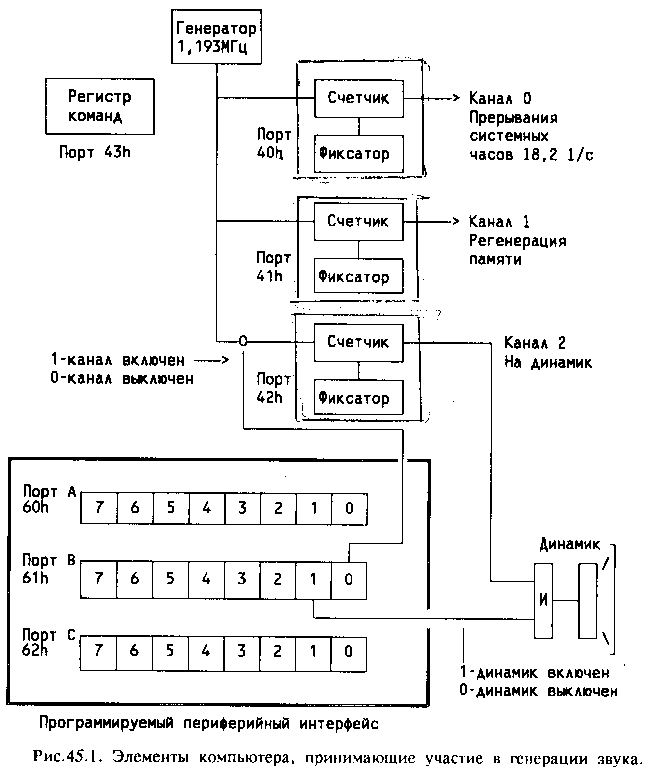
****

Рисунок 4 - Схема стандартного использования каналов таймера

Поскольку микросхема таймера 8254 работает независимо от процессора, то очень просто генерировать звук, который издается одновременно с выполнением других операций. Вы должны запрограммировать канал 2 этой микросхемы для генерации определенной частоты (см. табл.8 и описание «**Канал 2** таймера»), а затем программно задать длительность звучания этой частоты (**лучше использовать функцию 86h int15h,** **при этом программу следует запускать из WIN\_98 или MS DOS).** Работа таймера должна быть предварительно разрешена (см. рис.4) через порт B (адрес 61H, биты 0 и 1). Звук будет продолжаться до тех пор, пока биты 0 и 1 установлены. Поэтому необходимо сбросить бит 1 порта B в 0, иначе звук будет продолжаться, и может быть прекращен только перезагрузкой компьютера.

В данном примере генерируется частота 440Гц. Звук прекращается после нажатия любой клавиши на клавиатуре.

|  |  |  |  |
| --- | --- | --- | --- |
| ;разрешение канала 2 микросхемы 8254 установкой порта B | | | |
| PORT\_B | EQU | 61H | ;установка адреса порта B |
|  | IN | AL,PORT\_B | ;чтение его значения |
|  | OR | AL,3 | ;установка двух младших битов |
|  | OUT | PORT\_B,AL | ;посылаем байт в порт B |
| ;установка режима работы канала 2 | | | |
| COMMAND\_REG | EQU | 43H | ;адрес командного регистра |
| CHANNEL\_2 | EQU | 42H | ;адрес канала 2 |
|  | MOV | AL,10110110b | ;цепочка битов для канала 2 |
|  | OUT | COMMAND\_REG,AL | ;засылка в командный регистр |
| ;загрузка коэффицизнта деления частоты в регистр канала 2 | | | |
|  | MOV | AX,2705 | ;счетчик = 1190000/440 |
|  | OUT | CHANNEL\_2,AL | ;посылаем младший байт |
|  | MOV | AL,AH | ;сдвигаем старшиий байт в AL |
|  | OUT | CHANNEL\_2,AL | ;посылаем старший байт |
| ;ждем нажатия клавиши | | |  |
|  | MOV | AH,1 | ;номер функции прерывания 21H |
|  | INT | 21H | ;вызываем прерывание |
| ;выключение звука | | | |
|  | IN | AL,PORT\_B | ;получаем байт из порта B |
|  | AND | AL,11111100b | ;сбрасываем два младших бита |
|  | OUT | PORT\_B,AL | ;посылаем байт обратно |

Один из вариантов задания длительности звучания заданной частоты можно обеспечить выполнением следующих команд:

|  |  |  |  |
| --- | --- | --- | --- |
| ;создание цикла задержки | | | |
|  | MOV | AH,0 | ;номер функции чтения счетчика |
|  | INT | 1AH | ;получаем значение счетчика |
|  | MOV | CL,<задержка> | ;берем длину очередной ноты |
|  | MOV | CH,0 | ; в СХ теперь только длина |
|  | MOV | BX, DX | ;берем младшее слово счетчика |
|  | ADD | BX, CX | ;определяем момент окончания |
| L1: | INT | 1AH | ;берем значение счетчика |
|  | CMP | DX, BX | ;сравниваем с окончанием |
|  | JNE | L1 | ;неравно продолжаем звук |

Читаем значение счётчика времени суток BIOS, используя функцию 0 прерывания 1Ah, и добавляем к нему необходимое значение задержки (кратное числу импульсов по 1/18 секунды). Далее следует цикл, в котором считывается текущее значение счётчика времени суток и сравнивается с требуемой величиной задержки.

**Часы реального времени и системный таймер**

Начиная с IBM AT, персональные компьютеры содержат два устройства для управления процессами – часы реального времени (RTC) и собственно системный таймер. Часы реального времени реализованы на базе микросхемы (RT/CMOS RAM), которая получает питание от аккумулятора на материнской плате и работает даже тогда, когда компьютер выключен. Аппаратура часов реального времени использует только первые 14 байт этой микросхемы, а остальные предназначены для сохранения конфигурации компьютера.

**Конфигурация компьютера**

В процессе начальной загрузки компьютера программы DOS считывают показания часов, сохраняют их в ячейках системной области памяти и модифицируют в дальнейшем эти ячейки в соответствии с ходом работы системного таймера – по прерыванию INT 08h (18,2 раза в секунду). Системный таймер включает в себя три независимых канала и используется одновременно для управления контроллером ПДП, для управления динамиком и как генератор импульсов, вызывающий прерывание INT 08h. В идеале системные часы и часы реального времени должны идти одинаково. Реально это не всегда так. Для чтения или изменения показаний часов реального времени используется прерывание BIOS 1Ah. Это прерывание имеет несколько функций. Время в CMOS RAM хранится в упакованном двоично-десятичном формате, причём и при получении, и при установке времени число часов находится в регистре CH, число минут в CL и число секунд в DH.

Для обращения к часам реального времени на уровне портов ввода/вывода (адреса портов – 70h и 71h соответственно) необходимо выполнить следующие шаги:

* записать адрес считываемого (записываемого) регистра микросхемы в порт 70h;
* считать (записать) данные из регистра (в регистр) по адресу 71h.

***Следует отметить***, что чтение и запись данных порта 71h должны выполняться сразу после записи номера регистра в порт 70h. Кроме того, в процессе операций ввода/вывода с регистрами часов должны быть запрещены прерывания.

## 2.4. Возможности последовательного интерфейса RS-232 и программирование его контроллера.

**Контроллер последовательного интерфейса**

Контроллер последовательного интерфейса предназначен для обеспечения связи по протоколу RS-232C.

В настоящее время известны и другие реализации этого контроллера, как совместимые с рассматриваемым адаптером, так и не совместимые. В состав PC могут входить до четырех последовательных интерфейсов, работающих в стандарте RS-232C именуемых СОМ1 - СОМ4.

Каждый интерфейс связан с определенным уровнем контроллера прерываний:

* СОМ1 вызывает прерывание IRQ4 (Int 0Ch)
* COM2 вызывает прерывание IRQ3 (Int 0Bh)
* СОМ3 и СОМ4 не имеют стандартных векторов прерываний.

Каждое из устройств RS-232C представляет собой контроллер 8250, ос­нащенный 25- или 9- штырьковым разъемом на задней стенке корпуса PC. Наименования контактов стыка RS-232С представлены в таблице 2.

Таблица 2

**Наименования контактов стыка RS-232С**

| **Название**  **сигнала** | **Номер контакта** | | **Назначение** | **Направление** |
| --- | --- | --- | --- | --- |
| **9 шт.** | **25 шт**. |
| DCD | 1 | 8 | Связь модемов установлена | В РС |
| RX | 2 | 3 | Принимаемые данные | В РС |
| ТХ | 3 | 2 | Передаваемые данные | Из РС |
| DТR | 4 | 20 | Готовность РС к работе | Из РС |
| SG | 5 | 7 | Сигнальная земля | - |
| DSR | 6 | 6 | Готовность модема к работе | В РС |
| RТS | 7 | 4 | Запрос на передачу | Из РС |
| СТS | 8 | 5 | Готовность модема к передаче | Из РС |
| RI | 9 | 22 | Индикатор вызова | В РС |
| FG | - | 1 | Защитная земля | - |

**Состав контроллера последовательного интерфейса.**

В состав контроллера последовательного интерфейса входят следующие регистры [10]:

* регистры буферов приемника и передатчика;
* регистры разрешения и идентификации прерываний;
* регистры управления и состояния линии;
* регистры управления и состояния модема;
* регистры буфера делителя генератора.

В таблице 3 приведены адреса всех программно доступных регистров. Адреса в этой таблице даны относительно базового адреса контроллера. В таблице приведено значение бита DLAB регистра LCR, который управляет адресацией регистров. Именно этот бит делает возможным доступ к разным регистрам контроллера через порты с одним адресом, В приведенной таблице в графе «DLAB» стоит символ «X», если для адресации соответствующего регистра состояние данного бита несущественно. Более детально бит DLAB описан ниже.

Таблица 3

**Состав контроллера последовательного интерфейса**

| **Адрес** | **Операция** | **Регистр** | **DLAB** |
| --- | --- | --- | --- |
| 0 | W | Буфер передатчика (THR) | 0 |
| 0 | R | Буфер приемника (RBR) | 0 |
| 0 | R/W | Младший байт буфера де­лителя (Division Latch LSB) | 1 |
| 1 | R/W | Старший байт буфера делителя (Divisor Latch MSB) | 1 |
| 1 | R/W | Регистр разрешения преры­вания (IER) | 0 |
| 2 | R | Регистр идентификации прерывания (IIR) | X |
| 3 | R/W | Регистр управления линией (LCR) | X |
| 4 | R/W | Регистр управления модемом (MCR) | X |
| 5 | R | Регистр состояния линии (LSR) | X |
| 6 | R | Регистр состояния модема (MSR) | X |
| 7 | R/W | Неиспользуемый регистр (Scratch Register) | X |

Базовый адрес контроллера в зависимости от номера контроллера располагается в сегменте данных BIOS и приведен в таблице 4.

Таблица 4

**Базовые адреса и номера прерываний**

|  |  |  |
| --- | --- | --- |
| **Номер конт­роллера** | **Адрес в сегменте BIOS** | **Номер прерывания** |
| СОМ1 | 0040:0000 | IRQ4 (INT 0Ch) |
| COM2 | 0040:0002 | IRQ3 (INT 0Bh) |
| COM3 | 0040:0004 | не фиксирован |
| COM4 | 0040:0006 | не фиксирован |

Таблице 4 содержит адреса полей в области данных BIOS, в которых расположены базовые адреса контроллеров последовательного интерфейса. Базовые адреса контроллеров заносятся в сегмент данных BIOS программой POST (Power On Self Testing) при проверке после включения электропитания. Программа POST помещает базовые адреса контроллеров последовательно один за другим. Это означает, что между значащими полями не может быть нулевого поля.

Рассмотрим подробно назначение и содержимое регистров контроллера последовательного интерфейса.

***Регистр буфера передатчика (THR)***. Имеет адрес 0 относительно базового адреса контроллера. Данный регистр доступен только по записи и при значении бита разрешения доступа к делителю (DLAB) в регистре управления линией (LCR), рав­ном 0. Регистр THR содержит восемь битов данных (бит 0 яв­ляется младшим значащим разрядом и посылается первым в канал передачи).

***Регистр буфера приемника (RBR)***. Имеет адрес 0 относи­тельно базового адреса контроллера. Этот регистр доступен только по чтению (IN) и при значении бита разрешения досту­па к делителю (DLAB) в регистре управления линией (LCR), равном 0. Регистр RBR содержит восемь битов данных (бит 0 является младшим значащим разрядом и принимается первым из канала).

***Регистр буфера младшего байта делителя (Divisor Latch LSB)****.* Регистр имеет адрес 0 относительно базового адреса кон­троллера. Этот регистр доступен по чтению и записи только при значении бита разрешения доступа к делителю (DLAB) в регистре управления линией (LCR), равном 1. При записи в этот регистр нового значения делитель перезагружается немед­ленно.

