http://radio-mir.com
Индексы: 48925, 45995
7•2002

радиомир

Ваш компьютер

В НОВЫЙ ВЕК —
С НОВЫМ НАЗВАНИЕМ!

NEW CLIP

ДЕТИ
1000 ФОТО

БОЛЬШАЯ ЭНЦИКЛОПЕДИЯ ГЕОГРАФИЧЕСКИХ
БЛИЖНЕЕ ЗАРУБЕЖЬЕ
БЕЛАРУС

100% проверено PETROSOFT ВИРУСОВ нет

Microsoft
РУССКАЯ ВЕРСИЯ
Microsoft® Windows 2000
erver & Professional
Service Pack 2

PRESTIGE CLASSICS
Коллекция Классической Музыки
MOZART • SOUSA • BACH • GRIEG • LISZT
2 CD

СКОРАЯ ПОМОЩЬ ДЛЯ ВАШЕЙ СИСТЕМЫ
РЕАНИМАТОР
Факс-Модем и Интернет 2002
на диске:
- АОН и АнтиАОН
- Браузеры
- Все для общения
- Доступ до Интернет
- Скачка файлов
- ФАКС-программы
- FTP Клиенты
- Голосовое общение
- Почта
- Мультимедиа
- Инструменты
- WEB-дизайн

и другие

COMPACT disc DIGITAL DATA

ПОЗНАВАТЕЛЬНАЯ
ЭНЦИКЛОПЕДИЯ
Тайна Египетских Пирамид

ПОЛНЫЙ РУССКИЙ ПЕРЕВОД
CHESSMASTER 8000
Chessmaster 8000 - полный компьютеризированный шахматный симулятор с разными уровнями сложности для всех возрастных групп.
Забавный и удобный, Chessmaster 8000 отли-
ром к шахматам,
гр Chessmaster
удет интересно
System Requirements: P 150, Windows 95/98
2 в 1

РУССКАЯ и АНГЛИЙСКАЯ версии
Easy CD Creator
5.0 PLATINUM

mp3 коллекция
КИНО
диск 1

Красная акула

Учредитель и издатель:
ООО "Радиомир Пресс"

Свидетельство о регистрации
ПИ №77-9841 от 17.09.2001

Главный редактор
Ольга СТРЫЖАНКОВА

Зам. гл. редактора
Иван БЕЛЬСКИЙ

Редколлегия:
Сергей ДРОЗДОВСКИЙ,
Алексей КОНДРАШОВ,
Анатолий КОРБИТ,
Владимир КУЦЕНКО,
Елена ЛЕВИТМАН,
Николай МЕДВЕДЬ,
Геннадий ПЕЧЕНЬ,
Геннадий ШУЛЬГИН

Контактные телефоны:
в Москве (095) 105-99-89,
в Минске (017) 249-41-47

Компьютерная верстка —
Янина БЕЛЬСКАЯ

Техническая графика —
Наталья БЕЛЬСКАЯ,
Александр ГОРАЮТИН,
Мария КАЛАБУХОВА

Оформление обложки —
Наталья БЕЛЬСКАЯ,
Надежда БОГОМОЛОВА

Адреса для писем:
119454, РФ, г.Москва, а/я 37;
220095, РБ, г.Минск-95, а/я 199

E-mail: rl@radiopage.by
rm@radio-mir.com
WWW: http://radio-mir.com

Наши платежные раквизиты:
получатель: ООО "Радиомир Пресс",
ИНН 7729404741,
р/с 40702810900010000084
в ООО КБ "Агролромкредит"
ф-л Центральный, г.Москва,
корр. счет 30101810500000000109
БИК 044525109
Адрес банка: 113054, г.Москва,
ул.Зацепа, 43Б

Материалы для публикации принимаются в рукописном, печатном и электронном вариантах

Требования к графическим материалам рекламного хврактара в электронном виде: CorelDRAW до 9.0, все шрифты в кривых; bitmaps 300 dpi; TIFF 300 dpi; CMYK. Приложить печатную копию.

За достоверность рекламной и другой публикуемой информации несут ответственность рекламодатели и авторы. Мнение редвкции не всегда совпадает с мнениями авторов

Адрес редакции: 119454, РФ, г.Москва,
ул.Коштоянца, 6 — 233, тел. (095) 105-99-89

Подписано к печати 24.05.2002 г.
Формат 60 x 84 1/8.
Печать офсетная. 6 печ. л.
Цена свободная.

Отпечатано в типографии
ЗАО "Красногорская типография".
Заказ №1377.

радиомир
Ваш компьютер

ЕЖЕМЕСЯЧНЫЙ МАССОВЫЙ ЖУРНАЛ
С 1995 г. по 6/2001 г. издавался под названием
"Радиолюбитель. Ваш компьютер"

№7/июль/2002

ЧИТАЙТЕ В НОМЕРЕ

Полные листинги некоторых программ расположены по адресу
http://radio-mir.com

А.ИГНАТЕНКО,
E-mail: SRLSA@bsu.by

От шредингеровского кота к квантовому компьютеру

Введение

Сегодня мы встречаем компьютеры повсеместно — большие и маленькие, мощные и не очень. Выполняя множество различных вычислений, они помогают человеку в решении огромного круга задач, которые сам он не мог бы выполнить столь же быстро, точно и качественно. Порой современные вычислительные машины кажутся всесильными. Однако существует определенный круг задач, с которыми обычные компьютеры справляются еле-еле, а зачастую и вообще не поддаются решению. Самая, пожалуй, известная "тяжелая" задача для классического компьютера — это факторизация числа, т.е. разложение его на простые множители. Например, с факторизацией 129-значного числа 1600 современных компьютеров справились в течение 8 месяцев. Долго, не правда ли? А напрямую с такой задачей связано нахождение периода последовательности. Или еще пример сложной задачи — поиск в неупорядоченной базе данных. Хорошо, если в ней несколько сотен или тысяч элементов, а когда база насчитывает, например, миллиард элементов, то простым перебором можно и не обойтись. Полное решение этих проблем может прийти из области квантовой механики, при условии, что будет создан так называемый квантовый компьютер. Действительно, достаточно быстрое развитие квантовой механики привело к созданию целой новой науки под названием квантовая информатика, в которой рассматриваются три относительно самостоятельных раздела: квантовая криптография, квантовая телепортация и квантовые вычисления (компьютинг). В этом кратком обзоре мы лишь коснемся квантового компьютинга и некоторых связанных с ним понятий.

Суперпозиция состояний

В квантовой механике наблюдается большое количество явлений, которые не имеют аналогов в классической физике и вообще в окружающем нас мире. Например, если у нас есть некий прибор, то мы всегда можем определить, включен он или нет, т.е. практически очень трудно представить себе его включенным и выключенным одновременно. Другими словами, если в окружающем нас мире существует некая система, у которой есть два взаимно исключающих состояния, то она всегда будет находиться в каком-то одном из них. В квантовом мире — другая ситуация.

Поясним на конкретном примере. Возьмем любую двухуровневую систему, например, электрон, у которого спин может быть направлен "вверх" или "вниз" относительно, скажем, оси z. Известно, что состояние квантовой системы может быть описано волновой функцией, которая обладает свойством суперпозиции (здесь намеренно не употребляется термин "любая квантовая система", так как существуют и такие системы, которые нельзя описать волновой функцией). Это означает, что если электрон имеет состояние, в котором его спин направлен "вверх", и состояние со спином, направленным "вниз", то существует состояние, являющиеся суперпозицией двух предыдущих.

Представить такую ситуацию нелегко. Однако с точки зрения математики такое состояние системы описывается очень просто:

$$|\Psi\rangle = a|1\rangle + b|2\rangle.$$

В левой части равенства стоит волновая функция (такие скобки показывают, что на самом деле это вектор в некотором пространстве), а в правой части — суперпозиция двух состояний с соответствующими вероятностями. Числа a и b в общем случае комплексные, и сумма их квадратов должна равняться 1.

Однако и тут все не так просто. При попытке измерить такое состояние, оно разрушится. Пусть у нас есть прибор, который настроен на определение состояния 2. Тогда при взаимодействии прибора и нашей квантовой системы квантовая суперпозиция исчезнет, а уже другая волновая функция будет просто описывать состояние 2. Как мы видим, воздействие прибора приводит к необратимым изменениям структуры самой системы.

Чтобы немного оживить описанную картину, можно привести пример "шредингеровского кота". Его придумал сам Шредингер для иллюстрации явления суперпозиции. Пусть у нас есть закрытый ящик, в котором находится счетчик Гейгера, и ампула с ядом. Если счетчик улавливает частицу, запускается специальный механизм, и ампула разбивается. Поместим в ящик кота. Так как ящик закрыт, то мы не имеем ни малейшего представления о том, что происходит внутри — разбилась ампула или нет. С точки зрения квантовой механики данная ситуация описывается просто как суперпозиция живого и мертвого кота. А теперь откроем ящик. Понятно, что мы не сможем увидеть такую суперпозицию, как бы это ни было заманчиво — нам предстанет либо печальная картина, либо живой кот. Противоречие? Совсем нет, если вспомнить, что открыв ящик, мы тем самым произвели измерение системы, находящейся внутри его, и суперпозиция перешла в одно конкретное состояние.

Перепутанные состояния

Понятие "перепутанных состояний" (entangled states) вводится для описания систем, которые образованы из нескольких частиц, возможно, и пространственно разделенных. Шредингер называл перепутанными состояниями такие, при которых полное знание о состоянии всей системы не соответствует такому же полному знанию о состоянии ее частей.

Здесь можно привести классический пример. Пусть у нас есть двухуровневый атом и некоторое поле. В определенный момент времени атом пролетает область взаимодействия с данным полем. После непосредственного контакта атом и

поле оказываются пространственно разделенными. При этом состояние системы "атом плюс поле" оказывается перепутанным, так как состояние, скажем, поля зависит от того, в каком состоянии находится атом. Это явление можно описать следующим образом:

$$|\Psi\rangle = |атом\rangle_1|поле\rangle_1 + |атом\rangle_2|поле\rangle_2.$$

Если в данном случае следовать формулировке Шредингера, то действительно, нам известно, что атом и поле взаимодействовали, однако такое полное знание об общем состоянии системы не соответствует нашему знанию об отдельных частях (неизвестны конкретные состояния атома и поля). Следует отметить и тот факт, что такие состояния живут, как правило, в течение времени, большего, чем время взаимодействия.

Декогеренция

Декогеренция — процесс, при котором система перепутывается с состоянием своего окружения. Можно сказать, что информация о состоянии системы "записывается" в состоянии окружения. При этом теряется информация о связи коэффициентов (весов) состояний. Это означает, что если первоначально имела место некоторая суперпозиция состояний (с соответствующими вероятностями), то после декогеренции окружением суперпозиция станет смешанным состоянием, причем с измененными весами. Возвращаясь к шредингеровскому коту и атому, можно представить себе ситуацию следующим образом — атом рассмотреть как систему, а кота как окружение, тогда атом находится в смешанном состоянии со своим окружением. В качестве системы можно рассматривать и кота, тогда атом — окружение. Кстати, именно из-за существования явления декогеренции нам не удается наблюдать суперпозицию живого и мертвого котов, так как первоначальная суперпозиция запутывается с окружающей средой во время измерения.

Квантовый компьютер

В этом разделе мы рассмотрим модель компьютера, совсем не похожего на классического собрата. Шредингеровский же кот, то ли живой, то ли мертвый, поможет нам понять, как он действует.

Прежде всего вспомним, что единицей информации в классической теории является один бит, а его физическим носителем может быть любое устройство с двумя состояниями — выключатель, триггер или одноразрядный регистр. Обычно состояниям "выключено" и "включено" ставятся и соответствие значения 0 и 1 соответственно. Совершенно очевидно, что в регистре объемом n бит мы может разместить только одну последовательность нулей и единиц, а следовательно, только одно число. А теперь давайте представим себе, что в качестве носителя единицы информации выступает электрон. Пусть его состояние со спином "вверх" — это 1, а со спином "вниз" — 0. И первому, и второму состоянию соответствует своя волновая функция, но будет существовать и дополнительное состояние — суперпозиция 0 и 1. Таким образом, мы пришли к интересному выводу — с помощью такого квантового бита мы можем представить два числа одновременно!

Квантовый бит получил название "кубит". Нетрудно посчитать, что в квантовый регистр, подобный описанному выше, можно записать до 2^n чисел одновременно. Вдобавок, мы получаем возможность оперировать всем этим количеством чисел одновременно, что приведет к экспоненциальному увеличению быстродействия компьютера. Строго говоря, состояние квантового компьютера будет представлять собой очень сложное, запутанное состояние.

Такой подход предоставляет много возможностей. В частности, гораздо быстрее решается задача факторизации или нахождения периода длинной последовательности, а также поиска в полностью неупорядоченной базе данных. Уже разработаны алгоритмы, решающие такие задачи для квантового компьютера.

Однако не следует думать, что квантовый компьютер всесилен. В действительности он способен эффективно справляться с задачами, которые не требуют проверки экспоненциально большого количества решений. У квантового компьютера есть и еще одна интересная особенность — его вероятностный характер. Это означает, что решение задачи, в принципе, можно получить и за один шаг, но вероятность того, что оно верное, будет зависеть от сложности задачи. Также верно и то, что решение может быть получено со сколь угодно высокой вероятностью.

На данном этапе развития этого направления реально стоит вопрос о принципиальной возможности построения квантового компьютера. Главной проблемой является декогеренция. Взаимодействие с окружающей средой будет губительным для состояния компьютера, следовательно, перед создателями встает требование его полной изоляции от среды. Решение может быть найдено, если использовать в качестве кубитов, скажем, ядра атомов, которые хорошо изолированы вследствие особенностей физического строения, или ионы в магнитных ловушках. Следует помнить и о том, что квантовое состояние нельзя копировать. Измерение или наблюдение уже само по себе создает копию, следовательно, отсюда вытекает еще одно свойство квантового компьютера — его нельзя наблюдать вплоть до окончания вычислений. Непонятно также, каким образом можно будет поддерживать сложное состояние многих кубитов (для реальных вычислений их требуется порядка 50...100, но и такое их число пока невозможно контролировать).

Заключение

Проблем на пути создания квантового компьютера немало, однако шаги к их решению предпринимаются уже не один год. Сейчас созданы опытные модели квантовых компьютеров, состоящих только из семи кубитов, которые, однако, справились со своими первыми задачами — факторизовали число 15 и затратили на поиск элемента в неупорядоченной базе данных из четырех элементов всего один шаг. Быть может, эти маленькие успехи покажутся смешными, но квантовый компьютер делает только первые шаги. Трудно предсказать, смогут ли в ближайшие годы появиться достаточно мощные квантовые вычислительные машины, но сама попытка построить их может привести как к другим очень важным результатам, так и к новому пониманию сложных процессов, происходящих в загадочном квантовом мире.

Возможности библиотек Speech SDK: диалог с компьютером на человеческом языке

А.ТОМИЛИН.
E-mail: SRLSA@bsu.by

Сегодня мы хотим от компьютера чего-то большего, чем просто умения проводить вычисления сложных математических выражений. Век коммуникаций и интерактивности выдвигает свои требования к программным продуктам и расставляет свои акценты.

Уже сейчас, благодаря совместным усилиям программистов крупнейших компаний, занимающихся разработкой программного обеспечения, и богатому опыту, накопленному в научно-исследовательских лабораториях многих стран мира, речевые технологии, еще недавно казавшиеся прерогативой научно-фантастических произведений, становятся доступными для широкого использования в реальных программных приложениях.

Механизм распознавания речи обогащает ваш компьютер способностью к восприятию разговорной речи, то есть способностью определения того, что было сказано, и преобразованию этого в письменное представление. Синтез же речи представляет собой процесс воспроизведения текста "голосом" вашего компьютера.

В общих чертах, для того чтобы научиться синтезировать и распознавать человеческую речь, компьютер должен уметь выполнять ряд операций. Например, в случае синтеза речи этот ряд операций складывается из следующих пунктов:

- структурный анализ — обработка текста на предмет выделения слов, предложений, параграфов;
- предварительная обработка — анализ текста на наличие специальных языковых конструкций (аббревиатур, чисел, дат, времени и др.) и их преобразование с целью получения разговорных эквивалентов;
- преобразование слов в фонемы — звуковые единицы языка;
- сегментация речи — выбор тона, скорости произношения, величин пауз для реальности звучания воспроизводимой речи;
- воспроизведение составленных фонемных цепочек.

При реализации механизма распознавания речи можно выделить следующие этапы:

- составление грамматики — задание распознаваемых слов и шаблонов их использования;
- цифровая обработка — анализ спектральных характеристик входного звукового потока;
- фонемное распознавание — сравнение спектральных характеристик фонем, полученных на предыдущем этапе, со спектральными характеристиками фонем распознаваемого языка;
- распознавание слов — сравнение последовательностей выделенных фонем со словами и шаблонами из грамматики;
- результирующий вывод слов, отобранных распознавателем.

Какие же спектральные характеристики выбирать, по какому принципу их сравнивать? К настоящему моменту, в результате большого объема теоретических и экспериментальных исследований, было установлено, что наиболее эффективным способом построения реальных систем распознавания и синтеза речи является использование аппарата скрытых марковских моделей. Широкое применение скрытых марковских моделей обусловлено, прежде всего, их мощной и гибкой математической структурой, позволяющей легко адаптировать их к решению самых различных специфических задач обработки речевых сигналов, таких как распознавание изолированных слов с большим словарем, задач дикторонезависимого распознавания слитной речи и др.

Этот механизм основывается на том, что предварительно каждой речевой единице ставится в соответствие марковская модель, характеризуемая определенным набором характеристик. Считается, что каждая модель может пребывать в одном состоянии из конечного числа возможных. Полученные модели формируют так называемую кодовую книгу. Далее, для каждого неизвестного речевого элемента с помощью вероятностных методов ищется наиболее подходящая модель кодовой книги, а соответствующая этой модели речевая единица принимается за результат поиска.

Именно аппарат скрытых марковских моделей, как наиболее эффективный, послужил физической основой реализации многофункциональной библиотеки Speech SDK компании Microsoft. Библиотека содержит ревлизацию звуковых движков распознавания и синтеза речи, идентификации диктора по его голосу, а также набор функций на языке C++ для их использования в прикладных программах.

К настоящему времени, кроме упомянутого выше движка от Microsoft, на компьютерном рынке появилось достаточно большое количество звуковых движков других производителей (IBM, Lernout&Hauspie и др). С точки зрения абстрактного взгляда пользователя — все это движки, реализующие определенную, общую для всех функциональность. А следовательно, не существует ли какого-либо общего механизма настройки и управления ими, единого прикладного интерфейса программирования?

Ответом на этот вопрос служит Java Speech API, разработанный компанией Sun Microsystems и представляющий собой набор кросс-платформенных программных интерфейсов, предназначенных для поддержки распознавания и синтеза человеческой речи на языке Java. Хотелось бы акцентировать внимание на том, что Java Speech API не является голосовым движком непосредственно, это именно набор интерфейсов, характеризующих основные свойства и функциональные возможности движков, дающих пользователям возможность унифицированного доступа к настройкам и использования голосовых движков различных производителей.

Основные функциональные элементы звукового движка содержатся в классах и интерфейсах пакета javax.speech. Более специфичная информация, характерная для распознавателя или синтезатора речи, может быть найдена в пакетах javax.speech.recognition и javax.speech.synthesis.

Создание движка начинается с обращения к классу-координатору Central из пакета javax.speech:

- Synthesizer Central.createSynthesizer(EngineModeDesc mode);

- Recognizer Central.createRecognizer(EngineModeDesc mode).

Используя переменную mode, можно указать язык распознавания или настроить "голос" синтезатора (регулируя возраст, пол, скорость воспроизведения и многое другое).

Простейшим способом заставить движок сказать что-нибудь, является вызов функции синтезатора speak, передавая ей в качестве параметра обычную текстовую строку:

```
Synthesizer synth = Central.createSynthesizer(new SynthesizerModeDesc(Locale.ENGLISH));
// подготовка к воспроизведению
synth.allocate();
synth.resume();
// произнести строку "Hello"
synth.speakPlainText("Hello!", null);
// дожидаемся окончания воспроизведения
synth.waitEngineState(Synthesizer.QUEUE_EMPTY);
// освобождаем ресурсы
synth.deallocate();
```

Того же результата можно достичь с помощью строки, отформатированной в соответствии со стандартом разметки произносимой речи JSML (Java Speech Markup Language). В частности, чтобы синтезатор понял, что передаваемый ему на вход набор цифр является датой, можно воспользоваться тэгом <sayas class="date"> Дата </sayas>.

Несмотря на свою простоту, синтезатор очень гибок в настройке. За счет этого можно добиться потрясающей реальности воспроизведения голоса.

Более сложным как с точки зрения реализации, так и с точки зрения управления, является процесс распознавания речи. Однако Java Speech API предоставляет достаточно гибкий подход к использованию и настройке движков распознавания речи.

Уже сейчас многие производители поддерживают стандарт, разработанный Sun. Благодаря же Джонатану Киннерсли (Jonathan S.Kinnersley) из компании Cloudgarden, разработавшему мост между Microsoft Speech SDK и Java Speech API, любители языка Java получили доступ к движку от Microsoft, который и использовался автором для исследований возможностей общения с компьютером на человеческом языке.

Литература

1. Ronald A. Cole. Survey of the State of the Art in Human Language Technology (http://cslu.cse.ogi.edu/HLTsurvey/)
2. http://java.sun.com/
3. http://www.microsoft.com/

А.ГРИНЧУК,
г.Минск, РИВШ БГУ,
E-mail: gav@nihe.niks.by

Работа в системе Mathematica 3.0/4.0.

Основы программирования в Mathematica 3.0/4.0

Основным режимом работы в системе Mathematica является диалоговый режим, т.е. общение пользователя с программой происходит по принципу "вопрос—ответ". Между тем, система имеет встроенные программные средства, позволяющие на равных конкурировать со многими языками программирования. В данной статье мы рассмотрим основные конструкции языка программирования и сохранение программ во внешних файлах.

Наверное, одним из лидеров по использованию является условное выражение (условный переход). Создатели Mathematica не стали отступать от традиции и воспользовались обозначением If. Эта команда имеет три основные формы:

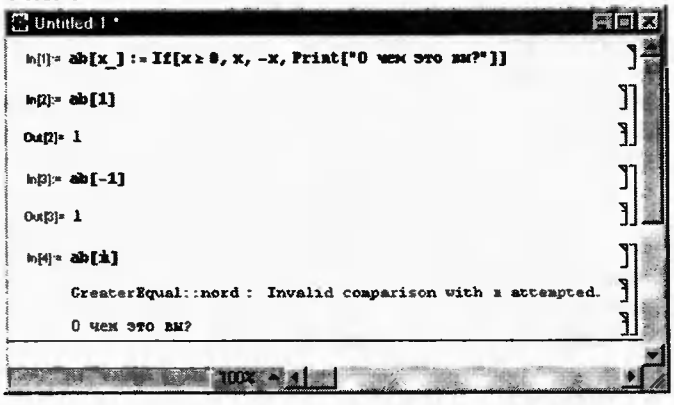
- **If**[условие, выражение_1];
- **If**[условие, выражение_1, выражение_2];
- **If**[условие, выражение_1, выражение_2, выражение_3].

При выполнении условия (TRUE—ИСТИНА) осуществляется переход к выражению 1, при невыполнении (FALSE—ЛОЖЬ) — к выражению 2. Если же при проверке условия не было получено ни TRUE, ни FALSE, осуществляется переход к выражению 3. В качестве примера определим команду для вычисления модуля действительного числа:

ab[x_] := **If**[x >= 0, x, -x, Print["О чем это вы?"]].

Для комплексных чисел, как известно, операция сравнения (больше или меньше) не определена, поэтому после системного сообщения об ошибке Mathematica переходит к третьему выражению — выводу сообщения "О чем это вы?" **(рис.1)**. Естественно, допускаются вложения условных операторов друг в друга, т.е. структуры вида **If**[..., **If**[ ], **If**[ ]].

Рис. 1

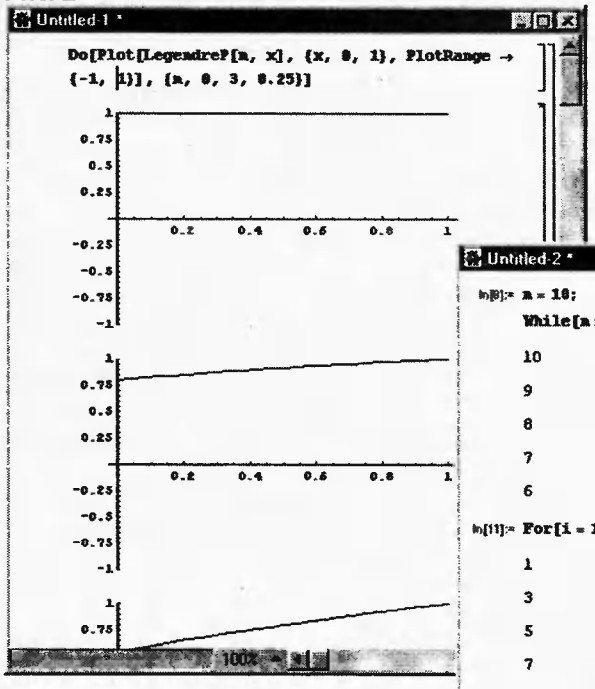


Точно так же соблюдаются традиции и в отношении циклов Do, For, While. Например, команда

Do[Plot[LegendreP[n, x], {x, 0, 1}, PlotRange ? {-1, 1}], {n, 0, 3, 0.25}]

приводит к построению графиков функций Лежандра порядков от 0 до 3 с шагом 0.25. Интересно, что этот набор

Рис. 2



графиков также анимируется двойным щелчком мыши по одному из кадров **(рис.2)**.

Цикл While имеет следующую структуру — **While[**условие, тело цикла; изменение управляющей переменной**]**. Например, после выполнения команд:

n = 10;

While[n > 5, Print[n] ; n-=1**]**

на экране появится набор чисел от 10 до 6 **(рис.3)**. Наверное, единственный необходимый здесь комментарий — это "расшифровка" команды n-=1. Эта команда уменьшает (знак "-") значение n на 1. Аналогично, команда n+=1 увеличивает n на 1 на каждом шаге, n*=2 — удвоение и т.д. Похожим образом организована и команда For:

For[i = 1, i <= 9, Print[i]; i += 2**]** — вывод чисел от 1 до 6 с шагом 2 **(рис.3)**.

Если судить по командам условных переходов и циклов, то Mathematica, казалось бы, не дает никакого преимущества по сравнению с обычными языками программирования. Однако нужно иметь в виду, что внутри этих команд можно использовать весь (и очень широкий) спектр математических возможностей системы. В качестве примера рассмотрим небольшую программу для поиска парных простых чисел (т.е. простых чисел, отличающихся на 2: 5 и 7, 17 и 19). В Mathematica имеется команда для вычисления i-го простого числа — **Prime[i]**. Для проверки используется условное выражение If[Prime[i + 1] - Prime[i] == 2, Print[Prime[i], " " Prime[i + 1]]]. Цикл For перебирает все простые числа от 1000-го до 1100-го **(рис.4)**:

For [i = 1000, i <= 1100, If[Prime[i + 1] - Prime[i] == 2, Print[Prime[i], " ", Prime[i + 1]]]; i += 1].

Часто внутри цикла требуется выполнить целую последовательность команд, например:

x = a

t = (x + 2)^3

t = Expand[t].

Эту последовательность можно объединить в одной функции, используя круглые скобки, а в качестве разделителя — точку с запятой:

f[x_] := (t = (x + 2)^3; t = Expand[t]; t).

Однако при этом переменная t является глобальной, и после завершения вычислений t принимает новое значение. Для вычислений с локальными переменными используются блоки и модули, выполняющие примерно ту же роль, что и подпрограммы в традиционных языках программирования. Опять-таки, последовательные шаги вычислений разделяются точкой с запятой, локальные переменные указываются в фигурных скобках:

Рис. 3

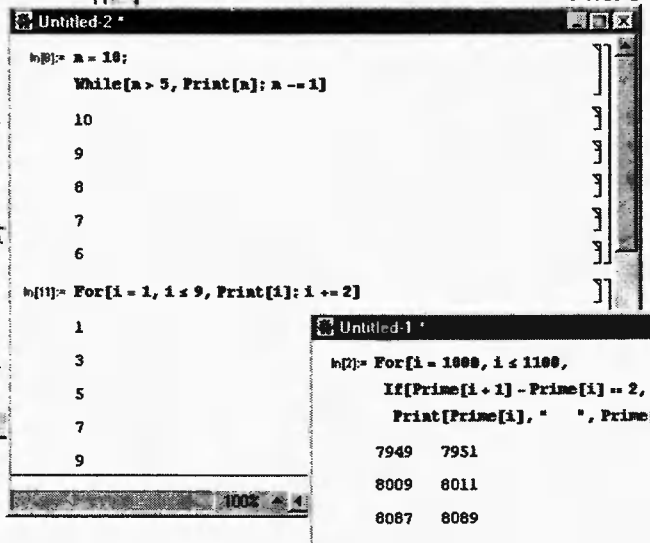


Рис. 4

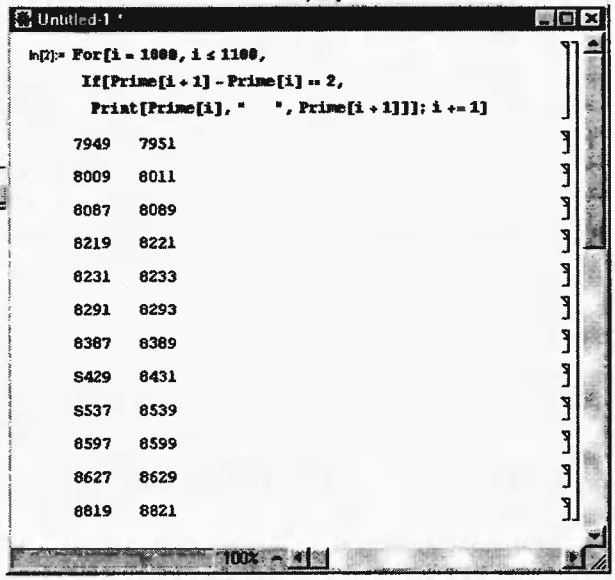
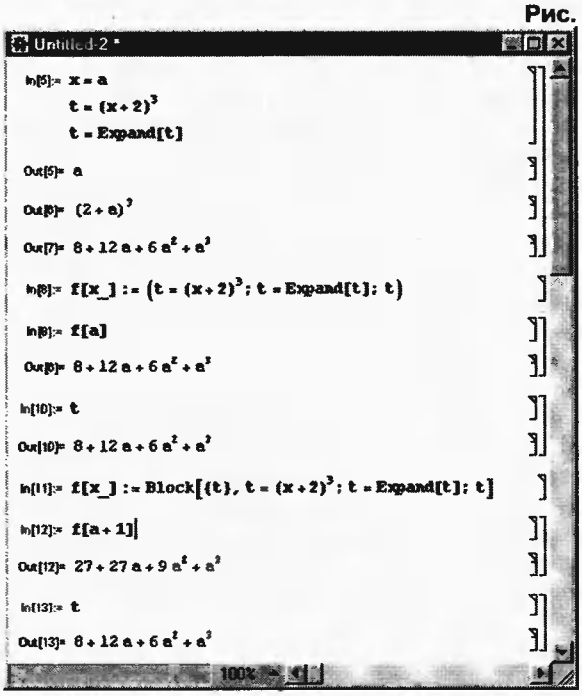


Рис. 5



f[x_] := **Block[{t}**, t = (x + 2)^3; t = Expand[t]; t]
f[x_] := **Module[{t}**, t = (x + 2)^3; t = Expand[t]; t].

Причем допускается присвоение значений локальным переменным внутри блоков и модулей:

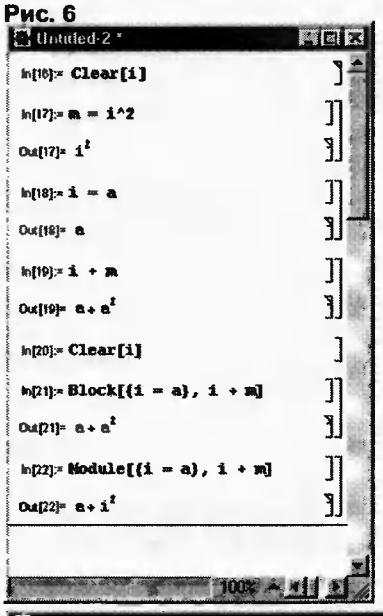
Block[{i = a}, ...].

Переменная t свое значение вне блока не меняет **(рис.5)**.

Разницу между блоками и модулями можно продемонстрировать на следующем примере **(рис.6)**:

```
m = i^2
i = a
i + m
Clear[i]
Block[{i = a}, i + m]      ;возвращает a + a^2
Module[{i = a}, i + m]     ;возвращает a + i^2
```

Рис. 6



В блоке вместо m подставляется i^2, после чего все i заменяются на a. В модуле, в отличие от блока, учитывается, в каком случае переменная i является локальной (i в первой степени затем заменяется на a), а в каком глобальной — в m=i^2 подстановка на применяется.

Естественно, что время от времени пользователю удается создать достаточно удобные и практичные программы, и возникает вопрос об удобстве их частого использования. В такой ситуации рекомендуется создавать пакеты расширения. Рассмотрим весь этот процесс на примере классической задачи — численное решение уравнения f[x]=0 методом Ньютона ($x_{n+1}=x_n-f(x_n)/f'(x_n)$). Здесь необходимо указать функцию (f[x_] := Cos[x] - x), точность решения ($\varepsilon = 10^{-5}$) и начальное приближение (x=1.0).

Итак, самый простой путь состоит в использовании цикла While **(рис.7)**:

```
y[t_] = t - f[t]/D[f[t], t];
While[Abs[x - y[x]] > ε, x = y[x]]; x
```

Этот же алгоритм решения можно оформить с использованием блоков или модулей

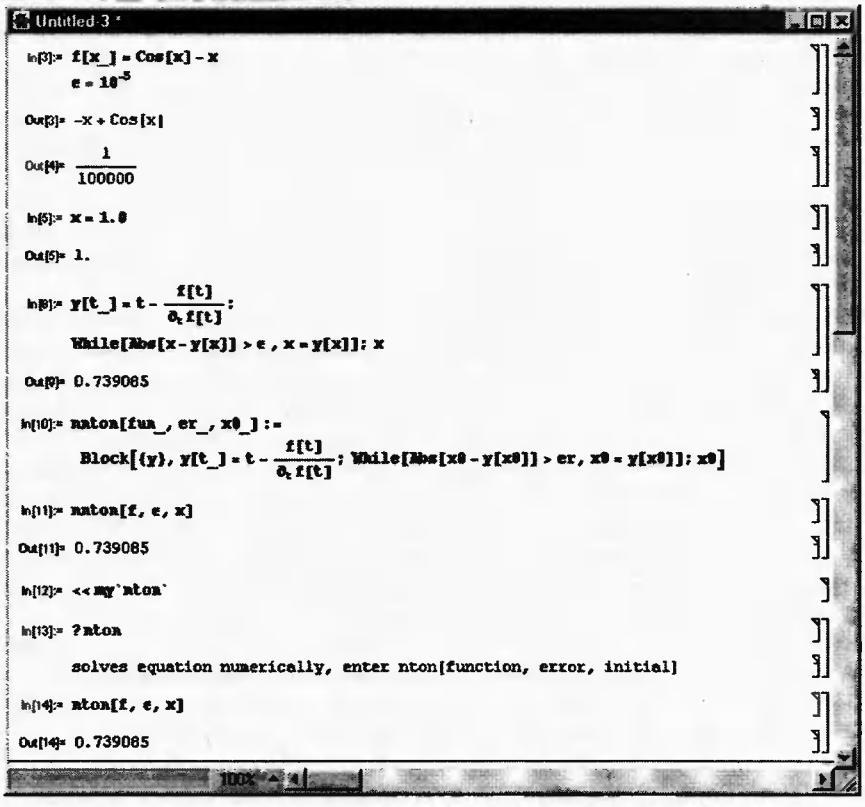
```
nnton[fun_, er_, x0_] :=
Block[{y},   y[t_] = t - f[t]/D[f[t], t];
While[Abs[x0 - y[x0]] > er, x0 = y[x0]];
 x0].
```

Тогда для вычисления корня уравнения достаточно выполнить команду nnton[f, ε, x].

