DOWNGRADE

N29 '2019

СОДЕРЖАНИЕ

ОТ РЕДАКТОРА

Тема этого номера – «Компьютерный андеграунд». Т.е. всё, что касается хакинга, вирусов, «экстремального программирования» (всё это только в историческом аспекте :-).

На этот раз статей по теме на удивление много – есть из чего выбрать. Неожиданно темой номера также стал компьютер БК – про него в №29 прислали аж три статьи: интервью с разработчиками игр для БК Дмитрием Барулиным (**Дес**) и Романом Подковыровым (**PC**), статья **Sh** про демо **BKvaders** и очень интересное продолжение статьи **Manwe** про цифровой звук на БК.

Есть статьи и на общие темы – скажем, окончание цикла **SysTools** о разработке различных утилит для редактора «Журналист», также прошу обратить внимание на очень интересную статью **Антиквара** по истории процессоров.

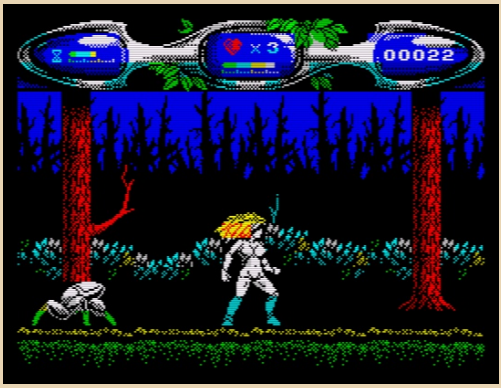
В общем, в этом номере много интересного – объём получился больше обычного.

Пользуясь случаем, поздравляю всех наших читателей с новогодними праздниками и желаю им приятно провести время в компании с нашим журналом. :-)

Как обычно, если у вас будут вопросы, пожелания, замечания, предложения (а, главное, статьи) – присылайте мне на e-mail uav16060 [zabyl_nazvanie_etoy_shtuki] mail.ru

uav1606

НОВОСТИ, СОБЫТИЯ, КОММЕНТАРИИ



Yandex Retro Games Battle 2019

В декабре Yandex подвёл итоги конкурса для разработчиков игр – Yandex Retro Games Battle 2019. Участникам требовалось создать игру произвольного жанра для ZX Spectrum 48K или 128K. Всего было прислано 19 работ. Победители получили крупные денежные призы. Ознакомиться со всеми работами (и скачать их) можно здесь:

https://rgb.yandex/

Первое место заняла **Valley of Rains** – аркада с отличной графикой и динамичным геймплеем (см. скриншот выше).

Опубликованы ранние исходные коды UNIX

К 50-летнему юбилею UNIX Музей компьютерной истории опубликовал исходники первой версии этой ОС. Коды были найдены в бумагах Денниса Ричи, они датируются 1970-1971 годами. Всего найдено около 190 страниц распечаток на ассемблере PDP-7 с рукописными пометками.

Коды содержат различные математические процедуры, а также исходники игры Space Travel, которая сыграла значительную роль в разработке UNIX.

Скачать все эти исходники можно здесь.

NAC объявила о приостановке производства аудиокассет

«Национальная аудиокомпания» США объявила о приостановке изготовления аудиокассет из-за проблем с сырьём – гамма-оксидом железа. Недостаток сырья связан с ремонтом на производящем его заводе. При этом NAC обещает, что в ближайшее время объёмы производства будут восстановлены.

Несмотря на то, что аудиокассеты считаются очень устаревшим носителем информации, они всё ещё достаточно популярны среди аудиофилов (наравне с виниловыми пластинками и CD).

Command & Conquer Remastered

Electronic Arts совместно с Petroglyph Games и Lemon Sky Studios занимается созданием HD-вариантов двух классических RTS – Command & Conquer и C&C: Red Alert. Обновлённые игры по-прежнему останутся в 2D, но получат поддержку высоких разрешений, обновлённый саундтрек и множество других улучшений. С ходом разработки вы можете ознакомиться здесь.

А здесь вы можете посмотреть тизер Command & Conquer: Remastered.

Постапокалиптическая ОС

Речь идёт о Collapse OS, разработанной Virgil Dupras. Эта операционная система предназначена для работы на микропроцессорах

Z80 (речь, скорее, не о «Спектруме», а о более новых реализациях) и на AVR. Автор позиционирует её как ОС для постапокалиптического мира, где будет проблемой найти PC или детали для них, зато Z80 и различных микроконтроллеров (которые часто используются во встраиваемых решениях) должно хватать.