***Регистр буфера старшего байта делителя (Divisor Latch MSB)****.* Регистр имеет адрес I относительно базового адреса контроллера. Этот регистр доступен по чтению и записи только при значении бита разрешения доступа к делителю (DLAB) в регистре управления линией (LCR), равном 1. При записи в этот регистр нового значения делитель перезагружается сразу.

***Регистр разрешения прерываний (IER)****.* Имеет адрес 1 относительно базового адреса контроллера. Этот регистр доступен по чтению и записи, но только при значении бита разрешения доступа к делителю (DLAB) в регистре управления линией (LCR), равном 0. Этот регистр позволяет управлять четырьмя типами прерываний, порождаемыми контроллером последова­тельного интерфейса. Формат регистра приведен ниже.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | IСМ | ICL | IFB | IDA |

*ICM* задает прерывание при изменении состояния модема:

1 - прерывание вырабатывается;

0 - прерывание запрещено.

*ICL* определяет прерывание при изменении состояния ли­нии приемника:

1 - прерывание вырабатывается;

0 - прерывание запрещено.

*IFB* задает прерывание при освобождении регистра буфера принимаемых данных:

1 - прерывание вырабатывается;

0 - прерывание запрещено.

*IDA* определяет прерывание при доступности принимаемых данных:

1 - прерывание вырабатывается;

0 - прерывание запрещено.

Биты 7-4 не используются и должны принимать значение 0.

Следует отметить, что все прерывания управля­ются также битом 3 (Out2) регистра управления модемом.

***Регистр идентификации прерывания (IIR)****.* Регистр имеет адрес 2 относительно базового адреса контроллера. Этот ре­гистр доступен только по чтению и позволяет получить инфор­мацию от контроллера о ждущем прерывании. Значение битов регистра приведено ниже.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | IType | | II |

Биты *IType* определяют тип ждущего прерывания, если оно хранится контроллером, что определяется битом II:

11 - изменилось состояние линии приемника;

10 - принимаемые данные доступны;

01 - освобожден регистр буфера;

00 - изменилось состояние модема.

Более подробная информация о приоритетах прерываний, условиях появления и условии сброса состояния прерывания приведена ниже в таблице 5.

Таблица 5

**Информация о ждущем прерывании**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IType** | **Прио­ритет** | **Тип** | **Условие появ­ления** | **Условие сброса** |
| 11 | 1 | Состояние линии прием­ника | Ошибка пере­полнения, чет­ности, посыл­ки или пауза | Операция чтения LSR |
| 10 | 2 | Доступность принимае­мых данных | Доступность принимаемых данных | Операция чтения RBR |
| 01 | 3 | Освобожде­ние регистра буфера пере­датчика | Освобождение THR | Операция чтения IIR или запись в THR |
| 00 | 4 | Состояние модема | Clear To Send, Data Set Ready, Ring Indicator или Data Carrier Detect | Операция чтения MSR |

Бит II является индикатором ждущего прерывания:

0 - контроллер последовательного интерфейса хранит прерывание;

1 - нет прерываний, ожидающих обработки.

Биты 7-3 регистра не используются и должны принимать значение 0.

***Регистр управления линией (LCR)****.* Регистр имеет адрес 3 относительно базового адреса контроллера. Этот регистр досту­пен по чтению и записи.

Значение данного регистра определяет формат передавае­мых данных в линию передачи данных контроллером последо­вательного интерфейса. Описание битов регистра приводится далее.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 7 | 1 | 0 |
| DLAB | SB | SP | EPS | PA | NSB | WLS | |

*DLAB* управляет доступом к регистрам буфера делителя. Если бит равен 1, операции чтения и записи по адресам 0 и 1 относительно базового адреса выполняются с регистрами буфе­ра делителя программируемого генератора. Для доступа к регистрам RBR, THR и 1ER бит должен иметь нулевое значение.

*SB* устанавливает состояние “пауза”, когда равен 1. В этом состоянии на выходе контроллера последовательного интер­фейса устанавливается значение 0, которое не может быть из­менено никакими другими действиями, хроме как переуста­новкой бита в 0.

*SP* управляет установкой режима неизменного бита контро­ля четности. Значение бита 1 задает режим, а значение 0 -отменяет. При установке бита SP в 1 должен устанавливаться в 1 и бит РА, т. е. эти два бита связаны. Когда значение бита EPS равно 0, посылается и контролируется значение бита кон­троля четности, равное 1 (Mark Parity), При единичном значе­нии бита EPS посылается и контролируется значение бита контроля четности, равное 0 (Space Parity).

*EPS* задает выбор режима контроля четности. Если бит установлен в 0 и бит РА установлен в 1, генерируется и проверя­ется четное количество единичных битов символа посылки и бита контроля четности. Если бит установлен в 1 и бит РА установлен в 1, генерируется и проверяется нечетное количество единичных битов символа посылки и бита контроля четности.

*РА* является битом разрешения контроля четности. Если бит установлен в 1, то генерируется бит контроля четности между последним битом передаваемого символа и стоп-битом.

*NSB* определяет количество стоп-битов в каждом символе, передаваемом контроллером последовательного интерфейса, и связан с длиной слова обмена (биты WLS). Если этот бит уста­новлен в 0, то генерируется и проверяется один стоп-бит при любой длине слова обмена. Если этот бит установлен в 1, то при длине слова обмена в *5* бит генерируется и проверяется 1.5 стоп-бита, а при любой другой длине слова обмена генерирует­ся и проверяется 2 стоп-бита. При асинхронной передаче понятие бита неразрывно связано с длитель­ностью сигнала, поэтому вполне возможна посылка нецелого числа стоп-битов. Это может потребоваться, если подключенное к компьютеру устрой­ство не программируется, а жестко настроено на анализ стоп-битов задан­ной длительности.

Биты *WLS* определяют длину слова обмена:

00-5 битов;

01-6 битов;

10-7 битов;

11-8 битов.

***Регистр управления модемом (MCR).*** Регистр управления модемом имеет адрес 4 относительно базового адреса контрол­лера. Этот регистр доступен по чтению и записи” С помощью регистра можно управлять работой модема.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | LB | Out2 | Out1 | RTS | DTR |

*LB* задаст режим “шлейф” (Loopback) для диагностических целей. При единичном значении этого бита происходит следу­ющее:

выход передатчика (SOUT) устанавливается в активное состояние;

* вход приемника (SIN) отсоединяется;
* выход сдвигового регистра передатчика подсоединяется к сдвиговому регистру приемника;
* четыре входных управляющих сигнала модема (CTS, DSR, DCD и RI) отсоединяются;
* четыре выходных управляющих сигнала модема (DTR, RTS, Out1 и Out2) подсоединяются к четырем управляю­щим входам модема;
* управляющие цепи модема принудительно устанавлива­ются в неактивное состояние.

В диагностическом режиме передаваемые данные сразу же принимаются. При этом полностью обеспечиваются прерыва­ния приемника и передатчика. Управление прерываниями так­же управляется регистром IER, однако источниками прерыва­ний в этом случае являются четыре младших бита регистра MCR вместо четырех управляющих входов модема. Система управления прерываниями может быть проверена в режиме “шлейф” записью в младшие 6 бит регистра LSR и младшие 4 бита регистра MSR. При установке любого из этих битов в 1 вырабатывается соответствующее прерывание (если оно разре­шено в регистре IER). Условие сброса состояния прерывания полностью соответствуют нормальному режиму работы.

Для возврата к нормальному режиму работы необходимо сначала перепрограммировать регистры для этого режима рабо­ты, а затем установить бит LB регистра MCR в значение 0.

*Out2* управляет сигналом Out2. При единичном значении бита, сигнал Out2 устанавливается равным 1. Сигнал Out2 уп­равляет генерацией прерываний контроллера последовательно­го интерфейса. При единичном значении сигнала контроллер генерирует прерывания в соответствии со значением регистра IER. При нулевом значении сигнала Out2 контроллер не гене­рирует прерываний независимо от значения регистра IER.

*Out1* управляет сигналом Outl. Если бит установлен в 1, сигнал Out1 устанавливается равным 1. При задании значения 0 сигнал устанавливается в нулевой уровень.

*RTS* управляет сигналом «запрос на передачу» (Request to Send). При значении этого бита, равном 1, сигнал “запрос на передачу” устанавливается равным 1. При задании значения 0 сигнал устанавливается в нулевой уровень.

*DTR* задает уровень сигнала «готовность терминала» (Data Terminal Ready). Если бит установлен в 1, сигнал “готовность терминала” устанавливается равным 1. При задании значения 0 сигнал устанавливается в нулевой уровень.

Биты 7-5 не используются и всегда устанавливаются в 0.

***Регистр состояния линии (LSR)****.* Регистр состояния линии имеет адрес 5 относительно базового адреса контроллера и до­ступен только по чтению. Регистр LSR предоставляет инфор­мацию о состоянии обмена данных.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | ТЕМТ | THRE | BI | FE | РЕ | OR | DR |

*ТЕМТ* является индикатором освобождения передатчика. Установка этого бита в 1 означает, что как регистр THR, так и регистр TSR свободны. Этотбит устанавливается в значение 0, если любой из регистров THR и TSR содержит символ.

*THRE* является индикатором освобождения регистра THR. Установка этого бита в 1 означает, что из регистра THR символ передан в сдвиговый регистр передатчика (TSR) и регистр THR готов принять следующий байт. Если в регистре IER разрешено прерывание по освобождению регистра THR, то при установке этого бита в значение 1 происходит также прерывание по осво­бождению регистра THR.

*BI* является индикатором состояния «пауза» (Break Interrupt). Состояние «пауза» фиксируется в том случае, если уро­вень принимаемого сигнала установлен в 0 на время приема полной посылки, т. с. общее время стартового бита, битов дан­ных, бита контроля четности и стоп-бита. Этот бит принимает значение 0 после операции чтения регистра LSR.

Биты с 4 по 1 являются индикаторами ошибки, и установка любого из этих битов в значение 1 приводит к порождению прерывания по состоянию линии приемника.

*FE* является индикатором «ошибки стоп-бита» (Framing Error). Ошибка стоп-бита фиксируется в том случае, когда в принятом символе не обнаружено корректного стоп-бита, т. е. бит, следующий за последним битом данных или за битом кон­троля четности (в случае контроля четности), имеет значение 0. Этот бит принимает значение 0 после операции чтения ре­гистра LSR.

*РЕ* является индикатором «ошибки четности» (Parity Error). Ошибка четности фиксируется, если в принятом символе обна­ружено некорректное значение бита контроля четности. Этот бит принимает значение 0 после чтения регистра LSR.

*OR* является индикатором «ошибки переполнения» (Overrun Error). Ошибка переполнения фиксируется в том случае, если при помещении очередного символа в регистр RBR обнаруже­но, что предыдущее содержимое этого регистра не считано и, таким образом, оно потеряно. Этот бит принимает значение 0 после операции чтения регистра LSR.

*DR -* индикатор доступности принимаемых данных. Этот бит всегда устанавливается в значение 1, когда приемником полностью принят символ и помещен в регистр RBR. Бит принимает значение 0 после операцией чтения из регистра RBR.

Бит 7 всегда устанавливается в значение 0.

***Регистр состояния модема (MSR)****.* Регистр имеет адрес 6 относительно базового адреса контроллера и доступен только по чтению. Регистр предоставляет информацию о состоянии управляющих линиях модема. Кроме того, этот регистр содер­жит четыре бита, которые отображают изменение состояния модема и устанавливаются в значение 0 после операции чтения из регистра MSR.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DCD | RI | DSR | CTS | DDCD | TERI | DDSR | DCTS |

*DCD* является инвертированным сигналом Data Carrier Detect (DCD). При установленном режиме “шлейфа” (бит LB регистра MCR имеет значение 1) этот бит эквивалентен биту Out2 регистра MCR.

*RI* является инвертированным сигналом Ring Indicator (RI). При установленном режиме «шлейфа» (бит LB регистра MCR имеет значение 1) эквивалентен биту Out1 регистра MCR.

*DSR* является инвертированным сигналом Data Set Ready (DSR). В режиме «шлейфа» (бит LB регистра MCR имеет зна­чение 1) эквивалентен биту DTR регистра MCR.

*CTS* - инвертированный сигнал Clear to Send (CTS), При установленном режиме «шлейфа» (бит LB регистра MCR имеет значение 1) этот бит эквивалентен биту RTS регистра MCR.

Биты DDCD, TERI, DDSR и DCTS являются ин­дикаторами изменения состояния модема и уста­новка любого из этих битов в значение 1 приво­дит к порождению прерывания по состоянию мо­дема, если оно разрешено в регистре IER.