Однако для часто применяемых функций и процедур в системе Mathematica удобнее создать т.н. пакет расширений. В нашем случае он будет выглядеть следующим образом:

```
BeginPackage["nton`"]
(*метод Ньютона*)
nton::usage = "solves equation numerically, enter
nton[function, error, initial]"
Begin["`Private`"]
nton[fun_, er_, x0_] := Block[{x = x0, y, t}, y[t_] = t - fun[t]/
D[fun[t], t];
While[Abs[x - y[x]] > er, x = y[x]]; x]
End[ ]
SetAttributes[nton, ReadProtected]
Protect[nton]
EndPackage[ ]
```

Рис. 7



Здесь:

- BeginPackage["nton`"] — имя пакета;
- nton::usage = "solves equation numerically, enter nton[function, error, initial]" — описание функций пакета;
- Begin["`Private`"] ... End[ ] — начало и конец содержательных частей пакета;
- nton[fun_, er_, x0_] := Block[{x = x0, y, t}, y[t_] = t - fun[t]/D[fun[t], t]; While[Abs[x - y[x]] > er, x = y[x]]; x] — функции пакета;
- SetAttributes[nton, ReadProtected] — установка атрибутов функции;
- Protect[nton] — защита функции от модификации.

Для создания пакета удобнее воспользоваться текстовым редактором "Блокнот". Затем пакет в виде файла с расширением *.m размещается в специально зарезервированной для этих целей директории — (основная директория Mathematica)\AddOns\ExtraPackages\My\nton.m. Поддиректориию My необходимо создать самостоятельно (выбор имени — My, либо же другого остается за пользователем, но именно имя созданной директории используется при вызове пакета).

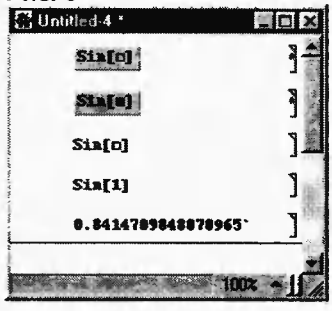
После этих операций ваша программа практически всегда готова к использованию. Для вызова пакета используется уже знакомого вида команда:

<<my`nton`.

Как и в стандартных пакетах, можно вызвать справку по работе с пакетом (рис.7), да и сама функция уже готова к работе: nton[f, ε, x].

Не забыты в системе также и средства визуального программирования. Например, предусмотрена возможность создания кнопок в рабочем документе, и выглядит этот процесс следующим образом **(рис.8)**:

Рис. 8



- **Input/CreateButton/Evaluate**,

- на кнопке ввести имя команды или функции;

Sin[□] — вводится Sin [**Esc** pl **Esc]** для неактивного поля;

Sin[■] — вводится Sin [**Esc** spl **Esc]** для активного поля;

- выделить при помощи мыши кнопку;

- **Input/EditButton** — установить флажок Button always active;

- выделить ячейку — **Cell/Cell Properties** (установить флажок — CellActive, снять — CellEditable).

При необходимости можно набор кнопок конвертировать в инструментальную палитру:

- выделить ячейки с кнопками;

- **File/Generate Palette From Selection**.

Дополнительно можно организовать ввод данных при помощи диалогового окна. Этот диалог организуется с помощью команды Input:

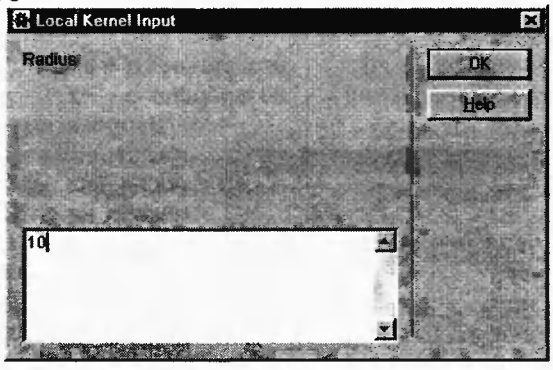
x := **Input**["Radius"].

Тогда при вычислениях с переменной x:

I = 2*Pi*x,

будет появляться диалоговое окно с поясняющей надписью Radius **(рис.9)**. В поле ввода необходимо задать значение переменной и нажать кнопку OK.

Рис. 9



И все же Mathematica — это система, ориентированная прежде всего на символьные вычисления. Это привносит свою специфику и в используемые программные средства. Поэтому в следующей статье мы еще раз обратимся к символьным выражениям и особенностям программной их обработки.

А.ГОРЯЧКИН,
г.Кыштым, Челябинской области.
E-mail: anatoliy_goryach@mail.ru

Диагноз поставит... Sandra

Существует определенный класс системных программ, позволяющих проверить конфигурацию персонального компьютера и работоспособность его устройств, произвести диагностику аппаратных средств и программного обеспечения PC. Такие программы называются утилитами. На сегодняшний день одной из наиболее распространенных программ-утилит является SiSoft Sandra 2001. В данной статье сделан краткий обзор этой популярной программы.

Разработчиком программы SiSoft Sandra 2001 является фирма SiSoftware (http://www.sisoftware.co.uk). Дата релиза версии 2001.5.8.11 — 30 мая 2001 г. Официальный сайт программы в Интернете находится по адресу http://www.sisoftware.co.uk/sandra

Свое название программа получила от сокращения (the **S**ystem **AN**alyzer, **D**iagnostic and **R**eporting **A**ssistant). SiSoft Sandra 2001 — это 32-разрядная информационная и диагностическая утилита, которая полностью поддерживает операционные системы Windows 95/98/Me. В последние версии программы включена поддержка Windows NT4/2000. К сожалению, пользователи Windows NT3 не смогут воспользоваться этой программой, потому что SiSoft Sandra 2001 не поддерживает данную операционную систему. Программу можно использовать для следующих целей:

- определения конфигурации компьютера;
- получения полной сводки о работе системы;
- проверки состояния системы;
- измерения производительности каждого из узлов и компонентов системы в отдельности;
- сохранения результатов работы программы на диске в виде текстового файла.

SiSoft Sandra способна тестировать аппаратные средства и системное программное обеспечение, установленное на компьютере. После тестирования программа выдает определенные рекомендации и советы по улучшению работы системы и увеличению ее производительности. С ее помощью можно также оптимально настроить систему на максимальное быстродействие. Она позволяет получить полезные и интересные сведения о технических характеристиках узлов и компонентов системы. Например, с ее помощью можно узнать, каковы текущие значения частот системной шины и шин расширения AGP, PCI, USB и ISA. SiSoft Sandra 2001 наглядно отображает в графическом виде производительность основных узлов компьютера, в частности, CPU, FPU, RAM, HDD, FDD и CD-ROM по отношению к эталонным системам. Sandra определяет все типы современных процессоров и другого "железа" — чипсетов, памяти, видеокарты, принтеров, портов, звуковых и сетевых карт, модемов и т.д. Программа выдает подробную (в том числе и недокументированную) информацию обо всей системе в целом.

Инсталляционный файл можно без особого труда найти в Интернете на любом сайте с бесплатным программным обеспечением. Скачать программу установки с сайта разработчика можно по следующим адресам:

http://www.sisoftware.co.uk/sandra/html/dload.htm
http://www.sisoftware.demon.co.uk/sandra/html/dload.htm

Размер в разархивированном виде невелик — 2,78 Мб. Установка SiSoft Sandra 2001 типична для программ Windows и поэтому не должна вызывать трудностей. Но прежде чем устанавливать программу, необходимо удалить любую предшествующую версию. Программа не требовательна к аппаратным ресурсам, и может быть установлена практически на любом компьютере. Минимальные системные требования — 486 DX2 66 МГц, 16 Мб RAM и 100 Мб HDD. После окончания процесса инсталляции программа при полной установке займет на жестком диске в папке "Program Files" 3,21 Мб. На рабочем столе создаются 3 ярлыка, а в главном меню — папка "SiSoft Utilities".

Существуют две версии этой программы:

- Sandra Standard — свободно распространяемая для домашнего пользования;
- Sandra Professional — коммерческая (ее стоимость — 29$).

Стандартная версия программы SiSoft Sandra 2001 включает в себя 50 отдельных модулей (в коммерческой версии — 70 модулей). Модули, входящие в состав SiSoft Sandra 2001, подразделяются на четыре группы:

- информационные модули;
- benchmark–модули;
- модули просмотра системных файлов (*.sys, *.bat , *.ini);
- тестовые модули.

Программа регулярно обновляется, обычно это происходит через каждые 3...4 месяца.

Перед запуском SiSoft Sandra, во избежание потери информации, настоятельно рекомендуется сохранить открытые файлы, документы и закрыть все другие приложения. Для запуска программы открываем ярлык "SiSoft Sandra 2001 Standard", и после загрузки программы нашему взору предстает картина, внешне очень напоминающая Панель управления Windows.

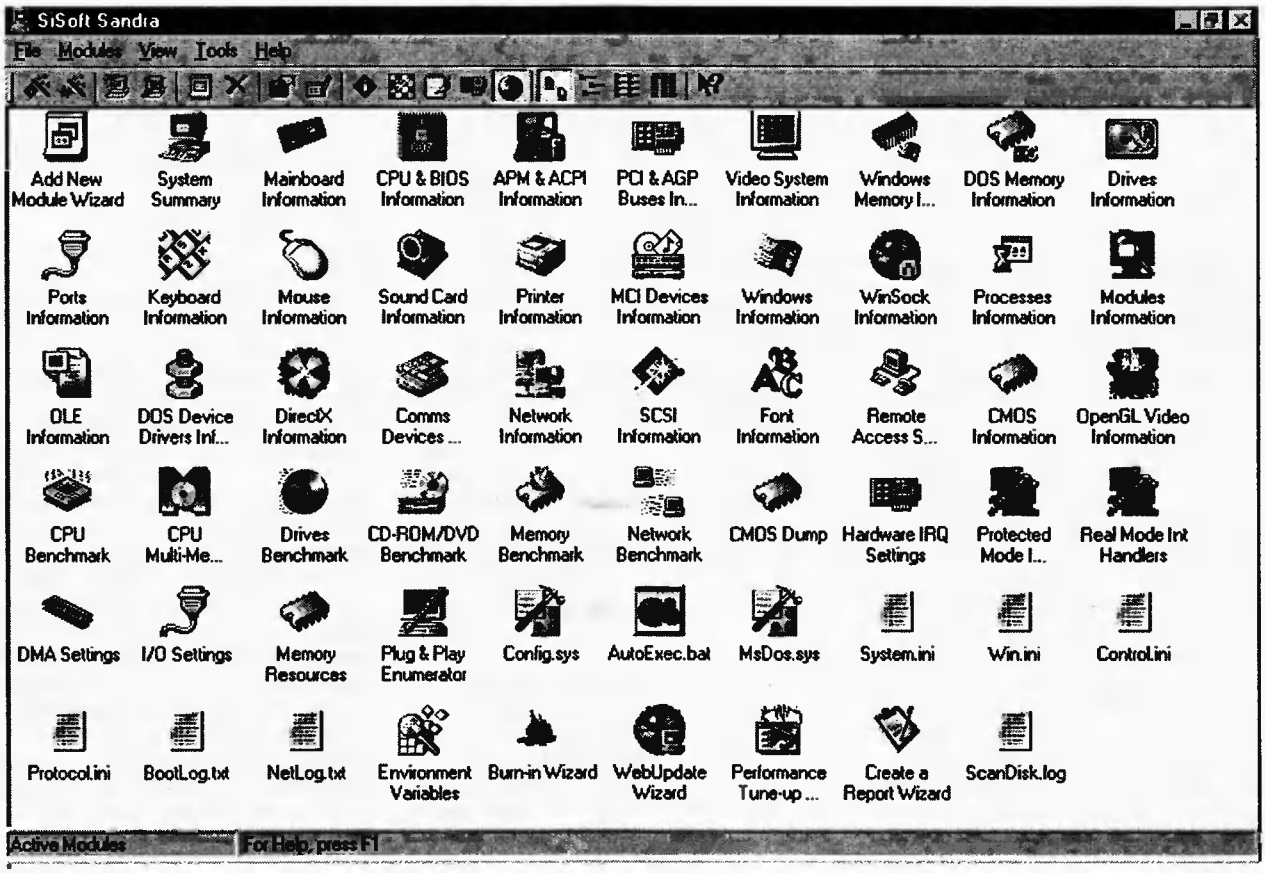
В первую очередь SiSoft Sandra интересна в плане определения быстродействия центрального процессора и других основных узлов системы. Теперь, для того чтобы измерить скорость CPU, достаточно "кликнуть" по кнопке "Benchmarking", находящейся на панели инструментов, и в окне программы выбрать иконку "CPU Benchmark". На рисунке показано одно из рабочих окон программы SiSoft Sandra.

При проведении тестов не стоит нажимать на кнопки клавиатуры и двигать мышью, о чем на экран выдается соответствующее предупреждение.

Помимо полного анализа системы, SiSoft Sandra способна выдавать детальную информацию о состоянии драйверов к комплектующим, сообщать при необходимости о замене этих драйверов ввиду устаревания или отсутствия их сертификации. Она предоставляет пользователю подробнейшую оперативную информацию о состоянии его компьютера — аппаратной части, использовании ресурсов памяти, дисковых накопителях — и производит сравнение с другими системами. Вся информация, рекомендации и советы выдаются в понятной и доходчивой форме. Sandra распознает практически все современное оборудование для персональных компьютеров. Это отличная программа для оценки производительности вашего компьютера.

Теперь немного о негативных моментах. Программа иногда "виснет", если до или после ее загрузки были запущены какие-либо другие приложения. Скорее всего, Sandra конфликтует с библиотеками, оставшимися в памяти от других программ. Поэтому лучше всего запускать утилиту сразу же после полной перезагрузки операционной системы, и во время ее использования не загружать другие программы.

Если вы хотите при существующей конфигурации максимально увеличить производительность вашего компьютера, не тратя при этом средств на приобретение новых компонентов, если вы желаете оптимизировать работу системы — то это программа для вас.



ЛИСТАЯ СТРАНИЦЫ

И.Голубцов а статье "Больше памяти, надежной и быстрой" **(ПЛ: компьютеры, №3/2002, С.147...150)** рассказывает **о достоинствах и недостатках используемых в настоящее время модулей оперативной памяти.**

EDO RAM (Extended Data Out RAM — память с расширенными аозможностями аывода данных) является разноаидностью памяти DRAM (Dynamic RAM — динамическая оперативная память) и представляет собой 72-контактный модуль в конструктивном исполнении SIMM (в настоящее время этот тип памяти используется только в устаревших системах). Аббревиатура SIMM расшифровывается как Single In-Line Memory Module — модуль с одной группой контактоа. Контакты располагаются с обеих сторон платы модуля, но попарно соединены между собой, дублируя друг друга.

С центральным процессором память EDO RAM работает асинхронно: первый вынужден приостанавливать свою работу при считывании очередного пакета данных. Как результат, возникают задержки (циклы ожидания), что является главным, по сравнению с современной синхронной памятью, недостатком асинхронных модулей.

SDRAM — синхронное динамическое ОЗУ. В его названии слово "синхронное" является ключевым. Память работает синхронно с центральным процессором, поэтому отсутствуют циклы ожидания. Память SDRAM выпускается в аиде 168-контактных модулей DIMM (Dual In-Line Memory Module). Отличительная особенность форм-фактора DIMM — это незааисимость электрических контактоа, расположенных по обеим сторонам платы. За каждый такт по шине памяти может пересылаться до 64 бит данных. Принимая во внимание, что большинство современных процессоров имеют 64-битную внешнюю шину данных, максимальная пропускная способность шины памяти SDRAM, функционирующей на частоте 133 МГц, состаляет порядка 1 Гбайт/с.

Очень важный параметр оперативной памяти — рабочая частота. Существуют модули, работающие на частотах 66, 100, 133, 150 и даже 166 МГц. Модули 66 и 100 МГц уже порядком устарели, поэтому в продаже они встречаются редко. Память SDRAM стандарта PC133 наиболее распространена, тогда как модули 150 и PC166 МГц предназначены для использования в "разогнанных" системах, и из-за достаточно высоких цен большой популярностью не пользуются.

Время ожидания — еще один параметр оперативной памяти, влияющий на ее быстродействие. Для модулей SDRAM оно равняется двум либо трем тактам. Чем меньше этот показатель, тем аыше быстродействие.

DDR SDRAM иначе называют SDRAM-II. Как в случае с модулями SDRAM, работа памяти этого типа основана на принципе синхронного взаимодействия с центральным процессором. Более того, технология производства обоих аидоа практически соападает. Внешне модули SDRAM и DDR SDRAM схожи, ведь они имеют один и тот же форм-фактор — DIMM. Отличия коснулись числа контактов (на модуле DDR SDRAM их стало больше — 184 против 168 у SDRAM) и аысоты модуля, что является следствием увеличения размера чипов памяти.

Главной особенностью DDR SDRAM является применение технологии DDR (Double Data Rate). Она позволяет за один такт пересылать уже не один, а два пакета данных, что вдвое увеличивает пропускную способность шины памяти.

Различают три вида DDR SDRAM: PC 1600, PC2100 и PC2700. В отличие от модулей SDRAM, цифры в индексах обозначают не частоту системной шины, а пропускную способность шины памяти в мегабайтах в секунду. Их рабочая частота составляет 200, 266 и 333 МГц соответственно. Модули PC2700 целесообразно использовать в компьютерах на базе DDR-чипсетов, поддерживающих частоту системной шины 266 и 333 МГц. Данные наборы системной логики были специально созданы для работы с памятью DDR.

Время ожидания у DDR SDRAM мало отличается от аналогичного параметра SDRAM — 2 или 2,5 такта (CL2 и CL2,5 соответственно). DDR SDRAM CL2,5 имеет наилучшее соотношение цены и производительности, тогда как модули CL2 быстрее, стоят дороже и используются в системах, предназначенных для работы с приложениями, обрабатыаающими большие объемы информации.

RDRAM расшифровывается как Rambus DRAM. Rambus — это компания, которая разработала и запатентовала данный тип памяти. Особенность RDRAM заключается в измененных по сравнению с SDRAM конструкции шины данных и рабочей частоте памяти: 64-битная шина данных разделена на четыре незааисимых канала по 16 бит каждый. Это позволило значительно увеличить рабочую частоту памяти. За счет применения технологии передачи двух пакетоа данных за 1 такт, рабочая частота памяти составила 800 МГц. Таким образом, пиковая пропускная способность шины памяти RDRAM равна 6,4 Гбайт/с, что в шесть раз больше, чем у SDRAM PC 133, и в три раза — чем у DDR SDRAM PC2100.

Но RDRAM имеет существенный недостаток — большое время ожидания (порядка 10...20 тактов), что заметно ограничивает быстродействиа в некоторых задачах. Особенно это проявляется при работе с программами Microsoft Office, где данные от памяти к процессору поступают маленькими блоками. Обращаться к памяти приходится часто, и при каждом запросе процессор аынужден ждать не менее 10 тактов. Однако в программах, требующих пересылки больших пакетов данных, например, в Adobe Photoshop, RDRAM заметно опережает и DDR SDRAM, и, тем более, SDRAM.

Для памяти RDRAM был разработан новый форм-фактор под названием RIMM. Модули DIMM и RIMM имеют почти одинакоаые размеры, но не могут работать друг с другом из-за механической и электрической несовместимости. Интересно, что для нейтрализации внешних электромагнитных полей модули RIMM помещают в металлический корпус.

Многоканальный принцип работы RDRAM накладывает определенные ограничения на компоновку модулей в системе. Их можно устанавлиаать только парами. При этом в комплекте с модулями RIMM поставляются специальные заглушки, которые необходимо устанавливать в свободные слоты. В противном случае система не будет работать.

Буферизированная память позволяет значитвльно уменьшить временные задержки при обращении к ОЗУ. В таких модулях есть несколько очень быстрых микросхем, играющих роль буфера. Эти микросхемы почти мгновенно принимают данные, освобождая системную шину. При этом запись пакета в ячейки осноаной памяти происходит уже без участия контроллера памяти, который в это время может обрабатывать очередной запрос от процессора.

В результате существенно увеличивается быстродействие как при "случайном", так и при последовательном доступе к ячейкам. Для обычных приложений, возможно, это не так важно. Зато применение подобных модулей в серверах давно стало стандартом де-факто.

Буферизация применяется только а модулях типа SDRAM и DDR SDRAM. Внешне буферизированные модули отличаются большей высотой печатной платы, тогда как контактный разъем имеет традиционный вид.

Ввиду того что осноаная область применания буферизированной памяти — серверы, требоаания к надежности гораздо жестче. Поэтому очень часто буферизированные модули оснащаются блоком контроля и исправления ошибок, так называемым ECC (Error Checking and Correction).

Подводя итоги, автор отмечает, что на данный момент существуют две основные конкурирующие технологии производства памяти — DDR SDRAM и RDRAM. Благодаря лучшему соотношению цены и производительности, предпочтительнее выглядит DDR SDRAM. Надолго ли хватит ее возможностей? Скорее всего, нет. Век SDRAM, несмотря на совершенствование технологии, постепенно подходит к концу. Определенный рывок DDR SDRAM еще сможет осуществить благодаря использованию принципа передачи четырех пакетов данных за один такт. Но в целом следует признать, что память SDRAM, вероятно, сойдет со сцены.

Что касается технологии RDRAM, здесь ситуация диаметрально противоположная. Применение сегментированной шины данных с малым количеством линий и сложной архитектуры памяти открывает большие перспективы для увеличения быстродействия модулей и пропускной способности шины данных. Уже сейчас на рынке предствлена память RDRAM с рабочей частотой 800 МГц. В ближайшем будущем планируется увеличить тактовую частоту и одновременно расширить шину данных до 32 и даже 64 бит. Однако все упирается а стоимость, которая и является сдерживающим фактором для быстрого развития RDRAM.

Последним достижениям фирмы Intel посвящена статья "Intel Pentium 4 Northwood: история продолжается" **(ПЛ: компьютеры, №3/2002, С.38...39).**

К началу 2002 г. корпорация Intel подготовила "даойной удар" по кошелькам пользователей, отмечается в статье. Вместе с долгожданным чипсетом i845 B-step (i845D), поддерживающим относительно недорогую память DDR SDRAM, на рынке появился продолжатель рода Pentium 4— процессор с ядром Northwood.

Табл. 1

Тип процессора	Pentium 4 Willamette	Pentium 4 Northwood
Проектные нормы, мкм	0,18	0,13
Площадь кристалла, мм²	217	146
Число транзисторов, млн шт.	42	55
Процессорный разъем	Socket 432/478	Socket 478
Частота процессора, ГГц	1,3...2,0	2,0...2,2
Частота системной шины, МГц	100 (400)	100 (400)
Размер кеша первого уровня	8 Кб (данные) + 12000 инструкций	8 Кб (данные) + 12000 инструкций
Частота кэша первого уровня	Частота ядра	Частота ядра
Ширина шины данных кеша первого уровня, бит	256	258
Размер кэша второго уровня, Кб	256	512
Частота кэша второго уровня	Частота ядра	Частота ядра

Новый Pentium 4 изготавливается по 0,13-мкм технологии с использованием медных соединений. Он включает 55 миллионов транзисторов. Суммарная длина проводников — около 3 км. Применение новой технологии позволило уменьшить размер кристалла на 30%, поднять тактовую частоту и в два раза увеличить объем кэша второго уровня — до 512 Кбайт **(табл.1)**. Новый Pentium 4 будет выпускаться в двух модификациях: 2,0 ГГц и 2,2 ГГц с индексом A в маркировке.

В новом процессоре используется существующий процессорный разъем Socket478. Это настоящий бальзам на душу пользователей, уставших от постоянной смены разъемов после появления новых моделей процессоров. Теперь большинство владельцев систем на базе Pentium 4 Willamette в исполнении под Socket478 могут спать спокойно. Для того чтобы перейти на новый процессор, им понадобится только обновить версию BIOS материнской платы. В худшем положении оказываются хозяева компьютеров на основе Pentium 4 Willamette Socket423 — им придется поменять системную плату.

Конфигурация системы, использованной для испытания нового процессора:

- материнская плата — Intel;
- процессор — Intel Pentium 4 2,2 ГГц;
- графический адаптер — ASUS 8200 GeForce3;
- жесткий диск — Maxtor D740X, 40 Гбайт, UDMA/100;
- звуковая карта — Creative Audigy Player 5.1;
- для видеокарты применен референсный драйвер от nVIDIA версии 23.11.

Быстродействие оценивалось с помощью программы Sandra Professional 2002:

- Dhrystone ALU 4122 MIPS;
- Whetstone FPU 1308 MFLOPS;
- Whetstone SSE2 2665 MFLOPS.

Для тестирования использовались также программы Quake 3 Arena DEMO (demo001) версии 1.11 и 3Dmark 2001 (табл.2). В последней использовались установки:

- кадровый буфер — двойной;
- цветовое разрешение текстур — 32 бит;
- полноэкранное сглаживание — выключено;
- Z-буфер — 24 бит;
- режим — Pure 3D Hardware.

Параметры системы в Quake 3 Arena:

- fullscreen — on;
- lighting — lightmap;
- geometric detail — high;
- texture detail — maximum;
- texture quality — 32 bit;
- texture filter — bilinear.

Результаты тестирования даны в **табл.2**.

Корпорация Intel сделала достойный подарок пользователям, считает редакция журнала "ПЛ: компьютеры". В свет вышел процессор, обладающий не только повышенной тактовой частотой, но и вдвое увеличенным объемом кеш-памяти второго уровня. Микроархитектура процессора гарантирует ему длительный срок службы и большие перспективы. Ожидается, что процессоры этой линейки будут производиться еще в течение примерно пяти лет. Производительность будет возрастать по мере роста количества программ, оптимизированных под потоковое расширение SSE2.

Табл. 2

Экранное разрешение (32-бит цвет)	3Dmark 2001 (3Dmarks)	Quake 3 Arena (кадры/с)
1280x1024	5705	135,4
1024x768	7168	183,3
800x600	8227	207,6

Результаты тестирования новейших чипсетов для Pentium 4 Northwood рассмотрены в статье "Большой куш" **(CHIP, №3/2002, С.46...51).**

Intel делает ставку на Pentium 4 Northwood и вместе с ним предлагает поддержку оперативной памяти типа DDR SDRAM. Связка Pentium 4 и быстрой, но дорогой памяти Rambus (чипсет i850), ограничивали использование первого. Чипсет i845 поддерживает медленную память SDRAM. И только новый чипсет i845D позволяет использовать достаточно быструю и недорогую память DDR SDRAM. Материнские платы на i845D могут работать с модулями памяти спецификаций PC1600 и PC2100. PC2700 этот чипсет не поддерживает. Основные характеристики упомянутых чипсетоа приведены а **табл.1**.

Фирма VIA с самого начала сделала ставку на DDR SDRAM и предложила первую работоспособную платформу с данным типом памяти для Pentium 4 — чипсет P4X266. Новый P4X266A характеризуется улучшенной по сравнению с предшественником пропускной способностью памяти. Кроме того, его южный мост VT8233A, а отличие от старого VT8233, поддерживает новый стандарт интерфейса — Ultra ATA 133.

Еще один шаг в направлении Pentium 4 — настройки BIOS. Теперь можно увеличить частоту системной шины до 133 МГц, что позволит процессорам в будущем работать с системной шиной на частоте 533 МГц (4x133 МГц). Реальная тактовая частота увеличивается в четыре раза за счет технологии Quad-Pumped (передача четырехкратно увеличенного пакета данных). На материнские платы с VT8233A можно

Табл. 1

Чипсет	i845	i845D	i850
Северный мост	MSH KC82845	MSH KC82845	MSH KC82850
Частота FSB, МГц	400	400	400
Интерфейс AGP	4x	4x	4x
Тип памяти	SDRAM (PC133)	SDRAM (PC133), DDR SDRAM (PC1600, PC2100)	RDRAM (PC600, PC700, PC800)
Максимальная пропускная способность памяти, Гбайт/с	1,06	1,06; 2,1	3,2
Максимальный объем памяти	3 Гбайт SDRAM	3 Гбайт SDRAM, 2 Гбайт DDR SDRAM	2 Гбайт RDRAM
Южный мост	ICH2	ICH2	ICH2
Два IDE-контроллера	Ultra ATA 100	Ultra ATA 100	Ultra ATA 100
USB-порты	4	4	4
Аудиокодек	AC'97	AC'97	AC'97

ставить до трех банков памяти PC100/PC133 или PC1600/PC2100.

Чипсет SiS645 способен обеспечить процессоры Pentium 4 по максимуму (можно использовать самую быструю DDR-память). Наряду с нынешними модулями оперативной памяти PC2100, он работает и с новейшими — PC2700.

Еще одна особенность SiS645 заключается в использовании для обмена данными между северным и южным мостами двунаправленной 16-битной шины MuTIOL-Link. Ее теоретическая пропускная способность равна 533 Мбайт/с, тогда как чипсеты конкурентов, P4X266A от VIA и интеловский i845D, способны только на половинный результат — 266 Мбайт/с. Это преимущество проявляется, когда винчестер, звуковая карта или USB-компоненты не мешают друг другу.

Для тестирования использованы материнские платы:

- ABIT BL7-RAID (чипсет i845 и SDRAM);
- TH7II-RAiD(чипсет i850);
- Tyan S2266 (чипсет P4X266);
- VIA P4XB (чипсет P4X266A);
- SIS SS51A (чипсет SiS645);
- MSI 845 Ultra (чипсет i845D).

Оперативная память:

- 256 Мбайт SDRAM PC133;
- 256 Мбайт DDR SDRAM PC2100;
- 256 Мбайт DDR SDRAM PC2700;
- 256 Мбайт RDRAM PC800.

Во всех случаях использовались:

- видеоускоритель Leadtek Winfast GeForce 3;
- звуковая карта Creative SoundBlaster LIVE!;
- винчестер IBM DTLA-307030 (Ultra ATA 100);
- ОС Windows 98 SE;
- видеодрайвер Detonator 21.83.

Тестирование проводилось с помощью игр Quake 3 Arena и Expendable и тестов 3DMark 2000 (DirectX 7), 3DMark 2001 (DirektX 8), Sysmark 2000 (оценка общей производительности при работе с различными приложениями) и Bench 32 (измерение скорости передачи данных). Его результаты приведены в **табл.2**.

i845D дополнил линейку чипсетов Intel для Pentium 4 до полного комплекта, делает вывод редакция "CHIP". Комбинация Pentium 4 и DDR SDRAM, как показали тесты, не так уж плоха, но конкуренты могут больше. Чипсет SiS645 с быстрыми модулями PC2700 опережает i845D, а P4X266A бьет их обоих. Против SiS645 с памятью PC2100 i845D выступал очень удачно, да и с другим соперником — VIA P4X266 — он боролся на равных. Но P4X266A здорово выиграл а пропускной способности памяти. В среднем он опережает i845D по данному параметру на 7%. SIS645 разочаровал своей работой с памятью PC2100, но при использовании PC2700 ситуация улучшалась. Неприступный бастион Intel — связка Pentium 4 и Rambus — показал результат всего на 2 Мбайт/с больше, чем SiS645, что очень мало для дорогих модулей RIMM.

Однако, в то время как матплата с i845D выполнила все тесты без "падения", чипсеты VIA и SiS645 несколько раз "зависали". i845D может быть рекомендован как беспроблемный чипсет для длительной и эффективной работы.

Табл. 2

Тест	i845 (PC133)	i845D (PC2100)	i850 (PC800)	SiS645 (PC2100)	SiS645 (PC2700)	P4X266 (PC2100)	P4X266 (PC2100)
Quake 3 Arena, 640x480, 16 бит, кадров в сек.	160	196	205	193	215	202	212
Expendable, 640x480, 16 бит, кадров в сек.	84	95	104	103	103	95	101
3DMark 2000, баллов	9215	11018	11489	8580	11407	10888	11435
3DMark 2001, баллов	7067	8126	8368	6858	8506	8090	8316
Sysmark 2000, баллов	208	219	227	221	225	221	225
Bench 32, баллов	294	461	589	515	587	476	570

Диапазон применения миниатюрных источников питания исключительно широк — от карманного фонарика до ноутбука. Велико разнообразие имеющихся в продаже батареек и аккумуляторов. *Б.Семенов* в статье "Энергетика цифровой фотографии" **(Hard'n'Soft, №2/2002, С 62...68)** рассказывает о том, **каковы особенности источников тока, применяющихся в современных цифровых фотоаппаратах и не только в них.**

В качестве автономных элементов питания в фотоаппаратуре используются химические источники тока — гальванические элементы (батарейки) и аккумуляторы. В химических источниках тока электрическая энергия вырабатывается за счет протекания окислительно-восстановительных реакций. Если в батарейках такая реакция носит необратимый характер, то в аккумуляторах активные вещества подобраны таким образом, что окислительно-восстановительный процесс носит обратимый характер. Таким образом, когда к клеммам разряженного аккумулятора прикладывают постоянное напряжение, он начинает накапливать энергию за счет восстановления активных компонентов. Однако в каждом цикле восстановление происходит не полностью. Это приводит к постепенному уменьшению емкости аккумулятора и выходу его из строя после определенного числа циклов зарядки/разрядки.

Самыми важными параметрами химического источника тока являются номинальное напряжение на клеммах и емкость. Под емкостью подразумевается количество электрической энергии, которую элемент питания выделяет при определенных условиях разряда. Емкость выражают в ампер-часах (А·ч), миллиампер-часах (мА·ч) или кулонах (1 А·ч = 3600 Кл). На батарейках редко указывают их емкость, а на аккумуляторах она обозначается практически всегда.

Важной характеристикой химического источника тока при использовании его во многих устройствах является внутреннее сопротивление, на котором падает часть электродвижущей силы, развиваемой источником. Чем меньше внутреннее сопротивление химического источника тока, тем лучше он отдает электроэнергию в моменты пиковых нагрузок.

До недавнего времени в большинстве цифровых фотокамер (ЦФК) использовались химические источники тока цилиндрической формы типа АА (обозначение, принятое в США и многих других странах). Эти батарейки или аккумуляторы у нас в быту именуют пальчиковыми, а их зарубежное бытовое название — Mignon. Они имеют также обозначение R6, принятое Международной электротехнической комиссией. В России и в ряде европейских стран они обозначаются как 316, в Японии — UM3. На самой батарейке, как правило, присутствует не менее двух-трех альтернативных обозначений. Высота элементов типа АА составляет 50,5 мм, диаметр — 14,5 мм. Менее применимы в цифровой фотоаппаратуре цилиндрические элементы типоразмера ААА (альтернативные обозначения — 286, PO3, UM4 и Micro, в нашем быту они носят название "мизинчиковые"). Диаметр их составляет 10,5 мм и высота — 44,5 мм. В типоразмерах АА и ААА выпускаются не только батарейки, но и аккумуляторы. Обычно в цифровых фотокамерах используется комплект из двух или четырех элементов.

Солевые батарейки, в которых используется жидкий электролит — устаревший источник тока. Они имеют небольшую емкость и малое пиковое энергопотребление. Интенсивное энергопотребление может привести к утечке электролита и порче фотоаппаратуры.

В алкалиновых (щелочных) батарейках электролит находится в связанном состоянии, поэтому они не протекают. Такие батарейки обладают, по сравнению с солевыми, более высокой емкостью и меньшим внутренним сопротивлением, что обеспечивает большую пиковую отдачу энергии. К тому же, щелочные батарейки дольше хранятся.

Еще более высокую емкость имеют батарейки улучшенной конструкции, маркируемые Super Alkaline battery (правда, и стоят они порядка 0,6...1,0 USD за шт.).

Наиболее высокую емкость и наименьшее внутреннее сопротивление из гальванических элементов для массового потребления сейчас имеют литиевые батарейки. Они стоят довольно дорого, что объясняется применением в них ряда новых технологий, поэтому был найден компромиссный вариант — литиевая батарейка дает напряжение 3 В, а по размеру соответствует двум элементам АА, сложенным вместе. Получается дешевле, чем два литиевых элемента по 1,5 В. В **табл.1** приведены параметры гальванических элементов.