Официальный сайт проекта:
https://collapseos.org/

Военные США отказались от дискет

Командование ядерными силами США всё-таки отказалось от использования восьмидюймовых дискет, которые ранее применялись в системе SACCS (Strategic Automated Command and Control System), основанной на компьютерах IBM Series/1. Дискеты использовались для передачи приказов о запуске ядерных ракет. Теперь же они заменены на высокозащищённые твердотельные накопители.

Более подробно с новостью можно ознакомиться здесь.

Умер Дмитрий Александрович Поспелов

30 октября в Москве в возрасте 86 лет скончался учёный Дмитрий Александрович Поспелов. Он известен как основоположник советской школы искусственного интеллекта, а также как автор двух десятков монографий и нескольких сотен статей на тему нечётких вычислений, ИИ, экспертных систем и т.д. С 1968 по 1998 годы он работал заведующим отделом проблем искусственного интеллекта в Вычислительном центре РАН.

25 лет первому баннеру

27 октября 1994 года Wired Magazine разместил на своём сайте рекламные баннеры более десятка компаний, включая AT&T, MCI, Volvo и других.

Конечно, реклама в интернете была и до этого, но, можно сказать, это был первый «типичный» пример платной баннерной рекламы в виде картинок-гиперссылок.

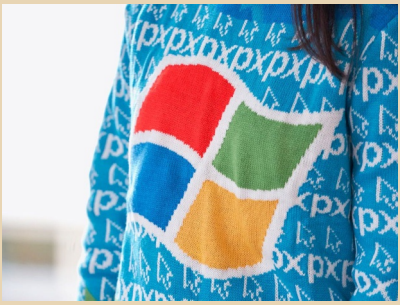
Подробнее здесь:
http://thefirstbannerad.com/

Дискета за $84000

Именно за такую сумму на RR Auction была продана 3.5”-дискета с автографом основателя Apple Стива Джобса.

Она содержит Macintosh System Tools 6.0 и датируется примерно 1988 годом. Такая высокая цена отчасти связана с тем, что Стив Джобс очень неохотно раздавал автографы.

Описание лота на RR Auction:
http://rrauction.com/PastAuctionItem/3420447



«Уродливые свитера» с Windows XP

В США есть традиция выпускать к Рождеству т.н. «уродливые свитера» (ugly sweaters). Название, видимо, связано с грубой вязкой и общим «неряшливым» стилем. В этом году компания Microsoft поддержала эту традицию, разослав своим сотрудникам и журналистам свитера в стиле Windows XP.

Неожиданно они стали популярными и компанию завалили вопросами «Где купить?»

Обзор подготовили:
Вячеслав Рытиков (eu6pc)
uav1606
Андрей Шаронов (Andrei88)
takkajuspe
RomFil

Развлечения на бытовом компьютере. Интервью с Романом Подковыровым и Дмитрием Барулиным

На рубеже 90-х годов наибольшей популярностью пользовался ZX Spectrum. 8-битные игровые приставки пока ещё не получили столь широкого распространения, а количество «домашних» пользователей IBM PC в масштабах страны стремилось к нулю, в основном из-за его неподъёмно высокой цены. Однако существовали в те времена и другие, менее известные платформы. Одна из которых – платформа, или даже можно сказать, целый мир компьютеров БК. Мир со своей архитектурой, программным обеспечением и талантами. Гости нашей сегодняшней рубрики – яркие представители этого мира. Мастера нулей и единиц, коллеги по игровому цеху, друзья, и просто хорошие ребята – Дмитрий Барулин (Дес) и Роман Подковыров (PC). Стараниями обоих Сергеевичей на свет появилось около десятка неплохих, для своего времени, игр.

Давайте познакомимся немного поближе – расскажите что-нибудь о себе: сколько вам лет, где родились и жили, учились – любую информацию, какой сочтёте нужным поделиться с нашими читателями.

РП: Родился, учился, пока не женился. Если серьёзно, родился и живу в Москве, окончил МГТУ им. Н.Э. Баумана, работал в основном инженером в разных сферах (электроника, ИТ, системы безопасности).

ДБ: Родился в Москве, сейчас живу также в Москве. Был замечательный период, когда я жил в Люберцах, остались очень яркие воспоминания. В данный момент всё хорошо. Женат, двое детей.

Москва, Васильевский спуск, «5 лет Поляроиду» (95-й или 96-й год, слева направо: Роман, Дмитрий)

Расскажите свою историю знакомства с БК. До него были другие компьютеры? Почему в конце концов решили остановиться на этой платформе?

РП: Купили мне в детстве, даже игрушку с мафона загрузить сначала не мог. Потом освоился, со временем писать/рисовать что-то начал. До него других не было. Видел где-то, у друзей или ещё где… Игрался иногда. ДВК, РК, «Спектрум», «Микроша», «Агат»… Их много было, всех не упомню. У самого потом до кучи были спек и УКНЦ.