*DDCD* является индикатором изменения сигнала Data Carrier Detect (DCD). Этот бит принимает значение 1 при изменении сигнала DCD после последней операции чтения регистра MSR.

*TERI* является индикатором заднего фронта сигнала RI. Этот бит принимает значение 1 при изменении сигнала RI с уровня логической единицы на уровень логического нуля.

*DDSR* является индикатором изменения сигнала Data Set Ready (DSR). Этот бит принимает значение 1 при изменении сигнала DSR после последней операции чтения регистра MSR.

*DTCS* является индикатором изменения сигнала Clear to Send (CTS). Этот бит принимает значение 1 при изменении сигнала CTS после последней операции чтения регистраMSR**.**

***Неиспользуемый регистр (Scratch Register)****.* Имеет адрес 7 относительно базового адреса контроллера и доступен по чте­ния и записи. Регистр не управляет контроллером и может быть использован в качестве рабочего регистра для хранения каких-либо данных.

***Программируемый генератор****.* Программируемый генера­тор служит для установки частоты контроллера последователь­ного интерфейса. Частота следования определяется как отно­шение частоты задающего генератора к делителю частоты. Ча­стота задающего генератора равна 1,8432 МГц. Дели­тель час­тоты представляет собой 16-ти битовое число, младший и старший байт которого загружаются по отдельности через ре­гистры буфера дели­теля. Делитель вычисляется как D=1843200/(bitrate\*16), где bitrate требуемая скорость (частота следования). После операции записи в любой из буферов делителя делитель перезагружается сразу же. В таблице 6 приведены необхо­димые значения делителя для получе­ния требуемой частоты следования.

Таблица 6

**Значения делителя**

|  |  |  |
| --- | --- | --- |
| **Требуемая час­тота следования (в бодах)** | **Значение делителя для получения требуемой частоты следования** | |
| **в десятичном виде** | **в шестнадцате­ричном виде** |
| 50 | 2304 | 0900h |
| 75 | 1536 | 0600Ь |
| 150 | 768 | 0300Ь |
| 300 | 384 | 0180h |
| 600 | 192 | 00C0h |
| 1200 | 96 | 0060h |
| 1800 | 64 | 0040h |
| 2400 | 48 | 0030h |
| 3600 | 32 | 0020h |
| 4800 | 24 | 0018h |
| 7200 | 16 | 00l0h |
| 9600 | 12 | 000Ch |
| 19200 | 6 | 0006h |
| 38400 | 3 | 0003h |
| 57600 | 2 | 0002h |
| 115200 | 1 | 000lh |

**Порядок инициализации контроллера последовательного интерфейса.**

Для подготовки контроллера к работе необходимо выполнить следующие шаги:

1. установить бит DLAB регистра LCR и записать делитель, задающий скорость обмена, в буфер делителя;
2. инициализировать регистр управления линией (LCR); при этом сбросить бит DLAB;
3. инициализировать регистр управления модемом (MCR);
4. инициализировать регистр управления прерываниями (IER) и, если прерывания разрешены, установить адрес программы обработки преры­ваний, а затем снять маску с соответствующего прерывания в регистре IMR контролера прерываний (порт 21h).

**2.5. Руководство по среде разработки Keil uVision.**

**Интегрированная система предназначена:**

* для проектирования программного обеспечения путем формиро­вания текстов программ на одном из языков программирования (ассемблер или Си) в специализированном текстовом редакторе:
* компиляции с получением исполнимого кода для микроконтрол­лера:
* формирования модели работы портов ввода - вывода, тестирова­ния программ микроконтроллера путем симуляции их выполне­ния:
* загрузки программ из компьютера в микроконтроллерную систему через интер­фейсы различных типов:
* обмена данными между персональными ЭВМ и микроконтрол­лерной системой в реальном времени.

Обычно по функциональным возможностям эти программы «позво­ляют»:

- задавать структуру проекта как совокупность перемещаемых или  
абсолютных программных модулей, разрабатываемых на вы­бранном языке программирования;

* в среде специализированного текстового редактора создавать тексты программных модулей на выбранном языке программи­рования:
* получать путем компиляции исполнимый файл, который содер­жит программу в виде кодов команд микроконтроллера:
* выполнять тестирование прикладной программы для микрокон­троллера на компьютере путем симуляции ее выполнения с динамиче­ским отображением состояния всех аппаратных ресурсов микро­контроллера и его окружения;
* загружать скомпилированную прикладную программу в микроконтроллерную систему через стандартный последовательный интерфейс (обычно RS-232);
* поддерживать двусторонний обмен данными между микрокон­троллерной системой и компьютером с отображением информации.

Современные программные системы, ориентированные на поддержку процесса проектирования, как правило, используют концепцию проектов.

*Проект* - это совокупность файлов, которые составляют некоторую разработку и фактически являются элементами описания *прикладной про­граммы* для выбранного микроконтроллера. Таким образом, каждый файл проекта может быть сформирован пользователем. Обычно в проекте поддер­живается организация прикладной программы для микроконтроллера на ос­нове модульного подхода, то есть прикладная программа может состоять из произвольного количества программных модулей (файлов), которые содер­жат взаимные ссылки. Целевые файлы проекта формируются автоматически системой из исходных файлов в процессе *построения при* *компиляции.* При построении проекта на основе исходных файлов формируются промежуточ­ные файлы, файлы, которые используются при симуляции, а также исполни­мые файлы в определенных форматах.

Базовым понятием является также понятие модели *контроллера.* Мо­дель контроллера определяет совокупность аппаратных ресурсов, которые учитываются компилятором и симулятором в составе системы. Каждая мо­дель контроллера соответствует аппаратной структуре определенной моди­фикации физического микроконтроллера семейства MCS-51. Таким образом, модель микроконтроллера - это совокупность следующих параметров:

* объемы внутренней и внешней памяти программ и данных;
* номенклатура и количество компонентов периферии, используе­мых в данной модификации микроконтроллера:
* конфигурационные параметры, используемые при симуляции, например, размер и скорость доступа для модели памяти EEPROM.

С каждым проектом связывается определенная модель микрокон­троллера, для которого создается программа в этом проекте. Таким образом, параметры модели непосредственно влияют на процесс компиляциипро­граммы и симуляцию ее выполнения при тестировании. Все операции с фай­лами проекта и описания моделей микроконтроллеров, перечисленными вы­ше, выполняются менеджерами проектов и моделей, в которых также обычно поддерживается концепция расширения списка моделей микроконтроллеров, относительно базового ядра.

**2.5.1 Разработка программных средств**

На первом шаге разработки программных средств формулируются технические требования к системе, и составляется блок-схема процесса решения нужных задач, которая обеспечит реализацию заданных требований. Блок-схема должна быть надлежащим образом структурирована, чтобы гарантировалась высокая эффективность логики программ.

Для того, чтобы было легко ориентироваться в файлах на компьютере, каждую задачу помещают в отдельную директорию (папку). Написание программы тоже следует начинать в отдельной папке, тем более, что как будет показано далее даже простейшая программа состоит из нескольких файлов.

В большинстве случаев программа состоит из нескольких программных модулей. Использование в составе одной программы нескольких программных модулей позволяет увеличить скорость трансляции программ, поручать написание программных модулей различным программистам, увеличивать понятность программ.

Принцип разбиения единой программы на программные модули заключается в том, что для реализации работы с отдельными узлами аппаратуры пишутся отдельные подпрограммы, которые относительно слабо связаны с остальными частями программы. Эти подпрограммы можно выделить в отдельную программу, которую можно хранить в отдельном файле, и транслировать отдельно от остальной программы. Часто одни и те же модули могут входить в состав нескольких программ, выполняющих совершенно различные задачи, но использующие при этом одни и те же устройства.

В качестве примера можно назвать работу с клавиатурой, индикацию различных видов информации, работу с последовательными портами, с АЦП, с ЦАП. Каждое из этих устройств может обслуживаться отдельными программными модулями. Этот список можно продолжать и далее, но для составления представления о программных модулях этого достаточно.

Программные проекты можно создавать, и поддерживать вручную, но в последнее время обычно используются различные системы поддержки разработок. Это создаёт ряд дополнительных преимуществ.

Часто программа пишется, и отлаживается на одной аппаратуре (например, на оценочных платах, предлагаемых фирмами изготовителями микросхем), а используется на другой. При этом программа при отладке незначительно отличается от программы, которая будет использоваться в реальной аппаратуре. Для этого в составе программного проекта создаются назначения проекта. В качестве примера назначений программного проекта можно назвать отладку и реализацию, а также версии программы.

**2.5.2. Создание программного проекта в интегрированной среде**

Работа с программными проектами начинается с создания нового файла проекта. Для создания файла проекта в интегрированной среде разработки программ можно воспользоваться главным меню, как показано на рисунке 3.1.

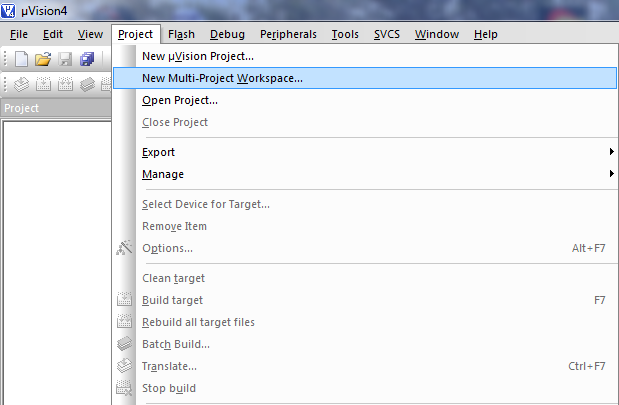


Рисунок 3.1 – Создание нового программного проекта

После создания новой директории и нового файла программного проекта, интегрированная среда программирования предлагает выбрать конкретную микросхему из семейства MCS-51, как это показано на рисунке 3.2.

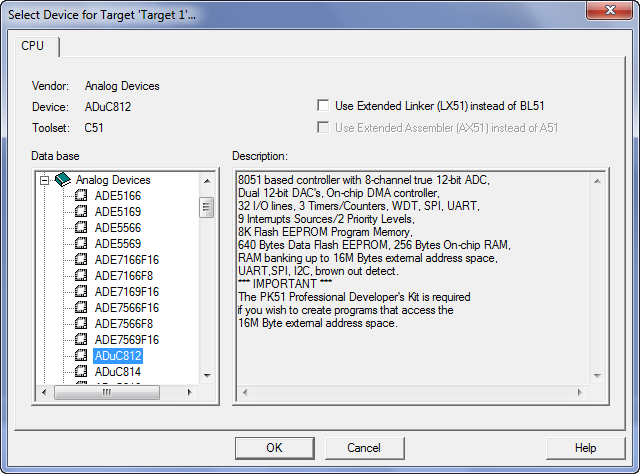


Рисунок 3.2. Диалоговое окно выбора конкретной микросхемы для программного проекта.

При этом в интегрированной среде программирования окно менеджера проекта приобретает вид, показанный на рисунке 3. Название назначения программного проекта можно изменить, щёлкнув манипулятором “мышь” по названию назначения программного проекта в окне менеджера проекта (Например: отладка, реализация или сопровождение). Точно так же можно изменить название устройства в составе программного проекта (Например: носимая радиостанция, автомобильная радиостанция, стационарная радиостанция или базовая радиостанция).

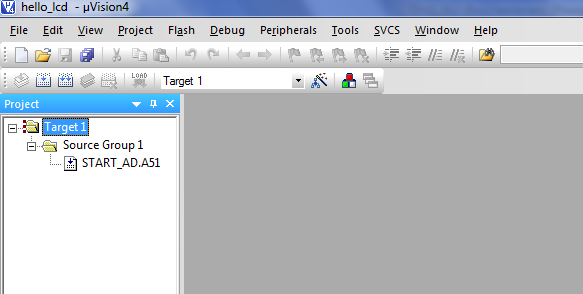
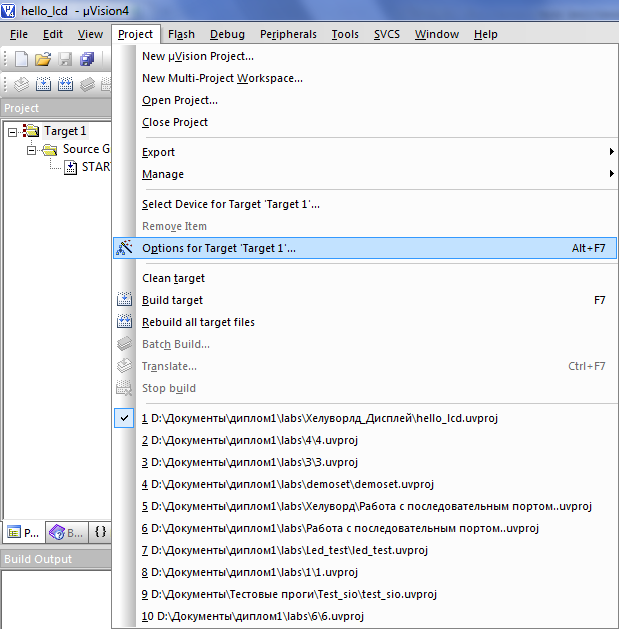


Рисунок 3.3 - Внешний вид окна менеджера проекта после создания программного проекта

**2.5.3. Настройка свойств программного проекта в интегрированной среде программирования Keil uVision**

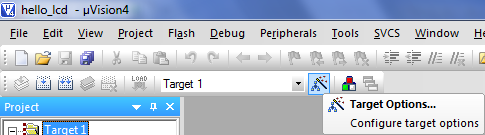
После создания программного проекта в интегрированной среде программирования Keil uVision IDE конечным файлом трансляции является абсолютный файл. Для загрузки в микросхему обычно используется HEX файл. Для создания этого файла необходимо включить соответствующую опцию в свойствах программного проекта.