Табл. 1

Тип	Типоразмер	Номинальное напряжение, В	Емкость, мА·ч	Внутреннее сопротивление, Ом	Срок хранения
Солевой	AA AAA	1,5 1,5	600...1200 300...600	0,5...1,0 0,5...1,0	2 2
Алкагиновый	AA AAA	1,5 1,5	2500...3000 1200...1500	0,1...0,3 0,1...0,3	5 5
Литиевый	AA	1,5...3,6	1700...5000	0,05...0,1	10

Существует множество аидов аккумуляторов с различными типами электродов и электролитов. Однако в современной фотоаппаратуре чаще всего применяются никель-кадмиевые (NiCd) аккумуляторы, никель-металл-гидридные (NiMH), литий-ионные (Li-Ion) и литий-полимерные (Li-Pol). Динамика разряда аккумуляторов значительно отличается от динамики разряда гальванических элементов. В процессе разряда гальванических элементов их напряжение плавно снижеется по мере уменьшения емкости. При разряде аккумулятора напряжение остается практически неизменным до некоторого порога, после которого резко падает. Минимальное непряжение, при котором элемент способен отдавать полезную энергию в определенных условиях разряда, называется напряжением отсечки. Для использования аккумуляторов большинства типов оптимальной считается температура в пределах от 10 до 30°С.

При работе с аккумуляторами типоразмеров АА и ААА необходимо иметь в виду, что их номинальное напряжение равно, как правило, 1,2 В. И хотя за рубежом уже созданы полноценные аккумуляторы, имеющие стандартный типоразмер АА или ААА и напряжение 1,5 В, но в отечественных магазинах их встретить пока не удалось.

Первыми аккумуляторами, нашедшими широкое применение в ЦФК, стали никель-кадмиевые. Они имеют ряд достоинств. Во-первых, быстрый и простой метод заряда. Во-вторых, относительно слабую чувствительность к неправильной эксплуатации. В-третьих, длительный срок службы (4...5 лет) при соблюдении условий эксплуатации и периодического обслуживания. В-четвертых, низкую цену.

Существенным недостатком NiCd-аккумуляторов является "эффект памяти", который проявляется, когда на подзарядку ставится не полностью разряженный аккумулятор с напряжением выше значения напряжения отсечки. Аккумулятор как бы запоминает значение напряжения, до которого был разряжен, и не отдает энергию потребителю, если напряжение будет ниже этого уроаня. Таким образом, реальная емкость аккумулятора уменьшается. "Эффект памяти" устраним путем тренировки элемента несколькими циклами зарядки/разрядки.

NiCd-аккумулятор допускает заряд током, численно (в миллиамперах) равным его номинальной емкости (в миллиампер-часах). Однако наиболее благоприятным с точки зрения продления срока службы принято считать медленный заряд током (а миллиамперах), численно равным примерно 0,1 от емкости (в миллиамперчасах). Допускается перезарядка аккумулятора до 20...40% сверх его номинальной емкости, но следует помнить, что избыточный заряд может привести к уменьшению емкости аккумулятора или к полной потере его работоспособности.

В никель-металл-гидридных аккумуляторах веществом, участвующим в окислительно-восстановительных реакциях, является водород, находящийся в связанном состоянии. Удельная емкость у них примерно на 30...50% больше, чем у NiCd-аккумуляторое, соответственно меньше габариты и вес при той же емкости.

К достоинствам NiMH-аккумуляторов можно отнести практически полное отсутствие "эффекта памяти" и экологическую чистоту. Именно эти обстоятельства приаели к тому, что во многих применениях сейчас NiMH-аккумуляторы вытесняют NiCd. Однако у них есть и ряд существенных недостатков. NiMH-аккумуляторы выделяют значительно большее количество тепла во время заряда и требуют реализации более сложного алгоритма работы зарядного устройства для обнаружения момента полного заряда. Некоторые модели даже содержат внутренний термодатчик для обнаружения момента, когда зарядку нужно прекращать. NiMH-аккумулятор не может заряжаться так быстро, как NiCd — время его заряда обычно вдвое больше, чем у NiCd. Уступают NiMH-аккумуляторы своим никель-кадмиевым собратьям и по сроку службы.

Для NiMH-аккумуляторов, с точки зрения продления срока службы, неполный разряд предпочтительнее глубокого. Плохо сказывается на долговечности этих устройств короткое замыкание — не рекомендуется разряжать их током, превышающим половину численного значения емкости.

К достоинствам аккумуляторов Li-Ion относятся высокая энергоемкость, низкий саморазряд и отсутствие "эффекта памяти". К недостаткам можно отнести пока еще относительно высокую цену, необходимость хранения в заряженном состоянии и подверженность процессу старения, даже если аккумулятор не используется. Снижение емкости наблюдается примерно после одного года с момента изготовления, поэтому не рекомендуется хранить аккумуляторы Li-Ion в течение длительного времени.

Литий-ионные аккумуляторы содержат внутреннюю схему управления и защиты, призванную ограничить пиковое напряжение каждого элемента во время заряда и предотвратить понижение напряжения элемента при разряде ниже допустимого уровня. Кроме того, для обеспечения безопасности при их эксплуатации должен быть ограничен максимальный ток заряда/разряда, и необходимо контролировать температуру. "Интеллектувльные" литий-ионные аккумуляторы (иначе называемые инфолитиевыми), которыми комплектуются, например, цифровые видеокамеры фирмы Sony, дополнены встроенным электронным устройством, информирующим об оставшемся времени работы и оставшейся емкости.

Литий-полимерные аккумуляторы — следующий этап в развитии литиевой технологии. Главная область их применения — сотовые телефоны и переносные компьютеры. По сравнению с аккумуляторами Li-Ion, Li-Pol допускают меньшее число циклов заряда-разряда (100...150) и небольшой максимальный ток нагрузки (порядка 0,2 от численного значения емкости). Li-Pol значительно уступают аккумуляторам других типов и по надежности.

Главное преимущество Li-Pol-аккумуляторов — в конструктивном исполнении. Технология их производства допускает изготовление в различных геометрических формах, нетрадиционных для обычных аккумуляторов. Благодаря этому свойству можно делать очень плоские аккумуляторы, а также аккумуляторы, полностью заполняющие имеющийся свободный объем в устройстве, где они используются. Основные параметры аккумулятороа приведены в **табл.2**.

Табл. 2

Тип	Типоразмер	Номинальное напряжение, В	Емкость, мА·ч	Напряжение отсечки, В	Внутренее сопротивление, Ом	Диапазон рабочих температур, °С	Количество циклов	Саморазряд, % в месяц
NiCd	AA	1,2	700...1500	1	0,01...0,1	-20...+60	1000...1500	20...30
NiCd	AAA	1,2	300...800	1	0,01...0,1	-20...+60	1000...1500	20...30
NiMH	AA	1,2	1500...2100	1	0,01...0,1	0...+50	500...1000	30...40
NiMH	AAA	1,2	800...1100	1	0,01...0,1	0...+50	500...1000	30...40
Li-Ion	Нестандартный	3,6	1700...4000	3	-	0...+40	300...1000	0,1...0,5
Li-Pol	Нестандартный	3,6	600...1200	3	-	0...+40	100...300	-

М.ДОЛИНСКИЙ,
г.Гомель.

Решение задач на графах

Существует целый класс олимпиадных задач по программированию, которые проще решаются, если ученик владеет определенным набором знаний, умений и навыков в области алгоритмов на графах. Это происходит потому, что такие задачи могут быть переформулированы в терминах теории графов.

Теория графов содержит огромное количество определений, теорем и алгоритмов. И поэтому данная статья не может претендовать и не претендует на полноту охвата темы. Однако, по мнению автора, предлагаемые сведения являются хорошим компромиссом между объемом материала и его "коэффициентом полезного действия" в практическом программировании и решении олимпиадных задач.

Необходимо отметить, что данный материал в существенной степени опирается на наличие у ученика определенных навыков в использовании рекурсивных функций и рекуррентных соотношений, которые, в частности, могут появиться при работе над материалами [9, 10] соответственно.

Далее статья построена следующим образом. Дается минимальное количество базовой теории, после чего приводится решение большого количества задач из Белорусских республиканских и международных (IOI — International Olympiad in Informatics) олимпиад по информатике для школьников.

Автору представляется наиболее эффективным следующий способ изучения излагаемого материала. После ознакомления с общими сведениями, разбор каждой новой задачи следует проводить в таком порядке. Прочитав условия, отложить статью в сторону и попытаться решить задачу самостоятельно. Если же вам покажется, что вы зашли в тупик, и дальнейшие размышления не могут привести к полезному результату, нужно возвращаться к материалу, прочитать полное решение и самостоятельно реализовать его на компьютере. Чем полнее и напряженнее будет проведенная вами работа над текущей задачей, тем больше шансов, что вы решите следующую задачу самостоятельно. Более того, работа над каждой из приведенных в данном материале задач значительно повысит шансы на то, что вы решите новые задачи такого класса, которые вам обязательно встретятся. И, наоборот, поверхностное чтение может привести, в лучшем случае, к тому, что вы просто поймете и, возможно, запомните решение десятка приведенных задач. Это, конечно, тоже не мало, ведь вам могут встретиться эти, либо похожие, задачи. Но автор ставит другую задачу — помочь научиться решать новые задачи такого класса.

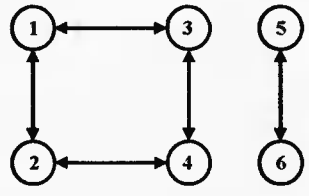
1. Общие сведения об алгоритмах на графах

Прежде всего несколько слов о том, как возникает понятие графа из естественных условий задачи. Приведем несколько примеров.

Пусть мы имеем карту дорог, на которой для каждого города указано расстояние до всех соседних с ним. Здесь два города называются соседними, если существует дорога, соединяющая непосредственно эти два города. Аналогично, можно рассмотреть:

- улицы и перекрестки внутри одного города. Заметим, что могут быть улицы с односторонним движением;
- сеть компьютеров, соединенных проводными линиями связи;
- набор слов, каждое из которых начинается на определенную букву и заканчивается на эту же или другую букву;
- множество костей домино. Каждая кость имеет 2 числа — на левой и правой половине кости;
- устройство, состоящее из микросхем, соединенных друг с другом наборами проводников;
- генеалогические деревья, указывающие родственные отношения между людьми;
- и, наконец, собственно графы, указывающие отношения между какими-либо абстрактными понятиями, например, числами.

Итак, неформально граф можно определить как набор вершин (города, перекрестки, компьютеры, буквы, цифры кости домино, микросхемы, люди) и связей между ними (дороги между городами; улицы между перекрестками; проводные линии связи между компьютерами; слова, начинающиеся на одну букву и закачивающиеся на другую или эту же букву; проводники, соединяющие микросхемы; родственные отношения, например, "Алексей — сын Петра"). Двунаправленные связи, например, дороги с двусторонним движением, принято называть ребрами графа. Однонаправленные связи, например, дороги с односторонним движением, принято называть дугами графа. Граф, в котором вершины соединяются ребрами, принято называть неориентированным графом. Граф, в котором хотя бы некоторые вершины соединяются дугами, принято называть ориентированным графом.



На **рисунке** приведен пример неориентированного графа с шестью вершинами.

2. Кратчайшие расстояния на графах

Графы могут задаваться матрицей смежности. Например, для приведенного выше графа матрица смежности выглядит следующим образом:

G	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	0	1	0	0
3	1	0	0	1	0	0
4	0	1	1	0	0	0
5	0	0	0	0	0	1
6	0	0	0	0	1	0

То есть G[i,j]=1, если есть дуга из вершины i в вершину j, и G[i,j]=0 в противном случае.

Часто также представляет интерес и матрица достижимости графа, в которой G[i,j]=1 если существует путь по ребрам (дугам) исходного графа из вершины i в вершину j.

Например, для графа, приведенного на рисунке выше, матрица достижимости будет иметь вид:

G	1	2	3	4	5	6
1	1	1	1	1	0	0
2	1	1	1	1	0	0
3	1	1	1	1	0	0
4	1	1	1	1	0	0
5	0	0	0	0	1	1
6	0	0	0	0	1	1

Граф называется взвешенным, если для его дуг вводятся веса — например, длины дорог между городами. Во взвешенном графе G[i,j] — вес дуги от вершины i к вершине j.

На взвешенных графах можно решать задачу нахождения кратчайшего расстояния между вершинами. Простейший для реализации способ нахождения кратчайших рас-

стояний от каждой вершины к каждой — метод Флойда:

```
for k:=1 to N do
  for i:=1 to N do
    for j:=1 to N do
      G[i,j]:=min(G[i,j],G[i,k]+G[k,j]);
```

В результате выполнения этого алгоритма в G[i,j] сформируется кратчайшее расстояние от вершины i до вершины j для всех пар вершин i и j в графе.

Замечания:

1. Начальные значения G[i,j] для вершин i и j, между которыми в исходном графе нет дуги, необходимо инициализировать заведомо чрезвычайно большой величиной GREAT, например, так:

```
for i:=1 to N do
  for j:=1 to N do G[i,j]:=GREAT;
```

где GREAT — подходящая по смыслу константа.

Не рекомендуется использовать maxlongint в качестве GREAT, поскольку в этом случае при сложении может возникнуть переполнение.

2. Попутное свойство алгоритма Флойда — возможность построить матрицу достижимости для исходного графа, поскольку если G[i,j] осталось равным GREAT, это и означает, что вершина j недостижима из вершины i.

3. Алгоритм Флойда работает также и в том случае, когда некоторые из ребер исходного графа имеют отрицательные веса (но гарантировано, что в графе отсутствуют циклы с отрицательным весом).

В случае, когда требуется находить расстояние между двумя конкретными вершинами, или расстояние от конкретной вершины до всех остальных, можно для ускорения работы или сокращения используемой памяти применить несколько более сложный метод Дейкстра (см. задачи из пунктов 2.2, 2.3).

2.1. Задача "Маневры"

(Республиканская олимпиада по информатике, 2000 г., автор — Перепечко С.Н.)

На территории некоторого государства с сильно пересеченной горной местностью идут военные маневры между двумя противоборствующими сторонами — "Синими" и "Зелеными". Особенности ландшафта и сложные климатические условия вынуждают подразделения обеих сторон размещаться только на территории некоторых населенных пунктов. Общее количество населенных пунктов в этом государстве равно N.

Тактика ведения боевых действий "Синих" рассчитана на нанесение противнику быстрых и внезапных ударов, что возможно лишь в том случае, если в операциях используются моторизованные части, а их передвижение происходит только по дорогам.

Разнообразие используемой боевой техники приводит к тому, что время перемещения различных боевых частей из одного пункта в другой оказывается различным и определяется величиной V_j — скоростью движения подразделений боевой части, расквартированной в j-ом населенном пункте.

Используя подавляющий перевес в технике, одна из сторон под кодовым названием "Синие" планирует организовать ночной налет на базы противника под кодовым названием "Зеленые" и полностью их разгромить. Все боевые подразделения "Синих" приступают к выполнению операции одновременно. Если боевая часть "Синих" врывается в населенный пункт, занятый "Зелеными", то, учитывая фактор внезапности, им удается полностью разгромить эту группировку.

К сожалению, блестящему проведению операции помешало то обстоятельство, что спустя время T после начала операции "Зелеными" был осуществлен радиоперехват сообщения о начавшейся операции. После радиоперехвата группировки "Зеленых" мгновенно рассеиваются в окрестных горах и остаются невредимыми.

Выясните, какое количество группировок "Зеленых" и в каких населенных пунктах удастся все-таки разгромить "Синим"?

Предполагается, что в начальный момент времени группировки "Зеленых" и "Синих" не могут находиться в одном и том же населенном пункте. Если сигнал тревоги поступает в тот момент, когда боевая часть "Синих" только врывается в населенный пункт, занятый "Зелеными", то используя превосходное знание местности, "Зеленым" все-таки удается скрыться в горах. Подавляющее превосходство в технике и живой силе позволяет боевым частям "Синих" организовать из каждой части любое количество экспедиций для разгрома "Зеленых". Ничто не мешает одной экспедиции за время проведения операции уничтожить несколько группировок.

Описание формата входных данных

Исходные данные задачи содержатся в файлах MAP.IN и TROOPS.IN Структура файла MAP.IN описывает карту местности. В первой строке этого файла содержатся два целых числа: N — количество населенных пунктов (0<N<256) и K — количество дорог, соединяющих эти населенные пункты (0≤K≤1000). Дороги нигде не пересекаются. В последующих K строках файла содержится схема дорог. В каждой строке записана пара двух натуральных чисел i и j и одно положительное вещественное число L_{ij}, означающее, что между населенными пунктами i и j существует дорога длиной L_{ij} километров (L_{ij}<1000).

Содержимое файла TROOPS.IN отражает размещение боевых частей воюющих сторон. Первая строка файла содержит число MF — количество боевых частей "Синих". Каждая из последующих MF строк содержит по два числа. Первое — целое число j — номер населенного пункта, в котором размещается часть, второе — вещественное неотрицательное число V_j — скорость движения боевых колонн этой части в километрах в час (V_j < 110). Далее в отдельной строке файла записано число MB — количество боевых группировок "Зеленых", за которым перечислены MB чисел — номера населенных пунктов, в которых эти группировки находятся. И, наконец, в последней строке файла хранится положительное вещественное число T, измеренное в часах (T < 24). Все числа в строках файлов разделены пробелами.

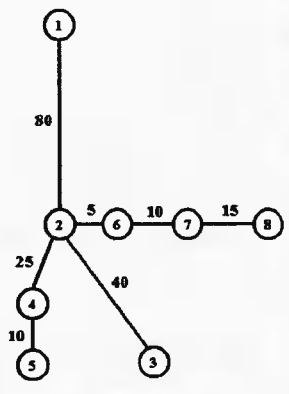
Описание формата выходных данных

Результат решения задачи необходимо вывести в текстовый файл VICTORY.OUT. В первую строку файла выводится количество разгромленных группировок, а во вторую — в порядке возрастания номера населенных пунктов, в которых эти группировки базировались.

Пример входных и выходных данных

MAP.IN	TROOPS.IN	VICTORY.OUT
8 7	2	2
1 2 80	1 50	4 8
2 4 25	6 20	
4 5 10	4 4 5 3 8	
6 2 5	2.0	
2 3 40		
7 6 10		
8 7 15		

На **рисунке** показана соответствующая примеру схема местности.



Кратко алгоритм решения задачи может быть описан так.

Методом Флойда вычисляем кратчайшие расстояния от каждой вершины до каждой. Цик-

лом по всем вершинам "Зеленых", циклом по всем вершинам "Синих". Если "Синие" успевают, значит, соответствующая группировка "Зеленых" разбита.

Рассмотрим решение задачи подробнее.

Вначале идет ввод исходных данных:

```
read(N,K);                      {ввод количества пунктов и дорог}
for x:=1 to N do        {инициализация дорог для метода Флойда}
  for y:=1 to N do a[x,y]:=1000000000000.0;
for i:=1 to K do
  begin
    read(x,y,z);
    a[x,y]:=z; a[y,x]:=z;     {корректировка матрицы смежности}
  end;
read(NumBlue);          {ввод расположения и скоростей "Синих"}
for i:=1 to NumBlue do read(Blue[i],VBlue[i]);
read(NumGreen);                  {ввод расположения "Зеленых"}
for i:=1 to NumGreen do read(Green[i]);
read(T);                        {ввод времени проведения операции}
```

Затем — расчет расстояний между всеми населенными пунктами методом Флойда:

```
for i:=1 to N do
   for x:=1 to N do
      for y:=1 to N do
         a[x,y]:=min(a[x,y],a[x,i]+a[i,y]);
```

Затем расчет количества уничтоженных группировок:

```
Deleted:=0;
for i:=1 to NumGreen do                    {для всех "Зеленых"}
  for j:=1 to NumBlue do                     {для всех "Синих"}
   if a[Green[i],Blue[j]]<Vblue[j]*T      {если "Синие" успеют}
     then
      begin
        inc(Deleted);              {уничтоженых инкрементируем}
        Num[Deleted]:=Green[i];          {номер заносим  в Num}
        break;                 {переходим к следующим "Зеленым"}
      end;
```

Наконец, в соответствии с требованиями задачи, сортируем номера уничтоженных группировок (в массиве Num) и выводим их:

```
Sort;
writeln(Deleted);
for I:=1 to Deleted do write(Num[i],' ');
```

Ниже приводится полный текст решения задачи.

Размерность исходного графа уменьшена до 100*100, поскольку 255*255 вещественных чисел не помещается в статической памяти Турбо-Паскаля (см. замечание в п.6).

```
{$R+,E+,N+}
program by00d1m3;
var
  a                    : array [1..100,1..100] of single;
  blue, green, Num     : array [1..100] of longint;
  Vblue                : array [1..100] of real;
  i,j,K,N,x,y,NumBlue,NumGreen,Deleted  : longint;
  z,T                  : real;

  function min(a,b:real):real;
      begin
          if a<b then min:=a else min:=b
      end;

  procedure Sort;
     begin
        for i:=1 to Deleted-1 do
           begin
              x:=Num[i]; y:=i;
              for j:=i+1 to Deleted do
                 if num[j]<x
                    then begin x:=Num[j]; y:=j; end;
              Num[y]:=Num[i]; Num[i]:=x;
           end;
     end;

begin
    assign(input,'map.in'); reset(input);
    assign(output,'victory.out'); rewrite(output);
    read(N,K);
    for x:=1 to N do
       for y:=1 to N do a[x,y]:=1000000000000.0;
    for i:=1 to K do
       begin
           read(x,y,z);
          a[x,y]:=z; a[y,x]:=z;
       end;
    read(NumBlue);
    for i:=1 to NumBlue do read(Blue[i],VBlue[i]);
    read(NumGreen);
    for i:=1 to NumGreen do read(Green[i]);
    read(T);

    for i:=1 to N do
       for x:=1 to N do
          for y:=1 to N do
             a[x,y]:=min(a[x,y],a[x,i]+a[i,y]);

    Deleted:=0;

    for i:=1 to NumGreen do
       for j:=1 to NumBlue do
          if a[Green[i],Blue[j]]<=Vblue[j]*T+0.00001
             then
                begin
                    inc(Deleted);
                    Num[Deleted]:=Green[i];
                   break;
                end;

  Sort;
  writeln(Deleted);
  for I:=1 to Deleted do write(Num[i],' ');

  close(input); close(output);
end.
```

2.2. Задача "Пирамида Хеопса"

(Республиканская олимпиада по информатике, 1996 г., автор — Котов В.М.)

Внутри пирамиды Хеопса есть N комнат, в которых установлены 2M модулей, составляющих M устройств. Каждое устройство состоит из двух модулей, которые располагаются в разных комнатах, и предназначено для перемещения между парой комнат, в которых установлены эти модули. Перемещение происходит за 0,5 условных единиц времени. В начальный момент времени модули всех устройств переходят в "подготовительный режим". Каждый из модулей имеет некоторый свой целочисленный период времени, в течение которого он находится в "подготовительном режиме". По истечении этого времени модуль мгновенно "срабатывает", после чего опять переходит в "подготовительный режим". Устройством можно воспользоваться только в тот момент, когда одновременно "срабатывают" оба его модуля.

Индиана Джонс сумел проникнуть в гробницу фараона (комната 1). Обследовав ее, он включил устройства и собрался уходить, но в этот момент проснулся хранитель гробницы. Теперь Джонсу необходимо как можно быстрее попасть в комнату N, в которой находится выход из пирамиды. При этом из комнаты в комнату он может попадать только при помощи устройств, так как проснувшийся хранитель закрыл все двери в комнатах пирамиды.

Необходимо написать программу, которая получает на входе описания расположения устройств и их характеристик (смотри описание формата ввода), а выдает значение оптимального времени и последовательность устройств, которыми надо воспользоваться, чтобы попасть из комнаты 1 в комнату N за это время.

Входной файл — input2.txt. Выходной файл — output2.txt.

Формат ввода:

```
N
M
R11 T11 R12 T12
...
RM1 TM1 RM2 TM2
```

Где:

- N (2≤N≤100) — количество комнат;
- M (M≤100) — количество устройств;
- Ri1 и Ri2 — номера комнат, в которых располагаются модули устройства i;
- Ti1, Ti2 (Ti1,Ti2 ≤1000) — периоды времени, через которые срабатывают эти модули.

Все числа — натуральные.

Пример

```
4
5
1 5 3 2
1 1 2 1
2 5 3 5
4 4 3 2
3 5 4 5
```

Оптимальное время — 8,5.

Искомая последовательность — 2 3 4.

Кратко алгоритм решения может быть изложен следующим образом.

Представив комнаты вершинами, а пары устройств (с временами срабатывания) — взвешенными дугами, можем применить алгоритм Дейкстра для поиска кратчайшего пути от первой вершины до последней. Точнее, алгоритм Дейкстра построит кратчайшие пути от первой до всех остальных, но в данной задаче нас интересует только кратчайший путь от первой до последней. Нужно учитывать, что дуги существуют только в моменты, кратные временам срабатывания.

Рассмотрим решение более подробно.

Ввод исходных данных:

```
read(N,M); for I:=1 to N do kG[i]:=0;
for i:=1 to M do
  begin
    read(r1,t1,r2,t2);
    nok:= (t1*t2) div Nod(t1,t2);
                          {nok – наименьшее общее кратное,}
                          {Nod – наибольший общий делитель}
                                              {чисел t1 и t2}
    inc(kG[r1]); G[r1,kG[r1]]:=r2; P[r1,kG[r1]]:=nok;
    inc(kG[r2]); G[r2,kG[r2]]:=r1; P[r2,kG[r2]]:=nok;
  end;
```

Исходный граф по условиям задачи неориентированный. Поэтому мы вводим обе дуги. При этом граф представляется в виде списка дуг, смежных с текущей:

- kG[i] — количество дуг из вершины i;
- G[i,j] — вершина, в которую из вершины i идет дуга под порядковым номером j;
- P[i,j] — вес дуги из вершины i в вершину G[i,j].

Для вычисления nod используется функция, основанная на алгоритме Евклида:

```
function Nod (a,b:longint):longint;
    begin
       while (a<>b) do  if a>b then a:=a-b else b:=b-a;
       Nod:=a;
    end;
```

Затем проводится подготовка к выполнению алгоритма Дейкстра:

```
for i:=1 to N do                                {для всех вершин}
  begin
    Labl[i]:=FREE;                        {помечаем как свободную}
    pred[i]:=1;                                 {предок – первая}
    d[i]:=maxlongint;                  {расстояние до первой – max}
  end;
Labl[1]:=DONE;                                {первая – обработана}
Pred[1]:=0;                                    {предок у первой – 0 }
d[1]:=0;                          {расстояние от первой до первой – 0}
for j:=1 to kG[1] do                {для всех дуг из первого ребра}
  d[G[1,j]]:=P[1,j]+0.5;             {расстояние до первой вершины}
                                                {равно весу ребра + 0.5}
```

Заметим, что в последней строке к весу ребер добавляется 0,5, поскольку по условиям задачи перемещение с помощью устройств из одной комнаты в другую занимает 0,5 единицы времени.

Далее идет основной цикл алгоритма Дейкстра

```
for i:=1 to N-1 do
  begin
    Поиск ближайшей к первой вершине из необработанных.
    Пересчет кратчайших расстояний через ближайшую вершину
    до вершин, смежных с ближайшей.
  end;
```

Первый этап — "Поиск ближайшей к первой вершине из необработанных":

```
MinD:=maxlongint;                                    {MinD – max}
for j:=1 to N do                                  {для всех вершин}
  if (Labl[j]=FREE)                          {если вершина свободна}
     and (d[j]<MinD)                  {и расстояние от нее до первой}
                                                 {вершины меньше MinD}
    then
      begin
        MinD:=d[j];                  {заносим это расстояние в MinD}
        Bliz:=j                    {вершину запоминаем как ближайшую}
      end;
Labl[Bliz]:=DONE;                    {найденную ближайшую помечаем}
                                                  {как обработанную}
```

Второй этап — "Пересчет кратчайших расстояний через ближайшую вершину до вершин, смежных с ближайшей":

```
for j:=1 to kG[Bliz] do  {для всех вершин смежных с ближайшей}
  if (Labl[G[Bliz,j]]=FREE){если вершина еще не обрабатывалась}
  then
    begin
      NewTime:=Calculat(d[bliz],P[bliz,j]);
                                        {вычислить расстояние до нее}
                  {в терминах задачи - время перемещения в нее}
      if d[G[Bliz,j]]> NewTime +0.5  {если новое время лучше,}
        then
          begin
           d[G[Bliz,j]]:= NewTime+0.5; {заносим его в массив D}
           pred[G[Bliz,j]]:=Bliz;     {указываем ближайшую как}
          end;                         {предыдущую для текущей}
    end;
```

При вычислении кратчайшего расстояния до текущей вершины через ближайшую (в терминах алгоритма Дейкстра) или времени перемещения из первой комнаты через ближайшую в текущую (в терминах данной задачи) используется функция Calculat (T,Tnok):

```
function Calculat (T:single; Tnok:longint):longint;
  var i : longint;
  begin
    TRes:=0;
    while (T>TRes) do inc(TRes,Tnok);
    Calculat:=TRes ;
  end;
```

Эта функция вычисляет время TRes (которое передается в вызывающую программу непосредственно через имя функции Calculat), ближайшее большее текущего времени T (T равно минимальному значению времени, которое потребовалось, чтобы оптимальным образом добраться из первой вершины до ближайшей вершины) и кратное Tnok — периоду срабатывания пары устройств, связывающих комнаты Bliz (ближайшая) и G[Bliz,j] (текущая).

После выполнения алгоритма Дейкстра будут сформированы два массива:

- d[j] — кратчайшее расстояние от начальной вершины до вершины j;

- pred[j] — предок вершины j на оптимальном маршруте.

По этой информации вывод результата осуществляется следующим образом:

```
tg:=N; i:=0;                          {начиная с последней вершины}
while tg<>0 do                {пока не закончились предшествующие}
  begin
    inc(i);              {инкрементируем количество вершин в пути}
    path[i]:=tg;                  {заносим текущую в массив пути}
    tg:=pred[tg];                  {переустанавливаем предыдущую}
  end;
writeln('Оптимальное время: ',D[N]:0:1);
write('искомая последовательность:');
for j:=i-1 downto 1 do write(' ',path[j]);
```

Ниже приведен полный текст решения задачи:

```
{$R+,E+,N+}
program by96dls2;
const
    FREE=1;
    DONE=2;
var
   G                              : array[1..100,1..100] of byte;
   P                              : array[1..100,1..100] of
longint;
   D                              : array[1..100} of single;
   Labl,pred,kG          : array[1..100] of byte;
   path                           : array[1..100] of byte;
  is,t1,t2,nok,Tres,NewTime   : longint;
  i,j,x,y,A,B,C,N,M,tg,Bliz   : byte;
  r1,r2                          : byte;
  Yes                            : boolean;
  MinD                           : single;

  function Nod (a,b:longint):longint;
     begin
         while (a<>b) do
             if a>b then a:=a-b else b:=b-a;
         Nod:=a;
     end;

  function Calculat (T:single; Tnok:longint):longint;
     var i : longint;
     begin
         TRes:=0;
         while (T>TRes) do inc(Tres,Tnok);
         Calculat:=TRes ;
     end;
begin
   assign(input,'input2.txt'); reset(input);
   assign(output,'output2.txt'); rewrite(output);
   read(N,M);
   for I:=1 to N do kG[i]:=0;
   for i:=1 to M do
       begin
           read(r1,t1,r2,t2); nok:= (t1*t2) div Nod(t1,t2);
           inc(kG[r1]); G[r1,kg[r1]]:=r2; P[r1,kg[r1]]:=nok;
           inc(kG[r2]); G[r2,kg[r2]]:=r1; P[r2,kg[r2]]:=nok;
       end;
  for i:=1 to N do
     begin
         Labl[i]:=FREE; pred[i]:=1;  d[i]:=maxlongint;
     end;
  Labl[1]:=DONE; Pred[1]:=0; d[1]:=0;
  for j:=1 to kG[1] do d[G[1,j]]:=P[1,j]+0.5;
  for i:=1 to N-1 do
      begin
          MinD:=maxlongint;
          for j:=1 to N do
             if (Labl[j]=FREE) and (d[j]<MinD)
                then begin MinD:=d[j]; Bliz:=j end;
          Labl[Bliz]:=DONE;

          for j:=1 to kG[Bliz] do
              if (Labl[G[Bliz,j]]=FREE)
                 then begin
                         NewTime:=Calculat(d[bliz],P[bliz,j]);
                         if d[G[Bliz,j]]> NewTime +0.5
                            then
                               begin
                                 d[G[Bliz,j}]:= NewTime+0.5;
                                  pred[G[Bliz,j]]:=Bliz;
                               end;
                      end;
      end;
  tg:=N; i:=0;
  while tg<>0 do
      begin
          inc(i); path[i]:=tg; tg:=pred[tg];
      end;
  writeln('Оптимальное время: ',D[N]:0:1);
  write('искомая последовательность:');
  for j:=i-1 downto 1 do write(' ',path[j]);
  close(input); close(output);
end.
```

(Продолжение следует)

Serrgio /HI-TECH,
http://hi-tech.nsys.by/
E-mail: serrgio@gorki.unibel.by

НИЗКОУРОВНЕВОЕ ПРОГРАММИРОВАНИЕ

(Продолжение. Начало в №№9-12/2001, №№1-6/2002)

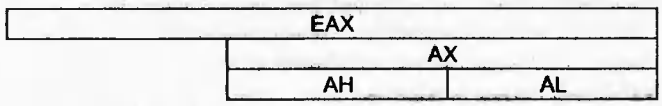
Программирование под WIN32

2. Формат данных (от лукавого)

#1. Как известно, объем жидкости измеряется в $см^3$;). Но когда мы покупаем, например, пиво, то измеряем его вовсе не в кубических сантиметрах, а в совершенно других единицах измерения: бутылках, банках, ящиках, канистрах, бокалах или, на худой конец, в литрах. Точно так и информация — с одной стороны, она, конечно же, измеряется битами, байтами, килобайтами и т.д., но с другой стороны — существуют и ее элементарные "потребительские куски". Как мы пьем глотками пиво, так и процессор потребляет информацию своими небольшими "глоточками" ;).

Ранее мы уже разбирались, что такое регистры и какими кусками они могут держать в себе информацию. Однако со времен процессора 8086 прошло очень много времени, и регистры немного подросли. Поэтому мы для начала познакомимся с *современным* вариантом "нездоровой" схемки из **главы 1.3. #3**, и только потом поговорим о данных.

Типичный **регистр общего назначения** выглядит следующим образом:



Что в нем изменилось? Да просто его (регистра) стало в два раза больше, и максимальное число, которое мы можем в этот регистр записать — это FFFFFFFFh (4294967295d, 11111111111111111111111111111111b). Однако обратите внимание (см. рисунок), что EAX — это все лишь **расширенная версия** регистра AX. То есть мы не можем обратиться к "старшей половине" EAX, как делали это, например, при обращении к частям AX через "половинки" AH|AL. Другими словами, если **mov AX,1234h** и парочка **mov AH,12h; mov AL,34h** — это "один черт", то про **mov EAX,12345678h** ничего похожего мы сказать не можем.

"Неделимые" регистры SI, DI, SP и BP также имеют свои расширенные версии (ESI, EDI, ESP и EBP), а вот сегментные

регистры CS, SS, DS, ES, GS и FS сия участь миновала, так как парочек наподобие ES:EBX (сегмент:смещение) вполне достаточно и сейчас. Заметим, что сегментные регистры попрежнему задействованы во всех инструкциях доступа к памяти (например, **mov ax,[var]** всегда идет через DS, а **call [funcvar]** использует оба — DS и CS), однако при программировании под Win32 об этом обычно можно не беспокоиться.

Повторюсь опять и опять — если простеньким программкам под Win32 сегментные регистры ***явно*** *использовать нет надобности, то это вовсе не значит, что они перестали использоваться процессором при адресации данных в "сегменте данных", "на стеке" и в строковых инструкциях. Не говоря уже про явное использование сегментных регистров в сУрьезных программах ;).*

На первых порах мы будем "замалчивать" существование сегментных регистров, но знайте, что утверждение об их "устарелости" — брехня.