ДБ: Компьютер БК-0010 был куплен в магазине «Электроника» в тот самый период, когда был на него самый ажиотажный спрос, мы стояли с родителями в очереди двое суток. Деньги были заработаны на бензоколонке мытьём машин.

Где и как вы научились программированию?

РП: Ну Бейсик, Фокал как-то сами попёрли, Паскалю в школе учили, потом на Delphi научился писать – для девушки лабы делать (она на программиста училась))). Ассемблер для БК тоже как-то само собой, книжечки, советы соратников.

ДБ: Программированием начал увлекаться с 4 класса, в кружок «Программирование и компьютерные игры» пошёл. Конкурс туда был огромный, но когда все узнали, что компьютерные игры – это только 5 минут в конце занятий, нас осталось человек 10. Мы тогда учились Бейсику на «Агате». Далее был ассемблер, на котором писал на БК. Ассемблер – только книжки, основная книжка была господина Зильберштейна, а вторую не помню, вроде штатный мануал, и плюс изучение кода других программистов.

Роман, а как, собственно, вы встретились с Дмитрием, и откуда возникла идея работать дуэтом?

РП: Встретились мы совершенно случайно. Меня привлёк номер телефона, который отличался от моего на одну цифру. Оказалось, что мы живём в одном подъезде. Писать вместе как-то само собой получилось. Дмитрий потом в Люберцы переехал, я к нему приезжал. Мы ж ещё музычку лабали. Так что в Люберцах я частенько бывал.

Расскажите подробнее свои мысли о платформе БК – какая у этого компьютера архитектура, какие были варианты исполнения, модели, что за периферия и модули расширения использовались. Какие бы плюсы и минусы, насколько компьютер был надёжен и так далее.

РП: Ну вопрос! Вспомнить бы. Какая архитектура? Проц, память, порты. Модели помню 0010, 0010-01, 11, 11М, периферия разная была. У меня был принтер, 2 дисковода, при желании джойстик, мышь, ковокс. Из расширений у меня стояла ПЗУшка с нортоном, потом поставил расширение памяти и грузил ДОС, какой хочу. Разгонял проц до 4.5 МГц, на 6 не тянул. Плюс – удобство, и для меня – доступность к ремонту, модернизации и пр. Минус – ресурсов всё же мало, хотя их никогда не бывает достаточно. Надёжность вполне устраивала. Вообще есть куча статей про историю развития БК.

ДБ: Архитектура PDP-11, которая в своё время была украдена вместе с технологией DEC Corporation, которая, к сожалению, не взлетела за границей, и там победила IBM. Соответственно, и у нас на базе этой архитектуры были построены машины ДВК-1, ДВК-2, УКНЦ. А потом всё благополучно умерло без поддержки. Ассемблеры IBM и «Спектрума» другие, сравнивать сложно. «Спектрум» восьмиразрядным был, а БК шестнадцатиразрядный, это плюс. А вот по объёму оперативки проигрывала – 16 против 48 КБ. Иногда на БК даже экранную память использовали под код.

Ещё у БК была очень интересная внешняя шина, через которую можно было подключать внешние блоки, самый знаменитый – это блок МСТД, в который был забит интерпретатор языка Фокал. Язык, который был создан только для БК, очень странный и нафиг никому не нужный.

Из периферии со временем появились и принтеры, и дисководы на 5-дюймовых дисках. В принципе, можно было много чего подключить, потому что железо это позволяло.

Viacheslav Slavinsky ©

Как по-вашему, кто был рядовым пользователем компьютеров БК? Если среди поклонников «Спектрума», а уж тем более РК86/«Ориона» было много радиолюбителей, то кто обычно использовал БК?

РП: Все, кто купил.)) Радиолюбителей много было, я например. Плюс умения держать паяльник или программировать в том, что другие пользователи давали возможность немного подзаработать.

ДБ: Да все те же самые. Техническая интеллигенция, инженеры и очень много молодёжи. Молодёжи, которая не увлеклась «Спектрумом».

Для спектрумистов, а уж тем более для пользователей IBM PC существовали сети на основе модемов – или нормальных аппаратных, или поддерживаемых программно, но были. А была ли подобная сеть или её аналог для компьютеров БК?

РП: На моей памяти не было. Может, где и были, но я не слышал. К сети подключился только когда купил первый пень (за бешеные деньги). То была FIDO. Ну и пара небольших сеточек.

ДБ: Насколько мне известно, сетей у нас не было, хотя технически это вполне было реализуемо, так что, возможно, под конец что-то и было.

Попадались сведения о подключении к БК музыкальных процессоров AY. Вы интересовались этим вопросом? Были ли эксперименты в данной области? Был ли у вас вообще интерес в области развития звуковых возможностей БК?