«Рисунок 3.4. Изменение свойств программного проекта через главное меню»

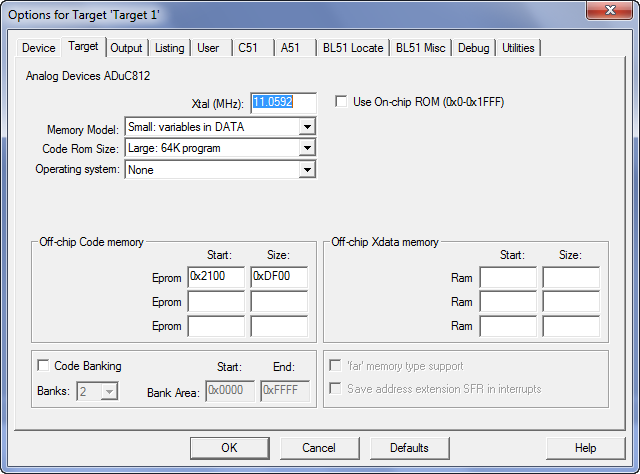
Изменить свойства программного проекта можно несколькими способами. Первый способ – воспользоваться главным меню, как показано на рисунке 4.

Второй способ – это нажать на кнопку изменения свойств программного проекта, как показано на рисунке 3.5.



«Рисунок 3.5. Изменение свойств программного проекта при помощи пиктограммы».

При этом на экране компьютера появляется диалоговое окно изменения свойств программного проекта как показано на рисунке 3.6. В этом окне необходимо ввести параметры внешней памяти программ и памяти данных.



«Рисунок 3.6. Диалоговое окно настройки свойств программного проекта»

При написании программ для лабораторного стенда SDK-1.1 нужно учитывать такую особенность разработки программы, что программа должна находиться во внешней памяти программ. Поэтому начальный адрес памяти программ необходимо установить за пределами внутренней памяти программ, т.е. 0x2000. Начальный адрес внешней памяти данных можно установить, начиная с половины страницы, т.е. с адреса 0x8000.

Затем необходимо установить выходные параметры программного проекта. Для этого открываем закладку выход (output), как это показано на рисунке 3.7. В этой закладке убеждаемся, что установлена галочка создания выходного загрузочного файла в hex формате. Для того, чтобы не загромождать директорию проекта файлами объектных кодов можно создать отдельную директорию. Например, с названием OBJ. Новая директория может быть создана после нажатия на кнопку “Select Folder for Object”.

Точно так же можно создать директорию (папку) для файлов листингов. Для этого необходимо выбрать закладку “Listing”. В файлах листингов помещается информация об ошибках, ассемблерный код и соответствующий ему машинный код программного модуля. Использование листингов позволяет оптимизировать программу, а при работе без интегрированной среды программирования и находить синтаксические ошибки программы. Отметим, что создание файлов листингов замедляет процесс трансляции. Обычно имя для папки листингов выбирают LST.

Необходимость создания отдельных папок для листингов и объектных кодов возникает при большом количестве программных модулей, а значит при большом количестве файлов в одной папке. Обычно директории для листингов и для объектных кодов располагают внутри папки программного проекта, где содержатся исходные тексты программы.

Для настройки параметров компиляции выбирается закладка “C51”. В этой закладке настраивается уровень оптимизации транслируемого программного модуля и цель оптимизации (по скорости работы программы или по размеру выходного файла). Кроме того, в этой закладке заносится адрес векторов прерывания.

После настройки свойств программного проекта в диалоговом окне, это окно закрывается нажатием кнопки “OK”. Если нужно отменить все сделанные изменения программного проекта то нажимается кнопка “отмена”.

Теперь можно подключать к программному проекту файлы с исходным текстом программных модулей. Для этого можно щёлкнуть правой кнопкой мыши по значку группы файлов в окне менеджера проектов, как это показано на рисунке 3.8, и выбрать опцию добавления файлов к программному проекту.

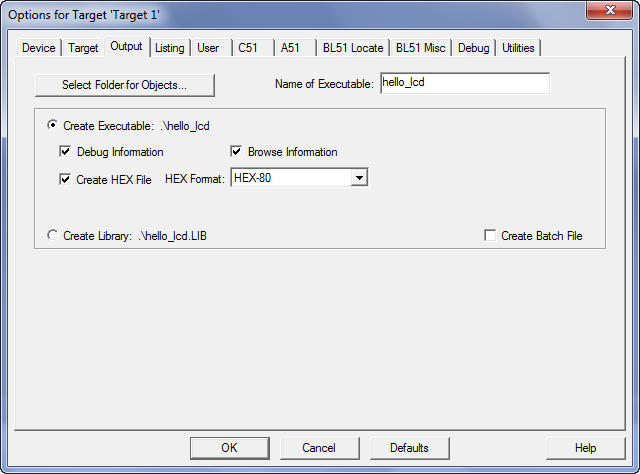


Рисунок 3.7 – Диалоговое окно настройки выходных параметров программного проекта

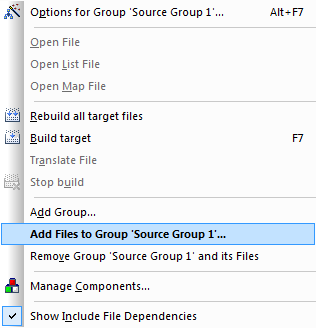


Рисунок 3.8 - Всплывающее меню менеджера проектов с выбранной опцией добавления файлов к программному проекту.

**2.5.4. Использование интегрированной среды программирования Keil uVision для трансляции программного проекта.**

Интегрированная среда программирования позволяет максимально облегчить трансляцию программных проектов. Так как параметры программного проекта уже настроены, то для трансляции исходного текста программного модуля достаточно загрузить исходный текст этого программного модуля в окно текстового редактора. Это можно сделать одним из способов, рассмотренных в предыдущих лабораторных работах.

После загрузки исходного текста программного модуля достаточно нажать на кнопку трансляции программного модуля, как это показано на рисунке 3.9.

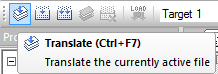


Рисунок 3.9 - Трансляция программного проекта при помощи кнопки «Translate»

Если же необходимо оттранслировать все программные модули вне зависимости имеются объектные модули или нет, и получить загрузочный файл, то нажимается кнопка «Build» или выбирается соответствующее меню, как показано на рисунке 3.10.

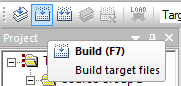


Рисунок 3.10 – Построение проекта с помощью кнопки «Build»

После построения в директории проекта должен появиться выходной файл в формате HEX-80 с именем, указанным в выходных параметрах проекта. На этом шаге разработку программы можно считать законченной.

**2.6 Отладка программы в среде Keil uVision**

**2.6.1 Способы отладки программ.**

Отладка программ заключается в проверки правильности работы программы и аппаратуры. Программа, не содержащая синтаксических ошибок тем не менее может содержать логические ошибки, не позволяющие программе выполнять заложенные в ней функции. Логические ошибки могут быть связаны с алгоритмом программы или с неправильным пониманием работы аппаратуры, подключённой к портам микроконтроллера.

Встроенный отладчик позволяет отладить те участки кода программы, которые не зависят от работы аппаратуры, не входящей в состав микросхемы микроконтроллера.

Для отладки программ обычно применяют три способа:

* Пошаговая отладка программ с заходом в подпрограммы;
* Пошаговая отладка программ с выполнением подпрограммы как одного оператора;
* Выполнение программы до точки останова.

Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор.

Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ.

**2.6.2 Использование встроенного отладчика программ.**

Вызов встроенного отладчика удобнее всего осуществить, нажав на кнопку отладчика «Start/Stop debug session» на панели инструментов «File» как показано на рисунке 3.11. Как и в предыдущих лабораторных работах для вызова отладчика можно воспользоваться главным меню интегрированной среды программирования или воспользоваться быстрой кнопкой ‘Ctrl+F5’.

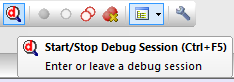


Рис. 3.11. Вызов встроенного отладчика с использованием кнопки на панели «File»

После этого внешний вид интегрированной среды программирования принимает вид, показанный на рисунке 3.12. В верхней части программы появляется дополнительная панель инструментов отладчика программ. В нижней части программы появляется окно просмотра памяти контроллера и окно контроля переменных Watch.

Окно просмотра памяти контроллера можно настроить на просмотр памяти программ или памяти данных, введя в диалоговое окно “адрес” ключ, двоеточие и адрес начальной ячейки памяти. Например:

d:0 – просмотреть память данных начиная с нулевой ячейки;

c:0 – просмотреть память программ начиная с нулевой ячейки;

x:0 – просмотреть внешнюю память данных начиная с нулевой ячейки.

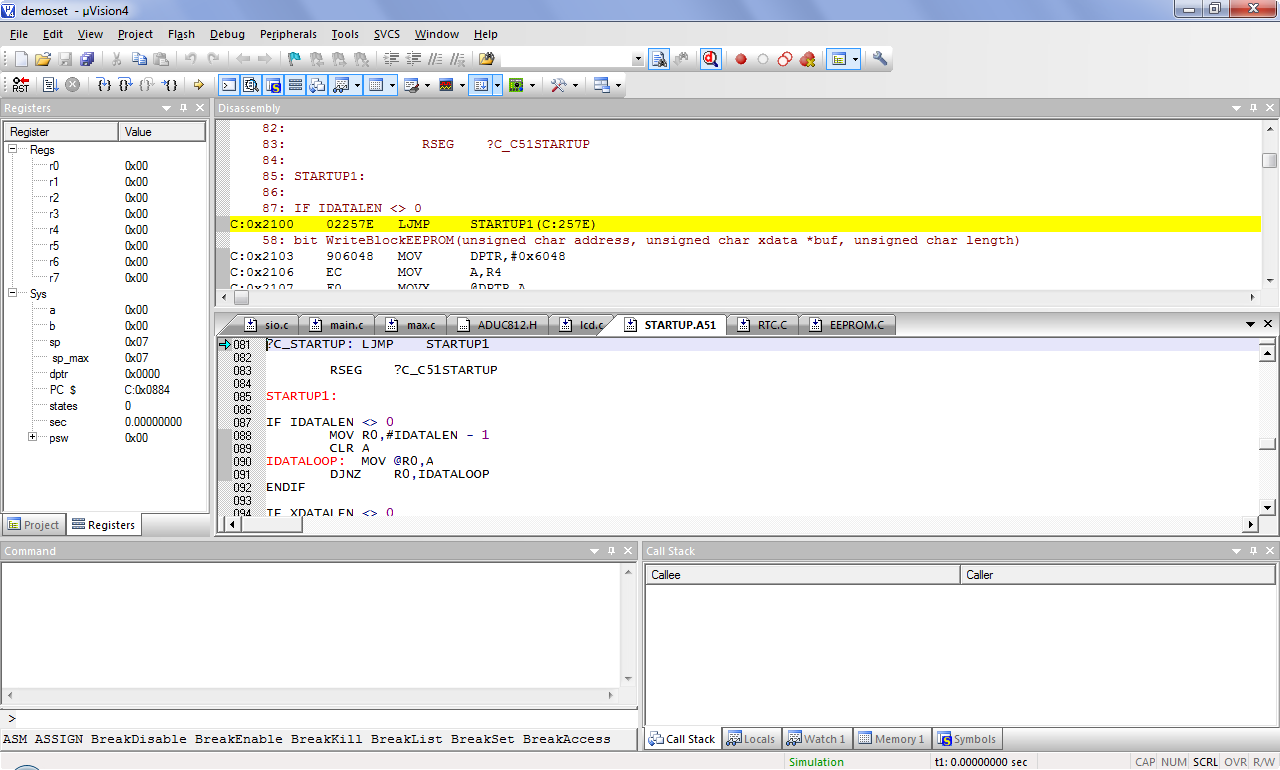


Рисунок 3.12 - Внешний вид интегрированной среды программирования в режиме отладки программ.