Однако, вернемся к нашим баранам.

#2. Для описания простейших типов данных на языке ассемблера используются так называемые **директивы резервирования и инициализации** данных, которые по сути являются указаниями **транслятору** на выделение определенного объема памяти.

Например, мы неоднократно использовали директиву DB (define byte), которая инициализировала столько байтов, сколько нашей душеньке было угодно. Например, **DB "Hello, World"** "резервировала и инициализировала" столько байтов, сколько символов нам вздумалось написать между кавычками или апострофами.

Однако, помимо DB, существует еще куча подобных директив. Я приведу вам их все. Некоторые из них вас ужаснут, однако расслабьтесь — вовсе не обязательно, что вы будете все их использовать. Но ознакомиться с ними — крайне желательно. Подозреваю, что впоследствии мы вернемся к этой главе еще не раз.

Для начала мы "построим" эти директивы согласно размеру памяти, который они резервируют. А заодно с трех сторон посмотрим на диапазон значений, которые эти "элементы данных" могут принимать:

DB	Байт, byte	1	0...FFh	0...255	-128...+127
DW	Слово, word	2	0...FFFFh	0...65535	-32768...+32767
DD	Даойное слово, double word	4	0...FFFFFFFFh	0...4294967295	-2147483648...+2147483647
DF	Указатель дальнего словв, far word	6	0...FFFFFFFFFFFFh	$0...2^{48}-1$	$-2^{47}...+2^{47}-1$
DQ	Учетверенное слово, quadword	8	0...FFFFFFFFFFFFFFFFh	$0...2^{64}-1$	$-2^{63}...2^{63}-1$
DT	10 байтов, ten bytes	10	0...FFFFFFFFFFFFFFFFFFFFh	$0...2^{80}-1$	$-2^{79}...2^{79}-1$

Расшифровывается эта табличка следующим образом.

Первая колонка — это сама инструкция. Здесь первая буква — D — от буржуйского *define*, т.е. "определить", а вторая — B, W, D и т.д. — от размера определяемого элемента данных, т.е. *byte* (байт), *word* (слово), *doubleword* (двойнов слово), и т.д., как во второй колонке. Третья колонка — размер в байтах. Четвертая — диапазон принимаемых значений в шестнадцатеричной системе счисления (хе..., я понимаю, что последние две строчки в таблице несколько длинноаты ;), однако, вроде, никто еще не придумал писать шестнадцатеричные числа в экспоненциальном виде). Пятая колонка — это соответствующий диапазон десятичных беззнаковых чисел, шестая — знаковых десятичных.

Во-первых, обратите внимание на инструкцию DF — это как раз случай *длинного сегментированного указателя* (32 бита смещения плюс 16 битов сегмента). И для него, кстати, есть синоним — DP.

Во-вторых, помимо целых чисел, как аргументы можно указывать *нецелые*. DD может использоваться для описания и хранения *коротких* (single-precision) нецелых чисел — $+/-10^{-38}...10^{38}$, DQ — для *длинных* (long, double) нецелых $+/-10^{-308}...10^{308}$, а DT — для *"временных"* (temporary) нецелых (диапазон не помню). Например:

```
dd 3.141
dq 1.2345678987654321
dt 0.0000000001
```

Вообще-то, если быть совсем точным, то помимо интерпретации значений в соответствующих ячейках как целых, они могут интерпретироваться следующим образом: DB — байт или знак; DW — int/unsigned или 16-битное смещение; DD — long, float, 16-битный far (сегмент:смещение) или 32-битный близкий (смещение) указатель; DF — 32-битный far (сегмент:смещение) указатель; DQ — double; DT — упакованное десятичное (packed BCD) число длиной 20 цифр или long double (оба — формата FPU).

Вот еще одна интересная табличка. Все-таки у нас программирование для дЗенствующих ;), а это подразумевает в первую очередь рассмотрение любого "объекта" с нескольких точек зрения. Первая колонка — это директива, а вторая — так называемое адресное выражение:

DW	16-битный адрес сегмента; смещение в 16-битном сегменте
DD	16-битный адрес + 16-битное смещение; 32-битный адрес
DF	16-битный адрес сегмента + 32-битное смещение

В принципе, все перечисленные выше директивы можно использовать для "задания" строкового значения. Однако имейте в виду, что в памяти они могут выглядеть совсем не так, как вы задали их в директиве, поэтому для инициализации строк мы всегда будем использовать DB. Следующая же "простынка" приводится исключительно в "образовательных" целях, и ни в коем случае не следует воспринимать ее как руководство к действию ;):

```
String_1 DB 'A'
String_2 DW 'As'
String_3 DD 'Asse'
String_4 DF 'Assemb'
String_5 DQ 'Assemble'
String_6 DT 'Assembler ' ;после r - пробел
String_7 DB 'Assembler is ruleZ forever!'
```

Вы должны четко усвоить одно — существует только 6 типов переменных: **байт, слово, двойное слово, "дальнее слово"** (давайте будем именно так называть "дальний указатель" в формате сегмент : 32-битное смещение), **учетверенное** слово и **"тенбайт"** ("10 байт"). Все остальное — от лукавого ;).

И, наконец, третья табличка. В тех случаях, когда мы захотим инициализировать локальную переменную, существующую только в пределах какой-либо функции, и умирающую, как только функция свое "отработает", мы будем использовать несколько другие директивы. Вот табличка их "соответствия":

DB	BYTE/SBYTE
DW	WORD/SWORD
DD	DWORD/SDWORD
DF	FWORD
DQ	QWORD
DT	TBYTE

Два варианта "соответствия" через слэш — это беззнаковое (unsigned) и знаковое (signed), соответственно.

#3. Как я уже говорил в п.1.#4 прошлого номера, вся документация, которую добренький Микрософт задарма предоставляет своим девелоперам, "заточена" под Си. А Си — это, да будет вам известно, все же язык программирования, пре-

тендующий на высокоуровневость. И те же 32 бита *двойного слова* могут быть обозваны и как *указатель на строку символов* (LPCSTR), и как *результат, возвращаемый функцией* (LRESULT), и даже, страшно подумать, *цветом* (COLORREF).

Вам это нравится? Мне тоже — нет. Однако, если мы хотим программировать приложения под Win32, нам нужно научиться использовать функции API. Информация же об этих функциях находится в Си-"заточенном" MSDN'е. Что же делать? Будем учиться "перезатачивать"!

Загрузите свой MSDN и поищите там, например, функцию ***WriteConsole*** (в закладке Index). Смотрим на страничку в описании:

Функция **WriteConsole** записывает в строку символов буфер экрана консоли (*console screen buffer*), начиная с текущей позиции курсора.

```
BOOL WriteConsole(
  HANDLE hConsoleOutput,          // хэндл буфера экрана
  CONST VOID *lpBuffer,           // буфер для записи
  DWORD nNumberOfCharsToWrite,    // число симоволов для записи
  LPDWORD lpNumberOfCharsWritten, // число записаных символов
                                  // (указатель на...)
  LPVOID lpReserved               // резерв
);
```

Слово BOOL перед нашей функцией означает, что результат, который она нам вернет через регистр EAX (все апишные функции возвращают результат через этот регистр) — булевский.

Читаем описание. Возвращаемое значение "не 0", если функция выполнилась успешно, и 0, если сглючила. А дальше, в скобках, какие-то непонятные парочки слов наподобие **HANDLE hConsoleOutput**... Как их прикажете понимать?

Первое слово из "сладкой парочки" — это тип переменной, а второе — ее "имя собственное". Извлечь необходимую нам информацию мы можем как из первого, так и из второго слова. Первое слово — это тип переменной, которых в Си "целый вагон с маленькой тележкой". "Разгрузить" его попробуйте самостоятельно, обратившись к разделу *Platform SDK* MSDN, к топику *Data Types*. Второе же слово — это "имя собственное" переменной, однако имя хитрое — с **префиксом**, по которому мы с той или иной степенью вероятности можем судить о **типе** этой переменной (это не потому что правила языка таковы, а потому что в Мелкософте решили все "кишки" переменной описывать в ее имени, причем как префикс, а не как суффикс — чтобы жизнь медом не казалась при работе с сорсами от Мелкософта; называется сия гадость "венгерская нотация").

Расшифровываю:

b	Байт	BYTE
w	Слоао	WORD
dw	Даойное слоао	DWORD
h	Хэндл	DWORD
lp	Указатель	DWORD
n	"Количество"	DWORD

Это — небольшой кусочек из так называемой **венгерской нотации**, которой Мелкософт старается следовать. Суть ее состоит в том, что имя переменной должно быть не просто идентификатором, но еще и нести в себе некоторые целевые характеристики. Притом не обязательно "размерные", но и "функциональные", с которыми вы можете ознакомиться в MSDN'овском топике *"Hungarian Notation"*.

#4. Теперь, когда мы краешком глаза взглянули на многообразие типов данных великого и претендующего на сильную могучесть языка Си, давайте попробуем перевести топик про WriteConsole на язык Эллочки-людоедки, то бишь ассемблерный:

hConsoleOutput. Префикс h, значит (смотрим на табличку) — DWORD.

CONST VOID *lpBuffer. CONST и VOID, конечно же, смущают, но в имени переменной есть префикс lp, значит — DWORD.

DWORD nNumberOfCharsToWrite. Префикс n, значит DWORD (хотя, собственно, при чем тут префикс? Про DWORD же прямым текстом написано!)

И так далее...

Остается открытым вопрос, что же это за "указатель" такой. По большому секрету скажу, что это самый обыкновенный **адрес**. А вот **хэндл**... брррр... в общем, это индекс в массиве данных, который позволяет таскать *вместо* всей *структуры данных* только ее относительно *короткий "идентификатор"* (сама же структура хранится, разумеется, где-то в глубинах системы, что, к тому же, позволяет обеспечить ее целостность, несмотря на неосторожные или агрессивные действия со стороны программы). Короче, хэндл — тот же адрес, только адрес является *индексом памяти (массива байт)*, а **хэндл** — индексом в некотором массиве, который лежит в этой памяти. Попробуйте это представить. Получилось?

На этой оптимистической ноте ;) переходим к следующей части — написанию "Hello, World" под Win32.

3. "Hello, World" и три халявы MASM32

#1. С легкой левой руки Дениса Ричи повелось начинать освоение нового языка программирования с создания простейшей программы "Hello, World". Ничто человеческое нам не чуждо — давайте и мы совершим сей грех.

В главе **1. Минимальное приложение** я уже рассказал о том, как работать в ассемблере с апишными функциями, однако вы наверняка не поняли ;). Это нормально, и не нужно из-за этого беспокоиться. Все станет более чем ясным после того как мы с вами напишем одну-две простенькие программки и разберем их по строчкам.

Заново перечитайте *"Минимальное приложение"* и набейте следующий исходник:

```
;-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
;        ПРОЦ, МОДЕЛЬ, ОПЦИИ, ИНКЛУДЫ, БИБЛИОТЕКИ ИМПОРТА
;-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

.386
.model flat,stdcall
option casemap:none

includelib kernel32.lib

SetConsoleTitleA PROTO :DWORD
GetStdHandle PROTO     :DWORD
WriteConsoleA PROTO   :DWORD,:DWORD,:DWORD,:DWORD,:DWORD
ExitProcess PROTO      :DWORD
Sleep PROTO            :DWORD

;-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
;                       СЕКЦИЯ КОНСТАНТ
;-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

.const

sConsoleTitle db 'My First Console Application',0
sWriteText    db 'hEILo, Wo(R)LD!!'

;-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
;                        СЕКЦИЯ КОДА
;-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

.code

;-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
;                  Самая Главная Процедура
;-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Main PROC
  LOCAL hStdout :DWORD         ;(1)

  ;титл консоли
  push offset sConsoleTitle    ;(2)
  call SetConsoleTitleA
```

```
  ;получаем хэндл вывода        ;(3)
  push -11
  call GetStdHandle
  mov hStdout,EAX

  ;выводим HELLO, WORLD!        ;(4)
  push 0
  push 0
  push 16d
  push offset sWriteText
  push hStdout
  call WriteConsoleA

  ;задержка, чтобы полюбоваться ;(5)
  push 2000d
  call Sleep

  ;выход                        ;(6)
  push 0
  call ExitProcess

Main ENDP

;-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

end Main
```

Вот две строчки из моего "батника" (*.bat), который позволяет не "париться" с командной строкой:

```
c:\tools\masm32\bin\ml /c /coff hello.asm
c:\tools\masm32\bin\link /SUBSYSTEM:CONSOLE /LIBPATH:c:\masm32\lib hello.obj
```

Обращаю внимание, что для сборки консольного приложения необходимо использовать ключ /SUBSYSTEM:CONSOLE. Несмотря на то что окошко, в котором оно запустится, до боли напоминает "сеанс MS-DOS", получившаяся программа — полноценное виндозное 32-битное приложение в формате PE. Ассемблируем, линкуем, запускаем, наслаждаемся...

#2. А теперь давайте устроим этому исходнику разборку.

Бряк 1. Таким образом мы определяем **локальную переменную** с именем **hStdout** и размером в двойное слово (**DWORD**). Почему *локальная*? А потому, что она существует только внутри процедуры **Main**, и если бы мы попытались обращаться к переменной **hStdout** за пределами этой процедуры, ассемблер бы ругал нас всякими нехорошими словами — в отличие от, скажем, константы **sWriteText**, имя которой "известно" в любом месте нашей программы.

Обратите внимания на префикс h в названии переменной. Это я просто оставил для себя памятку, что переменная заведена под **хэндл**.

Бряк 2. Апишная функция SetConsoleTitleA — устанавливаем титл (заголовок) для нашего консольного окошка. Вот выдержка из MSDN'a:

```
BOOL SetConsoleTitle(
  LPCTSTR lpConsoleTitle // new console title
);
```

Как видим, функция требует один-единственный параметр — указатель на строку символов, которую мы хотим вывести в заголовке окна. Строка должна заканчиваться нулем.

Команда **push offset sConsoleTitle** помещает в стек (**push**) адрес (**offset**) строки символов (помеченной как **sConsoleTitle**). Ну а далее следует, собственно, сам вызов (**call**) функции **SetConsoleTitle**.

Заметьте, для указания адреса используется префикс под названием **offset**. Это потому, что берется смещение (offset) относительно начала сегмента, которое и является "ближним адресом". Есть еще "дальние" адреса, в которых задействуется также сам сегмент, но это тема будущих разговоров — сейчас это нас не должно волновать.

Здесь у вас должен возникнуть вполне закономерный вопрос — почему мы дописали букву А в конец функции? В MSDN'e ведь нет никакой буквы А... Я отвечу на этот вопрос немного позже.

Бряк 3. Консоль мы можем использовать как **устройство ввода** (*input device*), **устройство вывода** (*output device*), устройство для **отчета об ошибках** (*error device*). Для того чтобы работать с этим "девайсом", мы должны получить его хэндл.

Рискуя быть обруганным педантичной частью программирующей общественности, попробую объяснить, что такое хэндл — популярно.

Например, есть у нас какое-либо *устройство*, обладающее целым рядом признаков, свойств, обработчиков, структур и т.д. Такого рода *"досье на устройство"* хранятся в операционной системе, но работать с ними напрямую, постоянно копируя всю сопутствующую информацию — никакого здоровья не хватит (не говоря уже о том, что есть опасность при копированиях создать несинхронизованные копии). И слава Богу, что от нас не требуется таскать весь этот багаж с собой — нам достаточно получить от системы *"квиток"*, однозначно идентифицирующий *"досье"*, и потом передавать его всем функциям, которые должны работать с соответствующим устройством. И все дела...

Теперь, как полноправный представитель программирующей общественности, сам же себя и обругиваю.

Во-первых, понятие хэндла — понятие всеобъемлющее, и оперировать им, говоря об устройствах, можно только с какой-то *специальной* целью.

Во-вторых, максимум, что общего можно сказать про все хэндлы разом — это только то, что в контексте, в котором данный хэндл существует, он *однозначно идентифицирует* некий уникальный объект, благодаря чему другие объекты могут работать с этим объектом, обращаясь к нему только посредством упоминания этого хэндла и не сомневаясь в том, что ЭТО — тот самый объект, который им нужен.

А уж "по совместительству" хэндлы могут нести разнообразную конкретную нагрузку. Например: указатель на структуру (в более общем случае — экземпляр класса), содержащую(щего) свойства (и методы), обслуживающие устройство. Или, при выделении зафиксированного блока памяти, хэндл может одновременно являться и указателем на начало этого самого блока. Или, если открываем файл, то хэндл — это просто порядковый номер открытого файла в контексте приложения; и т.д.

В-третьих, *во всех остальных перечисленных случаях навешивание конкретных смыслов на хэндлы — позорное отступление от священных принципов ООП!* Ибо хэндлы ради того и были придуманы, чтобы не нужно было никому объяснять их физического смысла. В этом их квинтэссенция и сверхзадача. (а также абстракция, на пару с инкапсуляцией)...

Во как завернул ;).

Хэндл на стандартное устройство ввода-вывода мы получаем при помощи следующей функции:

```
HANDLE GetStdHandle(
  DWORD nStdHandle   // input, output, or error device
);
```

Единственный параметр, который она от нас требует — указание, на какое устройство мы желаем получить "квиток"-хэндл. Вот табличка:

Хэндл стандартного ввода	-10
Хэндл стандартного вывода	-11
Хэндл "ошибок"	-12

Что нам нужно? Вывести строчку! Значит — запрашиваем хэндл для стандартного вывода, то есть перед вызовом функции "суем" в стек -11. После выполнения функции регистр EAX содержит столь желанный "хэндл стандартного вывода". Кладем этот хэндл в переменную **hStdout** (которую мы столь предусмотрительно определили на бряке 1) для последующего использования.

— Это что ж за безобразие? — воскликните вы. *— Что это за таблица такая нездоровая? Какие-то отрицательные числа, которые ни в жисть не запомнить! Хотим таблицу как в MSDN'е! Чтобы не -10, -11, -12, а длинные мнемонические STD_INPUT_HANDLE, STD_OUTPUT_HANDLE, STD_ERROR_HANDLE!*

Спокойно! Исходник, который мы сейчас рассматриваем, весьма точно отображает реальные процессы, происходящие в программе. Чуть позже мы приведем его к варианту в стиле Си и посмотрим, как можно использовать некоторые высокоуровневые конструкции, значительно облегчающие жизнь низкоуровневому программисту.

Бряк 4. Ну наконец-то, самое главное — функция, которая, собственно, и выводит на консоль строку символов. Вот ее описание:

```
BOOL WriteConsole(
  HANDLE hConsoleOutput,          // handle to screen buffer
  CONST VOID *lpBuffer,           // write buffer
  DWORD nNumberOfCharsToWrite,    // number of characters to write
  LPDWORD lpNumberOfCharsWritten, // number of characters written
  LPVOID lpReserved               // reserved
);
```

Расшифровываем. Перед вызовом функции WriteConsole мы должны поместить в стек целых пять параметров:

1. **Хэндл.** Какие проблемы? Мы его уже получили и предусмотрительно сохранили в переменной **hStdout**. Командой **push hStdout** заносим его в стек, и все дела.

2. **Указатель на строку символов**, которую мы хотим напечатать. Сама строка у нас определена в секции констант под именем **sWriteText**. Получить ее адрес мы можем при помощи **offset**. Укладываем все в одну строчку — **push offset sWriteText**. Два в одном — и адрес получаем, и в стек его заталкиваем :).

3. **Число символов**, которые мы хотим напечатать. В смысле — число "буковок" из строки **sWriteText**. Сколько символов в строке "hEllLo, Wo(R)LD!!"? Включая пробелы — 16d. Пишем — **push 16d**. Заметьте, функция WriteConsole не требует нуля в конце буфера!

4. **Указатель на переменную**, в которой будет возвращено **число напечатанных символов**. Функция нам любезно сообщает, сколько символов из шестнадцати ей удалось напечатать. И требует переменную, в которую эту информацию ей занести. Давайте сделаем вид, что она нам не нужна, то есть напишем 0. Ничего страшного не случится, а в ошибочности подобного рода игнорирований убедимся в следующей главе. Пишем — **push 0**, но для себя оставляем пометку, что *что-то функция от нас все же хотела.*

5. **Резерв.** Так сказать, зарезервировано для следующих версий. Смело пишем — **push 0**.

Теперь, когда мы разобрали все параметры, обратите внимание на то, что MSDN'овская очередность параметров не соответствует той очередности, в которой мы записываем их в стек в нашем исходнике. Вернитесь еще раз к **Минимальному приложению, п.12** и внимательно прочитайте пункты соглашения **stdcall**. Теперь понятно?

Бряк 5. Дабы мы успели полюбоваться результатом трудов своих праведных, при помощи функции **Sleep** вызываем программную задержку в 2 секунды. Думаю, с параметрами вы без труда разберетесь.

И, наконец, **бряк 6** — выход из программы.

Вообще-то, правильный стиль предполагает явное освобождение всех занятых ресурсов по минованию надобности в них, в том числе и хэндлов, несмотря на то что они автоматически закрываются ExitProcess'ом. Но будем надеяться, что если мы не сделаем это в такой маленькой программулине как наша, ничего страшного не случится. Естественно, "формат цэ" не в счет.

#3. Теперь делаем первый шаг по приведению нашего сырца в более читабельный вид.

Итак, первое, с чем мы ознакомимся — это эквиваленты, прописанные в файле /MASM32/windows.inc.

Мы уже сталкивались с MSDN'овской табличкой:

Value	Meaning
STD_INPUT_HANDLE	Standard input handle
STD_OUTPUT_HANDLE	Standard output handle
STD_ERROR_HANDLE	Standard error handle

Однако вместо мнемонического интуитивно-понятного аргумента STD_OUTPUT_HANDLE вносили в стек значение -11, неизвестно откуда взятое.

Давайте напишем сразу же после директивы **includelib** следующую строчку:

```
STD_OUTPUT_HANDLE equ -11d
```

А строчку **push -11** заменим на **push STD_OUTPUT_HANDLE**.

Что получилось? Программа откомпилировалась без проблем, ибо в самом начале листинга мы прописали equ [ивален]. Проще говоря, мы сказали ассемблеру: "если ты встретишь в тексте программы STD_OUTPUT_HANDLE, то имей в виду, что это то же самое, что и -11". Другими словами, завели нечто типа константы (не переменную!) с именем STD_OUTPUT_HANDLE и значением -11.

Теперь откройте файл **windows.inc** и полюбуйтесь его содержимым. Там целая куча "эквивалентов", наподобие вышерассмотренного! И чтобы воспользоваться этой халявой, вовсе не обязательно копировать ту или иную константу через буфер обмена. Можно поступить намного проще — добавить в исходник директиву **include [путь к файлу] windows.inc**. В ответ на это ассемблер сам извлечет из **windows.inc** всю имеющуюся в этом файле информацию и преподнесет ее транслятору на блюдечке с голубой каемочкой.

#4. Вторая халява, которой мы с вами воспользуемся — это "инклуды" (давайте именно так будем называть файлы *.inc) с прототипами функций. Мы уже рассматривали, что такое прототипы, и какую роль они играют при линковке нашей программы с библиотеками импорта. Конечно же, мы можем сами, на основе MSDN'овкого описания функции, вывести ее прототип, но зачем нам приумножать сущности сверх необходимого? Ведь в MASM32 для каждой из библиотек импорта есть и одноименный файл с прототипами. В нашем примере мы использовали функции kernel32 и для этого линковали его с библиотекой kernel32.lib? Ну а соответствующий файл с прототипами называется kernel32.inc!

Что может быть проще? Из нашего исходника вырезаем к черту блок с прототипами, а на его место лепим директиву **include [путь] kernel32.inc**. Компилим, и, как говорят по телеку, "теперь вы можете забыть об этих неудобных промокающих ..." (ууупс... опять пошли брутальные фантазии; время начинать новый абзац...).

Теперь, пожалуй, пришло время сдержать свое обещание и объяснить — какого черта мы к концу функции WriteConsole прилепили букву "A". Объясняю — а потому что нет в винде функции WriteConsole!

#5. ...зато есть функции **WriteConsoleA** и **WriteConsoleW**. "A" — это если вы хотите напечатать строку в формате ASCII (т.е. каждый знак занимает один байт), а "W" — если в Unicode (W — от wide, широкий. В Unicode знаки не 8-битные, а 16-битные, и занимают два байта). Подобные окончания имеют только те функции, которые тем или иным образом работают со строковыми значениями. Функция **ExitProcess**, например, подобного буквенного окончания не имеет — посудите сами, не все ли равно, на каком национальном языке завершать работу приложения?

Откроем файл **kernel32.inc** и пристально посмотрим на его содержимое, в частности, на следующее:

```
WriteConsoleA PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD
WriteConsole equ <WriteConsoleA>
```

Как видим, команда разработчиков MASM32 позаботилась не только о простыне прототипов, но и о "независимости" нашего исходника от выбранной кодировки. То есть, для того чтобы "перезаточить" программу под UNICODE, нам вовсе не нужно заменять окончание A на W в имени функции. Достаточно просто приинклюдить другой файл с прототипами и эквивалентами наподобие

```
WriteConsoleW PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD
WriteConsole equ <WriteConsoleW>
```

и не "париться" с переписыванием исходника.

Надо отметить, в MASM32 подобного "юникодного" инклуда нет, однако вы легко можете сделать его сами.

#6. И, наконец, третья, самая большая "халява" — это маленькая фенечка, использование которой сразу же превращает макроассемблер из *языка кодирования* в *язык программирования*!

С помощью этой "фенечки" целый блок инструкций:

```
push 0
push 0
push 16d
push offset sWriteText
push hStdout
call WriteConsoleA
```

мы с легкостью можем заменить одной-единственной строчкой:

```
invoke WriteConsoleA, hStdout, offset sWriteText, 16d, 0, 0
```

Обратите внимание, что при использовании этой команды параметры мы передаем слева направо, в той же очередности, что и вещает нам MSDN. В отличие от простыни "пушей" с "каллом" в конце.

#7. Теперь самый главный момент... Затаите дыхание!

В свете вышесказанного, вышерасписанного и вышерасжеванного наш исходник принимает весьма красивый "высокоуровневый" вид:

```
.386
.model flat,stdcall
option casemap:none

includelib kernel32.lib
include windows.inc
include kernel32.inc

.const

sConsoleTitle db 'My First Console Application',0
sWriteText db 'hEILo, Wo(R)LD!!'

.code

Main PROC
  LOCAL hStdout :DWORD

  invoke SetConsoleTitle, offset sConsoleTitle
  invoke GetStdHandle, STD_OUTPUT_HANDLE
  mov hStdout,EAX
  invoke WriteConsole, hStdout, offset sWriteText, 16d, NULL, NULL
  invoke Sleep, 2000d
  invoke ExitProcess, NULL

Main ENDP

end Main
```

А что? Самое время выпить бутылочку пива ;).

Программирование алгоритмов с использованием функций и процедур. Создание модулей

А.ШАКИРИН,
г.Минск, БАТУ, каф.ВТ.

Цель этой статьи — освоить методику создания модулей, содержащих процедуры и функции, и научиться использовать их в проекте.

1. Пример создания приложения

Необходимо создать Windows-приложение, которое выводит таблицу значений функции

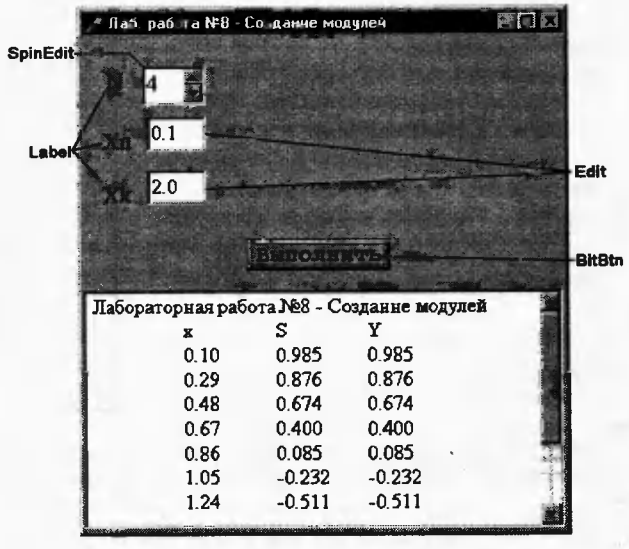
$$Y(x)=\left(1-\frac{x^2}{2}\right)\cos x-\frac{x}{2}\sin x$$

и ее разложения в ряд в виде суммы

$$S(x)=\sum_{n=0}^{n}(-1)^n\frac{2n^2+1}{(2n)!}x^{2n}$$

для значений x от x_n до x_k с шагом $h=(x_k-x_n)/10$. Для этого надо написать модуль, в котором вычисление значений $Y(x)$ следует оформить в виде функции, а вычисление $S(x)$ — в виде процедуры. Затем необходимо подключить модуль к проекту и выполнить созданное приложение.

Один из возможных вариантов панели интерфейса создаваемого приложения показан на **рисунке**.



1.1. Размещение компонентов на Форме

Разместим на Форме компоненты Label, Edit, SpinEdit, BitBtn и Memo.

Сохраним модуль под именем UnModul.

1.2. Создание модуля и подключение его к проекту

В соответствии с поставленной задачей создадим модуль, в котором вычисление значений $Y(x)$ оформим в виде функции, а вычисление $S(x)$ — в виде процедуры. Для создания модуля откроем в главном меню пункт File и щелкнем мышью на опции New...(Новый...). Delphi откроет панель New Items (Репозиторий), в которой сделаем активной пиктограмму **Unit** (Модуль) и нажмем кнопку OK. Откроется окно с "пустым" модулем Unit2. С помощью опции Save As... меню File сохраним модуль в папке с файлами проекта, присвоив ему, например, имя UnFuncProc.

В этом модуле операторы вычисления $Y(x)$ в виде подпрограммы-функции F и операторы вычисления $S(x)$ в виде подпрограммы-процедуры Sum оформим по правилам создания модулей.

Для подключения модуля UnFuncProc к проекту необходимо сделать активным окно с текстом модуля UnModul, затем

в меню File выбрать опцию Use Unit... и в открывшемся окне Use Unit указать имя используемого модуля — UnFuncProc. Убедитесь в том, что в разделе Implementation модуля UnModul появился оператор **Uses UnFuncProc;**, который Delphi вставила в текст модуля UnModul.

Откройте главный файл проекта и убедитесь в том, что проект не содержит посторонних модулей и файлов.

1.3. Текст модуля UnFuncProc

```
unit UnFuncProc;

 interface
 var
  n:integer;       // количество слагаемых в сумме S
 Function F(x:extended):extended;
 Procedure Sum(x:extended;Var s:extended);

 Implementation Function F(x:extended):extended;
  begin
   result:=(1-x*x*0.5)*cos(x)-0.5*x*sin(x);
  end;
 Procedure Sum(x:extended;Var s:extended);
  var
   c:extended;
   k:integer;
  begin
   c:=-x*x*0.5;
   S:=1;
  for k:=1 to n do
   begin
    s:=s+c*(2*k*k+1);
    c:=-c*x*x/((2*k+1)*(2*k+2));
   end;
  end;
end.
```

1.4. Текст модуля UnModul

```
Unit UnModul;

interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs,StdCtrls, ExtCtrls, Spin, Buttons;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    SpinEdit1: TSpinEdit;
    BitBtn1: TBitBtn;
    procedure FormCreate(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
Type
 func=function(x:extended):extended; // функциональный тип
 proc=procedure(x:extended;Var s:extended); // процедурный тип
var
  Form1: TForm1;

implementation

uses UnFuncProc;        // Delphi подключает модуль UnFuncProc

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  SpinEdit1.text:='4'; // начальное значение N
  Edit1.text:='0.1';   // начальное значение Xn
  Edit2.text:='2.0';   // начальное значение Xk
  Memo1.Clear;
  Memo1.Lines.Add('Лабораторная работа №8 — Создание модулей');
end;

{В процедуре Tablica вычисляется и выводится}
{таблица значений x, S(x) и Y(x)}
procedure Tablica(Sum:proc;F:func;n:integer;xn,xk,h:extended);
 var
  x,y,s:extended;
  begin
   Form1.Memo1.Lines.Add(#9+'x'+#9+'S'+#9+'Y');
                                        // заголовок таблицы
   x:=xn;
   repeat                               // цикл по x
    Sum(x,s);      // вызов процедуры Sum для вычисления S(x)
    y:=F(x);       // обращение к функции F для вычисления Y(x)
    Form1.Memo1.Lines.Add(#9+FloatToStrF(x,ffFixed,5,2)
                                        // вывод x
                  +#9+FloatToStrF(s,ffFixed,6,3)
                                        // вывод S
                  +#9+FloatToStrF(y,ffFixed,6,3));
                                        // вывод Y
    x:=x+h;
   until x>xk;
  end;
procedure TForm1.BitBtn1Click(Sender: TObject);
  var
   xn,xk,h:extended;
begin
  n:=StrToInt(SpinEdit1.Text); xn:=StrToFloat(Edit1.Text);
  xk:=StrToFloat(Edit2.Text); h:=(xk-xn)*0.1;          // шаг h
  Tablica(Sum,F,n,xn,xk,h);
             // вызов процедуры Tablica для вычисления таблицы
 end;
end.
```

2. Индивидуальные задания

В заданиях необходимо вывести на экран таблицу значений функции $Y(x)$ и ее разложения в ряд $S(x)$ для значений x от x_n до x_k с шагом $h=(x_k - x_n)/10$. Близость значений $S(x)$ и $Y(x)$ во всем диапазоне значений x указывает на правильность вычисления $S(x)$ и $Y(x)$.

№	x_n	x_k	S(x)	n	Y(x)
1	0,1	1	$1+\frac{\ln 3}{1!}x+\frac{\ln^2 3}{2!}x^2+\ldots+\frac{\ln^n 3}{n!}x^n$	8	3^x
2	$\frac{\pi}{5}$	$\frac{9\pi}{5}$	$\cos x+\frac{\cos 2x}{2}+\ldots+\frac{\cos nx}{n}$	18	$-\ln\left\|2\sin\frac{x}{2}\right\|$
3	$\frac{\pi}{5}$	$\frac{4\pi}{5}$	$\sin x-\frac{\sin 2x}{2}+\ldots+(-1)^{n+1}\frac{\sin nx}{n}$	6	$\frac{x}{2}$
4	0,1	0,8	$x\sin\frac{\pi}{4}+x^2\sin 2\frac{\pi}{4}+\cdots+x^n\sin n\frac{\pi}{4}$	12	$\frac{x\sin\frac{\pi}{4}}{1-2x\cos\frac{\pi}{4}+x^2}$
5	0,1	0,8	$x+\frac{x^5}{5}+\ldots+\frac{x^{4n+1}}{4n+1}$	16	$\frac{1}{4}\ln\frac{1+x}{1-x}+\frac{1}{2}arctg$
6	0,1	1	$1+\frac{\cos x}{1!}+\ldots+\frac{\cos nx}{n!}$	14	$e^{\cos x}\cos(\sin x)$

Выберите из таблицы два варианта индивидуальных заданий. Создайте модуль, в котором вычисление значений $S(x)$ реализуйте в виде подпрограммы-процедуры, а вычисление значений $Y(x)$ — в виде подпрограммы-функции. На панели интерфейса установите компонент, с помощью которого реализуйте возможность выбора соответствующего варианта задания и вывод таблицы значений x, $S_i(x)$, $Y_i(x)$, где i — номер варианта. Созданный модуль подключите к проекту и выполните приложение.

Литература

1. В.В.Феофанов. Delphi 3. Учебный курс. — М.: Нолидж, 2001.
2. Э.Возневич. Delphi. Освой самостоятельно. — М.: Восточная книжная компания, 1996.
3. Дж.Матчо, Д.Р.Фолкнер. Delphi. — М.: БИНОМ, 1995.
4. М.Канту. Delphi 2 для Windows 95/NT. — М.: ООО "Малип", 1997.