При использовании встроенного отладчика программ для контроля переменных можно воспользоваться окном Watch. В большинстве случаев это намного выгоднее, чем использовать просмотр памяти данных. Переменные в этом окне отображаются в том формате, в котором они были объявлены в программе. Для добавления переменной в окно Watch достаточно щёлкнуть правой кнопкой мыши по переменной в окне отладчика программ, как это показано на рисунке 3.13.

Окно просмотра переменных содержит две закладки «Watch #1» и «Watch #2». Это позволяет группировать переменные, по какому либо признаку, например по отлаживаемым подпрограммам. При добавлении переменной Вы выбираете номер окна просмотра переменных.

Кроме просмотра глобальных переменных, которые существуют на протяжении всей программы, окно просмотра переменных содержит закладку «Locals». Эта закладка позволяет отслеживать локальные переменные, которые существуют только внутри подпрограммы. Вводить имена локальных переменных в эту закладку не нужно. Они появляются в этой закладке автоматически, как только Вы попадаете в подпрограмму, в которой используются локальные переменные.

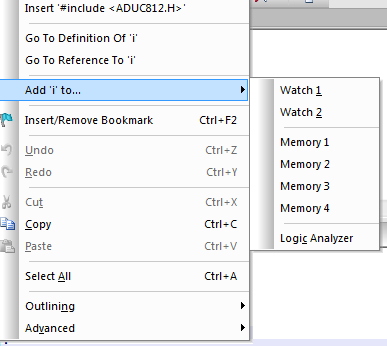


Рис. 3.13 - Всплывающее меню для добавления переменной в окно просмотра Watch.

При отладке программ на языке программирования ассемблер очень важно контролировать содержимое внутренних регистров микроконтроллера. Это позволяет сделать закладка «Regs» в окне менеджера проектов, показанная на рисунке 3.12. В этом окне можно проконтролировать содержимое регистров текущего банка, указателя стека и программного счётчика, содержимое аккумуляторов A и B, а также состояние рабочих флагов микроконтроллера в регистре PSW.

Один оператор программы может быть выполнен нажатием кнопки F11. Если вызов подпрограммы рассматривается как один оператор, то пошаговая отладка программы осуществляется нажатием кнопки F10.

Использование точек останова позволяет пропускать уже отлаженную часть программы. Для того, чтобы установить точку останова, можно воспользоваться кнопкой «Insert/Remove breakpoint» на панели файлов или воспользоваться главным или всплывающим меню. Перед тем как нажать на кнопку установки точки останова, необходимо установить курсор на строку исходного текста программы, где необходимо остановить выполнение программы.

Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору.

После того, как установлены все необходимые точки останова осуществляется выполнение программы в свободнобегущем режиме. Для этого можно воспользоваться кнопкой «Run» или нажать на кнопку F5 на клавиатуре.

Может возникнуть ситуация, что программа не передаёт управление ни одному из операторов, на которых установлены точки останова. В этом случае для прекращения выполнения программы следует воспользоваться кнопкой «Stop» или нажать на кнопку «Esc» на клавиатуре.

Точка останова может быть использована многократно. Иногда же возникает необходимость однократно пропустить часть операторов. В этом случае можно воспользоваться кнопкой выполнения программы до курсора «Run to cursor line». При нажатии на эту кнопку программа будет выполняться до тех пор, пока управление не будет передано оператору, на котором находится курсор. Как только это произойдёт, выполнение программы будет остановлено, и можно будет проконтролировать переменные и продолжить выполнение программы в пошаговом или свободнобегущем режиме.

**2.7 Загрузка программы в стенд SDK-1.1**

Следующий шаг - загрузка программы в память стенда SDK-1.1. В зависимости от загрузчика, установленного в стенде. В данный стенд может быть установлен один из двух загрузчиков:

* Резидентный загрузчик HEX-202;
* Универсальный загрузчик UL3, состоящий из резидентного загрузчика HEX-202 и инструментов загрузки ПО PM3P.

В первом случае загрузка осуществляется с помощью инструментальной системы T167B, в том виде, в котором программа была получена из среды Keil uVision, то есть в виде файла формата HEX-80. Для загрузки программы необходимо запустить инструментальное средство T167B с помощью следующей команды: «t167b lfile load\_t.167», где load\_t.167 – заранее подготовленный сценарий.

Во втором случае перед загрузкой выходной файл программы в формате HEX-80 не обходимо преобразовать в двоичный файл. Для преобразования и загрузки файла программы используется инструментальное средство M3P, которое по своей сути является интерпретатором языка FORTH. Команды преобразования и загрузки записываются в специальный сценарий, который обычно имеет имя load.m3p. Запускается сценарий командой «m3p lfile load.m3p»

Для удобства программиста можно настроить среду Keil uVision на использование одной из инструментальных систем для загрузки ПО в стенд. Делается это следующим образом: В окне опций проекта, изображенном на рисунке 3.7, необходимо перейти на вкладку «Utilities», как показано на рисунке 3.14.

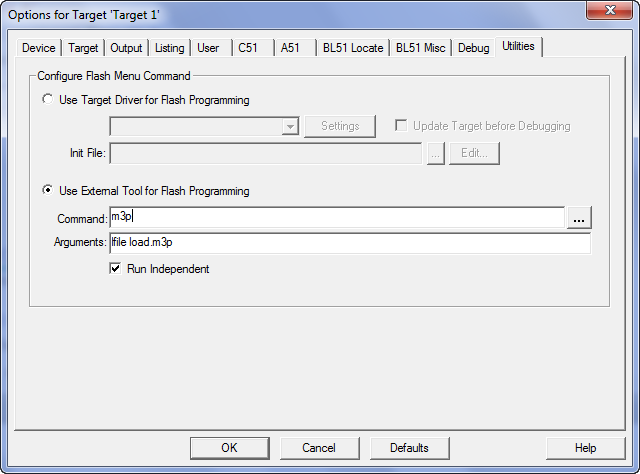


Рисунок 3.14 – Настройка внешних средств для загрузки ПО.

Далее необходимо выбрать пункт «Use External Tool fo Flash Programming», в поле «Command» ввести команду «m3p», а в поле «Arguments» - команду «lfile load.m3p». Также необходимо установить флаг «Run Independent», так как при неустановленном флаге, в случае «зависания» программы «m3p», среда Keil uVision перестанет отвечать на запросы пользователя. Она будет дожидаться завершения работы программы «m3p».

После этого можно загружать ПО в стенд с помощью пункта «Download» в меню «Flash», как показано на рисунке 3.15

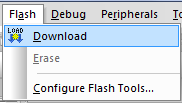


Рисунок 3.15 – Загрузка ПО в стенд.

# 3. ОБЩИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Как всегда на первоначальном этапе изучения нового материала, тре­буется освоить значительный объем терминологии и другой «стартовой» ин­формации. Поэтому при выполнении данной лабораторной работы, рекомен­дуется раздел «теоретические сведения» читать до конца несколько раз под­ряд (3...5), каждый раз все более и более «углубляясь» в мелкие, и, как пра­вило, более «сложные» детали.

Самостоятельные ответы на контрольные вопросы и задания, обычно «резко» углубляют знания по изучаемой теме.

Отчетная часть отчета по выполнению лабораторной работы, в соот­ветствии требованиям к оформлению отчета, должна содержать минималь­ный объем тезисных материалов.

Для успешного выполнения лабораторных работ требуется домашняя подготовка. При домашней подготовке студенты должны ознакомиться с данным описанием, рекомендованной литературой и лабораторным зада­нием, ответить на контрольные вопросы, выполнить домашнее задание. Отчет должен содержать данные, указанные в описании лабораторной ра­боты. Допуск к выполнению лабораторных заданий дается преподавателем только после проверки выполнения студентом домашних заданий и собе­седования с целью выявления понимания существа выполняемой работы.

Предполагается, что студенты знакомы с архитектурой, системой команд и приемами программирования на языке ассемблера процессора 8086. Из лекций и в результате самостоятельного изучения [1-19] необхо­димо знать: систему прерываний, поддерживаемую процессорами семейства X86; структуру, режимы работы и программирование контроллера прерываний 8259А; структуру, режимы работы и программирование интервального таймера 8254, а также режимы работы контроллера последовательного интерфейса RS-232.

**3.1 Порядок выполнения лабораторной работы в процессе домашней подготовки**

1. Установить программу…….. на домашнем компьютере.

2. Внимательно прочитать инструкцию пользователя к программе ……(см. Приложение).

3. Запустить программу в соответствии с указаниями инструкции пользователя.

4. Вызвать программу редактор.

5. Записать свою программу на языке ассемблера.

6. Сохранить ее, дав имя с расширением asm.

7. Откомпилировать исходную программу в hex -файл.

13. В случае удачного исхода компиляции перейти к следующему пункту. При выявлении синтаксических ошибок внести исправления в строки, указанные компилятором и повторить ассемблирование до получения положительного результата.

14. Перейти в режим отладки. В процессе отладки использовать стандартные отладочные режимы, предоставляемые симулятором:

- пошаговый режим выполнения проверяемой программы с заходом или без захода в подпрограммы;

- с остановом в контрольных точках.

15. Анализируя содержимое используемых в программе регистров или ячеек памяти после выполнения очередной команды или в точках останова, убедиться в правильности функционирования разработанной программы.

16. Записать разработанный проект на диск.

17. Распечатать текст программы.

18. Начертить схему управления данными (алгоритм) для рассмотренной задачи.

19.

20.

……

# 4. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ № 1

## 

## Изучение режимов работы, системы команд и дополнительных способов адресации процессоров семейства IA-32.

**Цель работы** состоит в изучении архитектурных особенностей процессоров IA-32 их дополнительных способов адресации, написании и отладке 32-разрядных консольных Windows – приложений на языке ассемблера.

## 4.1. Домашняя подготовка

1. Изучить программно-доступные регистры процессоров IA-32, их дополнительные способы адресации, структурную организацию памяти, подготовить ответы на контрольные вопросы.[17,19]
2. Ознакомиться c режимами работы процессоров IA-32 [2,3,17,19].
3. Изучить методику написания 32-разрядных консольных Windows – приложений [5,17,18].
4. Ознакомиться с принципом отладки программ, написанных на языке ассемблера, с помощью отладчика OllyDbg.
5. Составить на языке ассемблера программу в соответствии с номером своего варианта.
6. Сделать описание группы команд, указанных в варианте задания.

## 4.2. Лабораторное задание

* 1. Загрузить под управлением отладчика программу и исследовать её работу. Выполнить программу в пошаговом режиме отладчика.
  2. Проверить правильность полученных результатов.
  3. Ответить на контрольные вопросы.
  4. Оформить отчет, который должен содержать: описание алгоритма отлаженной программы и её текст на языке ассемблера, описание команд программы и программно-доступных регистров IA-32.

## 4.3. Варианты задания

Вариант 1

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. Процедура умножения 2-х произвольных 16-ти разрядных чисел без знака с помощью операций сдвига, сложения (вычитания), результат работы процедуры отобразить на мониторе.
2. Сделать описание регистров защищённого режима работы МП 80386.

Вариант 2

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. Процедура вычисления простых чисел. Максимальное количество найденных простых чисел использовать как параметр для процедуры; результат работы процедуры отобразить на мониторе.
2. Сделать описание группы арифметических команд.

Вариант 3

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. Процедура вычисления первых n – чисел последовательности Фибоначчи; процедура в цикле вычисляет заданное число (n) чисел, результат вычисления сохраняет в массиве и отображает на мониторе.
2. Сделать описание группы логических команд.

Вариант 4

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. Написать программу, которая устанавливает флаг Zero, если переданное ей в регистре EAX целое число является простым. Программа должна в цикле выводить запрос пользователю на ввод числа, а затем отображать сообщение, является ли это число простым.
2. Сделать описание группы команд пересылки данных.

Вариант 5

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. Процедура, в которой 4-х байтовое упакованное десятичное число преобразуется в числовую ASCII-строку. Процедуре в регистре EAX передается упакованное число, а в регистре ESI – адрес буфера, в котором будет храниться ASCII – строка. Результаты преобразований отобразить на мониторе.
2. Сделать описание группы команд передачи управления.

Вариант 6

1. Построить схему алгоритма решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. Написать процедуру, подсчитывающую число единичных битов в 32 – разрядном регистре; число в процедуру передаётся через стек. Результаты подсчёта отобразить на мониторе.
2. Сделать описание регистров реального режима работы МП 80386.

Вариант 7

1. Построить схему алгоритма решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. Написать процедуру поиска последнего пробела в строке символов заданной длины. Адрес строки использовать как параметр для процедуры. Результат работы программы: адрес байта за последним пробелом в строке.
2. Сделать описание группы команд условного перехода.

Вариант 8

1. Построить схему алгоритма решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. Написать процедуру, которая возвращает в регистре EAX длину строки, адрес которой использовать как параметр для процедуры.
2. Сделать описание группы команд безусловного перехода.

Вариант 9

1. Написать процедуру преобразования в символьную форму упакованного четырёхразрядного двоично-десятичного числа, введённого с клавиатуры. Ввод числа и вывод результата преобразования на экран из основной программы. Программа должна состоять из сегментов данных, кода и стека.

2. Сделать описание регистров защищённого режима работы МП 80386.

Вариант 10

1. Написать процедуру, в которой 4-байтовое упакованное десятичное число преобразовывается в числовую ASCIIZ-строку. Процедуре передаётся в регистре EBX упакованное число, а в регистре ESI – адрес буфера, в котором будет храниться ASCIIZ-строка. Результаты преобразований отобразить на мониторе.
2. Сделать описание группы команд сравнения.

Вариант 11

1. Написать процедуру поиска и замены первого пробела в строке символов заданной длины символом «\*». Адрес строки использовать как параметр для процедуры. Результат работы программы: исходная и преобразованная строки.
2. Сделать описание группы команд для работы со строками.

Вариант 12

1. Построить схему алгоритма решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. Написать процедуру определения количества *различных* цифр в 7-разрядном BCD-числе. Число вводить с клавиатуры. Программа должна состоять из сегментов данных, кода и стека.
2. Сделать описание группы команд сдвига.

Вариант 13

1. Построить схему алгоритма решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. В сегменте данных задан массив из 3-х четырёхбайтных чисел. Написать процедуру проверки этих чисел на кратность 4. Числа кратные четырём вывести на экран.
2. Сделать описание группы команд управления циклом.

Вариант 14

1. Построить схему алгоритма решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. В сегменте данных задан массив из 3-х четырёхбайтных чисел. Написать процедуру проверки этих чисел на кратность 16. Числа кратные 16 вывести на экран.
2. Сделать описание группы команд работы со строками.

Вариант 15

1. Построить схему алгоритма решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера: преобразовать шестнадцатеричное число без знака из регистра EDX в десятичный формат и вывести его на монитор.
2. Сделать описание группы команд условного перехода.

Вариант 16

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. В сегменте данных задан массив из 15 однобайтных чисел. Написать процедуру поиска максимального числа в массиве. Вывод максимального числа на монитор в символьной форме из основной программы. Программа должна состоять из сегментов данных, кода и стека.
2. Сделать описание группы команд десятичной коррекции результата.

Вариант 17

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. Дан массив из 10 двухбайтовых чисел. Найти среди них наименьшее и вывести на монитор в символьной форме. Программа должна состоять из сегментов данных, кода и стека.

1. Сделать описание способов адресации процессоров IA-32.

Вариант 18

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. В сегменте данных задана переменная для хранения 32-разрядного шестнадцатеричного числа. Написать процедуру преобразования в символьную строку 32-разрядного шестнадцатеричного числа, адрес переменной использовать как параметр для процедуры. Результат преобразования вывести на монитор.
2. Сделать описание способов адресации процессоров IA-32.

Вариант 19

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. В сегменте данных задан массив из 10 однобайтовых чисел. Написать процедуру вычисления среднего арифметического этих чисел. Результат вычисления вывести на монитор в символьной форме. Программа должна состоять из сегментов данных, кода и стека.

2.Сделать описание арифметических команд при операциях с BCD числами.

Вариант 20

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. В сегменте данных задан массив из 10 однобайтовых чисел. Написать процедуру вычисления квадратов этих чисел и записи их в другой массив. В сегменте данных зарезервировать место под второй массив. Программа должна состоять из сегментов данных, кода и стека.

2.Сделать описание группы команд передачи управления.

Вариант 21

1. Построить алгоритм решения задачи, а затем написать процедуру на языке ассемблера определения количества слов в строке введённой с клавиатуры. Адрес строки использовать как параметр для процедуры. Программа должна состоять из сегментов данных, кода и стека.

2. Сделать описание группы команд работы со строками.

Вариант 22

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. В сегменте данных задан массив из 10 чисел от 0 до 9. Написать процедуру: изменить порядок следования чисел на обратный. Результат вывести на монитор. Программа должна состоять из сегментов данных, кода и стека.
2. Сделать описание группы команд работы со строками.

Вариант 23

1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. В сегменте данных задан массив из 10 чисел от 0 до 9. Написать процедуру: поменять местами четные и нечетные элементы(1 2 3 4 =>2 1 4 3). Преобразованный массив показать на мониторе. Программа должна состоять из сегментов данных, кода и стека.
2. Сделать описание способов адресации процессоров IA-32.

Вариант 24

* + - 1. Построить алгоритм решения задачи, а затем написать 32-разрядную терминальную программу на языке ассемблера. В сегменте данных задан массив из 7 однобайтовых шестнадцатеричных чисел. Написать процедуру преобразования в символьную форму однобайтного числа; числа из массива в процедуру передаются через регистр AL. Вывод результатов преобразования из основной программы.

1. Сделать описание группы команд работы со строками.

## 4.4. Контрольные вопросы

1. Как найти остаток от деления целого положительного числа на степень двойки (деление с помощью сдвига)?
2. Напишите фрагмент программы, в которой заданное число умножается на 10 с помощью сдвига и сложения.
3. Напишите фрагмент программы, которая позволяет убедиться, что команды INC и DEC не влияют на состояние флага переноса.
4. Алгоритм формирования физического адреса памяти из логического?
5. Размер адресного пространства В/В в МП X86, как адресуются порты В/В?
6. Составьте последовательность команд, заменяющую команду XLAT?
7. Какие команды позволяют организовать обмен между различными сегментами данных?
8. В чем состоит различие в использовании флагов CF и OF?
9. Какие ограничения присущи командам условного перехода?

# 5. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ №2

## Программирование контроллера прерываний 8259А и организация прерываний в IBM PC

**Цель работы** состоит в изучении организации прерываний в IBM PC с использованием контроллера прерываний 8259А.

## 5.1. Домашняя подготовка

* Изучить блок – схему алгоритма (см. далее рис.5), поясняющего последовательность программирования БИС 8259А.
* Ознакомиться со схемой подключения БИС 8259А к шина IBM PC (см. далее рис.6).
* Ознакомиться с назначением и структурой таблицы векторов прерываний.
* Составить на языке Ассемблера программу инициализации БИС 8259А в соответствии с алгоритмом, приведенным на рисунке 2: режим приоритетов простой; сигнал прерывания IRQj воспринимается по фронту; адрес контроллера прерываний - 20H; новый базовый вектор для IRQ0 задан в таблице 7.
* Подготовить ответы на контрольные вопросы.

## 5.2. Лабораторное задание

1. Отладить **(под управление MS DOS)** введенную программу и показать результат работы программы преподавателю.
2. Составить отчет о проделанной работе.
3. Ответить на контрольные вопросы.

## 5.3. Варианты задания

Таблица 7

**Варианты задания**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер варианта** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| Базовый вектор | 60Н | 68Н | 70Н | 78Н | 80Н | 88Н | 90Н | 98Н |
| Номер варианта | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Базовый вектор | C8Н | C0Н | A0Н | A8Н | D8Н | B0Н | D0Н | B8Н |
| Номер варианта | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| Базовый вектор | По согласованию с преподавателем | | | | | | | |

**Пояснения к заданию**

*Следует помнить, что данное задание рассчитано на работу под управлением MS DOS*.

В качестве основной программы может быть использована любая программа пользователя. Задача основной программы перехватить прерывание от клавиатуры или от таймера *с учётом нового базового вектора ведущего контроллера,* сохранить старые значения векторов прерываний и каким-то образом оставить программу в памяти компьютера (см. далее примеры программ обработки прерываний).

В данной работе рассматривается обработка только двух источников внешнего прерывания: от таймера IR0 и от клавиатуры IR1. *Для демонстрации работы нового обработчика прерываний (ISR) нельзя использовать системные функции MS DOS.*

Блок реинициализации БИС 8259А содержит команды аналогичные командам блока инициализации за исключением команды установки типа прерывания.

При составлении основной программы следует разобраться с приведенными ниже примерами программ (см. раздел «примеры обработки аппаратных прерываний»), а также с системными функциями 25h и 35h.

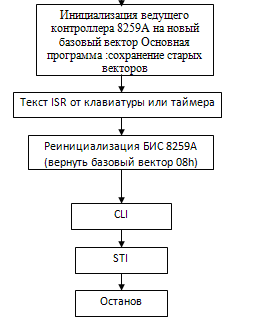


Рисунок 5 – Алгоритм программы лабораторного задания

## 5.4. Контрольные вопросы

1. Для чего в программе обработчика прерываний необходимо указывать команду EOI?
2. Что произойдет, если обработчик прерываний прервется сигналом того же уровня?
3. В каком случае становится важен вопрос об анализе приоритетов прерываний?
4. Можно ли командой CLI запретить программные прерывания?
5. Объяснить понятие «вектор прерывания»?
6. Как определить адрес вектора прерывания?
7. Что делает команда IRET.
8. Какие адреса используются для портов ведущего и ведомого контроллеров?
9. Какие действия необходимо провести для смены базового вектора прерываний?
10. Объяснить понятие «вложенное прерывание»?
11. Назначение внутренних регистров контроллера?
12. Назначение таблицы векторов прерываний?

# 6. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ № 3

## Программирование таймера 8254 и генерация звука.

**Цель работы** состоит в получении навыков программирования (на примере синтеза музыкальных фраз) и возможностей использования таймера 8254 в вычислительных системах.

Современные компьютеры оснащаются двумя подсистемами таймеров, параллельно отсчитывающими текущее время. Один таймер расположен в микросхеме с низким потреблением энергии (КМОП-микросхеме), которая при выключении питания компьютера продолжает работать, получая энергию от встроенного в компьютер аккумулятора. Этот таймер, обычно, называют часами реального времени. Другой таймер (его часто называют системным) работает, как и все остальные узлы компьютера, только когда компьютер включён. При выполнении задания по данной лаб. работе необходимо изучить оба таймера, что позволит наиболее оптимально составить текст программы.

## 6.1. Домашняя подготовка

* Ознакомиться с теоретической частью лабораторной работы №3
* Изучить режимы работы БИС 8254.
* Подготовить ответы на контрольные вопросы.
* В соответствии с вариантом задания, составить программу на языке ассемблера, генерирующую мелодию, предусмотрев возможность повторения фразы ''m'' раз. Варианты задания на лабораторную работу приведены в таблице 8.

## 6.2. Лабораторное задание

Ввести текст программы, составленной в соответствии с вариантом задания. Отладить введенную программу и показать результат работы программы преподавателю.

Ответить на контрольные вопросы.

Составить отчет о проделанной работе.