Pentium глазами программиста

П.СОКОЛЕНКО /HI-TECH,
г.Ростов-на-Дону,
E-mail: pts@icomm.ru
http://hi-tech.nsys.by/
http://www.macro.aaanet.ru

Презентация первого микропроцессора Pentium I прошла в США 22 марта 1993 г., а 15 ноября 2000 г. уже в Москве был представлен Pentium 4. За эти семь с небольшим лет компания Intel выпустила 18 разных моделей микропроцессоров (далее МП) семейства Pentium, относящихся к трем разным поколениям — пятому, шестому и седьмому. Тактовая частота первого Pentium I составляла 66 МГц, а первого Pentium 4 — уже 1,4 ГГц. В настоящее время на компьютерном рынке появились Pentium 4 с частотой около 2 ГГц, а представители компании Intel утверждают, что при используемой в них микроархитектуре и технологии производства достижима частота 3 ГГц. Но если бы реальные достижения компании Intel заключались только в этом, не было бы повода для написания данного эссе.

Тактовая частота является одним, но не единственным из факторов, от которых зависит производительность МП. Кроме того, она не может повышаться беспредельно, поскольку, начиная с некоторых ее значений, в игру вступает время распространения сигнала между элементами МП. На практике впервые с этим фактором столкнулись разработчики МП семейства Alpha, а чуть позже и Intel. Поэтому уже давно ведутся интенсивные поиски альтернативных решений увеличения производительности МП при неизменной тактовой частоте.

Эволюция семейства Pentium сопровождалась воплощением в кремнии именно альтернативных способов повышения производительности микропроцессоров. Наиболее важными из них являются:

- ускорение доступа к оперативной памяти;
- расширение возможностей конвейерной обработки команд;
- включение в систему команд групповых операций.

Ускорение работы с памятью сокращает вынужденные паузы в работе МП при чтении или записи данных, т.е. это косвенный способ повышения производительности. Прямыми способами являются конвейерная обработка инструкций и выполнение вычислений над группами данных, причем последнее позволяет достичь наиболее существенного повышения производительности.

Программиста, как правило, не интересует производительность МП сама по себе, ему важно знать, как она достигается. Иначе говоря, какие средства и приемы он может использовать при программировании для ускорения процесса выполнения создаваемых задач. Об этом и пойдет речь в данной статье.

Оптимизация работы с памятью

Оперативная память является внешним устройством, подключаемым к системной шине, поэтому время доступа к ней регламентируется частотой, на которой работает шина. Современные технологии позволили поднять эту частоту у Pentium 4 до 400 МГц, но она всегда будет меньше тактовой частоты МП. Если вы разделите тактовую частоту на частоту системной шины, то узнаете, сколько тактов занимает прямое обращение к памяти.

Начиная с Intel 486, в состав МП входит сверхбыстрое ассоциативное запоминающее устройство, называемое кеш (cache — буфер). Это посредник между процессором и оперативной памятью, в котором хранятся данные вместе с их адресами в ОЗУ, отсюда название "ассоциативная память". У семейства Pentium кеш имеет двухуровневую структуру, первый уровень (L1) всегда входит в состав МП и, в зависимости от модели, имеет объем 8 или 16 Кб. Второй уровень (L2), в зависимости от модели МП, может располагаться как в одном корпусе с процессором, так и на материнской плате, его объем составляет сотни килобайт. Данные и команды хранятся в двух независимых кешах первого уровня.

В большинстве случаев при выборке команд или операндов МП обращается не к оперативной памяти, а к кешу, исключением является, например, работа с видеопамятью, когда кеширование недопустимо. Если содержимое запрошенного адреса находится в кеше, то для чтения или записи, как правило, нужен один такт работы МП. В противном случае (кеш-промах) придется ждать, пока данные будут скопированы из памяти в кеш. Поэтому ***реальное ускорение*** доступа к данным происходит, только если к моменту запроса они находятся в кеше.

Для того чтобы данные попали в кеш до момента их использования, применяется простейший механизм предварительной загрузки. Он заключается в том, что при каждом кеш-промахе из памяти считывается строка, обычно содержащая 32 байта. Расчет сделан на то, что в программах чаще используются данные, расположенные в памяти подряд, друг за другом (смежные), и поэтому все байты строки будут рано или поздно востребованы. Если это не так, то время, затраченное на загрузку невостребованных байтов в кеш, потрачено впустую.

Перед загрузкой в кеш принудительно очищаются 5 младших разрядов адреса, поэтому адреса младших байтов хранящихся в кеше строк кратны 32. Отсюда следует простое правило, которое надо соблюдать при размещении данных:

Массивы, матрицы и другие структуры надо располагать в разделе данных программы, начиная с адресов, кратных 32, а часто используемые одиночные переменные группировать так, чтобы адрес первой из них был кратен 32. Сказанное относится не только к данным, но и к командам, поэтому циклы, подпрограммы и прочие часто используемые фрагменты тела программы надо располагать в сегменте кодов начиная с адресов, кратных 32.

В течение длительного времени отсутствовала возможность управления работой кеша, и программист мог только располагать данные и команды, руководствуясь приведенным правилом. Правда, существовали две команды, одна из которых очищала кеш без сохранения данных в памяти (INVD), а вторая — с сохранением (WBINVD). Но их можно не принимать в расчет.

Первые команды для активного вмешательства в работу кеша появились у Pentium III, а у Pentium 4 их состав пополнился. Теперь при работе с оперативной памятью и кешем можно выполнять следующие манипуляции:

- загружать в кеш строку данных (PREFETCH), не приостанавливая выполнение вычислительных операций;
- очистить строку кеша и соответствующие байты оперативной памяти (CLFLUSH);
- выполнять запись непосредственно в оперативную память, минуя кеш, с помощью команд пересылки данных (MOVNTQ, MOVNTDQ, MOVDQU);

- инициировать операции обмена данными между кешем и оперативной памятью и ждать их завершения (SFENCE, LFENCE, MFENCE).

Размер строки, с которой работают новые команды, зависит от модели МП, его можно определить с помощью команды идентификации процессора CPUID. Объем информации, возвращаемой данной командой, зависит от модели МП, достаточно сказать, что в технической документации на Pentium 4 ее описание занимает 10 страниц.

Таким образом, первым важным нововведением в последних моделях семейства Pentium является возможность программного управления взаимодействием кеша и оперативной памяти.

Особенности конвейерной обработки

При описании способов оптимизации программ для МП семейства Pentium в технической документации много говорится о необходимости учета особенностей конвейерной обработки. Но во многих случаях проверка приводимых примеров выявляет несостоятельность рассуждений авторов. Создается впечатление, что они имели в виду некий абстрактный конвейер, а не его конкретную реализацию. Давайте попробуем разобраться в этом вопросе.

В семействе Intel конвейер (pipeline) впервые появился у МП 486, у Pentium I конвейер состоял уже из двух труб (pipe), это называлось суперскалярной архитектурой. Дальнейшее развитие шло в направлении все большего дробления конвейера на составные части или ступени (у Pentium 4 их 20) и упрощения функций (микроопераций), выполняемых этими частями.

Для достижения наибольшей производительности разработчики дважды радикально изменяли микроархитектуру процессоров Intel. Первый раз это произошло при создании первого МП 6-го поколения Pentium Pro, когда Intel впервые отказалась от способа исполнения команд (CISC), принятого в семействе x86, и перешла на динамический способ (RISC) их исполнения. Второй раз это произошло при разработке микропроцессора 7-го поколения Pentium 4, когда на смену динамическому исполнению команд пришла микроархитектура NetBurst.

Введение конвейерной обработки преследует две основные цели:

- совмещение выполнения нескольких операций в одном такте;

- вовлечение в активную работу как можно большего числа элементов МП.

Но на пути к их достижению есть серьезное препятствие, не позволяющее использовать все преимущества конвейерной обработки. Составляя любую программу, программист исходит из предположения, что образующие ее команды выполняются последовательно друг за другом, а процессор пытается совместить выполнение нескольких команд, а иногда и изменять порядок их выполнения.

Независимо от структуры конвейера, совместному выполнению нескольких команд задачи, состоящей только из инструкций, выполняемых МП Pentium I, препятствуют различные факторы. Вот некоторые из них:

- особенности реализованного в задаче алгоритма, например, последующие команды используют результаты вычислений предыдущих команд;

- особенности выполнения команд, например, один из сомножителей должен находиться в регистре-аккумуляторе, поэтому невозможно начинать вторую операцию умножения, пока не закончится первая;

- использование общего ресурса, например, стека ячеек оперативной памяти или числовых регистров FPU (процессора для работы с вещественными числами);

- условные ветвления, до их выполнения невозможно точно предсказать, какая команда будет выполняться следующей.

По этим и другим причинам при выполнении обычных программ возможно лишь частичное совмещение выполнения нескольких команд. По оценкам аналитиков, исследовавших разные МП, относящиеся к разным семействам (Intel, Alpha, Athlon и пр.), в среднем удается совместно выполнять немного более полутора (1,5) команд без каких-либо специальных изменений в выполняемых задачах. Самое интересное, что эта цифра практически не зависит от структуры конвейера.

Автор исследовал, как отразилось усложнение структуры конвейера на выполнении простейшего фрагмента программы, формирующего в регистре EAX сумму четырех целых чисел. Для этого вычислялось количество тактов, затрачиваемых МП на выполнение четырех команд, приведенных в примере 1.

Напоминание: в состав всех МП Pentium входит счетчик тактов (Stamp Counter), и существует команда RDTSC, считывающая его содержимое в регистры EDX:EAX, но она выполняется **только** в защищенном режиме. Опросив счетчик до и после, можно узнать, сколько тактов занимает выполнение интересующей вас группы команд.

Числа A, B, C и D хранились в 32-разрядных переменных с именами arg_a, arg_b, arg_c и arg_d, расположенных в оперативной памяти и выровненных так, чтобы они находились в одной строке кеша. Для того чтобы исключить из результатов измерений время, необходимое для чтения чисел из памяти, строка предварительно загружалась в кеш.

Пример 1. Сложение четырех целых чисел

```
mov     eax,arg_a       ; eax = arg_a, запись числа A в регистр eax
add     eax,arg_b       ; eax = eax + arg_b, сложение A + B
add     eax,arg_c       ; eax = eax + arg_c, сложение A + B + C
add     eax,arg_d       ; eax = eax + arg_d, сложение A + B + C + D
```

Результаты измерений оказались такими: Pentium I выполнял команды за 9 тактов, Pentium MMX — за 13 тактов, Pentium III — за 11 тактов. Как видите, усовершенствованный конвейер Pentium III проигрывает простому конвейеру Pentium I два такта. Только не надо делать далеко идущих выводов, результаты относятся к данному примеру и не более того.

Разработчикам МП, не хуже чем аналитикам и пользователям, известны недостатки системы команд x86, ограничивающие возможности конвейерной обработки. Поэтому они создали дополнительную систему команд, о которой речь пойдет ниже. Впрочем, некоторые изменения были внесены и в систему команд общего назначения, так теперь называется старая система команд x86.

Как уже говорилось, работу конвейера могут нарушать команды условных ветвлений. В этом случае процессор может начать предварительную обработку не той команды, которая будет выполняться после ветвления. В состав конвейера входит специальный блок, который пытается прогнозировать следующую выполняемую команду, но прогноз есть прогноз. Поэтому в большинстве инструкций по оптимизации рекомендуется свести к минимуму количество условных ветвлений в программе.

Начиная с МП Pentium Pro, в систему команд общего назначения включены две группы команд, выполняющих условную пересылку целых (CMOVcc) и вещественных (FCMOVcc) чисел. Они копируют содержимое операнда

источника в приемник только при выполнении условия, указанного в имени команды. Если эти команды использовать, например, для выбора одного из двух чисел, то из текста программы исключается команда условного ветвления.

Предположим, что в регистрах EAX и EBX находятся два числа A и B, меньшее из них надо записать в EAX. В примере 2 показано, как это можно сделать с помощью условного ветвления и условной пересылки.

Пример 2. Варианты определения меньшего из двух чисел

```
; 1. использование условного ветвления
cmp   eax,ebx   ; eax – ebx, сравнение A и B
jbe   L1        ; если меньше или равно, то исключаем пересылку
mov   eax,ebx   ; eax = ebx, выполняем пересылку
L1:             ; следующая команда
; 2. использование условной пересылки
cmp   eax,ebx   ; eax – ebx, сравнение A и B
cmova eax,ebx   ; если больше, то eax = ebx
```

Обратите внимание на то, что при выполнении условия в первом варианте исключается пересылка, а во втором, наоборот, выполняется. Поэтому в командах JBE и CMOVA указаны противоположные условия ("не больше" и "больше").

В начале борьбы с условными ветвлениями в инструкциях по оптимизации программ для Pentium I, который не выполнял команду условной пересылки, приводились примеры нахождения меньшего из двух чисел без использования условного ветвления. Позже они перекочевали в инструкции по оптимизации программ для Pentium 4. Один из вариантов, предложенный Агнером Фогом, приведен в примере 3. Предполагается, что числа находятся в регистрах EAX и EBX, а меньшее из них окажется в регистре EAX. В комментариях к командам этого примера буквой C обозначен C-разряд регистра флагов, он устанавливается в единицу, если из меньшего числа вычитается большее.

Пример 3. Вариант нахождения меньшего из двух чисел

```
sub   ebx, eax  ; ebx = ebx - eax (если ebx <eax то C=1)
sbb   ecx, ecx  ; если C=0 то ecx = 0, иначе ecx= -1 (код 0FFFFFFFFh)
and   ecx, ebx  ; ecx = ecx & ebx (в ecx окажется 0 или содержимое ebx)
add   eax, ecx  ; eax = eax + 0 или eax = eax + ebx - eax = ebx
```

Аналогичные трюки полезны в тех случаях, когда **невозможно** выполнение команд условных ветвлений по результатам выполнения операций, например, при использовании новых команд, не вырабатывающих признаки, характеризующие результат операции.

В остальных случаях включать подобные примеры в программы не имеет смысла даже для Pentium I, поскольку он содержит лишнюю команду по сравнению с первым вариантом примера 2 и выполняется на несколько тактов дольше. Пример 3 иллюстрирует то, с чего мы начали данный раздел. Формально исключение условного ветвления улучшает работу конвейера, но фактически замедляется выполнение задачи. Так что важнее?

Еще одно замечание к вопросу об оптимизации программ. Процессоры Pentium намного совершеннее своих предшественников, поэтому трезво анализируйте и критически оценивайте рекомендации по замене одних команд другими — многие советы морально устарели. Особенно это относится к замене умножения сложениями или выполнению операций с вещественными числами с помощью команд общего назначения и другим подобным трюкам. Лучше подумайте об использовании новых команд, которые предоставляют вам новые возможности для оптимальной реализации алгоритмов.

Подведем итог сказанному. Задачи, состоящие только из команд общего назначения, не могут использовать все преимущества конвейерной обработки. Поэтому при их программировании основное внимание надо уделять общеизвестным способам оптимизации алгоритмов и программ и рациональному использованию возможностей кеша. А всяческие изменения, направленные на улучшение спариваемости инструкций, можно отложить до лучших времен.

(Окончание следует)

Работа с буфером обмена в Delphi

И.ШИРКО,
E-mail: FDC@tut.by

Буфер обмена позволяет приложениям обмениваться данными: графикой, текстом, файлами и т.д. Вообще говоря, буфер обмена представляет собой область памяти, которая обрабатывается при помощи функций Win API. В Delphi есть модуль ClipBrd, который позволяет использовать буфер обмена в своих программах, но для начала нужно подключить модуль к проекту в разделе **Uses**.

В буфере обмена используются разные форматы. Вот некоторые из них:

- CF_TEXT — текст;
- CF_BITMAP — точечный рисунок;
- CF_METAFILEPICT — рисунок в формате MetaFile (*.wmf);
- CF_PICTURE — объект типа Tpicture.

Рассмотрим обработку этих форматов.

CF_TEXT

Для обработки текстовых данных можно использовать свойство AsText класса TClibBoard. Оно содержит текст, который в данный момент находится в буфере обмена. При помощи этого свойства можно также поместить строку в буфер обмена.

Пример:

```
//если в буфере обмена содержится текст, то показываем его
//в диалоговом окне
  if clipboard.HasFormat(CF_TEXT) then
 showmessage(clipboard.astext)
//иначе помещаем в буфер обмена строку "qwerty"
  else clipboard.astext:='qwerty';
```

Получить текст из буфера можно и таким способом:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyHandle: THandle;
  TextPtr: PChar;
  MyString: string;
begin
//открываем буфер обмена
  ClipBoard.Open;
try
//получаем дескриптор данных
  MyHandle := Clipboard.GetAsHandle(CF_TEXT);
//получаем данные
  TextPtr := GlobalLock(MyHandle);
//преобразуем в строку
  MyString := StrPas(TextPtr);
//разблокирование дескриптора
  GlobalUnlock(MyHandle);
finally
```

```
//закрытие буфера обмена
  Clipboard.Close;
end;
  showmessage(MyString);
end;
```

CF_BITMAP

Разместите на форме компоненты Image1:Timage и Button1:Tbutton. Для обработки события OnClick кнопки запишите процедуру, которая поместит точечный рисунок в Image1:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if clipboard.HasFormat(CF_BITMAP )then
    Image1.Picture.Bitmap.Assign(clipboard);
end;
```

Поместите на форму еще одну кнопку, при нажатии на которую в буфер обмена скопируется точечный рисунок компонента Image1:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  clipboard.Assign(image1.picture.bitmap);
end;
```

CF_METAFILEPICT

Здесь все почти так же, как и с точечным рисунком, только объект Bitmap нужно заменить объектом Metafile:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if clipboard.HasFormat(CF_METAFILEPICT) then
    image1.Picture.MetaFile.Assign(clipboard);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  clipboard.Assign(image1.picture.Metafile);
end;
```

CF_PICTURE

Этот формат данных объединяет другие графические форматы. Т.е. его можно использовать и тогда, когда рисунок в буфере обмена представлен в формате CF_BITMAP, и тогда, когда рисунок представлен в формате CF_METAFILE.

Пример:

```
//получение изображения из буфера обмена
 procedure TForm1.Button1Click(Sender: TObject);
begin
  if clipboard.HasFormat(CF_PICTURE) then
    image1.Picture.Assign(clipboard);
end;
//копирование изображения в буфер обмена
procedure TForm1.Button2Click(Sender: TObject);
begin
  clipboard.Assign(image1.picture);
end;
```

Следует заметить, что для объекта Ticon не существует специального формата, поэтому, перед копированием в буфер обмена, иконку нужно преобразовать в точечный рисунок при помощи следующего алгоритма:

```
var
  ico:TIcon;
  bit:TBitmap;
begin
 met:=TIcon.Create;                    {создаем объект TIcon}
 bit:=TBitmap.Create;                  {создаем объект TBitmap}
 ico.LoadFromFile('icon.ico');         {загружаем файл icon.ico}
 bit.Width:=ico.Width;
 bit.Height:=ico.Height;       {приравниваем размеры картинки}
                                                 {и Tbitmap}
 bit.Canvas.Draw(0, 0, ico); {прорисовываем иконку в Tbitmap}
 bit.SaveToFile("Icon.bmp");           {сохраняем полученное}
                                       {изображение в  файл}
end;
```

В.ЗУБКО /HI-TECH,
E-mail: vzubko@chat.ru
WWW: http://hi-tech.nsys.by

Защита файлов в Excel

Как создать защищенный файл в Excel

Предположим, вы работаете с файлом (таблицами) в Excel и при этом хотите, чтобы кроме вас (или доверенных лиц) никто не мог бы иметь доступа к нему. Для этого вы можете защитить файл паролем (Файл — Сохранить как — Параметры — Ввод пароля). После всех подтверждений файл будет сохранен. При последующих попытках открыть его, Excel будет предлагать ввести пароль, и если тот окажется неверен, то вы (или кто-то другой) содержимое файла увидеть не сможете.

Примечание: речь идет о версии Excel 97.

Что происходит с файлом при сохранении его с паролем?

Создадим новый файл. В первое поле таблицы занесем отладочную строку, например 50 букв "А". Сохраним файл с паролем. Пусть он будет тоже характерным, например, "123456789012345" (максимальная длина, которую позволяет ввести программа — 15 символов). Создадим также файл с той же строкой из букв "А", но защищать его паролем не будем.

Теперь попытаемся сравнить оба файла. Длины защищенного и незащищенного файлов не отличаются. Попытаемся найти отличия в их содержимом. Проще сверять файлы не визуально, а с помощью небольшой программы, которая считывает в два буфера содержимое файлов, а потом побайтно сравнивает их, выводя несовпадающие байты и их номер в файле в протокол.

Оказывается, что файлы Excel имеют что-то вроде 512-байтного заголовка, который абсолютно одинаков для защищенного и незащищенного файлов. Далее, начиная с байта в позиции 0214h, в файле начинаются отличия:

```
0214h: защищенный: 86h, незащищенный: E1h
0216h: защищенный: 00h, незащищенный: 02h
0218h: защищенный: 2Fh, незащищенный: B0h
...
```

и так далее. Попробуем найти нашу тестовую строку в файле. Оказывается, ее там нет. Как нет и пароля в явном виде. Что ж, на такой легкий вариант никто и не рассчитывал.

Один из простейших способов зашифровать данные — это выполнить операцию хог (исключающее ИЛИ) над байтами шифруемой информации. В самом простом случае на каждый байт накладывается одна и та же хог-маска. Для расшифровки достаточно повторить ту же самую операцию хог с той же маской — поскольку эта операция при выполнении четное число раз приводит к начальному результату:

```
(ЛюбойБайт xor Mask) xor Mask = ЛюбойБайт
```

Убедимся, что Excel не так прост. Для этого достаточно пройти все 256 байтов масок, накладывая их на все байты файла и создавая новый файл.

Неудачным поиском тестовой строки по таким новым файлам легко проверить, что программа все еще хитрее нас.

Итак, мы имеем множество отличающихся байтов в защищенном и незащищенном файлах. Точнее, их более 60%. Очевидно, что сам пароль сейчас как "иголка в стоге сена", поэтому нам не хватает дополнительных эксперименталь-

ных данных, для того чтобы хотя бы знать, где примерно находится то, во что Excel превратил исходный пароль.

Напрашивается следующий опыт — сохранить множество одинаковых файлов с одним и тем же паролем и содержимым (скажем, все с той же тестовой строкой). Создадим, к примеру, 10 таких файлов. Сравнивать их опять же удобно по вышеприведенному принципу, немного усложнив алгоритм — сначала читаем в некий буфер один из файлов, а потом сравниваем второй файл с первым, несовпадающие байты фиксируем, для остальных файлов просто выводим байты в несовпадающих позициях. В протокол выведем строки типа:

```
Номер_байта Байт1 Байт2 ... Байт10,
```

где `Байт1 Байт2 ... Байт10` — байты 10 файлов, причем `Байт1` всегда отличен от `Байт2`.

Что же обнаруживается в результате такого сравнения? Оказывается, что очень значительная доля одинаковых с точки зрения пароля и содержащейся в них информации файлов, зашифрованных одной и той же копией Excel, оказалась различной (около 30%)!

Выявленное многообразие шифрования только на первый взгляд кажется осложняющим дальнейший анализ. В самом деле, вот у нас есть блоки байтов, которые сильно отличаются от файла к файлу, а вот есть наборы байтов, которые остаются неизменными, причем их расположение и длина от файла к файлу не меняются. Тут может быть два варианта — либо измененный "пароль" принадлежит неизменной части (частям), либо переменной. Остальные варианты пока можно отбросить из-за их сложной реализации с точки зрения программиста.

Преположим, что измененный "пароль" принадлежит к постоянной части (частям) файла. Но это находится в противоречии с тем фактом, что значительная часть файла, а следовательно и информация, по крайней мере частично, изменяется. Ведь если "пароль" постоянен, то неоткуда взять дополнительный источник, вносящий изменения — дата или время создания файлов на эту роль не годятся, ибо при копировании защищенных файлов они могут быть изменены.

Сделав такое допущение, будем искать характерные изменяющиеся блоки в протоколе. Вероятно, что длина их должна быть порядка длины пароля, как вероятно и то, что расположение их находится в "любимых" программистами местах файла — либо в начале файла, либо в его конце.

Остановимся пока на первом варианте, ибо мы уже обнаружили, что структура файла сделана по принципу заголовок (информация о файле) — информативное содержание, а не наоборот.

Раз сам заголовок не изменяется, перейдем к несовпадающим байтам после него. Сразу можно заметить, что с позиции 0222h начинается постоянная по длине (48 байтов) изменяющаяся группа байтов. Далее можно обнаружить и другие такие группы. Например, для пароля длиной в один символ на месте данных файла (тестовой строки) находятся видоизменяющиеся группы длиной 14 байтов, перемежаемые постоянными группами длиной 8 байтов (найти в зашифрованном файле позицию тестовой строки легко — достаточно узнать ее позицию в незашифрованном файле).

Как же можно "вручную" выбрать из таких групп ту, которая содержит в искаженном виде пароль? Предположим, что программа сначала считывает такую группу, запрашивает пароль, сверяет его по внутреннему алгоритму с этой группой, и только в случае успешной проверки продолжает распаковывать файл. Для примера изменим любой один байт из первой группы в 48 байтов и попробувм открыть его в Excel.

Оказывается, в этом случае Excel стабильно выдает сообщение — неверный пароль! А вот если мы изменим другие переменные байты, скажем, в позиции 025Ch, он выдаст сообщение типа "Не могу прочитать файл", или вообще появится традиционное "Программа выполнила ...".

На этом месте явно наступило время обдумать полученные "в лоб" результаты и обогатиться новыми знаниями. Ведь нам совершенно не известно, как программа обрабатывает эти 48 байтов вместе с паролем.

Криптография и все-все-все

Пусть мы пока не знаем, как поступает программа с этими 48 байтами, но можно предположить, что существует некоторая однозначная функция проверки пароля и этого блока:

```
F = F(пароль, 48 байтов с позиции 0222h).
```

Например, CRC (контрольная сумма) 48 байтов — скажем, просто их побайтное сложение по два, должно образовывать последовательность из 16 байтов. Эти байты, например, представляют собой сам пароль. Или не сам пароль, а пароль, на который наложена 16-байтная постоянная хог-маска, взятая из последних 16 байтов.

В случае использования алгоритмов, подобных этому, сразу возникает вопрос — а что, если кто-то узнает (скажем, украдет или подсмотрит исходный код Excel, или просмотрит его под отладчиком) принцип действия алгоритма? Тогда ***любой*** пароль элементарно вычисляем — достаточно проделать вышеописанные операции над этими 48 байтами, повторив действия программы, чтобы получить пароль для любого файла. Иначе говоря, существует функция G, обратная к функции F:

```
Пароль = G(48 байтов с позиции 0222h).
```

Очевидно, что в этом случае хакерами становились бы уже в детском саду. Да и парни из команд, подобных MS, знают, с кем имеют дело. В чем же тут хитрость?

Обогатимся, так сказать, знаниями, которые накопило к настоящему времени человечество в области выстраивания заборов и капканов. Скорее всего, "против нас играют" не программеры, а какие-нибудь ученые-математики с их заумными формулами и доказательствами.

Наберем в поисковике чего-нибудь со словами crypto. Крипто — это криптография, так по науке называются способы защиты данных. Искать лучше на русскоязычных сайтах — лишних трудностей нам не нужно. Вот, скажем, попался нам www.cryptography.ru — сайт МГУ по этой самой криптографии.

Начинаем читать статью "Введение в криптографию" под редакцией В.В.Ященко. Оказывается, многие современные алгоритмы шифрования (да, скорее всего, и наш) строятся вот по какому принципу — если используется некий алгоритм создания и проверки некоторой измененной информации (пароля), то не должно существовать однозначной функции, позволяющей получить сам пароль по этой зашифрованной информации. То есть все, что можно — это брать пароль и сравнивать его каким-нибудь способом с зашифрованными данными (функция F), а вот из самой зашифрованной информации однозначно пароль получить нельзя (обратная функция G). Причем отсутствие этой самой обратной функции строго не доказано (!), но сейчас все же полагают, что способы выбора функций F достаточно надежны. Проверяют надежность того или иного способа многократными попытками взлома (!).

Хорошо, поверим в то, что не существует способа сразу вычислить пароль. Но ведь никто нам не мешает узнать алгоритм проверки программой пароля, т.е. узнать функцию F. Узнав эту функцию, мы можем методом перебора поискать пароль, "подкидывая" все множество возможных паролей на вход функции F — как только сравнение окажется верным, мы нашли пароль. Такой алгоритм прост до неприличия, но в чем же хитрость защиты — ведь перебором однозначно можно найти нужный пароль? Смысл в том, что время, необходимое на перебор всех-всех вариантов, огромно, хотя и конечно.

Пусть процедура проверки одного пароля занимает 1 мкс. Пусть также мы ищем пароль длиной 10 символов. Число возможных символов, которые можно ввести с клавиатуры, будет примерно 100. Тогда число всех возможных вариантов пароля составит:

$$100^{10} = 10^{20} \text{ вариантов.}$$

А время, необходимое на перебор всех вариантов, будет равно:

$$10^{20} \times 10^{-6} = 10^{14} \text{ секунд!!!}$$

Увы, мы живем явно меньше. Тем не менее, интересно все же узнать, как именно Excel применяет такие алгоритмы шифрования.

Алгоритмы шифрования, используемые в Excel

Итак, перед нами явно какой-то из этих алгоритмов или их комбинация. В той же статье В.В.Ященко читаем, что Word и другие MS-продукты используют некие алгоритмы RC4 и MD5. Использовать они их начали не сразу, раньше можно было, например, импортировать данные из Excel-файла в Access из любого защищенного файла, пароль при этом почему-то никто не спрашивал.

Что же такое RC4, MD5 и тому подобные штучки? Схематично их алгоритмы можно представить таким образом — шифруемая информация (последовательность битов) должна быть, с одной стороны, скрыта во множестве "мусора" (тривиальный пример — сложение каждого байта строки с байтами строки-константы), а с другой стороны — рассеяна по полученному множеству таким образом, чтобы связи между шифруемыми битами были потеряны, но не окончательно — ведь еще предстоит проверка (например, меняем четные байты пароля на нечетные).

Возьмем, к примеру, алгоритм шифрования RC4. Он состоит из подготовительной части и самого шифрования.

Пусть у нас есть пароль "Password". Формируем две таблицы длиной 256 байтов. Первая (S-таблица) будет вначале содержать числа 0, 1, 2, ..., 255, а вторую (K-таблицу) заполним паролем примерно так:

K = "PasswordPassword..."

(пароль ведь может быть меньше 256 байтов). После этого введем индексы i и j, j вначале равен 0. Выполним 256 замен между таблицами:

```
i = 0,1.. 255, j = (j + S[i] + K[i]) and 255,  xchg S[i],S[j].
```

Полученная таблица S называется таблицей подстановок.

Теперь пусть есть некоторая информация, которую надо зашифровать — массив байтов Info[0,1,2...]. Определим два счетчика Q1 и Q2 с начальными значениями 0. При шифрации каждого байта Info[] выполняем следующие действия:

```
Q1 = (Q1 + 1) and 255, Q2 = (Q2 + S[Q2]) and 255,
xchg S[Q1], S[Q2], T = (S[Q1] + S[Q2]) and 255,
Gamma = S[T],
и,  наконец – вот оно:
Info[..] = Info[..] xor Gamma.
```

Таким образом, мы формируем таблицу подстановок S и с ее помощью проводим операции "исключающее ИЛИ" (xor) над элементами шифруемой информации. Как уже было сказано ранее, четное число операций xor над байтом дает исходный байт. Поэтому достаточно провести еще раз ту же самую операцию над зашифрованным алгоритмом RC4 массивом Info[], чтобы получить исходный Info[].

С алгоритмом MD5 все гораздо сложнее. Но случайно мне под руку подвернулся его исходник, правда, на Си, но читать можно. MD5 — это по сути не алгоритм шифрования, а так называемая ***хеш-функция***. Слово громкое, но это всего лишь сверхуникальная последовательность из 16 байтов для любой последовательности байтов любой длины. Грубо говоря, это некая особая выжимка из массива чисел. Работает даже тогда, когда массив пуст. Алгоритм поражает своей тяжестью — именно не сложностью! В нем, так же как и в RC4, есть секция инициализации, которая состоит в заполнении 4 двойных слов (32 бита) — пусть это будет массив state[1..4] — следующими числами:

```
67452301      ; state[1]:=MagValue1
EFCDAB89      ; state[2]:=MagValue2
98BADCFE      ; state[3]:=MagValue3
10325476      ; state[4]:=MagValue4,
```

а также в инициализации двух 32-битных счетчиков нулевыми значениями. Разобраться в MD5 гораздо сложнее, чем в RC4 — да это нам и не нужно. Нам пока важно иметь общее представление о механизмах, которые мы будем искать в Excel. А найдя их, мы найдем уже готовый код проверки пароля от MS.

"Микроскоп" SoftIce и Excel в роли "инфузории-туфельки"

Итак, мы имеем смутное представление о том, что же мы хотим найти в Excel. Посмотрим в отладчике SoftIce на то, как программа работает с файлом и введенным паролем.

Запускаем SoftIce. Выбираем в браузере зашифрованный файл. Нажимаем Ctrl-D. Сейчас файл будет открыт, и из него будет прочитано несколько блоков файла и, видимо, в том числе наш блок из 48 байтов. Чтение из файла осуществляется через API-функции SetFilePointer (Установить указатель в файле) и ReadFile (Прочитать блок байтов из файла). Находятся эти функции в модуле kernel32.dll. Формат вызова у этих функций примерно такой:

1. `SetFilePointer( FileHandle: DWORD; Pos: DWORD; Rezerv: DWORD; FromWhat: DWORD ) : DWORD;`

Где:

- `FileHandle` — дескриптор файла;
- `Pos` — указатель в файле;
- `Rezerv` — (резерв?) передают NULL;
- `FromWhat` — с начала файла (`FILE_BEGIN = 0;`), с конца файла (`FILE_CURRENT = 1;`), с текущей позиции (`FILE_END = 2;`).

2. `ReadFile( FileHandle: DWORD; AddrBuffer: DWORD; BuffSize: DWORD; AddrForBytesRead: DWORD; Rezerv: DWORD; ): DWORD;`

Где:

- `FileHandle` — дескриптор файла;
- `AddrBuffer` — адрес буфера чтения;
- `BuffSize` — сколько байтов читать;
- `AddrForBytesRead` — сколько байтов будет прочитано.

Устанавливаем breakpoint'ы на эти две функции:

```
bpx SetFilePointer
bpx ReadFile
```

и продолжаем выполнение программы (Ctrl-D). После клика на нашем файле в браузере мы почти сразу же "выпадаем" в отладчик на первый breakpoint по `SetFilePointer`. Ага, смотрим переданные параметры и видим, что произошел следующий вызов:

```
SetFilePointer( Handle: ...; Pos: 0; Rez: 62E2D4;
FromWhat: FILE_BEGIN).
```

Т.е. мы собираемся читать что-то с начала файла. Пропускаем этот и еще несколько вызовов, пока не обнаруживаем вот это:

```
015F:7FF66B88 call ReadFile
( ReadFile( Handle: ...; AddrBuffer: 64107Ah; BuffSize:
10h; AddrForBytesRead: 62B8F0h; Rezerv: 0;) ).
```

Т.е. читаем 16 байтов в буфер 64107Ah. Выполняем эту функцию:

```
G @SS:ESP,
```

и видим, что прочли как раз первые 16 байтов с позиции 0222h. Далее мы видим еще два интересующих нас вызова:

```
015F:7FF66B88 call ReadFile
( ReadFile( Handle: …; AddrBuffer: 62BA54h; BuffSize: 10h;
AddrForBytesRead: 62B8F0h; Rezerv: 0;) )
```

и

```
015F:7FF66B88 call ReadFile
( ReadFile( Handle: …; AddrBuffer: 62BA44h; BuffSize: 10h;
AddrForBytesRead: 62B8F0h; Rezerv: 0;) ).
```

А это, собственно, и есть чтение второго и третьего блока, по 16 байтов каждый, из тех самых 48 байтов.

Продолжая дальше выполнять программу, мы видим, как на экране появляется приглашение к вводу пароля. Введем характерный пароль, чтобы потом можно было легко искать его в памяти. После подтверждения ввода повторно "вываливаемся" в отладчик на тех же самых чтениях 48 байтов в той же самой последовательности.