## 6.3. Варианты задания

**\*(**В таблице 8 Hz обозначает частоту сигнала в Гц, а Т – длительность сигнала в мс.).

Таблица 8

**Варианты задания**

| **№** | | **Частота и длительность нот** | | | | | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Hz | 330 | 262 | 330 | 262 | 349 | 330 | 294 | 196 | 196 | 196 | 196 | 247 | 262 | 262 | 262 |
| T | 250 | 250 | 250 | 250 | 250 | 250 | 500 | 250 | 250 | 250 | 125 | 125 | 250 | 250 | 500 |
| 2 | Hz | 349 | 330 | 294 | 262 | 392 | 392 | 349 | 330 | 294 | 262 | 392 | 392 | 349 | 392 | 392 |
| T | 250 | 250 | 250 | 250 | 500 | 500 | 250 | 250 | 250 | 250 | 500 | 500 | 250 | 250 | 250 |
| 3 | Hz | 349 | 349 | 330 | 294 | 294 | 262 | 349 | 349 | 330 | 294 | 294 | 262 | 349 | 349 | 330 |
| T | 250 | 250 | 500 | 250 | 250 | 500 | 250 | 250 | 500 | 250 | 250 | 500 | 250 | 250 | 500 |
| 4 | Hz | 440 | 440 | 440 | 440 | 415 | 440 | 494 | 440 | 415 | 494 | 494 | 494 | 494 | 440 | 494 |
| T | 125 | 125 | 125 | 125 | 62 | 62 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 62 | 62 |
| 5 | Hz | 175 | 196 | 208 | 175 | 208 | 262 | 233 | 208 | 196 | 175 | 156 | 156 | 208 | 208 | 233 |
| T | 125 | 125 | 250 | 250 | 250 | 125 | 125 | 125 | 125 | 500 | 125 | 125 | 250 | 250 | 250 |
| 6 | Hz | 175 | 147 | 196 | 147 | 175 | 147 | 196 | 147 | 175 | 233 | 220 | 196 | 175 | 147 | 196 |
| T | 125 | 125 | 750 | 750 | 125 | 125 | 750 | 750 | 250 | 250 | 250 | 250 | 125 | 125 | 125 |
| 7 | Hz | 233 | 233 | 233 | 294 | 233 | 233 | 175 | 175 | 175 | 233 | 175 | 175 | 147 | 147 | 156 |
| T | 250 | 125 | 125 | 125 | 125 | 250 | 250 | 125 | 125 | 125 | 125 | 250 | 250 | 125 | 125 |
| 8 | Hz | 494 | 494 | 494 | 494 | 440 | 440 | 392 | 392 | 370 | 330 | 494 | 494 | 587 | 494 | 440 |
| T | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 250 | 250 | 125 | 125 | 125 | 125 | 125 |
| 9 | Hz | 330 | 330 | 247 | 247 | 330 | 370 | 330 | 311 | 277 | 310 | 330 | 330 | 392 | 392 | 294 |
| T | 125 | 125 | 125 | 125 | 250 | 125 | 125 | 250 | 125 | 125 | 250 | 250 | 125 | 125 | 125 |
| 10 | Hz | 220 | 294 | 394 | 494 | 594 | 694 | 700 | 294 | 350 | 700 | 1000 | 500 | 330 | 220 | 440 |
| T | 125 | 250 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 |
| 11 | Hz | 440 | 440 | 440 | 440 | 494 | 440 | 415 | 494 | 494 | 494 | 523 | 494 | 440 | 523 | 523 |
| T | 125 | 125 | 125 | 125 | 125 | 250 | 125 | 125 | 250 | 125 | 125 | 250 | 125 | 125 | 125 |
| 12 | Hz | 494 | 494 | 494 | 494 | 440 | 440 | 392 | 392 | 370 | 330 | 494 | 494 | 494 | 494 | 440 |
| T | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 250 | 250 | 125 | 125 | 125 | 125 | 125 |
| 13 | Hz | 392 | 440 | 392 | 440 | 392 | 392 | 440 | 440 | 392 | 392 | 440 | 392 | 440 | 392 | 440 |
| T | 500 | 500 | 500 | 500 | 250 | 250 | 250 | 250 | 250 | 250 | 500 | 500 | 500 | 500 | 500 |
| 14 | Hz | 330 | 440 | 523 | 523 | 587 | 523 | 494 | 1000 | 587 | 523 | 494 | 523 | 330 | 440 | 523 |
| T | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 250 | 250 | 125 | 125 | 125 |
| 15 | Hz | 392 | 392 | 392 | 440 | 494 | 440 | 392 | 494 | 440 | 392 | 392 | 392 | 392 | 440 | 494 |
| T | 250 | 250 | 250 | 250 | 500 | 250 | 250 | 250 | 250 | 500 | 250 | 250 | 250 | 250 | 500 |
| 16 | Hz | 349 | 349 | 349 | 392 | 440 | 392 | 349 | 440 | 392 | 349 | 349 | 349 | 349 | 392 | 440 |
| T | 250 | 250 | 250 | 250 | 500 | 250 | 250 | 250 | 250 | 500 | 250 | 250 | 250 | 250 | 500 |
| 17 | Hz | 494 | 523 | 494 | 392 | 494 | 523 | 494 | 392 | 494 | 523 | 494 | 392 | 494 | 523 | 494 |
| T | 250 | 500 | 250 | 500 | 250 | 500 | 250 | 500 | 250 | 500 | 250 | 500 | 250 | 500 | 250 |
| 18 | Hz | 330 | 349 | 392 | 349 | 330 | 294 | 349 | 330 | 294 | 262 | 294 | 330 | 262 | 330 | 349 |
| T | 250 | 250 | 250 | 250 | 250 | 125 | 125 | 250 | 250 | 500 | 125 | 125 | 250 | 250 | 250 |
| 19 | Hz | 523 | 494 | 440 | 330 | 523 | 494 | 440 | 330 | 494 | 440 | 330 | 262 | 494 | 440 | 415 |
| T | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 |
| 20 | Hz | 466 | 466 | 392 | 294 | 466 | 440 | 392 | 610 | 466 | 440 | 392 | 294 | 466 | 440 | 392 |
| T | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 375 | 250 | 250 |

## 

## 6.4. Контрольные вопросы

1. Каким образом представляются гармонические колебания, соответствующие определённой ноте в ЭВМ?
2. Каким образом можно обеспечить требуемую длительность звучания ноты?
3. Режимы работы БИС 8254?
4. Объясните назначение и использование канала 0 таймера в IBM PC?
5. Объясните назначение и использование канала 1 таймера в IBM PC?
6. Объясните назначение и использование канала 2 таймера в IBM PC?
7. Форматы управляющего байта для задания режима работы таймера.
8. Последовательность программирования БИС 8254.

# 7. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ № 4

Изучение программного обеспечения систем управления на базе учебного лабораторного комплекса SDK -1.1.

**Цель работы:**

Изучение архитектуры учебного лабораторного комплекса SDK-1.1, написание и отладка программ для подсистем ввода-вывода лабораторного комплекса SDK -1.1.

## 7.1. Домашняя подготовка

* Ознакомиться с теоретической частью лабораторной работы.
* Изучить назначение и возможности лабораторного комплекса SDK-1.1 [5,7,17]
* Разработать и реализовать драйверы светодиодного индикатора и DIP-переключателя контроллера SDK-1.1.
* Написать тестовую программу с использованием разработанных драйверов по алгоритму, соответствующему варианту задания.
* **При реализации вариантов задания необходимо использовать таймеры микроконтроллера AduC812 в режиме прерывания**.
* Изучить средства загрузки откомпилированной программы в стенд SDK-1.1.
* Подготовить ответы на контрольные вопросы.

**7.2. Лабораторное задание.**

1. На основе программы, составленной в соответствии с вариантом задания, создать программный проект в интегрированной среде Keil uVision (см. раздел 2.5) и откомпилировать исходную программу в hex –файл для загрузки в стенд.
2. Отладить программу на учебном стенде SDK-1.1 и показать результат работы программы преподавателю.
3. Ответить на контрольные вопросы.
4. Составить отчет.

## 7.3. Варианты заданий на лабораторную работу:

……….

**7.4 Теоретические сведения**

Ввод-вывод данных в стенде SDK-1.1 осуществляется с помощью портов микроконтроллера и микросхемы ПЛИС, которая имеет 8 регистров, отображаемых во внешнее адресное пространство памяти контроллера ADuC812. Через микросхему ПЛИС к микроконтроллеру подключены: матричная клавиатура, ЖКИ, последовательный порт ввода вывода, звуковой излучатель и линейка светодиодов.

Для управления светодиодами служит регистр SV. Адрес 080007H. Значение после сброса 00000000B.

Таблица 1 – Назначение битов регистра SV.

|  |  |  |
| --- | --- | --- |
| **Биты** | **Поле** | **Описание** |
| 0..7 | D0..D7 | Биты управления светодиодами. Подача логической «1» зажигает светодиоды. |

Управление параллельным портом и ввод-вывод данных через него осуществляется с помощью регистров EXT\_LO, EXT\_HI и ENA.

Регистр ENA - Регистр управления портами ввода-вывода, звуком, сигналом INT0 и прерыванием от клавиатуры. Адрес 080004H. Значение после сброса x0100000B.

Таблица 2 – Назначение битов регистра ENA.

|  |  |  |
| --- | --- | --- |
| **Биты** | **Поле** | **Описание** |
| 7 | - | Не используется |
| 6 | KB | В полной конфигурации при записи логического «0» прерывание от клавиатуры запрещается. Если бит установлен в «1», то прерывание от клавиатуры разрешено. В упрощенной конфигурации бит KB всегда равен нулю, т.е. прерывание клавиатуры запрещено. |
| 5 | INT0 | При записи логического «0» в этот бит на вход INT0 ADuC812 также попадает логический «0». Бит можно использовать для формирования внешнего прерывания для микроконтроллера. |
| 2..4 | SND0-SND2 | Выход звукового ЦАП. Задает уровень напряжения на динамике. Позволяет формировать звуковые сигналы различной тональности и громкости. |
| 1 | EN\_HI | Управление старшими 8 разрядами (биты 8..15) 16-разрядного порта ввода-вывода. Если записать в EN\_HI логический «0», то порт ввода-вывода переводится в Z-состояние и появляется возможность чтения данных из EXT\_HI. При записи в данный бит логической «1» порт переключается на вывод и данные, записанные в регистр EXT\_HI, попадают на выход порта ввода-вывода. |
| 0 | EN\_LO | Управление младшими 8 разрядами (биты 0..7) 16-разрядного порта ввода-вывода. Если записать в EN\_LO логический «0», то порт ввода-вывода переводится в Z-состояние и появляется возможность чтения данных из EXT\_LO. При записи в данный бит логической «1» порт переключается на вывод и данные, записанные в регистр EXT\_LO, попадают на выход порта ввода-вывода. |

Регистр EXT\_LO имеет адрес 080002H, его значение после сброса 00000000B.

Таблица 2 - Назначение битов регистра EXT\_LO

|  |  |  |
| --- | --- | --- |
| **Биты** | **Поле** | **Описание** |
| 0..7 | D0..D7 | Регистр EXT\_LO позволяет считывать и записывать биты 0..7 параллельного порта. |

Регистр EXT\_HI (его адрес 080003H) работает аналогично регистру EXT\_ LO, и отвечает за биты 8..15 параллельного порта.

DIP переключатель служит для принудительного «обнуления» линий INT0/1, T0/1, а также линий 0-7 параллельного порта ПЛИС. Он замыкает соответствующие линии через резисторы 100 Ом на корпус. Для того, чтобы принудительно «обнулить» соответствующую линию, необходимо установить соответствующий переключатель в положение «ON». Соответствие номеров переключателей и линий приведено в таблице.

Таблица 3 – DIP переключатель.

|  |  |
| --- | --- |
| **№ переключателя** | **Линия** |
| 1 | Вход INT0 ADuC812 (P3.2). |
| 2 | Вход INT1 ADuC812 (P3.3). |
| 3 | Вход T0 ADuC812 (P3.4). |
| 4 | Вход T1 ADuC812 (P3.5). |
| 5 | Линия 0 параллельного порта ПЛИС MAX. |
| 6 | Линия 1 параллельного порта ПЛИС MAX. |
| 7 | Линия 2 параллельного порта ПЛИС MAX. |
| 8 | Линия 3 параллельного порта ПЛИС MAX. |
| 9 | Линия 4 параллельного порта ПЛИС MAX. |
| 10 | Линия 5 параллельного порта ПЛИС MAX. |
| 11 | Линия 6 параллельного порта ПЛИС MAX. |
| 12 | Линия 7 параллельного порта ПЛИС MAX. |

## 7.5. Контрольные вопросы

……….

…………..

# 8. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ № 5

# Изучение возможностей последовательного интерфейса RS-232 и программирование его контроллера.

**Цель работы** состоит в изучении режимов работы СОМ–порта и приобретении навыков программирования на уровне портов.

## 8.1. Домашняя подготовка

* Ознакомиться с теоретической частью к данной лабораторной работе [12,13,16].
* Ознакомиться со схемой соединения двух компьютеров через последовательный порт [12,13,16].
* Ответить на контрольные вопросы.
* В соответствии с вариантом, составить программу на языке ASSEMBLER.

## 8.2. Лабораторное задание

1. Отладить программу, составленную в соответствии с вариантом задания.
2. Продемонстрировать преподавателю работу отлаженной программы на компьютере.
3. Составить отчёт.

## 8.3. Варианты задания

таблица 9

**Варианты задания**

| **№** | **Действие** | **Формат обмена** | **Прерывание** |
| --- | --- | --- | --- |

***Формат обмена в таблице описан в следующем виде***: x1,x2,x3,x4,

где x1 – скорость обмена, x2 – способ контроля четности, x3 – количество бит посылки, x4 – количество стоп-битов.

***Обозначения способов контроля четности***:

N – без контроля четности;

O – с контролем четного количества единичных битов символа посылки;

E – с контролем нечетного количества единичных битов символа посылки;

M – с контролем значения бита кон­троля четности, равным 1;

S – с контролем значения бита кон­троля четности, равным 0.

В графе “Прерывания” указано нужно ли использовать прерывания при приеме (передача и тест осуществляется без использования прерывания).

Если указано, что надо использовать прерывания, необходимо написать обработчик прерывания и запрограммировать контроллер последовательного интерфейса и контроллер прерываний в необходимый режим. Если нет, то необходимо проверять доступность принимаемых данных, используя бит DR регистра LSR.

## 

# *Приложения*

# *Приложение 1*

Таблица 1

# Терминальные функции Win32

|  |  |  |
| --- | --- | --- |
| **Функция** | **Описание** | |
| AllocConsole | Создать новый терминал для вызывающего процесса | |
| CreateConsoleScreenBuffer | Создать буфер экрана для терминала | |
| FillConsoleOutputAttribute | Устанавливает цвет символов и фона для указанного числа текстовых ячеек | |
| FillConsoleOutputCharacter | Выводит символ на экран указанное число раз | |
| FlushConsolelnputBuffer | Очищает входной буфер терминала | |
| FreeConsole | Отключает терминал от вызывающего процесса | |
| GenerateConsoleCtrlEvent | Посылает указанный сигнал группе обработки терминала, совместно использующей терминал, назначенный вызывающему процессу | |
| GetConsoleCP | Определяет номер входной кодовой страницы, которая используется в терминале, назначенной вызывающему процессу | |
| GetConsoleCursorlnfo | Определяет информацию о размере и внешнем виде курсора для указанного экранного буфера терминала | |
| GetConsoleMode | | Определяет текущий режим ввода для входного буфера терминала или текущий режим вывода для экранного буфера терминала | |
| GetConsoleOutputCP | | Определяет номер выходной кодовой страницы, которая используется в терминале, назначенной вызывающему процессу | |
| GetConsoleScreenBufferlnfo | | Определяет информацию об указанном экранном буфере терминала | |
| GetConsoleTitle | | Возвращает строку заголовка для текущего окна терминала | |
| GetConsoleWindow | | Определяет дескриптор окна, используемого для терминала, которая назначена вызывающему процессу | |
| GetLargestConsoleWindowSize | | Возвращает размер максимально возможного окна терминала | |
| GerNumberOfConsoleInputEvents | | Возвращает число непрочитанных введенных записей во входном буфере терминала | |
| GetNumberOfConsoleMouseButtons | | Возвращается число кнопок мыши, используемой в текущем терминале | |
| GetStdHandle | | Возвращается дескриптор для стандартного устройства ввода, стандартного устройства вывода либо устройства вывода сообщений об ошибках | |
| HandlerRoutine | | Функция, определяемая в пользовательской программе, которая используется совместно с функцией SetConsoleCtrlHandler | |
| PeekConsoleInput | | Читает данные из указанного входного буфера терминала без удаления их из буфера | |
| ReadConsole | | Читает введенные символы из указанного входного буфера терминала и удаляет их из буфера | |
| ReadConsolelnput | | Читает данные из указанного входного буфера терминала и удаляет их из буфера | |
| ReadConsoleOutput | | Читает символы и цветовые атрибуты из указанного прямоугольного блока символьных ячеек буфера экрана терминала | |
| ReadConsoleOutputAttribute | | Копирует указанное количество цветовых атрибутов символов и фона из последовательности символьных ячеек буфера экрана терминала | |
| ReadConsoleOutputCharacter | | Копирует указанное количество символов из последовательности символьных ячеек буфера экрана терминала | |
| ScrollConsoleScreenBuffer | | Перемешает блок данных в буфере экрана терминала | |
| SetConsoleActiveScreenBuffer | | Назначает указанный экранный буфер  в качестве отображаемого в настоящий момент  буфера экрана терминала | |
| SetConsoleCP | | Назначает номер входной кодовой страницы для терминала, назначенному вызывающему процессу | |
| SetConsoleCtrlHandler | | Добавляет или удаляет процедуру обработки терминала HandlerRoutine в список (или из списка) функций обработки вызывающего процесса | |
| SetConsoleCursorlnfo | | Устанавливает размер и внешний вид курсора для указанного буфера экрана терминала | |
| SetConsoleCursorPosition | | Устанавливает курсор в указанную позицию текущего буфера экрана терминала | |
| SetConsoleMode | | Устанавливает текущий режим ввода для входного буфера терминала или текущий режим вывода для экранного буфера терминала | |
| SetConsoleOutputCP | | Устанавливает номер выходной кодовой страницы, которая используется в терминале, назначенному вызывающему процессу | |
| SetConsoleScreenBufferSize | | Изменяет размер указанного буфера экрана терминала | |
| SetConsoleTextAttribute | | Устанавливает атрибуты цвета символов и фона, выводимых на экран терминала | |
| SetConsoleTitle | | Изменяет строку заголовка для текущего окна терминала | |
| SetConsoleWindowInfo | | Устанавливает размер и положение окна буфера экрана терминала | |
| SetStdHandle | | Устанавливает дескриптор для устройства стандартного ввода, стандартного вывода и стандартного устройства вывода сообщений об ошибках | |
| WriteConsole | | Выводит строку символов в буфер экрана терминала, начиная с текущего положения курсора | |
| WriteConsoleInput | | Записывает данные напрямую во входной буфер терминала | |
| WriteConsoleOutput | | Записывает символы и цветовые атрибуты в указанный прямоугольный блок символьных ячеек буфера экрана терминала | |
| WriteConsoleOutputAttribute | | Копирует указанное количество цветовых атрибутов символов и фона в последовательность символьных ячеек буфера экрана терминала | |
| WriteConsoleOutputCharacter | | Копирует указанное количество символов  в последовательность символьных ячеек буфера  экрана терминала | |

**Терминал.** Это 32-разрядное окно командной строки системы Windows, на котором можно отображать цветные текстовые строки. С точки зрения программы пользо­вателя оно работает в текстовом режиме. По умолчанию размер окна установлен в 25 строк по 80 колонок в каждой строке.

**Стандартное устройство ввода**. По умолчанию таким устройством является кла­виатура, хотя его всегда можно переопределить из командной строки при запуске приложения, указав, например, имя файла или последовательный порт.

**Стандартное устройство вывода**. По умолчанию таким устройством является мо­нитор, хотя его всегда можно переопределить из командной строки при запуске приложения, указав, например, имя файла, параллельный или последовательный порт.

**Функция SetConsoleTitle**

Эта функция позволяет изменить содержимое строки заголовка окна терминала.

.data

titleStr BYTE "Заголовок окна",0

.code

INVOKE SetConsoleTicle, ADDR titleStr

**Функция GetConsoleScreenBufterinfo**

Эта функция возвращает информацию о текущем состоянии окна терминала. Ей передается два параметра: дескриптор терминала и адрес структуры, в которую помещается разнообразная информация о текущем состоянии окна терминала. Вот ее прототип:

GetConsoleScreenBufferlnfo PR0TO,

outHandle: DWORD, ; Дескриптор буфера экрана терминала

pBufferInfo: PTR CONSOLE\_SCREEN\_BUFFER\_INFO

**Функция SetConsoleScreenBufferSize**

Эта функция позволяет задать размер буфера экрана в виде количества столбцов и строк. Вот ее прототип:

SetConsoleScreenBufferSize PR0T0,

outHandle:DWORD, ; Дескриптор вывода на терминал

dwSize:COORD ; Новый размер буфера экрана

**Управление курсором**

Среди функций Win32 API предусмотрены также функции для изменения размера, внешнего вида и положения на экране курсора. В них используется важная структура данных, называемая CONSOLE\_CURSOR\_INFO, с помощью которой указываются пара­метры курсора. Вот ее определение:

CONSOLE\_CURSOR\_INFO STRUCT

dwSize DWORD ?

bVisible DWORD ?

C0NSOLE\_CURSOR\_INF0 ENDS

В поле dwSize структуры указывается размер курсора в процентах (число от I до 100) относительно высоты символьной ячейки. Если значение поля bVisible истинно, кур­сор отображается на экране.

Функция GetConsoIeCursorlnfo

Эта функция возвращает информацию о размере курсора и виден ли он на экране или нет. Кроме дескриптора терминала ей передается объектная переменная типа

CONSOLE\_CURSOR\_INFO:

GetConsoIeCursorlnfo PROTO,

outHandle: DWORD, ; Дескриптор терминала

pCursorInfo: PTR CONSOLE\_CURSOR\_INFO ; Параметры курсора

По умолчанию размер курсора равен 25. Это означает, что курсор будет занимать 25% символьной ячейки.

**Функция SetConsoleCursorlnfo**

С помощью этой функции можно задать размеры курсора и отобразить или скрыть его на экране. Кроме дескриптора терминала, ей передается объектная переменная типа CONSOLE\_\_CURSOR\_INFO:

SetConsoleCursorlnfo PROTO,

outHandle: DWORD, ; Дескриптор терминала

pCursorlnfo: PTR CONSOLE\_CURSOR\_INFO ; Параметры курсора

**Функция SetConsoleCursorPosition**

Эта функция задает горизонтальную X и вертикальную Y координаты положения кур­сора на экране. В качестве параметров ей передаются выходной дескриптор терминала и структурная переменная типа COORD:

SetConsoleCursorPosition PROTO,

outHandle: DWORD, ; Дескриптор терминала

coords: COORD ; Координаты X,Y положения курсора

**Изменение цвета текста**

Существует два способа изменения цвета текста, отображаемого в окне терминала. Во-первых, вы можете изменить текущий цвет текста, вызвав функцию SetConsoleTextAttribute, что повлияет на все последующие операции вывода текста на тер­минал. Во-вторых, можно установить цветовые атрибуты определенных ячеек на экране, вызвав функцию WriteConsoleOutputAttributa.

**Функция SetConsoleTextAttribute**

С помощью этой функции задаются цвета символов и фона, что повлияет на все по­следующие операции вывода текста на терминал. Вот прототип функции:

SetConsoleTextAttribute PROTO,

outHandle: DWORD, ; Дескриптор терминала

nColor: DWORD ; Цветовые атрибуты

Значение атрибутов цвета хранится в младшем байте параметра nColor. Цвета коди­руются точно так же, как и при работе с видеофункциями BIOS.

**Функция WriteConsoIeOutputAttribute**

Эта функция копирует массив цветовых атрибутов в последовательность символьных ячеек буфера экрана терминала, начинающийся с указанной позиции. Вот ее прототип:

WriteConsoleOutputAttribute PROTO,

outHandle: DWORD, ; Дескриптор терминала

pAttribute: PTR WORD, ; Адрес массива атрибутов

nLength: DWORD, ; Число ячеек

xyCoord: COORD, ; Координаты первой ячейки

lpCount: PTR DWORD ;Переменная, содержащая реальное число записанных ячеек

Параметр pAttribute — это адрес массива слов, каждый элемент которого содержит в младшем байте цветовые атрибуты для соответствующей ячейки буфера экрана. Длина массива задается в параметре nLength. Координаты начальной ячейки в буфере экрана задаются с помощью параметра xyCoord. После вызова функции переменная, адрес ко­торой указан в параметре lpCount, будет содержать реальное число записанных ячеек.

# *Приложение 2*

**ЛИТЕРАТУРА**

* 1. Агуров П.В. Последовательные интерфейсы ПК. Практика

программирования - СПб.: БХВ-Петербург. 2004. - 496 с.

1. Вегнер В.А, Крутяков А.Ю. и др. Аппаратура персональных компьютеров

и её программирование. М: Радио и связь, 1995.

1. Гук М. Аппаратные интерфейсы ПК. Энциклопедия.- СПб: Питер, 2002. – 404с.
2. Ирвин Кип. Язык ассемблера для процессоров Intel, 4-ое издание.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2005. – 912 с.
3. Кулаков В. Программирование на аппаратном уровне: специальный справочник. 2-е издание. – СПб.: Питер, 2003.
4. Магда Ю.С. Программирование последовательных интерфейсов. - СПб.: БВХ-Петербург, 2009. – 304 с.
5. Новиков Ю.В., Калашников О.А., Гуляев С.Э. Разработка устройств сопряжения для персонального компьютера типа IBM PC, под редакцией Ю.В.Новикова. Практическое пособие – М.: ЭКОМ., 1997 – 224 с.
6. Рудаков П.И., Финогенов К.Г. Язык ассемблера: уроки программирования. – М.:ДИАЛОГ – МИФИ, 2001.-640 с.
7. Сван Т. Освоение Turbo Assembler.– К.: Диалектика, 1996.– 544с.
8. Скенлон Л. Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблера: Пер. с англ. - Москва: Радио и связь. 1989г., 336с.
9. Финогенов К.Г. Использование языка Ассемблера. Учебное пособие для вузов. - М.: Горячая линия –Телеком, 2004.