Итак, мы получили 3 вызова с одной и той же точки (:7FF66B88) на чтение 16 байтов 48-байтного блока. Это явно подпрограмма, одна и та же подпрограмма. Поэтому нужно найти код, который делает что-то вроде этого:

```
; Прочесть первые 16 байтов
; Что-то сделать
; Прочесть вторые 16 байтов
; Что-то сделать
; Прочесть третьи 16 байтов
; Что-то сделать
; Видимо, сравнить пароль
```

Последовательно возвращаясь из вложенных процедур с точки ":7FF66B88", находим именно такой код (`buff2`, `buff3` и `buff4` — 1-й, 2-й и 3-й блоки по 16 байтов):

```
308B7793:
; ...
      lea edx,[edi+116h]    ; <- 0064107A - buff2
; Прочитать buff2
308B77A3:
      call 3080AF24         ; Прочесть buff2
; ...
      call [esi+10h]        ; Выполнить код 308B75D5
308B77B5:
; ...
      call [esi+08]         ; Вызвать 308B74FD
308B77BF:
; ...
; Прочитать buff3
      lea edx,[ebp-14h]     ; <-0062BA54 - buff3 !!!
      call 3080AF24         ; Прочесть там buff3
      test eax,eax          ; Ошибка ?
      jz 308B778A           ; Нет
; ...
308B77DA:
      call esi              ; Вызвать 308B7548
; ...
; Прочитать buff4
      lea edx,[ebp-24h]     ; <-0062BA44 - buff4 !!!
      call 3080AF24         ; Прочесть там buff4
; ...
      call esi              ; Вызвать 308B7548
; ...
      call 309AD4D2         ; MD5Init( MD5_CTX - контекст
                            ; [0062B9DC] )
; ...
      call 309AD4FC         ; MD5Update( MD5_CTX - контекст
                            ; MD5, буф., дл.буф. )
      lea ecx,[ebp-8Ch]     ; <-0062B9DC - адрес контекста
      call 309AD5A9         ; MD5Final( контекст: 0062B9DC,
                            ; digest: +58h)
; Сравнить вычисленные данные.
308B781B:
      mov ecx,ebx           ; <-10h
      lea edi,[ebp-24h]     ; <- buff4, уже отксореный ранее по RC4
      lea esi,[ebp-34h]     ; <- 0062BA34 - digest к 0062B9DC
                            ; из 10h байтов
      xor eax,eax
      rep cmpsb             ; Сравнить блок из 16 байтов
                            ; в 222h файла.
;     jnz ...               ; Пароль не верен?
```

Вот примерно и ясен каркас подпрограммы проверки пароля.

Остается пройтись по подпрограммам, которые вызываются из этого участка кода (начиная с чтения первых 16 байтов и до последней проверки сравнением двух строк длиной по 16 символов).

Оказывается, что алгоритм проверки довольно-таки запутанный. Можно сказать, что это дикая смесь RC4 и MD5. Схематично он выглядит так:

- прочесть первые 16 байтов из 48;
- получить MD5-хеш от пароля;
- от этого хеш'а и первых 16 байтов образовывать новый MD5-хеш, фактически от ("MD5(пароль)+16 байтов" x 16) ;
- еще раз (!) от полученного хеш'а и пароля образовать новый MD5-хеш;
- полученный хеш использовоать как пароль для формирования таблицы подстановок RC4;
- прочесть вторые 16 байтов из 48;
- выполнить RC4-расшифровку этих 16 байтов, таблица подстановок уже получена;
- прочесть третьи 16 байтов из 48;
- выполнить RC4-расшифровку и этих 16 байтов;
- от уже расшифрованных RC4 двух блоков по 16 байтов получить MD5-хеш;
- и только сейчас сравнить его с расшифрованным по RC4 третьим блоком из 16 байтов!!!

Где-то внутри этого алгоритма также происходит формирование буфера, необходимого для расшифровки самого файла — например, той же таблицы подстановок для RC4.

Время выполнения проверки одного пароля на P-120, Windows98 составляет в среднем около 800 мкс. Т.е для проверки всех вариантов паролей длиной 2 символа необходимо:

~100 x 100 x 0,0008 = 8 секунд!

От редакции. На homepage журнала http://radio-mir.com выложены:

- password.asm — дизассемблированный код Excel, проверяющий пароль и подбирающий к файлу test.xls пароль длиной в 2 символа.
- md5.h, mdfile.c — алгоритм хеш-функции MD5 на Си.

Взаимодействие ADO Data Control и DataGrid ActiveX Control

Р.КОНОНЕНКО,
Днепропетровская обл.
E-mail: hostadmin@slavgorod.dp.ua

Начнем с того, что я расскажу, чем занимаюсь — пишу под заказ бухгалтерские программы для малого частного бизнеса. Почему для малого? Потому что не веду план счетов в программе, от этого программа становится легче и дешевле — хотя для меня большой разницы нет.

По роду работы я очень часто использую DataGrid и всегда — базы данных Access. Почему именно Access? Да потому что их легче всего конструировать. А так как данных в таблицах очень много, и они за разный период, то все это на экране я отображаю при помощи DataGrid. Поэтому изучил я его очень хорошо и хочу поделиться своим опытом.

Итак, начнем. На форме помещаем DataGrid (далее DG) и Adodc (далее ADO). ADO делаем невидимым или помещаем за пределы формы, в окне свойств кликаем на строке *Пользовательский*. Открывается диалог настройки ADO. На вкладке *General* выбираем *Use Connection String* и давим на кнопочку *Build*. В открывшемся диалоге на вкладке *Поставщик данных* выбираем Microsoft Jet 3.51 OLE DB Provider (для Access 2000 и XP — версия Jet 4.0). Потом кнопка *Далее*, и в поле для ввода базы данных пишем имя нашей базы. Если не указывать полный путь, то программа будет открывать базу из своей папки, что дает некоторые преимущества, в частности — нет привязки к пути. Для успокоения совести кликаем на кнопке *Проверить подключение*. Жмем *Ок* и возвращаемся к первому диалогу. Если в программе требуется вход по паролю, то вводим эти данные на вкладке *Authentication* и переходим на вкладку *RecordSource*.

И вот здесь нужно решить, как будет использоваться DG. Если нам необходимо отображать всю таблицу в том виде, в каком она заполняется, то в списке *Command Type* выбираем *2-adCmdTable* и ниже в списке — имя таблицы. А если нам нужно динамически менять условия отбора, то в списке *Command Type* выбираем *1-adCmdText* и в поле *Command Text (SQL)* пишем *"SELECT * FROM Имя_Таблицы"*. На этом настройка ADO закончена.

Теперь переходим к настройке DG. В свойствах *DataSourse* выбираем имя нашего ADO. Потом кликаем правой кнопкой на DG и выбираем меню *Retrieve Fields*. После этого DG обращается к ADO и загружает список полей. Затем в этом же меню выбираем *Properties*. В открывшемся диалоге будем настраивать DG.

Вкладка *General*, основные настройки:

- *Caption* — отвечает за содержимое текста над названиями колонок;
- *ColumnHeaders* — отвечает за то, будет ли слева отображаться колонка для выбора строк;
- *HeadLines* — количество строк, отведенных под текст названия колонок;
- *RowHeigth* — высота строк в твипах.

Все остальное и так понятно.

Вкладка *Keyboard* отвечает за то, как DG будет реагировать на нажатие кнопок при навигации по DG. Я все опции отключаю.

Вкладка *Columns* отвечает за порядок размещения и названия колонок в DG.

Вкладка *Layout* (здесь задаются параметры для каждой колонки):

- *Locked* — что-то фиксирует. Не помню что, но я флажок ставлю;
- *AllowSizing* — разрешает изменение ширины колонки юзером. Я отключаю;
- *Visible* — отмечаем, если нужно отобразить колонку;
- *WrapText* — разрешает функцию переноса текста в колонке;
- *Button* — по-моему, если в данной колонке в таблице находится список, то можно использовать этот список и в DG;
- *DividerStyle* — определяет тип сетки;
- *Alignment* — задает выравнивание текста;
- *Width* — значение ширины колонки.

Вкладки *Color* и *Font* — здесь все должно быть понятно.

Вкладка *Split* — это настройка области. DG позволяет иметь несколько областей и настраивать каждую в отдельности. Здесь убираем все флажки, кроме *Locked*, единственное что настраиваем, так это скроллбар.

Вкладка *Format* — здесь настраивается формат отображения данных в колонках.

Тут, вроде бы, все и так понятно, но есть маленькая хитрость. К примеру, нужно отображать цифры с двумя знаками после запятой и прижать их к правому краю. Но крайняя правая цифра располагается так близко к сетке, что почти сливается с ней. Выход — выбираем колонку, где эти самые цифры отображаются. Для этого в *Format Type* выбираем *Custom*, и в поле *Format String* пишем 0.00 ("ноль-точка-ноль-ноль" — и все без кавычек). Количество нулей после точки определяет разрядность дробной части. После нулей ставим пробел. Вот этот пробел при отображении и вставляется после цифры, что отодвигает число в ячейке чуть левее и не дает ему сливаться с сеткой.

Теперь пара нюансов для тех, кому нужно отображение строк в DG по определенному условию, и кто в ADO прописал *1-adCmdText* а в поле *Command Text (SQL)* — *"SELECT * FROM Имя_Таблицы"*. В ходе процедуры для изменения содержимого делаем так:

```
...
Adodc.RecordSource = "Здесь текст команды SQL на выборку данных"
Adodc.Refresh
DataGrid.Refresh
...
```

После этих строк DG отображает то, что вы указали в запросе Adodc.

От редакции. На web-странице журнала выложен пример использования описанного в статье комбинированного способа доступа к БД на Visual Basic через DAO и ADO (файл DBdemo.zip).

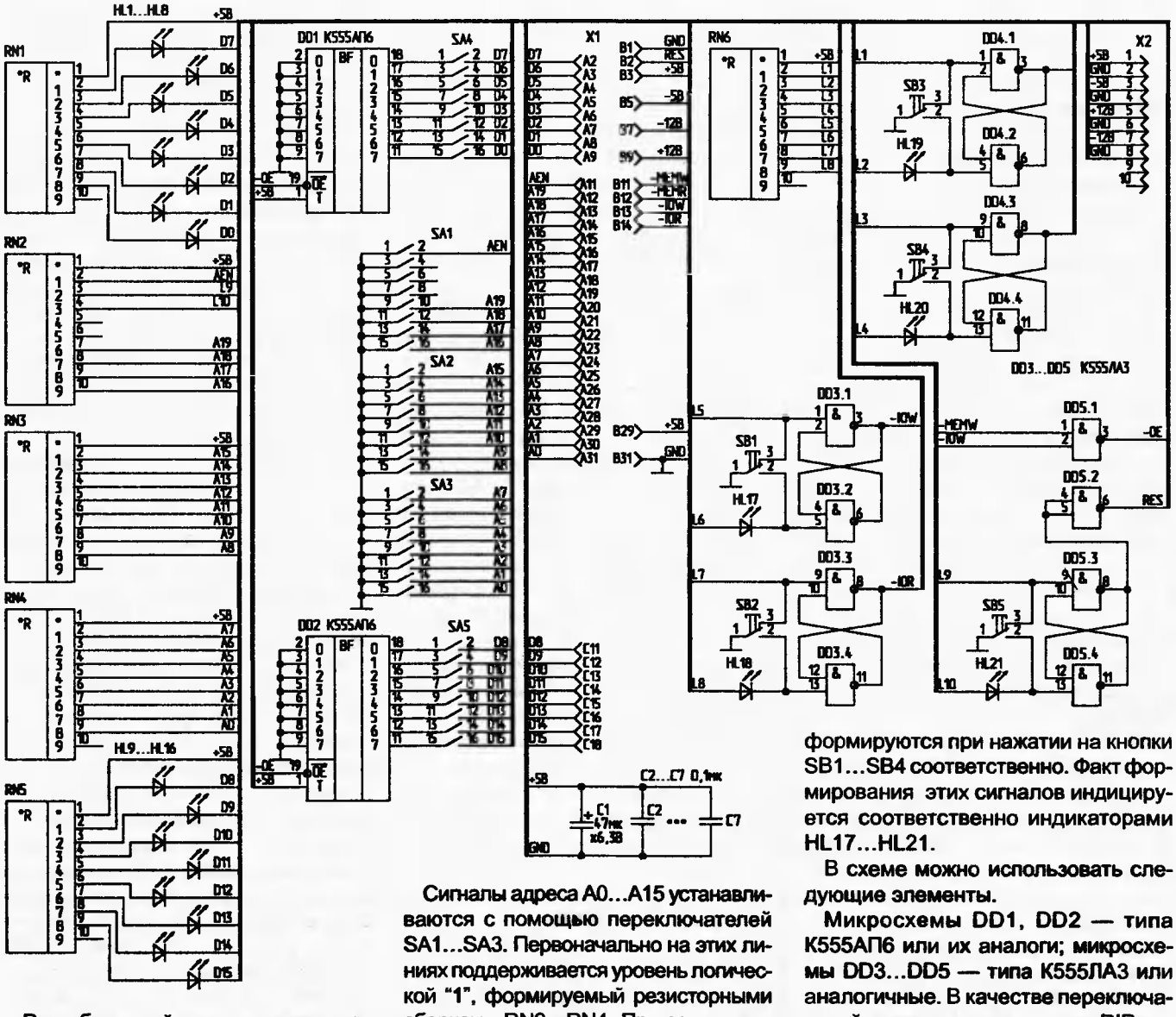
Эмулятор интерфейса ISA

О.ВАЛЬПА,
E-mail: sandh@narod.ru

В настоящее время компьютеры IBM PC стали самыми массовыми. Многие адаптеры для этих компьютеров подключаются через интерфейс (шину) ISA. Поэтому при ремонте таких адаптеров или отладке адаптеров собственной разработки необходимо устройство, позволяющее эмулировать интерфейс ISA. Именно такое устройство я и предлагаю.

Разработанный мною эмулятор интерфейса ISA состоит из распространенных недорогих деталей и достаточно прост в изготовлении. Данный эмулятор позволяет в ручном режиме формировать статические сигналы обращения к 16-разрядным устройствам памяти или к устройствам ввода-вывода по любому адресу от 0 до 1 Мб. Интерфейс ISA подробно описан в технической литературе, например в [1], а в последнее время и в сети Интернет.

Электрическая принципиальная схема эмулятора приведена на рисунке. Принцип его работы основан на формировании сигналов адреса, данных для записи, сброса, записи или чтения памяти (устройств ввода-вывода).



Сигналы адреса A0...A15 устанавливаются с помощью переключателей SA1...SA3. Первоначально на этих линиях поддерживается уровень логической "1", формируемый резисторными сборками RN2...RN4. При замыкании соответствующего контакта в группе переключателей, на соответствующей адресной линии устанавливается уровень логического "0". В разомкнутом положении на данной линии будет присутствовать уровень логической "1". Аналогично формируется и сигнал AEN.

Сигналы D0...D15 (данные для записи) выставляются с помощью переключателей SA4 и SA5. Индикаторы HL1...HL16 отображают выставленные для записи и считываемые из адаптера данные. Светящийся индикатор означает, что данный разряд находится в состоянии логического "0", не светящийся — логической "1".

Сигнал сброса "RES" формируется при нажатии на кнопку SB5 и выставляется на шину активным уровнем логической "1". Сигналы записи и чтения памяти (-MEMW", -MEMR) и устройств ввода-вывода (-IOW, -IOR) формируются при нажатии на кнопки SB1...SB4 соответственно. Факт формирования этих сигналов индицируется соответственно индикаторами HL17...HL21.

В схеме можно использовать следующие элементы.

Микросхемы DD1, DD2 — типа К555АП6 или их аналоги; микросхемы DD3...DD5 — типа К555ЛА3 или аналогичные. В качестве переключателей желательно применить DIP-переключатели импортного производства в силу их высокой надежности. Кнопки SB1...SB5 — типа КМ1-1. В качестве индикаторов можно использовать любые светодиоды, например, типа АЛС307АМ(БМ), или их импортные аналоги с током свечения не бо-

лее 5 мА. Резисторные сборки RN1...RN6 — типа HP1-4-9M или импортные аналоги с сопротивлением 1 кОм. Конденсатор C1 — типа K50-35 или любой электролитический соответствующего номинала. Конденсаторы C2...C7 могут быть типа KM-5 или любые керамические. Разъем X1 — это розетка SLOT-98 стандартного интерфейса ISA. Разъем питания X2 — типа BH-10 или любой другой с необходимым количеством контактов. Можно также применить вместо X2 одиночные гнезда или клеммы.

Монтаж схемы эмулятора может быть выполнен на макетной плате проводными соединениями. При желании можно изготовить и печатную плату, разводка которой не составляет труда.

Я рекомендую после изготовления эмулятора нанести на органы управления и контроля соответствующие надписи для удобства работы. Надписи можно, например, распечатать с помощью принтера на липкой бумаге и покрыть ее скотчем для защиты от влаги и грязи.

Эмулятор не нуждается в отладке, и в случае отсутствия ошибок в монтаже начинает работать сразу же после включения. Для проверки его работы достаточно убедиться в формировании всех сигналов на разъеме X1 при переключении соответствующих переключателей и кнопок.

При проверке адаптера с шиной ISA поступайте следующим образом. Отключите питание от эмулятора. Вставьте в разъем X1 испытываемый адаптер. Установите адрес необходимой области памяти или регистра ввода-вывода адаптера и включите питание.

Теперь при нажатии кнопки SB5 будет сформирован сигнал сброса для адаптера, при нажатии кнопки SB2 или SB4 — сигнал чтения регистра ввода-вывода или памяти соответственно. На индикаторах HL1...HL16 будет видно прочитанное слово в двоичном формате. Аналогично можно записать в адаптер произвольные данные, выставляемые переключателями SA4 и SA5. Таким образом, можно проследить прохождение сигналов от разъема эмулятора к микросхемам адаптера, читать и записывать в нужные области данные и найти причину неисправности отлаживаемого устройства.

Удачи вам всем в творчестве.

Литература

1. Гук М. Аппаратные средства IBM PC. Энциклопедия. — С-Пб.: Питер Ком, 1998.

Комфортная работа с Windows

А.ЗАЙЦЕВ,
602265, Владимирская обл.,
г.Муром, а/я 427,
E-mail: alexey_375@mail.ru

Я хочу поделиться с читателями опытом работы с операционными системами Windows 9x, дать полезные (как мне кажется) советы по оптимальной настройке и поддержанию работоспособности системы. Я намеренно не касаюсь вопросов инсталляции ОС и принципов работы с ней — об этом написано немало книг, статей, есть и справочная система Windows. Моя статья адресована читателям, в общих чертах знакомым с устройством компьютера, основными понятиями, способным самостоятельно найти "Панель управления" и клавишу AnyKey. Более опытные пользователи, возможно, тоже узнают что-нибудь интересное.

Также я приведу краткий обзор программ (служебных и общего применения), которые могут оказаться полезными в работе. Возможно, у этих программ есть более удобные аналоги, но мне о них неизвестно. Все перечисленные в статье программы тестировались и показали неплохую скорость работы даже на компьютере с AMD 5x86 133 МГц и 16 Мб ОЗУ.

Допустим, имеется компьютер со свежеустановленной операционной системой (или с давно установленной, но ведь все равно можно попытаться что-то улучшить). Еще нужно желание поэкспериментировать, здравый смысл и творческий подход к делу. Я думаю, у читателей журнала с этим все в порядке.

Итак, операционная система установлена. Далее необходимо проверить драйверы устройств Plug and Play, при необходимости добавить недостающие или заменить неправильно определившиеся. Это делается на вкладке "Панель управления\Устройства".

После этого я рекомендую произвести настройку изображения в соответствии с возможностями видеокарты и монитора — установить оптимальное разрешение, глубину цвета и повысить частоту развертки. Дело в том, что по умолчанию Windows устанавливает частоту развертки 60 Гц, а этого мало даже для довольно старого "железа", и тем более недостаточно для длительной комфортной работы с компьютером. Для доступа к настройкам изображения легче всего щелкнуть правой кнопкой мыши в любом месте экрана и выбрать "Свойства".

Если тип монитора или видеокарты при установке системы определялся неправильно, настройки частоты могут быть недоступны. В этом случае можно воспользоваться программой **HzTool** (www.hem.passagen.se/doxx) или более сложной, но более удобной при работе с современными видеокартами — **PowerStrip** (www.entechtaiwan.com/ps.htm). Если есть возможность, тип развертки лучше выбрать построчный (Non Interlacted).

Если вы планируете устанавливать программы с компакт-диска, имеет смысл отключить его автозапуск. Для этого нужно на вкладке "Панель управления\Система\Устройства" выделить "Устройство для чтения компакт-дисков" и нажать "Свойства", а затем снять галочку напротив соответствующей функции.

Если в компьютере установлено мало памяти (много ее практически никогда не бывает) или запускается сразу несколько задач, операционная система записывает часть данных в так называемый файл подкачки, расположенный на жестком диске. Разумеется, это сильно замедляет работу компьютера, особенно когда файл подкачки находится в фрагментированной области в конце дискового пространства. А это произойдет неизбежно, так как Windows по умолчанию использует динамический файл подкачки — отводит место по мере потребности (для сравнения — в ОС Linux для файла подкачки предусмотрен отдельный раздел, форматированный особым образом). Для повышения быстродействия нужно установить постоянный файл подкачки, для этого на вкладке "Панель управления\Система\Быстродействие\Виртуальная память" в строках "Минимум" и "Максимум" указать одинаковые значения — порядка 100...200 Мб. Благодаря такой настройке, сразу после установки ОС и дефрагментации диска, файл подкачки будет создан в еще не фрагментированной области сразу за системными файлами.

Еще лучших результатов можно достичь, если переместить наиболее часто используемые файлы в начало диска. Перемещение файла в начало диска уменьшает время доступа к нему, что при частом обращении способно значительно ускорить работу. Утилита **Speed Disk** из комплекта Norton Utilites (www.symantec.com) позволяет не только производить дефрагментацию файлов, но и оптимизирует их размещение. С ее помощью папки и отдельные файлы, в том числе и файл подкачки, можно переместить к началу или концу дискового пространства.

Важным элементом системы Windows является браузер. Даже если вы не собираетесь выходить в Интернет или же являетесь поклонником Netscape Navigator или Opera, вам наверняка придется установить хотя бы IE4 (лучше сразу ставить IE5). Дело в том, что при установке браузера в систему устанавливается множество библиотек (.DLL) и компонентов (.COM), без которых некоторые программы просто откажутся работать или потеряют часть настроек. При работе в Интернете нужно ставить самую свежую версию браузера — это уже вопрос безопасности (хотя специалисты по этому вопросу сильно "не советуют" использовать ОС и браузер от одного разработчика).

Копировать и открывать файлы можно, конечно, путем перетаскивания мышью, но гораздо удобнее делать это с помощью программы-оболочки. Помоему, лучшей на данный момент является **Windows Commander (WC)**. С ее помощью работать с архивами почти так же удобно, как с обычными папками. Настраиваемая панель инструментов позволяет создавать кнопки для быстрого вызова приложений (а в версии 4.54 я неожиданно обнаружил возможность вызова контекстного меню — все, что заботливо "навешивается" на правую кнопку мыши, стало доступным). Кроме того, встроенная система просмотра (Lister) умеет просматривать тексты в различной кодировке и большого объема (нет "блокнотного" ограничения в 60 Кб), графические файлы и HTML. Возможности поиска файлов и папок в WC гораздо шире, чем у проводника Windows, возможен поиск в архивах. Раздобыть Windows Commander в виде 30-дневной версии можно на сайте разработчика (www.ghisler.com) или на русскоязычном www.wincmd.ru, полностью посвященном этой программе.

Итак, операционная система настроена, но для поддержания ее в рабочем состоянии и устранения поломок необходимо время от времени сохранять копию системного реестра и критически важные файлы. Для этого удобно использовать программу **Reanimator** (www.annatar.narod.ru/download/software/3rdparty/rw.zip). Она обладает понятным интерфейсом и может быть настроена на работу в автоматическом режиме. В случае невозможности загрузки Windows достаточно запустить Reanimator из командной строки MS-DOS и выбрать одну из ранее сохраненных копий системных файлов.

Для тонкой настройки ОС может пригодиться утилита **WinSEr** (www.multiworks.narod.ru). С ее помощью можно, например, спрятать настройки в меню "Пуск", запретить вывод заставок при загрузке и завершении работы системы или убрать "стрелки" у ярлыков. Более глубокие настройки доступны из программы **Xteq X-setup** (www.xteq.com/products/xset), но для ее использования нужно хорошо разбираться в функционировании системы, и она "говорит" на английском языке.

Для просмотра и конвертирования различных графических файлов удобно использовать популярную бесплатную утилиту **IrfanView** (www.irfanview.com). Она позволяет просматривать группы файлов в автоматическом режиме (слайд-шоу), а при помощи дополнительных плагинов — слушать музыку, смотреть flash-анимацию и avi-файлы.

На старых машинах (486, младшие Pentium) для прослушивания музыки в формате MP3 не хватает быстродействия. В этом случае может выручить не очень удобная, но неприхотливая программа **Fmod Media Player** (www.fmod.org/ifmoddownload.html). С ее помощью наверняка удается найти компромисс между системными требованиями и качеством звука.

Программа **XZView** (www.coderu.com) является аналогом менеджера задач в Windows NT. Она позволяет просмотреть список текущих процессов и удалить приложение, а также показывает и может удалить ветви реестра, отвечающие за запуск программ при загрузке ОС (удалять можно все, кроме индикатора раскладки клавиатуры internet.exe). Более подробную информацию о запущенных программах и их компонентах можно получить с помощью утилиты **PrcView** (www.teamcti.com).

Скопировать текстовую информацию из всплывающих меню, окон сообщений и других труднодоступных мест можно с помощью программы **StringTheif** (www.cooler.vov.ru). Если же она не поможет, часть экрана можно скопировать в буфер обмена или сразу в файл с помощью **ScreenHunter** (www.wisdom-soft.com).

Более широкими возможностями обладает программа **SnagIt** (www.techsmith.com) — она может записывать происходящее на экране монитора в текстовый, графический, AVI-файл или даже посылать по электронной почте. С ее помощью можно полностью скопировать как рисунок содержимое многостраничного PDF-файла или защищенной от копирования Web-страницы (хотя механизм управления, на мой взгляд, не совсем отлажен).

Для обратного преобразования графического файла в текст необходима программа распознавания текста (OCR). Бесспорным лидером в этой области является **FineReader** компании ABBYY Software (www.abbyy.ru). Она способна почти безошибочно читать текст низкого разрешения (скриншоты) или плохого качества (страницы мелкого текста). Для этого лучше не соглашаться на смену разрешения на стандартное, и первые сотню-две символов пройти в режиме обучения. Недостаток этой программы — не очень удобная работа с таблицами и проблемы с их корректным сохранением, тут лучше использовать **CuneiForm** (www.cognitive.ru, www.cuneiform.ru). Эта программа имеет более простой интерфейс и умеет сохранять результаты в виде форматированного пробелами текста ("табличный текст"). Но она больше любит крупные шрифты. Обе программы можно раздобыть в виде демоверсии на 30 запусков или 30 часов работы.

Для обработки текстов есть очень удобный текстовый редактор **UltraEdit** (www.idmcomp.com, www.ultraedit.com). В нем встроена поддержка HTML и некоторых языков программирования, Hex-редактор и множество поддерживаемых кодировок. Кроме того, есть возможность про-

изводить поиск и формировать список строк, содержащих заданную фразу, не только в открытом документе, но и в группе файлов. Программа распространяется в виде 45-дневной демо-версии, дистрибутив умещается на дискете.

Большая часть документации и описания к программам чаще всего пишутся на английском языке. Поэтому если вы владеете им недостаточно хорошо, без переводчика не обойтись. На мой взгляд, лучшим является **"Сократ Персональный"** (www.ars.ru). Он умеет встраиваться в help-файлы Windows (в панели инструментов появляется кнопка "Перевести"), автоматически определять язык и переводить содержимое буфера обмена. Распространяется в виде демо-версии (через 21 день пропадает возможность сохранить результат перевода), дистрибутив "весит" всего 6 Мб.

Для чтения электронных книг в текстовом, DOC- и HTML-форматах отлично подходит программа **BookView** (www.bookview.i-connect.com). Она позволяет настраивать шрифты, цвет текста и фона, имеет режим автопрокрутки, умеет читать из архива и запоминать место, на котором чтение было прервано.

Желающим немного отвлечься от работы могу порекомендовать установить программу **DxBall2** (www.LongbowDigitalArts.com). Это игра-арканоид, с очень красивой графикой и звуковыми эффектами.

Безопасное удаление программ и различных файлов можно производить с помощью программы **McAfee Uninstaller** (www.mcafee.com). Она позволяет удалять временные файлы (рекомендуются к удалению .TMP, .BAK, .GID, .OLD) и лишние ключи реестра, отслеживать связи между программами и их компонентами. Кроме того, с ее помощью можно архивировать программы, перемещать их в другую папку или даже на другой компьютер (создается что-то вроде дистрибутива, включающего файлы программы, ветви реестра и инсталлятор). Неплохой деинсталлятор есть и в пакете Norton Utilites.

Когда все необходимые программы установлены и настроены, имеет смысл заархивировать текущую установку системы. Тогда в случае необходимости полную работоспособность системы можно будет восстановить за 15...20 минут. Если у вас жесткий диск большого объема (что сейчас далеко не редкость), а ОС и наиболее важные программы установлены на небольшом логическом диске C:, его можно заархивировать целиком с помощью программы **Norton Ghost** фирмы Symantec (www.symantec.com). Тогда в случае серьезных неполадок достаточно запустить восстановление из архива, и система будет гарантированно восстановлена. Следует только помнить, что архив должен быть размещен в разделе, отформатированном в системе FAT16, иначе из-под DOS он будет не виден.

Часто установка программы может быть запущена только с дистрибутивного компакт-диска, но не из дистрибутива, записанного на винчестер. Во многих случаях проблему можно обойти, используя подстановку. Для этого нужно запустить "Сеанс MS-DOS" и в командной строке набрать:

SUBST F: C:\DISTRIB.

После этого в папке "Мой компьютер" появится диск F:, содержимое которого будет полностью повторять содержимое папки DISTRIB, и запущенная с него программа установки, скорее всего, не заметит подвоха. После успешной инсталляции программы необходимо в том же сеансе MS-DOS набрать:

SUBST F: /D

для отмены подстановки.

В ОС Windows нет возможности распечатать список файлов, содержащихся в отдельной папке, для этого используются специальные программы. Однако то же самое можно сделать штатными средствами DOS. Допустим, необходимо сформировать список файлов, содержащихся в папке Windows. Для этого нужно запустить "Сеанс MS-DOS" и в командной строке набрать:

DIR C:\WINDOWS>C:\TEMP\LIST.TXT.

В результате этого в папке TEMP будет создан файл LIST.TXT, содержащий список файлов папки Windows. Если после команды DIR поставить ключ /s, этот файл будет также содержать списки файлов, содержащихся в каждой вложенной папке.

Операционные системы Windows 9x могут функционировать без файла AUTOEXEC.BAT, но даже при его отсутствии имеет смысл его создать и записать в него две строчки:

SET TEMP=C:\TEMP

SET TMP=C:\TEMP,

а также создать в корневой директории диска C: папку TEMP для временных файлов. Без этого временные файлы, созданные различными программами, окажутся разбросанными по всему диску.

Для автоматической очистки "Корзины" при каждой перезагрузке ОС в файл AUTOEXEC.BAT необходимо вписать строчку

@DELTREE C:\RECYCLED/Y

Удаленная этой командой папка Recycled при необходимости будет вновь создана операционной системой.

Иногда в пакетном файле бывает необходимо эмулировать нажатие клавиши Enter для подтверждения какого-либо действия. Это можно сделать с помощью текстового файла. Допустим, необходимо при каждой загрузке ОС записывать в файл дату и время. Для этого вполне подходят команды DATE и TIME, но при их выполнении ожидается ввод новой даты (времени) или просто нажатие Enter. Нужно открыть "Блокнот", нажать Enter и затем сохранить полученный файл (2 байта) в корневой директории под каким-нибудь именем, например, ENT.TXT. Затем в файл AUTOEXEC.BAT добавляются строки

TYPE ENT.TXT | DATE > C:\LOG.TXT

TYPE ENT.TXT | TIME > C:\LOG.TXT.

При следующей перезагрузке в корневой директории будет создан файл LOG.TXT, содержащий дату и время, и впоследствии он будет дополняться. Кроме полезной информации, этот файл содержит лишние строки: "Текущая дата", "Введите новую дату" и т.п., но для созданного средствами DOS файла это простительно.

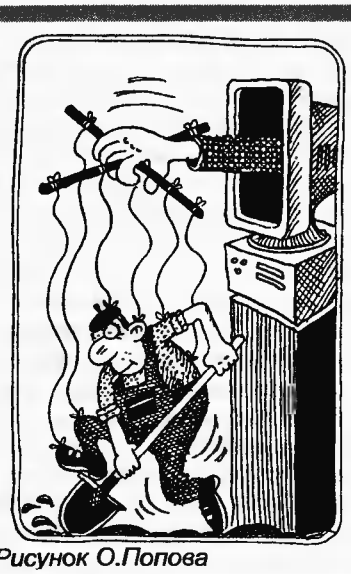


Рисунок О.Попова

Полезные советы для пользователей Windows 9x/Me

А.ГОРЯЧКИН,
г.Кыштым, Челябинской области.
E-mail: anatoliy_goryach@mail.ru

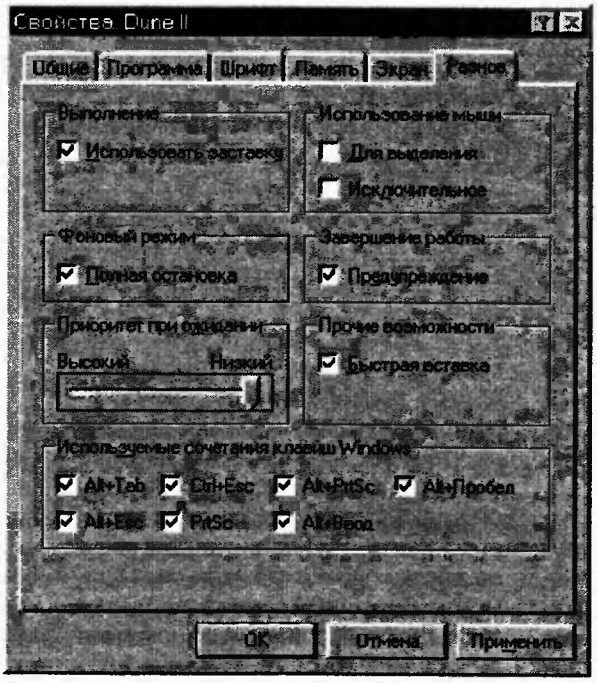
В этой статье мне хотелось бы дать ряд полезных советов пользователям Windows 9x/Me.

Совет первый — для тех, кто до сих пор продолжает пользоваться DOS-приложениями (разнообразными старыми программами и играми). Не стоит удивляться, но именно они — наши старые добрые DOS'овские игрушки, запускаемые из-под Windows 9x/Me, в большинстве случаев используют центральный процессор "на всю катушку" — практически на все 100% (!), безжалостно разогревая его до "белого каления". Как ни прискорбно, но это, к сожалению, факт. В справедливости данного утверждения можно убедиться с помощью штатной программы Windows 9x/Me "Системный монитор", проконтролировав показатель "Использование процессора".

Из-за столь негативного воздействия на CPU, применение каких-либо DOS-программ, запускаемых из-под Windows 9x/Me, вряд ли будет целесообразно. Поэтому постарайтесь отказаться от их применения, если не хотите укоротить жизнь своему процессору. Тем более, что сейчас на рынке программного обеспечения можно без особого труда найти Windows-версии практически всех DOS-программ.

Ну а для тех, кто упорно не желает в силу своих привычек отказываться от запуска DOS-приложений из-под Windows — следующие рекомендации.

Старайтесь как можно реже использовать программы MS-DOS. Загружайте программы на короткий период — не более одного часа. За это время процессор не успеет сильно разогреться. Для запуска программ MS-DOS пользуйтесь ярлыками (PIF-файлами) — не ленитесь их создавать. Эти PIF-файлы (PIF — Program Information File) нужны для оптимизации параметров запуска программ MS-DOS. Установленные пользователем параметры будут использоваться при каждом запуске программы с помощью созданного PIF-файла. При запуске программы непосредственно из окна сеанса MS-DOS или с помощью файлового менеджера типа Norton Commander эти параметры использованы не будут.



Допустим, вы уже создали ярлык к программе MS-DOS (PIF-файл). Становимся на его значок курсором и щелкаем правой кнопкой мыши, вызывая тем самым контекстное меню. Далее выбираем из контекстного меню команду "Свойства". Откроется окно "Свойства" с шестью вкладками. На вкладке "Разное" (**см. рисунок**) в опции "Фоновый режим" установите флажок в строке "Полная остановка". Установка этого флажка блокирует возможность использования программой каких бы то ни было системных ресурсов, если программа неактивна. А в опции "Приоритет при ожидании" ползунком установите приоритет "Низкий". Эта оптимизация параметров запуска программы MS-DOS позволит до минимума снизить использование процессора при переходе в фоновый режим. Но следует учитывать, что не все программы MS-DOS при простое или при переходе в фоновый режим отдают управление операционной системе Windows 9x/Me.

Совет второй. Если вам не привычно, как в русифицированной версии Windows Millennium Проводник представляет тип PIF-файлов — ярлыков к DOS-программам: "Работа в текстовом режиме (в командной строке)", то откройте следующий раздел системного реестра:
`HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES\piffile`.

В данном разделе должен находиться строковый параметр "По умолчанию". Измените значение этого параметра, например, на "Ярлык к программе MS-DOS". Закройте редактор реестра и перезагрузите компьютер.

Совет третий. Тот, кто "переходил" с Windows 9x на Windows Millennium, наверное, обратил внимание на появившуюся папку _RESTORE в корневой директории. В этой папке программа восстановления системы System Restore в автоматическом режиме, без участия пользователя, ежедневно сохраняет системные "снимки", для того чтобы пользователь мог произвести восстановление системы, если будет нарушена ее работа (произвести корректный откат к тому рабочему состоянию, когда система была полностью работоспособна). Но, как показывает практика, полное восстановление системы произвести обычно не удается, или в этом нет особой необходимости. Более того, папка _RESTORE может занимать приличное место на винчестере — от 200 Мб до 13% объема дискового пространства, и к тому же, System Restore притормаживает производительность системы, контролируя все наши действия по удалению

файлов. Для того чтобы отключить программу восстановления системы, откройте диалоговое окно свойств системы, щелкнув значок на Панели управления, и выберите вкладку "Быстродействие". Нажмите кнопку "Файловая система", щелкните вкладку "Устранение неполадок", затем установите флажок "Настройка восстановления системы".

Совет четвертый. Периодически перед дефрагментацией для освобождения дискового пространства делайте очистку жесткого диска от ненужных временных и резервных файлов. Применяйте для этого не только штатную утилиту Windows — CLEANMGR.EXE, но и удаляйте все файлы с расширениями *.tmp, *.—, *.bak, *.old, *.prv, *.gid, *.dir вручную, используя для удобства средство поиска файлов и папок. Также без всякого вреда можно удалять все файлы с нулевой длиной. Рекомендуется удаление файлов производить с помощью Проводника, т.е предварительно помещая файлы в Корзину. В этом случае нужные системе файлы вам удалить все равно не удастся.

Совет пятый. Перед удалением старой операционной системы и установкой новой полезно сохранить свою полученную и отправленную почту. Находится она в следующих файлах: C:\WINDOWS\ Application Data\Identities\{F2873893-7275-487E-A9EA-41A833EA9341}\ Корпорация Майкрософт\Outlook Express\Отправленные.dbx и там же — Входящие.dbx. Если в Outlook Express вами были созданы дополнительные папки, например, "Полученные" или "Переписка" — сохраните файлы Полученные.dbx и Переписка.dbx. Затем, после установки новой операционной системы, в Outlook Express создайте такие же дополнительные папки, какие были раньше. После выхода из программы скопируйте свои сохраненные почтовые файлы формата DBX вместо тех, что созданы в новой системе Windows.

Совет шестой (о сохранении адресной книги). Адресную книгу перед переустановкой Windows также можно сохранять, чтобы каждый раз заново не создавать нужные вам контакты. Находится она в C:\WINDOWS\Application Data\ Microsoft\Address Book в виде файла file_name.wab. Сохраните этот файл где-нибудь в укромном месте, например: C:\Мои документы\Почта\Address Book. После переустановки системы скопируйте сохраненный file_name.wab поверх вновь созданного.

Совет седьмой. О подключении к Интернету через прокси-сервер. Если вы хотите затратить меньше времени и соответственно финансовых средств на загрузку файлов из Интернета, то подключайтесь к Интернету через прокси-сервер вашего провайдера. Для этого в свойствах браузера нужно включить соответствующую опцию и указать адрес и порт прокси-сервера, которые надо узнать у вашего провайдера.

Волшебства здесь никакого нет, просто, если вы или какой-либо другой компьютерщик, пользующийся услугами того же провайдера что и вы, посещали какую-либо страницу в Интернете или скачивали для себя информацию, то копии всего контента сохраняются в кеше на жестких дисках прокси-сервера. Поэтому прокси-сервер является как бы промежуточным звеном между вашим web-браузером и WWW-сервером. Фактически вы будете получать всю запрашиваемую информацию не с удаленного web-сервера, а с прокси-сервера вашего провайдера, но только в том случае, если эта информация там уже есть.

Совет восьмой. При выполнении дефрагментации жесткого диска, требующей значительного времени, возможен запуск экранной заставки Windows. Это приводит к остановке и повторному запуску процесса дефрагментации. В Windows 9x/Me можно применить автоматическую блокировку запуска заставки на время работы утилиты дефрагментации диска (Defrag.exe). Это будет означать, что во время дефрагментации диска запуск заставки будет невозможен, а после окончания дефрагментации произойдет разблокировка. Для этого выполните следующее:

- запустите редактор реестра (regedit.exe);
- откройте раздел системного реестра HKEY_CURRENT_USER\ Software\ Microsoft\Windows\ CurrentVersion\ Applets\Defrag\ Settings;
- в разделе "Settings" создайте подраздел "DisableScreenSaver";
- установите для параметра "По умолчанию" в данном разделе значение "YES" (без кавычек);
- закройте редактор реестра и перезапустите компьютер.

***Примечание*:** запуск заставки Windows не будет блокироваться в двух случаях:

- если дефрагментация была запущена при помощи программы "Планировщик заданий";
- если дефрагментация выбранного диска была завершена, и программа ожидает ввода пользователя.

Совет девятый. Для ускорения загрузки операционной системы и увеличения объема свободной оперативной памяти можно отменить двухсекундную паузу после появления строки с надписью Starting Windows... и отключить показ на экране монитора логотипа Microsoft Windows. Также можно отключить загрузку ряда ненужных драйверов — DoubleSpace, DriveSpace, DoubleBuffer. Для этого нужно отредактировать системный файл Windows MSDOS.SYS. Находится этот файл в корневом каталоге и имеет текстовую структуру. Но предварительно с этого файла нужно снять атрибуты "системный" и "только для чтения", для того чтобы можно было свободно его редактировать. Суммарный выигрыш по уменьшению времени загрузки может составлять до шести секунд. Ниже приведен для примера фрагмент файла MSDOS.SYS после его редактирования:

```
[Options]
BootDelay=0
Logo=0
DblSpace=0
DrvSpace=0
DoubleBuffer=0
```

И.РОЩИН,
Fido: 2:5020/689.53
ZXNet: 500:95/462.53
E-mail: asder_ffc@softhome.net
WWW: http://www.ivr.da.ru

Повышение степени сжатия музыкальных модулей

Введение

Чтобы программа занимала меньше места на диске, ее часто сжимают каким-либо упаковщиком. Если в программе содержатся музыкальные модули, то, повысив степень их сжатия, можно добиться дополнительного сокращения длины программы. Это особенно необходимо, если длина программы не должна превышать определенную величину — например, при написании intro.

Для повышения степени сжатия модулей можно использовать специальный алгоритм сжатия, учитывающий их структуру. Тут, однако, есть свои недостатки. Во-первых, при разработке такого алгоритма придется подробно разбираться в формате модулей (а у каждого музыкального редактора формат свой). Во-вторых, для распаковки модуля потребуется специальный распаковщик, который, естественно, займет определенное место. В результате может получиться так, что модуль упакуется лучше, но за счет наличия распаковщика размер программы даже возрастет, либо уменьшится, но незначительно, не оправдывая затраченных усилий.

Но есть другой способ — так преобразовать модуль, чтобы он более эффективно сжался обычным упаковщиком, тем же, которым сжимается вся программа. Тогда проблемы, связанные с необходимостью наличия специального распаковщика, отпадут. Также, как мы увидим дальше, не возникнет необходимости подробно разбираться в формате модулей. Об этом способе и пойдет речь.

О музыкальных модулях

Как известно, музыку пишут с помощью специальных программ музыкальных редакторов. Чтобы использовать написанную мелодию отдельно от редактора, ее подвергают специальной обработке — компиляции. Компилятор может быть как встроенным в редактор, так и независимым.

Музыкальный модуль может быть откомпилирован без плеера (тогда там будут только данные о мелодии) или с плеером, т.е. с набором процедур для проигрывания (тогда в модуле сначала идет плеер, а за ним — данные). Длина плеера, как правило, постоянна для данного компилятора, а длина данных зависит от конкретной мелодии.

Для вызова процедур плеера (обычно таких процедур три) используются точки входа, находящиеся по определенному смещению от начала плеера.

Первая процедура — INIT (инициализация), вызывается перед началом проигрывания.

Вторая процедура — PLAY (проигрывание), вызывается каждые 1/50 секунды (обычно синхронно с маскируемыми прерываниями). Время работы этой процедуры составляет от нескольких процентов до нескольких десятков процентов от промежутка между двумя прерываниями (в зависимости от редактора, в котором создан модуль, и от используемого плеера), так что на проигрывание музыки тратится лишь небольшая часть времени процессора.

Третья процедура — STOP (заглушение звука). Она может и не использоваться в конкретной программе.

Основная идея

И в данных о мелодии, и в плеере первоначальные значения некоторых ячеек памяти могут не использоваться при проигрывании (назовем такие ячейки неиспользуемыми). То есть либо к этим ячейкам вообще не происходит обращения, либо до чтения в них что-то записывается, и первоначальное значение пропадает. Таким образом, первоначальные значения этих ячеек можно сделать какими угодно, и это не окажет никакого влияния на процесс проигрывания мелодии.

В данных о мелодии не использоваться при проигрывании могут, например, название мелодии или данные о конце длинного сэмпла (если мелодия такова, что после одной ноты через короткий промежуток времени следует другая, и в результате конец сэмпла никогда не проигрывается). В плеере неиспользуемыми могут быть некоторые элементы частотной таблицы (где указывается значение делителя частоты для каждой ноты) — ведь, как правило, в мелодии используются не все ноты. Также в плеере могут не использоваться некоторые обработчики так называемых "команд" (не путать с командами процессора), таких как "изменить темп", "изменить частоту ноты" — ведь обычно в мелодии присутствуют не все возможные команды. Может не использоваться первоначальное значение определенных ячеек памяти, выделенных для хранения переменных при работе плеера. Может быть и другая неиспользуемая информация.

Так вот, если знать все неиспользуемые ячейки модуля, то их можно обнулить, и тогда модуль сожмется гораздо лучше — ведь обычно такие ячейки образуют последовательности, а последовательности одинаковых байтов хорошо сжимаются.

Еще чуть-чуть подумав, нетрудно догадаться, что лучше не обнулять ячейки, а заполнять их значением, взятым из предыдущей используемой ячейки — тогда сжатие станет еще более эффективным. В самом деле, если раньше был участок длиной n, занятый нулевыми байтами, то теперь будет участок длиной как минимум n+1, занятый одинаковыми байтами, а значит, и сожмется он лучше.

А как узнать адреса неиспользуемых ячеек, ничего не зная о формате модуля? Это можно сделать, эмулируя действия процессора при работе плеера и отслеживая при этом все операции с памятью.

Отмечу, что такой способ определения неиспользуемых ячеек памяти может быть использован не только для "ZX-Spectrum" и не только для обработки музыкальных модулей.

Реализация

Для определения неиспользуемых ячеек заведем в памяти массив с количеством элементов, равным длине обрабатываемого модуля. Сначала обнулим все элементы. Нулевое значение будет обозначать, что к соответствующей ячейке памяти не было обращения. При эму-

ляции, если произошло обращение к какой-либо ячейке памяти (естественно, относящейся к модулю), проверяем — это первое обращение к данной ячейке или нет? Нас будет интересовать только первое обращение — т.е. когда соответствующий элемент массива равен нулю. Если выполняется чтение из ячейки, вместо нуля записываем 1, а если запись — 2.

Если после окончания эмуляции элемент массива равен нулю, значит, к соответствующей ячейке модуля не было обращений; если элемент равен 1 — значит, значение этой ячейки было прочитано; если равен 2 — значит, первоначальное значение не было прочитано, а на его место было записано другое. Понятно, что ячейки, которым соответствуют значения 0 и 2, и будут искомыми.

Теперь более подробно об эмуляции. При проигрывании музыки вначале вызывается процедура INIT, затем — процедура PLAY (определенное количество раз), и, если надо, процедура STOP. Рассмотрим, как производится эмуляция действий процессора при выполнении какой-либо из этих процедур.

Имеются следующие структуры данных — область памяти, в которой располагается стек выполняемой процедуры, и массив, где хранятся значения регистров при ее выполнении (так называемые эмулируемые регистры; их я буду в дальнейшем обозначать, добавляя "EM" к названию регистра — например, EM_HL). Перед выполнением процедуры в вышеописанный массив записываются начальные значения регистров, при этом в EM_SP помещается адрес конца области памяти, выделенной под стек (не забываем — стек растет вниз), а в EM_PC — адрес начала процедуры.

Выполнение происходит по отдельным командам. Рассмотрим выполнение одной команды. Значения регистров на момент ее выполнения известны. Адрес начала команды также известен (он содержится в EM_PC).

Во-первых, определяется длина команды. Далее определяется, прозводит ли команда чтение из памяти или запись в память, и если да, то по каким адресам (понятно, что из ячеек, в которых хранится код команды, также произойдет чтение). Эти сведения помещаются в массив с информацией об используемых ячейках памяти.

Если выполняемая команда не является командой передачи управления, то она копируется в специальный буфер и непосредственно запускается на выполнение. Перед этим содержимое массиаа регистров загружается в регистры процессора, а после выполнения — значения регистров процессора записываются в массив. EM_PC увеличивается на длину выполненной команды и, таким обрезом, указывает теперь на следующую команду.

Выполнение команды передачи управления происходит с помощью специальной подпрограммы (для каждой такой команды подпрограмма своя).

Если текущая выполняемая команда — RET, и при этом значение EM_SP совпадает с адресом конца области памяти, выделенной под стек, то считается, что эта команда обеспечивает выход из выполняемой процедуры. На этом выполнение процедуры завершается.

Ниже приведен дамп программы, откомпилированной с адреса #6000. Если программа должна располагаться с другого адреса, то, конечно, дамп будет другим, но на практике необходимость в расположении программы с другого адреса возникает редко.

```
#6000: 1814 1821 183E 00C0 A517 0080 0080 00C0
#6010: 05C0 9A10 08C0 2A0A 6054 5D13 ED4B 0860
#6020: 0B36 00ED B02A 0E60 CD5F 60D8 2A12 60E5
#6030: 2A10 60CD 5F60 E1D8 2B7C B520 F22A 1460
#6040: CD5F 60C9 2A06 60ED 5B0A 60ED 4B08 601A
#6050: 3D28 042B 7E23 7723 130B 78B1 20F1 C922
#6060: 8060 213A 5C22 4964 2A0C 6022 7364 F321
#6070: 0000 E5CD 7F60 E123 30F8 FBA7 C837 C911
#6080: 0000 2100 0022 7564 2277 64CD 9364 1A32
#6090: 7564 13FE CB28 1EFE ED28 41FE DD28 69FE
#60A0: FD28 6526 006F 2901 D164 097E E5E6 0747
#60B0: 2176 6418 7ECD 9364 1A32 7664 3E02 32BF
#60C0: 612A 8060 2323 2280 601A E607 FE06 2007
#60D0: ED5B 5564 CD93 64CD 5D64 A7C9 CD93 641A
#60E0: 3276 6413 FECB CAA5 61FE EDCA A561 FEDD
#60F0: CAA5 61FE FDCA A561 2600 6F29 01D1 6609
#6100: 7EFE 80CA A561 1824 CD93 641A 3276 6413
#6110: FEED CAA5 61FE DDCA A561 FEFD CAA5 61FE
#6120: CB28 6126 006F 2901 CD68 097E E5E6 0747
#6130: 2177 6405 2809 CD93 641A 7713 2318 F411
#6140: 7564 A7ED 527D 32BF 61ED 5B80 6019 2280
#6150: 60E1 7E23 CB77 201D E57E 1F1F 1F1F E60F
#6160: 11A9 61CD C764 E17E E60F 1111 62CD C764
#6170: CD5D 64A7 C97E 117B 62CD C764 2A80 607C
#6180: B5C0 37C9 CD93 641A 3277 6413 CD93 641A
#6190: 3278 64CD EB61 2A80 6011 0400 1922 8060
#61A0: CD5D 64A7 C937 3E01 C952 00B9 61CB 61D3
#61B0: 61DB 61E3 61EB 6105 6221 7564 1600 3E00
#61C0: D602 5F19 5E23 56CD 9364 C9CD B961 13CD
#61D0: 9364 C9ED 5B59 64CD 9364 C9ED 5B57 64CD
#61E0: 9364 C9ED 5B55 64CD 9364 C92A 4B64 3A75
#61F0: 64FE DD28 032A 4964 3A77 644F 179F 4709
#6200: EBCD 9364 C9ED 5B73 64CD 9364 13CD 9364
#6210: C952 0021 6234 623C 6244 624C 6254 626E
#6220: 6221 7564 1600 3ABF 61D6 025F 195E 2356
#6230: CD97 64C9 CD21 6213 CD97 64C9 ED5B 5964
#6240: CD97 64C9 ED5B 5764 CD97 64C9 ED5B 5564
#6250: CD97 64C9 2A4B 643A 7564 FEDD 2803 2A49
#6260: 643A 7764 4F17 9F47 09EB CD97 64C9 ED5B
#6270: 7364 1BCD 9764 1BCD 9764 C9C9 62E8 62D0
#6280: 62D5 62DA 62DF 6294 6300 6380 6346 6399
#6290: 63C0 6305 634B 6372 639E 630A 6350 63A3
#62A0: 630F 6355 63A8 6314 635A 63AD 632E 6319
#62B0: 635F 63B2 631E 6364 63B7 6323 6369 6311
#62C0: 64DD 632D 64F7 6335 6321 5A64 35C0 1818
#62D0: 0140 4018 0D01 0040 1808 0101 0118 0301
#62E0: 0001 3A5B 64A0 A9C8 2175 6416 003A BF61
#62F0: 3D5F 197E 2A80 604F 179F 4709 2280 60C9
#6300: 0140 4018 2101 0040 181C 0101 0118 1701
#6310: 0001 1812 0104 0418 0D01 0004 1808 0180
#6320: 8018 0301 0080 3A5B 64A0 A9C8 1852 2A55
#6330: 6422 8060 C92A 4B64 3A75 64FE DD28 032A
#6340: 4964 2280 60C9 0140 4018 2101 0040 181C
#6350: 0101 0118 1701 0001 1812 0104 0418 0D01
#6360: 0004 1808 0180 8018 0301 0080 3A5B 64A0
#6370: A9C8 ED5B 8060 2A73 642B 722B 7322 7364
#6380: 2175 6416 003A BF61 D602 5F19 5E23 56ED
#6390: 5380 60C9 0140 4018 2101 0040 181C 0101
#63A0: 0118 1701 0001 1812 0104 0418 0D01 0004
#63B0: 1808 0180 8018 0301 0080 3A5B 64A0 A9C8
#63C0: 2A73 64ED 5B0C 60A7 E5ED 52E1 1100 0028
#63D0: 075E 2356 2322 7364 ED53 8060 C9ED 5B55
#63E0: 64ED 4B59 642A 5C64 CD93 641A 130B BD28
#63F0: 6C78 B120 F318 66ED 5B55 64ED 4B59 642A
#6400: 5C64 CD93 641A 1B0B BD28 5278 B120 F318
#6410: 4C2A 5564 ED5B 5764 ED4B 5964 EBCD 9364
#6420: EBCD 9764 2313 0B78 B120 F118 302A 5564
#6430: ED5B 5764 ED4B 5964 EBCD 9364 EBCD 9764
#6440: 2B1B 0B78 B120 F118 1400 0000 0000 0000
#6450: 0000 0000 0000 0000 0000 0000 00ED 7390
#6460: 6431 4964 FDE1 DDE1 E1D1 C1F1 D908 E1D1
#6470: C1F1 3100 0000 0000 00F3 ED73 7364 315D
#6480: 64F5 C5D5 E5D9 08F5 C5D5 E5DD E5FD E531
#6490: 0000 C93E 0118 023E 0232 C364 E5D5 C52A
#64A0: 0660 ED4B 0860 09EB ED52 C1D1 E1D0 E5D5
#64B0: 2A06 60EB A7ED 5238 0BED 5B0A 6019 7EA7
#64C0: 2002 3601 D1E1 C926 006F 2919 5E23 56EB
#64D0: E901 0003 0001 0301 0001 0001 0002 0001
#64E0: 0001 0001 0001 3001 0001 0001 0002 0001
#64F0: 0042 0003 0001 0401 0001 0001 0002 0001
#6500: 0042 0101 0001 4001 0001 0001 0002 0001
#6510: 0042 0203 0003 0201 0001 0001 0002 0001
#6520: 0042 0301 0003 2001 0001 0001 0002 0001
#6530: 0042 0403 0003 0101 0001 5001 5002 0501
#6540: 0042 0501 0003 1001 0001 0001 0002 0001
#6550: 0001 0001 0001 0001 0001 0001 0001 5001
#6560: 0001 0001 0001 0001 0001 0001 0001 5001
```

```
#6570: 0001 0001 0001 0001 0001 0001 0001 5001
#6580: 0001 0001 0001 0001 0001 0001 0001 5001
#6590: 0001 0001 0001 0001 0001 0001 0001 5001
#65A0: 0001 0001 0001 0001 0001 0001 0001 5001
#65B0: 0001 0501 0501 0501 0501 0501 0501 0001
#65C0: 0501 0001 0001 0001 0001 0001 0001 5001
#65D0: 0001 0001 0001 0001 0001 0001 0001 5001
#65E0: 0001 0001 0001 0001 0001 0001 0001 5001
#65F0: 0001 0001 0001 0001 0001 0001 0001 5001
#6600: 0001 0001 0001 0001 0001 0001 0001 5001
#6610: 0001 0001 0001 0001 0001 0001 0001 5001
#6620: 0001 0001 0001 0001 0001 0001 0001 5001
#6630: 0001 0001 0001 0001 0001 0001 0001 5001
#6640: 0001 0001 0001 0001 0001 0001 0001 5001
#6650: 0041 0601 7043 0743 0843 0901 0702 0001
#6660: 0041 0A41 0B43 0C00 0043 0D43 0E02 0001
#6670: 0041 0F01 7043 1002 0043 1101 0702 0001
#6680: 0041 1201 0043 1302 0043 1401 0002 0001
#6690: 0041 1501 7043 1601 7043 1701 0702 0001
#66A0: 0041 1841 1943 1A01 0043 1B00 0002 0001
#66B0: 0041 1C01 7043 1D01 0043 1E01 0702 0001
#66C0: 0041 1F01 0043 2001 0043 2100 0002 0001
#66D0: 0080 8080 8080 8080 8080 8080 8080 8080
#66E0: 8080 8080 8080 8080 8080 8080 8080 8080
#66F0: 8080 8080 8080 8080 8080 8080 8080 8080
#6700: 8080 8080 8080 8080 8080 8080 8080 8080
#6710: 8080 8080 8080 8080 8080 8080 8080 8080
#6720: 8080 8080 8080 8080 8080 8080 8080 8080
#6730: 8080 8080 8080 8080 8080 8080 8080 8080
#6740: 8080 8080 8080 8080 8080 8080 8080 8080
#6750: 8001 0001 0001 0003 0201 0041 0B01 0001
#6760: 0001 0001 0001 0003 2080 0041 0B80 0001
#6770: 0001 0001 0001 0003 0280 8080 8001 0001
#6780: 0001 0001 0001 0003 2080 8080 8001 0001
#6790: 0001 0001 0001 0003 0280 8080 8080 8001
#67A0: 0001 0001 0001 0003 2080 8080 8080 8001
#67B0: 0001 0080 0001 0003 0280 8080 8080 8080
#67C0: 8001 0001 0001 0003 2080 8080 8080 8080
#67D0: 8080 8080 8080 8080 8080 8080 8080 8080
#67E0: 8080 8080 8080 8080 8080 8080 8080 8080
#67F0: 8080 8080 8080 8080 8080 8080 8080 8080
#6800: 8080 8080 8080 8080 8080 8080 8080 8080
#6810: 8001 5401 5001 5001 5080 8080 8080 8080
#6820: 8001 5401 5001 5001 5080 8080 8080 8080
#6830: 8041 2241 2380 8080 8080 8080 8080 8041
#6840: 2441 2580 8080 8080 8080 8080 8080 8080
#6850: 8080 8080 8080 8080 8080 8080 8080 8080
#6860: 8080 8080 8080 8080 8080 8080 8080 8080
#6870: 8080 8080 8080 8080 8080 8080 8080 8080
#6880: 8080 8080 8080 8080 8080 8080 8080 8080
#6890: 8080 8080 8080 8080 8080 8080 8080 8080
#68A0: 8080 8080 8080 8080 8080 8080 8080 8080
#68B0: 8080 8080 8080 8080 8080 8080 8080 8080
#68C0: 8080 8080 8080 8080 8080 8080 8001 0003
#68D0: 0001 0301 0001 0001 0002 0001 0001 0001
#68E0: 0001 3001 0001 0001 0002 0001 0042 0003
#68F0: 0001 0401 0001 0001 0002 0001 0042 0101
#6900: 0001 4001 0001 0001 0002 0001 0042 0203
#6910: 0003 0201 0001 0001 0002 0001 0042 0301
#6920: 0003 2001 0001 0001 0002 0001 0042 0403
#6930: 0003 0101 0002 6002 6003 0601 0042 0501
#6940: 0003 1001 0001 0001 0002 0001 0001 0001
#6950: 0001 0001 0001 0001 0002 6001 0001 0001
#6960: 0001 0001 0001 0001 0002 6001 0001 0001
#6970: 0001 0001 0001 0001 0002 6001 0001 0001
#6980: 0001 0001 0001 0001 0002 6001 0001 0001
#6990: 0001 0001 0001 0001 0002 6001 0001 0001
#69A0: 0001 0001 0001 0001 0002 6001 0002 0602
#69B0: 0602 0602 0602 0602 0601 0002 0601 0001
#69C0: 0001 0001 0001 0001 0002 6001 0001 0001
#69D0: 0001 0001 0001 0001 0002 6001 0001 0001
#69E0: 0001 0001 0001 0001 0002 6001 0001 0001
#69F0: 0001 0001 0001 0001 0002 6001 0001 0001
#6A00: 0001 0001 0001 0001 0002 6001 0001 0001
#6A10: 0001 0001 0001 0001 0002 6001 0001 0001
#6A20: 0001 0001 0001 0001 0002 6001 0001 0001
#6A30: 0001 0001 0001 0001 0002 6001 0001 0001
#6A40: 0001 0001 0001 0001 0002 6001 0041 0601
#6A50: 7043 0743 0843 0901 0702 0001 0041 0A41
#6A60: 0B43 0C00 0043 0D43 0E02 0001 0041 0F01
#6A70: 7043 1002 0043 1101 0702 0001 0041 1201
#6A80: 0043 1302 0043 1401 0002 0001 0041 1501
#6A90: 7043 1601 7043 1701 0702 0001 0041 1841
#6AA0: 2643 1A01 0043 1B00 0002 0001 0041 1C01
#6AB0: 7043 1D01 0043 1E01 0702 0001 0041 1F01
#6AC0: 0043 2001 0043 2100 0002 0001 00-
```

Чтобы проверить, не было ли допущено ошибок при вводе дампа, вычислите контрольную сумму соответствующего участка памяти с помощью следующей процедуры:

```
        ORG     #8000

        LD      DE,0
        LD      HL,#6000
        LD      BC,#ACD

M1      LD      A,E
        ADD     A,(HL)
        LD      E,A
        JR      NC,M2
        INC     D
M2      INC     HL
        DEC     BC
        LD      A,B
        OR      C
        JR      NZ,M1

        RET
```

После окончания ее работы содержимое регистровой пары DE должно равняться #D6C9.

Руководство пользователя

Полный текст программы обработки музыкальных модулей, основанной на описанном выше принципе, выложен на web-страницу журнала, а здесь я расскажу, как ею пользоваться.

Сразу же замечу, что адреса, по которым располагаются как сама программа, так и структуры данных, можно при необходимости изменить (например, если по тем же адресам должен располагаться обрабатываемый модуль).

Итак, вам нужно обработать модуль с плеером или без (если модуль без плеера, то плеер должен быть загружен отдельно). Вам должна быть известна следующая информация — адрес и длина модуля, адреса процедур INIT, PLAY и STOP плеера (если STOP не используется — адрес считается равным #0052), количество вызовов процедуры PLAY (если модуль зациклен и проигрывается "бесконечно", то количество вызовов устанавливается таким, чтобы цикл был проигран два раза).

Так как для хранения количества вызовов процедуры PLAY используется двухбайтная переменная, количество вызовов не может превышать 65535. Исходя из того что между двумя вызовами проходит 1/50 с, получаем, что максимальное время проигрывания модуля — 21 минута 50,7 с . Я не стал усложнять программу, выделяя под переменную три байта для увеличения максимального времени проигрывания, так как для подавляющего большинства модулей это время не превышает нескольких минут. Тем не менее, изменение программы при необходимости не представляет каких-либо сложностей.

Итак, устанавливаете значения соответствующих переменных в тексте программы, компилируете ее, затем загружаете модуль и вызываете процедуру "эмуляция с обнулением массива RW_MEMORY" (это массив с информацией об используемых ячейках памяти). Для вызова этой процедуры используется точка входа по смещению 0 от начала программы.

Эмуляция займет довольно продолжительное время, при этом вы будете слышать замедленное проигрывание мелодии. Если после ее окончания флаг C будет сброшен, то все в порядке; если же установлен — значит, при эмуляции встретилась неподдерживаемая недокументированная команда (см. раздел "Ограничения эмуляции"). Адрес этой команды вы сможете узнать из переменной EM_PC.

После окончания эмуляции модуль следует перезагрузить — вдруг при его проигрывании значения каких-либо

ячеек изменились? Затем вызовите процедуру "установка новых значений неиспользуемых ячеек памяти" (точка входа по смещению 4 от начала программы). После этого остается только записать обработанный модуль на диск.

Когда нужно обработать один за другим несколько модулей, удобнее устанавливать значения переменных не в исходном тексте программы с последующей перекомпиляцией, а непосредственно в коде программы с помощью отладчика. Смещения переменных от начала программы приведены в **табл.1**. При установке значений не забывайте, что они должны быть в формате low-high — сначала младший байт, потом старший.

Если в вашей программе используются несколько модулей без плеера, проигрываемые отдельным плеером, и вы хотите обработать только этот плеер, то порядок действий следующий. Устанавливаете значения соответствующих переменных в тексте программы, подставляя при этом вместо адреса начала модуля и длины модуля соответственно адрес начала плеера и длину плеера. Компилируете программу, в отладчике загружаете плеер и первый музыкальный модуль. Вызываете процедуру "эмуляция с обнулением массива RW_MEMORY". После окончания эмуляции загружаете второй модуль и вызываете процедуру "эмуляция без обнуления массива RW_MEMORY" (точка входа по смещению 2 от начала программы). Так же повторяете и со всеми остальными модулями. Так как очистка массива с информацией об используемых ячейках памяти не производится, информация в нем будет накапливаться — если при проигрывании хотя бы одного модуля значение какой-то ячейки плеера будет использоваться, она будет помечена в массиве. Затем перезагружаете плеер, вызываете процедуру "установка новых значений неиспользуемых ячеек памяти" и записываете обработанный плеер на диск.

Табл. 1

Переменная (2 байта, low-high)	Смещение от начала программы
Адрес модуля	6
Длина модуля	8
Адрес массива с информацией об используемых ячейках памяти	10
Адрес вершины стека для эмуляции (стек растет вниз!)	12
Адрес процедуры INIT	14
Адрес процедуры PLAY	16
Количество вызовов процедуры PLAY	18
Адрес процедуры STOP	20

Получаемый выигрыш

А стоит ли овчинка выделки? Еще бы! :-) Для примера я взял десять PT3-модулей с плеером, которые были представлены на demoparty "Chaos Constructions 1", и сравнил степень их сжатия до и после обработки. Результаты приведены в **табл.2**. Упаковка модулей производилась с помощью HRUST 1.3 в режиме normal.

Если подсчитать средний выигрыш, получится 8,5% от исходной длины модуля, т.е примерно 850 байтов для модуля длиной 10000 байтов.

В **табл.3** приведена аналогичная информация для десяти PT3-модулей без плеера, которые также были взяты из представленных на demoparty "Chaos Constructions 1".

Как видим, здесь средний выигрыш составляет около 3% от длины исходного модуля, т.е. примерно 150 байтов для модуля длиной 5000 байтов.

Табл. 2

Файл	Длина, байтов	Старая длина после упаковки		Новая длина после упаковки		Выигрыш		
		Байтов	% от исходной	Байтов	% от исходной	Байтов	% от исходной длины	% от старой длины после упаковки
F_ANTROP.C	5963	3258	55	2521	42	737	12	23
IN2LIGHT.C	11404	4479	39	3583	31	896	8	20
squeak.C	12871	5809	45	5269	41	540	4	9
i'veCCCC.C	10226	4185	41	3491	34	694	7	17
SUMMER.C	13239	5905	45	5436	41	469	4	8
LostMemo.C	6739	3507	52	2877	43	630	9	18
Another.C	5627	3151	56	2453	44	698	12	22
FroznABC.C	12800	5683	44	5272	41	411	3	7
simpnice.C	5357	2942	55	2102	39	840	16	29
spacefly.C	6885	3627	53	2953	43	674	10	19

Табл. 3

Файл	Длина, байтов	Старая длина после упаковки		Новая длина после упаковки		Выигрыш		
		Байтов	% от исходной	Байтов	% от исходной	Байтов	% от исходной длины	% от старой длины после упаковки
ARCTLAND.m	4318	1736	40	1662	38	74	2	4
HAPPYLIL.m	3408	1189	35	1089	32	100	3	8
TO STAR.m	5177	1798	35	1715	33	83	2	5
SURPRIZE.m	6495	1515	23	1445	22	70	1	5
flies.m	4896	1849	38	1784	36	65	1	4
Zvaigzde.m	4438	2193	49	1929	43	264	6	12
SEA.m	3036	1457	48	1381	45	76	3	5
ACID GOA.m	3930	1479	38	1036	26	443	11	30
bugsMind.m	4176	1567	38	1489	36	78	2	5
PEACE.m	7124	2297	32	2146	30	151	2	7

О скорости эмуляции

При эмуляции выполнение программы сильно замедляется — в среднем в 125 раз (для разных команд замедление неодинаково). Это,однако, не значит, что 5-минутный модуль будет обрабатываться более 10 часов. Как уже упоминалось выше, проигрывание музыки занимает лишь небольшую часть процессорного времени (для модулей, созданных в Pro Tracker 3 — примерно 8%). Соответственно, 5-минутный PT3-модуль будет обработан примерно за 50 минут.

Да, быстрым этот процесс не назовешь. Каждую выполняемую команду требуется распознать, определить способ ее выполнения (непосредственным запуском или с помощью специальной подпрограммы), узнать, происходит ли при ее выполнении чтение из памяти или запись в память (и по каким адресам) и изменить при необходимости содержимое соответствующего элемента (элементов) массива, где хранится информация об обращении к ячейкам. На это и уходит время.

Чтобы повысить скорость эмуляции, можно, вместо того чтобы всегда выполнять программу по отдельным командам, во время выполнения выявлять участки программы, в которых нет команд перехода за пределы участка, и либо нет команд чтения/записи, либо есть, но адреса операндов фиксированы (тогда информация об обращении к соответствующим ячейкам памяти будет обработана при первом выполнении участка). Также в участке не должна происходить самомодификация кода. Тогда при повторном выполнении такого участка не будет обращения ни к одной ячейке памяти, кроме тех, к которым было обращение при его первом выполнении. Таким образом, вместо покомандного выполнения достаточно скопировать этот участок в буфер и непосредственно запустить на выполнение, что будет значительно быстрее.

При этом, правда, появятся дополнительные затраты времени на определение таких участков и занесение информации о них в специальный список; на определение того, что началось выполнение такого участка; на отслеживание записи в участки (если произошла такая запись, код участка может быть модифицирован, и такой участок должен быть исключен из списка). Так что, возможно, повышение скорости эмуляции не окажется значительным. Но это я предлагаю проверить самостоятельно.

Ограничения эмуляции

Так как программа предназначена для выполнения довольно специфической задачи — обработки музыкальных модулей, то эмуляция Z80 была несколько упрощена. Но если вы захотите обрабатывать с помощью этой программы не музыкальные модули, а что-нибудь еще, то должны иметь представление об ограничениях эмуляции:

- каждая команда выполняется при запрещенных прерываниях (то есть при выполнении команды HALT все зависнет);
- не поддерживаются недокументированные команды, начинающиеся с префикса ED, а также с последовательностей префиксов DD ED и FD ED;
- не поддерживается выполнение последовательности префиксов DD и FD (например, DD DD FD DD FD);
- команды блочного ввода-вывода (INIR, INDR, OTIR, OTDR) не выполняются (помечены в таблице команд как неподдерживаемые недокументированные команды), т.к. их эмуляция потребовала бы наличия специальных подпрограмм, определяющих, к каким ячейкам памяти будет обращение (так же, как и для команд LDDR, LDIR, CPDR, CPIR);
- эмуляция регистра R не производится;
- при выполнении команд LDIR и LDDR не отслеживается ситуация, когда при записи очередного байта происходит изменение кода команды (код этих команд считывается из памяти в каждом цикле выполнения);
- команды RST выполняются, но обращение к ячейкам памяти при их выполнении не учитывается;
- переключение банков памяти специально не отслеживается.

От редакции: на web-странице журнала http://radio-mir.com выложены полный листинг и дамп кодов программы обработки музыкальных модулей. Желающие могут получить их и из редакции по предварительному запросу.

Привязка видеосигнала к уровню "черного"

А.ПЕХИМЕНКО,
Казахстан, Кустанайская обл., г.Рудный,
E-mail: paalex@hotmale.com

На некоторых компьютерах нет привязки уровня видеосигнала к уровню "черного". В этом случае черный квадрат повышенной яркости на черном фоне смотрится на экране почти как белый, и иногда при чередовании цветов может происходить сбой синхронизации. Владельцам таких компьютеров предлагаемая схема может понадобиться. Также эта схема необходима для режима "аппаратный мультиколор 1 цвет — 1 бит" — иначе соседние цвета могут смотреться неправильно.

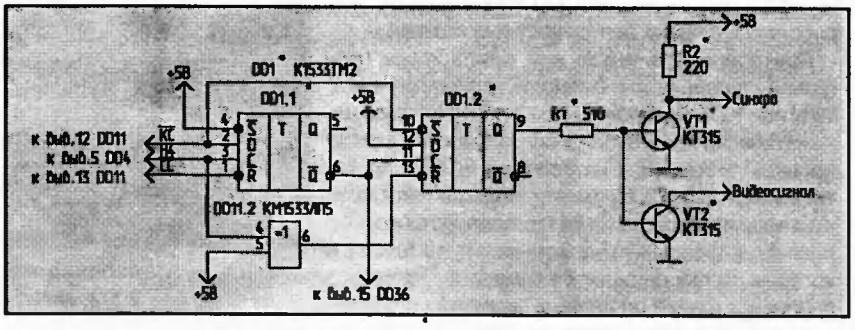
Сама по себе, эта схема не является новой. В отличие от схемы аппаратного "мультиколора" [1], это не моя авторская разработка — я ее только выдернул из схемы "зеленого" "Scorpion ZS-256" и опробовал на "Ленинграде".

Схема привязки (**см. рисунок**) состоит из двух дополнительных транзисторов, микросхемы К1533ТМ2 (можно другой серии) и пары резисторов. Предварительно на плате "Ленинграда" удаляется диод VD10, идущий на транзистор видеосигнала; выводы 4 и 6 микросхемы DD11 и вывод 15 микросхемы DD36 отрезаются от платы. Затем производятся соединения в соответствии с рисунком.

Хотя приведенная схема предназначена для доработки компьютера типа "Ленинград", для других компьютеров она будет аналогичная. Все элементы доработки монтируются поверх одной из микросхем оригинальной платы (например, DD11) или на свободном месте.

Литература

1. А.Пехименко. Режим мультиколор для Spectrum. — Радиомир. Ваш компьютер, 2002, №5, С.43.



По страницам электронных изданий

Блокировка порта #1FFD на "Scorpion ZS-256"

Вашему вниманию предлагается несложная доработка компьютера "Scorpion ZS-256", которая поможет значительно облегчить жизнь его владельцам.

В последнее время развелось довольно значительное количество программ, авторы которых адресуют порт по половине адреса, пользуясь для переключения страниц командой OUT (#FD),A. Такие программы отказываются работать на "Скорпионах", т.к. "Скорпиону" требуется полный адрес порта.

Предлагаемая доработка позволяет запускать подобные программы. Она блокирует порт #1FFD, за счет чего "проходит" команда OUT (#FD),A, и удается без приложения дополнительных усилий посмотреть "SATISFACTION" или "INSULT".

Следует помнить только одно — программы, которые "лазят" в ПЗУ через RST8, с кнопкой работать не будут. Это происходит из-за того что RST8 находится под контролем "теневика", которому для работы необходима 8-я страница и, естественно, порт #1FFD.

Итак, теперь более подробно. Кнопку можно нажимать в любой момент, находясь в boot'е или в самой программе. Не следует нажимать кнопку только находясь в главном меню компьютера, т.к. там тоже "хозяйничает" "теневик".

Например, вам необходимо запустить ранее не работавшую на "Скорпионе" программу. Вы идете в TR-DOS, нажимаете кнопку, запускаете программу — RUN "имя программы" и с удивлением видите, что программа работает!!! Или еще проще — нажали ENTER в главном меню, "вывалились" в TR-DOS с загрузкой boot'а, нажали кнопку, запустили программу. Единственный известный boot, который не дружит с кнопкой — старинный boot Трубинова (вероятно, он использует RST).

Обратите внимание, что нажатие кнопки не мешает вам использовать теневик!!! При нажатии "Magic", порт #1FFD разблокируется, и вы без проблем окажетесь в любимом "теневике". Вторым управляющим сигналом для разблокировки порта является, естественно, "Reset". Т.е. по "сбросу" или "мэджику" компьютеру "возвращаются" его порт и "лишние" страницы.

Таким образом, выявляется еще один плюс предлагаемой доработки — кнопка позволяет "обманывать" защиты "Скорпиона", основанные на "забивании" 8-й страницы наугад всякой "дрянью". И хотя при придумывании доработки этот эффект не брался в расчет, он является еще одним аргументом в пользу ее использованиия.

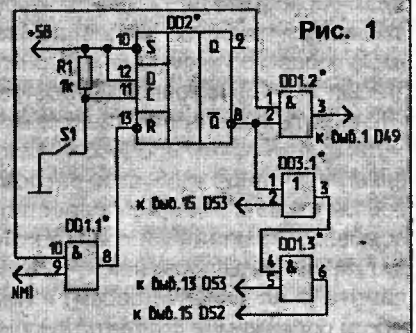


Рис. 1

Теперь несколько поясняющих слов о схеме (**рис.1**). Позиционные обозначения "родных" элементов "Scorpion ZS-256" приведены в соответствии с исходной схемой компьютера, новые элементы отмечены "звездочкой". Предварительно необходимо отсоединить вывод 1 микросхемы D49 от соответствующей дорожки печатной платы, не разорвав саму дорожку. Также необходимо перерезать проводник между выводом 13 м/с D53 и выводом 15 м/с D52. Затем следует произвести соединения в соответствии со схемой доработки. В качестве кнопки S1 можно использовать любой микропереключатель, DD1* — К1533ЛИ1, DD2* — К1533ТМ2, DD3* — К1533ЛЛ1. Удобно использовать половинку триггера ТМ2, которая осталась от "турбирования ВГ" (разумеется, если у вас это сделано). Номера выводов этой микросхемы (DD2* на рис.1) приведены с учетом этого факта. Микросхемы DD1* и DD3* устанавливаются на свободное место платы компьютера. Их незадействованные (пока) элементы пригодятся для дальнейших доработок. Для большего удобства рекомендуется сделать индикацию (**рис.2**). Теперь вы никогда не запутаетесь — при включенной блокировке порта светодиод светится, при выключенной — нет.

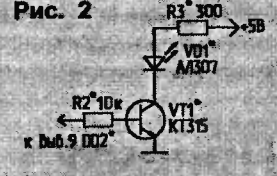


Рис. 2

В заключение хочется развеять сомнения нерешительных. За два года эксплуатации данной доработки не было выявлено ни одного "минуса" — ведь пока вы не нажмете на кнопку, доработка никак не влияет на работу схемы компьютера.

Подготовил Е.Зарецкий.

BestView 2.12

BV_CREATOR,
г.Москва.
Fido: 2:5020/689.53
ZXNet: 500:95/462.53
E-mail: asder_ffc@softhome.net
WWW: http://www.ivr.da.ru

Не прошло и года :), как вышла очередная версия BestView — одной из лучших программ для просмотра и запуска файлов на "ZX-Spectrum". Я хочу рассказать о новых возможностях программы.

При просмотре текста или дампа кодов программы теперь можно выбирать кодировку — ALT, WIN или KOI. Кодировка может определяться автоматически (для этого был разработан оригинальный алгоритм, о котором вы можете прочитать в журнале [1]). Просматриваемый текст можно отформатировать на 64 символа в строке. Это позволяет с большим удобством читать тексты, в которых строки длиннее 64 символов. Также реализован быстрый поиск файла по первым символам его имени.

Стало удобнее получать листинги BASIC-программ, т.к. можно отключать отображение внутреннего пятибайтного представления чисел в программе. При просмотре текстовых, BASIC- и ASM-файлов первая страница текста теперь быстрее появляется на экране. Выполнены и другие небольшие усовершенствования, а также исправлены замеченные ошибки. При этом увеличение длины программы составило менее 3 Кб.

BestView 2.12 выложена на моей web-странице — http://www.ivr.da.ru. Кроме того, я могу выслать ее вам по E-mail, для этого напишите мне по адресу asder_ffc@softhome.net. Также я могу добавить ваш адрес в список рассылки, так что вы будете получать новые версии BestView сразу после их выхода.

Литература

1. И.Рощин. Автоматическое определение кодировки текста. — Радиомир. Ваш компьютер, 2002, №№5-6.

Красная акула

А.ВЕНДИЛОВСКИЙ,
г.Минск.

2009 г. Мир разрывается в локальных войнах, в Азии война принимает масштабный характер, в нее волей-неволей оказываются втянуты все крупные государства. Противостояние западной и исламской культур достигло своего апогея, когда американцы стали применять оружие массового поражения в ответ на террористические акты радикальных восточных партий. После распада Советского Союза в мире больше не осталось противовеса США, способного изменить ситуацию. Россия, втянутая в бесконечные внутренние конфликты, с подорванной экономикой, больше не представляла реальной силы. Мир все сильнее скатывался на порог Армагеддона. Но именно в России небольшая группа ученых смогла сделать невероятное открытие...

2010 г. Под эгидой спецслужб, лучшие физики России заканчивают разработку единой теории поля, которую не успел завершить Эйнштейн. Уже первые выкладки и эксперименты показали, что человечество обретает теперь власть не только над пространством и энергией, но и над временем! В обстановке строжайшей секретности, под прямым руководством Президента, военные приступают к созданию первого в мире темпорального оружия под кодовым названием "Молот Времени". Цель — произвести некоторые изменения в недалеком прошлом, чтобы изменить весь последующий ход истории.

Единственное ограничение, которое поставила перед учеными природа, было в том, что перебросить во времени можно было предмет не самых больших габаритов и на весьма ограниченное время. И невозможно было влиять на самые ключевые фигуры в истории. Нельзя было вернуться в прошлое, чтобы уничтожить Гитлера или другой исторический персонаж. Казалось, само Время защищает узловые этапы, не позволяя человеку переписать его заново. Но возможность изменить ход истории все же оставалась. Как цель для глобальных исправлений был выбран самый тяжелый для страны временной отрезок — Вторая мировая война. Военные историки принялись за работу, поднимая все архивы, выискивая узловые точки событий, а далеко за Уралом, под гранитной толщей скал, куда не мог проникнуть взор даже лучшего спутника-шпиона, в спешном порядке отстраивалась пусковая установка.

2011 г. Маршал рассматривал огромный зал, посреди которого стоял красавец К-50 с полным боевым комплектом, готовый к старту. Несмотря на то что они находились на глубине почти трех километров, этот вертолет мог спокойненько взлететь здесь, не боясь задеть стены искусственной пещеры. Повсюду сновали ученые, в последний раз проверявшие систему контроля запуска. Мелодичный голос из динамиков во всех лабораториях настойчиво предупреждал о начале обратного отсчета и требовал, чтобы персонал покинул опасную зону.

— Наша первая цель, товарищ маршал, находится в 1942 г. 12 июня 1942 г. советские войска послали большую эскадрилью бомбардировщиков для атаки по наступающим силам врага. Из-за малой осведомленности о расположении войск противника, их курс пролегал через мощную систему противовоздушной обороны немцев на этом участке фронта. Практически все самолеты были уничтожены, и наши войска остались без воздушной поддержки. Мы поставили перед нашими пилотами задачу — уничтожить систему немецкой ПВО до пролета наших эскадрилий и вывести из строя вражеский аэродром...

В этот момент лопасти вертолета стали вращаться, и через минуту он уже висел в нескольких метрах от пола. Сам пол внезапно стал раздвигаться, и взору присутствующих открылась темная, казавшаяся бездонной шахта, откуда, словно змеи, выскочили синеватые молнии, жадно облизывающие стены шахты и корпус вертолета.

... пять, четыре, три, два, один, генератор запущен! — послышалось из динамика, когда тысячи беснующихся электрических разрядов сплелись в один клубок, и ярчайшая вспышка ударила по глазам людей. Когда они снова обрели зрение, то увидели, что вертолета в пещере уже не было.

— Удачи вам, сынки! — тихо произнес маршал, глядя на таймер, отсчитывающий время до возвращения экипажа.

1942 г. Герр Шнитке проверял расположение своих сил — по данным разведки, очень скоро над ними будет пролетать эскадрилья русских бомбардировщиков, которую никак нельзя было пропустить к развертывающимся войскам. Он внимательно смотрел на восток и поэтому пропустил удивительное природное явление. С абсолютно чистого неба ударили в землю молнии, разламывая деревья и на мгновения затмив даже солнце. Резкий громовой удар принес с собой терпкий запах озона. А через несколько минут перед изумленным офицером появился странный летательный аппарат с огромными вращающимися винтами. Этот агрегат на секунду ЗАВИС над базой, и Шнитке смог рассмотреть красные звезды на борту этого чудо-самолета. За многие годы, отданные службе в вермахте, он ни разу не встречался с подобной машиной и не слышал, чтобы нечто подобное было у Советов. Прежде чем Шнитке успел отдать первые приказания своим солдатам, две ракеты, сорвавшиеся с корпуса вертолета, превратили в щебень ближайшие зенитные орудия.

С этого момента воцарился хаос, неопознанный объект стрелой носился вдоль базы, просто сметая любые очаги сопротивления. Странное оружие безошибочно находило цели, словно по невидимой

ниточке двигаясь раз за разом за отчаянно уворачивающимся противником. Шнитке вжался в землю, отчаянно повторяя про себя, что этого просто не может быть, что все это дурной сон, и ему надо немедленно проснуться. Лопающиеся как грецкий орех танки, чья броня оказалась бессильна перед оружием из будущего, крики ужаса и боли, беспорядочная стрельба — люди падали подкошенные беспощадными пулеметными очередями с неба или погибали от взрывов ракет и собственной техники. Через несколько мгновений два снаряда попали в цистерны с горючим, и яростная волна багрового пламени, ломая и пожирая все на своем пути, захлестнула базу. Мало кто видел, как взмывший ангелом смерти из багрового пламени вертолет растворился в не менее яростном сполохе молний. На горизонте появились черные точки. Эскадрилья советских бомбардировщиков приближалась к месту гремевшего несколько минут назад боя...

Первая мысль, которая пришла мне в голову, едва диск с игрой попал в мои руки, что это еще один продвинутый вертолетный симулятор. Но, к моей великой радости, я ошибался — предо мной лежал добротный образец так называемой "спинномозговой" аркады, которой так не достает всем нам в последнее время.

Первое впечатление было весьма хорошим. Камера показывала вертолет в верхней части экрана, прицел прорисован аккурат посередине. Как в любой честной аркаде, физической моделью вертолета здесь и не пахнет. Так что смело можете забыть обо всех гримасах натурализма авиасимуляторов последнего времени и от всей души выписывать в воздухе всевозможнейшие кренделя, стараясь при этом не врезаться в землю или дерево. Хотя только на высшем уровне сложности контакт с земной и прочей поверхностью приводит к фатальным результатам, да и то не всегда — обычно дело заканчивается лишь уменьшением количества "жизни" аппарата, повреждения которого абсолютно не влияют на поведение модели в воздухе. Забегая вперед, хочу сразу сказать, что именно деревья и будут вашими главными противниками, но об этом немного позже. Забудем и о сложном управлении — пять клавиш и мышь, не нужен даже джойстик. Просто и со вкусом.

Первые миссии нельзя назвать сложными, они больше похожи на обучение, когда вы должны приноровиться к управлению летающей акулой Хотя это вызывает и огорчение, т.к. миссий всего пятнадцать, а по-настоящему серьезные проблемы возникают лишь к середине игры. Но с самого начала, как и в любой "спинномозговой" аркаде, большой приток адреналина обеспечен, когда вы будете поливать свинцом местность и отправлять ракеты направо и налево, вслед очень юрким танкам, не забывая уворачиваться от огня противника.

Когда эта игра была запущена на моем компьютере, я с чистой совестью решил пройти для начала первый уровень (только пятнадцать минуточек, одним глазком, и сразу за работу), но скоро несколько потерял счет времени. Работа вскоре была заброшена (какая может быть работа, когда война идет?), и вся толпа сослуживцев с жаром обсуждала, как правильно нужно было проходить то или иное укрепление врага или освобождать очередной поселок от гитлеровцев. Даже Шеф не выдержал и подключился к диалогу, ностальгически вспоминая времена, когда дискеты были большими, игры маленькими, а аркады переживали свои золотые времена.

Графика в игре... Пока летишь высоко, можно любоваться лесом, горами, дымкой, а вот при ближайшем рассмотрении, даже при максимальной детализации, все выглядит несколько угловато и коряво. Лес выглядит очень куцым, впрочем, как и любые другие постройки. И это при том, что сама модель вертолета очень качественно и красиво прорисована! Часто попадается глюк, когда полигоны "заходят" друг в друга, и ваш вертолет может несколько провалиться в ближайшее здание, или колонна танков смело "проходит" сквозь погибшего собрата. Небогат и выбор моделей для врагов — все танки, вышки и прочие недруги просто на одно лицо. По моим скромным подсчетам получилось видов пять, не больше. Видимо, разработчики рассуждали, что поскольку с высоты детали оформления все равно не разглядишь, то и зачем тогда стараться. Ангары и дома — такие же "штампованные", что несколько разочаровывает. Взрыв цистерн очень зрелищен, жаль, что все остальные пиротехнические эффекты прорисованы несколько слабовато для подобной аркады.

На закуску остается разобраться, имеется в этой игре хороший геймплей или нет. Чтобы все миссии не были похожи друг на друга как близнецы-братья, разработчики стараются давать нам, по сути, различные задачи — то сопроводи колонну танков (предварительно зачистив местность от представителей комитета по встрече из вермахта), то уничтожь все системы ПВО, то помоги прорваться солдатам советской армии из окружения... Врагов много, а вот оружия маловато. На всю игру нам дается лишь два вида ракет и пулемет с неограниченным запасом патронов. Вот и приходится не только на гашетку давить, но и думать, стоит ли этот танк такой ценной ракеты, или же приберечь ее на черный день, который может очень скоро наступить, стоит вашей "Акуле" обогнуть этот пригорочек. Ну а не попавшая в цель ракета может не только испортить настроение до конца дня, но и привести к тому, что уровень придется проходить заново. А враг не дремлет, едва завидит вас, как сразу открывает шквальный и, хочу заметить, очень прицельный огонь, не позволяя вам расслабиться ни на секунду, заставляя вращать мышь с таким усердием, что коврик начинает дымиться. Передохнуть можно лишь в промежутке между миссиями, пока загружается очередной уровень и читается очередной брифинг.

Вывод— несмотря на все недочеты, у "G5 Software" (http://www.redshark.g5software.com) стильная аркадка, немного коротковатая, но весьма завлекательная.

Диск предоставлен интернет-магазином "MediaCraft" www.mediacraft.shop.by

К.КЛИЩЕНКО,
г.Минск.

Fear of the Dark

А-А.
"Крик"
А-А2.
"Крик-2"
Чува-к.
"Очень страшное кино"

Пугаться, знаете, приятно (иногда). Перед покупкой этой игры я обещал себе, что играть в нее буду только ночью, при выключенном свете и в наушниках (для остроты ощущений). После первого знакомства я сразу же поменял ночь на день (причем на очень яркий день), свет включил, а наушникам предпочел колонки, отставленные подальше. Правда, вскоре я понял, что не так уж эта игра и страшна, но ее начало аыполнено в достаточно мрачных и подавляющих тонах.

Жанр Survivol Horror ("ужасы на аыживание") прижился в основном на приставках. На PC долгое время единственным представителем была серия "Alone in the Dark". Первая игра этой серии внесла в мир компьютерных игр такое новшевство как полигональные персонажи, помещенные в заранее нарисованные декорации. Четвертая часть (та, с которой я вас знакомлю) недалеко ушла от родоначальницы. В ней также все, кроме участников событий, нарисовано заранее, а поэтому мечтать о более удачных видах, чем присутствующие, не приходится. Кроме того, любой предмет полигональной природы выделяется как голый афроамериканец в Антарктике. Это и хорошо, и плохо. Хорошо тем, что если нечто режет вам глаз, то это точно участвует в действе, к тому же, не приходится по сто раз нажимать "пробел" в поисках нужного места взаимоотношений с миром. Правда, хитрые авторы предусмотрели такую особенность. Так что в игре хватает обьектов, которые не полигональны, но с которыми по сюжету вам необходимо войти в контакт. Часто их можно найти по характерному для водки "Распутин" подмигиванию (т.е. по блеску при попадании на них луча света от вашего фонарика). Причем блестит даже бумага. А те немногие, но обычно крайне важные вещи, которые не умеют мигать, авторы оставляют вашей интуиции. Плохой особенностью является нарушение целостности восприятия мира — монстр не может прятаться за грудой вещей, а вы не можете сильно поменять обстановку без просмотра мультика.

Ну ладно, если уж заговорил о погружении в атмосферу, то надо продолжать. Вам ни разу не дадут просто открыть дверь — все это сопровождается немедленной сменой локации. Т.е. вам не придется орать, когда из-за двери на вас кто-то выпрыгнет. Он, конечно, может выпрыгнуть, но только после высылки дипломатической делегации, либо на собственной территории. Я имею в виду, что монстр либо телепортируется к вам в виде синей вспышки (правильно, даже в армии учат, что враг обозначается синим цветом), либо вы выходите из комнаты, делаете некое действие (обычно имеющее непосредственное отношение к сюжету), а потом по возвращении обнаруживаете на ранее очищенном месте пачку монстров. На логичный вопрос о том, откуда же они здесь взялись, вы скорее всего услышите гордое: "Выкристаллизовались". Хорошо хоть не сублимировались. И дело даже не в злоупотреблении героями продукцией завода "Кристалл". Просто авторская фантазия наделила создания тьмы родственностью с кристаллами. Правда, не только структура интересна в этих чудищах (она, кстати, ни на что не влияет). Основной нашей игровой особенностью является светочувствительность (некоторых можно отогнать только светом). Вообще чувствуется, что авторы решили поиграть с ранее недоступной игрушкой — освещенностью. Тут и незаменимый фонарик, с помощью которого надо искать предметы и пугать хороших монстров, и выключатели. Если вы входите в комнату, то прежде всего поищите выключатель. Так как осматриваться лучше при включенном свете (видно больше деталей, да и тени не мешают — ну не будешь же постоянно направлять на них фонарь, а иначе кусают). Но иногда придется наоборот — выключать иллюминацию, чтобы заметить или сделать то, что от вас скрывает с-свет.

С-сюжет ведут два пакос-с-с-тных героя — мужчина-детектив и женщина-ученый. У каждого с-свои тайны, и мы надеемс-ся, что они рас-скроют их, только объединившис-сь с-с нами. Они бы ничего и не узнали и ходили бы в непоняттках, но в доме еще остались людиш-шки, которые либо не нужны нам, либо с-сильны. Да что там людиш-шки, в доме слишком много испис-саных белых листов, с наш-шими и ненаш-шими рас-с-скрытыми тайнами, и мы не можем их уничтожить. Но ничего, ни агентишка, ни женщина не узнают наши с-секреты, и не остановят нас-с-с. Тогда мы будем править этим миром. И не надейтесс им помочь. Они наши. С-создания Тьмы.

А-А-А-А-А-А. Помогите героям, я уже не в силах.

Умри, пис-сака! А мы будем с-следить за ними и с-соединим с-с нами, с-с-с нами они с-с-станут с-с-сильнее и помогут нам. И вы поможете с-с-сделать мир темнее. Потому что герои падут, а мы ос-с-с-существим план.

P.S-S-S-S. С-С-С-С-Свет-т нужно победитьс-с-с...

КУПЛЮ, ПРОДАМ, ОБМЕНЯЮ

*Для публикации бесплатных объявлений **некоммерческого характера** о покупке и продаже компьютерных комплектующих и программ, их текст можно присылать в письме по адресам:*

119454, г.Москва, а/я 37 или
220095, г.Минск-95, а/я 199,

*передавать по тел. в Минске **(017) 221-01-10**, либо через E-mail:*

rl@radiopage.by
rm@radio-mir.com
*WWW: **http://radio-mir.com***

Бесплатно высылаю схемы на любые темы и любой сложности, с подробным описанием и рисунком печатной платы. Для получения каталога схем высылайте конверт с обратным адресом.
357081, Ставропольский кр., Андроповский р-н, ст.Воровсколесская, ул.Центральная, 91.
Ширяев А.

Куплю контроллер дисковода "Электроника МС8414" для ПК "Электроника МС1502".
E-mail: mishsv@om.msi.ru

Требуется информационная поддержка по ремонту CD-ROM, CD-RW, DVD-ROM (только от жителей г. Минска).
Тел. 8-029-660-58-11, Сергей.

Ищу инструкцию по переделке игровой приставки Nintendo в генератор телевизионных испытательных сигналов.
Тел. в г.Бобруйске (8-0225) 55-41-64.
E-mail: ox68@rambler.ru

Куплю программное обеспечение: игры, графические и текстовые редакторы для компьютера "Искра-1030М" (СМ 5508, ДОС версии 3.30), инструкцию и описание принтара СМ 6337.
453500, Башкортостан, г.Белорецк, а/я 21.
Сафонов С.Н.
Тел. (34792) 4-45-67 (спросить Сергея).

Куплю компьютер "Радио-86 РК" с монитором и дисководом и документацию, программатор для PIC-контроллеров, литературу по микросхемам памяти.
Алексей.
E-mail: leoh1985@mail.ru

Недорого **продам или обменяю** струйный принтар EPSON COLOR STiLUS-600 с засохшей головкой.
Тел. в Воронеже (0732) 72-46-62. Виталий.
E-mail: ra3qg@mail.ru

Продаю недорого (можно по частям) коллекцию схем и мануалов по телефонии и видеотехнике, на бумаге и CD. Перечень вышлю.
426028, г.Ижевск, а/я 1502.
E-mail: danna100@chat.ru

Куплю двухкоординатный графопостроитель ПДП (ЛКД 4-003).
E-mail: angel_of_darkness@pisem.net

Продам акустику "Radiotehnika" (S-30), двухполосную.
Константин.
E-mail: kos_vr@mail.ru

Меняю на комплектующие для IBM **или продам** генераторы ПАЛ/СЕКАМ (габериты 160x60x35 мм), питание КРОНА (9В), литературу и CD-ROM по TV.
620138, г.Екатеринбург, а/я 220. Александр.

Продам коллекцию радиолюбительских CD-ROM (схемы, программы, электронные книги).
620039, Екатеринбург, а/я 172. Сергей.
E-mail: banan@pisem.net

Уважаемые читатели!

Подписаться на наши журналы (индексы: **48996** —"Радиомир", **48924** — "Радиомир. КВ и УКВ", **48925** — "Радиомир. Ваш компьютер") во II полугодии 2002 г. можно по каталогу Агентства "Роспечать", **стр. 235**, или по каталогу РО "Белпочта" "Газеты и журналы Республики Беларусь", **стр. 124** (раздел "Издания РФ, распространяемые по прямым договорам").

Те, у кого возникли проблемы с подпиской, могут получить их из редакции. Там же можно заказать имеющиеся в наличии отдельные номера журналов за предыдущие годы.

Год	Имеющиеся в наличии номера			Расценки на 1 экз. (с учетом пересылки)		
	Радиолюбитель	Радиолюбитель. Ваш компьютер	Радиолюбитель. КВ и УКВ	По России и СНГ (рос. руб.)	По Беларуси (бел. руб.)	По Украине (гривны)
1998	3, 5, 8	1 — 5, 8 — 12	1, 5, 6, 7, 11, 12	25	800	6
1999	2, 3, 9, 10, 11	1 — 6, 10, 11, 12	1 — 3, 5, 6	30	1000	7
2000	2 — 12	1 — 12	4 — 12	35	1200	8
2001	2 — 6	1 — 6	2 — 6	40	1400	8,5
	Радиомир	Радиомир. Ваш компьютер	Радиомир. КВ и УКВ	По России и СНГ (рос. руб)	По Беларуси (бел. руб.)	По Украине (гривны)
2001	7 — 12	7 — 12	7 — 12	40	1400	9
2002	1 — 12	1 — 12	1 — 12	45	1500	10

Наши платежные реквизиты:
- для жителей России и стран СНГ (кроме Беларуси) —
получатель: ООО "Радиомир Пресс", ИНН 7729404741,
р/с 40702810900010000084 в ООО КБ "Агропромкредит", ф-л Центральный, г.Москва,
корр. счет 30101810500000000109, БИК 044525109
Адрес банка: 113054, г.Москва, ул.Зацепа, 43Б;
- для жителей Беларуси —
получатель: УП "РЛД", УНН 190218688
р/с 3012000004882 в ф-ле №524 АСБ "Беларусбанк" в г.Минске код 121.
Адрес банка: 220028, г.Минск, ул.Физкультурная, 31.

На бланке почтового перевода необходимо очень четко написать свой почтовый индекс, полный адрес, фамилию, имя и отчество полностью. В графе "Для письма" необходимо точно перечислить, какие конкретно номера какого из журналов Вы заказываете.

При оплате платежным поручением нужно предварительно выписать счет.

Вся информация по E-mail: rm-sales@radio-mir.com или по тел. в г.Минске (017) 221-01-10

Приобретение отдельных номеров журналов

В РОССИИ:

В **ЗАО "Интерпресс — Экспресс"**: тел. в Москве (095) 208-85-55, 208-67-55, e-mail: inter@aif.ru

В магазинах радиодеталей **"ЧИП и ДИП"**:
- г.Москва, ул.Гиляровского, д.39 (ст. метро "Проспект Мира" — радиальная);
- г.Москва, ул.Ивана Франко, д.40, к.1, стр. 2 (платф.Рабочий поселок, 15 мин. от Белорусского вокзала);
- г.Москва, ул.Беговая, д.2;
- г.Ярославль, ул.Нахимсона, 12, тел. (0852) 27-57-15.

На радиорынках в Москве: Митинском (места R4, S8, K52, E50, G56), Царицынском (место 121).

В интернет-магазине изд-ва **"DMK-Press"**: www.dmkpress.ru с бесплатной доставкой по Москве.

В Москве в издательстве **"Солон-Р"**: ул.Садовая-Кудринская, д.11 (ст. метро Баррикадная, ст. метро Маяковская).
Время работы: с 09.00 до 18.00, кроме субботы, воскресенья.
Телефон: (095) 254-44-10.
Радиорынки: Царицынский (место 13/А), Митинский (ряд 8, контейнер 12, место Z25).
E-mail: Solon-R@coba.ru

В **"Фонде содействия радиолюбителям-радистам"**:
г.Москва, 4-й Войковский проезд, 10, тел. (095) 150-09-72, 150-04-16 (ст. метро "Войковская").

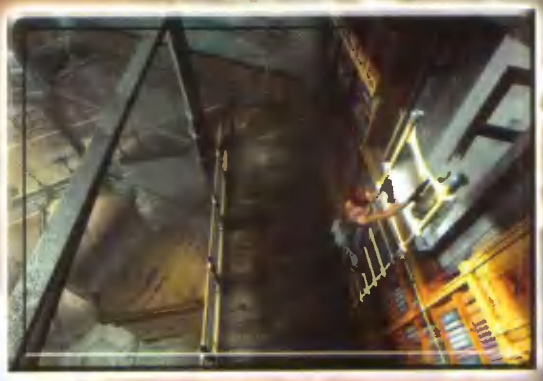
На УКРАИНЕ:

В Киеве в фирме **"Торм"**, тел. (044) 227-72-73.

В БЕЛАРУСИ:

В Минске в магазине **"Книга XXI век"**, пр.Ф.Скорины, д.92, тел. (017) 264-27-97 (ст.метро "Московская").

ALONE IN THE DARK




ПОЛНОСТЬЮ НА
ФАРГУС
РУССКОМ ЯЗЫКЕ
HEROES III
OF MIGHT AND MAGIC
ЖИЗНЬ ПРОТИВ СМЕРТИ
Life vs Death
1/2
2 IN 1
PC CD ROM
ПОЛНОСТЬЮ НА РУССКОМ ЯЗЫКЕ
Русский Проект
DIABLO II
LORD OF DESTRUCTION
ОСТРОВ ДЖУНГЛЕЙ
МедиаХауз
СБОРНИК
ЛУЧШИЕ
РУССКИЕ ВЕРСИИ
РУССКИЕ
КВЕСТЫ
ПЕТЬКА 3
ВОЗВРАЩЕНИЕ АЛЯСКИ
НА РУССКОМ ЯЗЫКЕ
SOLDIER OF FORTUNE II
DOUBLE HELIX
PC ACTIVISION
• Autobahn Racing • eRacer • Hot Wired •
СУПЕР ГОНКИ
2002
КРАСНАЯ АКУЛА
Бука
2В1
ПОЛНЫЙ РУССКИЙ ПЕРЕВОД
+ АНГЛИЙСКАЯ ВЕРСИЯ
GUILTY GEAR X
JOIN THE FIGHT TO SAVE THE WORLD!
TEEN
T