РП: AY, конечно, подключали. Повсеместно причём. У меня не было. Я ковоксом увлекался, музыку под него немного писал. А ещё собирал и продавал его, немножко этим зарабатывал. Но он не прижился из-за ограниченности ресурсов. AY – он же синтезатор, а Covox – банальный ЦАП, просто воспроизводящий WAV.

ДБ: Сопроцессор AY подключали, также под него даже был редактор, под него можно было писать. Ещё подключали Covox (ещё одна музыкальная приблуда наравне с AY, но дешевле и проще). Вообще музыки много существовало под них, одно время было модно писать демки, на которых под смену красивых картинок играла соперная музыка (сопер – сопроцессор). И был неплохой многоголосый музыкальный редактор для этого.

Москва, Парк Победы (примерно 98 год, слева направо: Дмитрий, Роман)

Поведайте о своём пути в игровой мир. (Как пришла идея взяться за разработку игр, какой была первая игра, много ли было затрачено времени, помогал ли вам кто-нибудь и т.д.)

РП: Идея вообще была изначально. Даже когда у меня не было БК, мы с приятелем писали какие-то игрульки на Бейсике спека. Ну а потом я занимался в основном художеством, ну и кодил по мелочи. Написал самостоятельно полноценную игрульку – аналог STONE AGE с PC. Даже редактор лабиринтов под неё. Но, увы, это уже никому было не нужно. Ну, может, некоторым фанатам БК. Короче, код потерялся, что печально.

ДБ: После покупки БК-0010 началась самая весёлая эпопея – это поиск новых игр на аудиокассетах. Это были обмены, переписывания, было очень круто, если у человека был двухкассетник, можно было переписать игру. После этого выяснилось, что, оказывается, пишут игры практически мои ровесники, и, естественно, захотелось делать то же самое. Первая моя игра была «Плевок», я её послал на конкурс фирмы «Снек». Это была единственная на тот момент фирма, которая распространяла игры для БК-0010. Я написал игру, в которой рандомно определялось, кто дальше плюнет, отправил кассету в «Снек», но ответа так и не получил. То ли что-то не дошло, то ли игру восприняли слишком несерьёзно и не взяли на конкурс. Но джинн был выпущен из бутылки, и меня было уже не остановить. В основном поначалу делал аналоги спектрумовских игр, так как, во-первых, оттуда можно было переть графику (были специальные программки), во-вторых, софта для «Спектрума» было в разы больше, и очень хотелось, чтобы понравившиеся игры были и на БК.

Роман, расскажите историю Atlantic Software. Насколько мне известно, до совместной работы с Дмитрием вы входили в её состав.

РП: «Атлантик» изначально был в составе меня, Сергея Кудина и Стаса Мекерина. С Серёгой мы сделали OVERFLY, а со Стасиком издавали газету МОРГ. CHAIN была написана как реклама OVERFLY. Ну, проги всякие по мелочи были. А потом всё как-то само по себе сошло на нет. Никто ни с кем не ссорился. До сих пор иногда общаемся.

Общий список игр под вашим авторством: BMX, OVERFLY, CHAIN, MPHREY, PIGAME, CORPSE, ERIC AND THE FLOATERS. Это полный перечень, или мы что-то упустили? Какую из созданных вами игр вы считаете лучшей и почему?

РП: Ох, много чего было. BMX чисто **Деса**, про OVERFLY и CHAIN см. выше. MPHREY – это извращение какой-то простенькой игрухи (не помню название), PIGAME – это уже мы с **Десом** вместе, на пределах возможности 11M, CORPSE тоже совместно, ERIC – **Дес** написал, а я потом просто графику переделал. Лучшая у ATLANTIC, конечно, OVERFLY, а у PIP (собственно BDS&PRS) – PIGAME. Лучшие потому, что вложено было всего по максимуму, ну и получилось замечательно, на мой взгляд. OVERFLY, например, распространялся со своим форматом дискеты, а про PIGAME я уже написал – использовали все возможности 11M. Да, пропустили, был целый «Пи-пакет» с нашими переделками других игр, тоже затерялся. Может, у кого и осталось что-то.

ДБ: О, MPHREY великая затея была. Я стырил картинки из игры Humphrey со «Спектрума», сконвертировал графику, сделал красивую заставку, с названием решил не заморачиваться и просто убрал две первые буквы. А в качестве самой игры воткнул примитивнейшую игру Gapper с БК, где по прямоугольной сетке ездит точка, а за ней гоняется крестик.

Также поправлю **РС**-а – Pigame писалась всё-таки для БК-0010-01.

Ну и «Пи-пакет», на мой взгляд, был шедевр. Мы с **РС**-ом взламывали чужие игры, ставили, обычно, вечную жизнь, потом по полной меняли графику и, по возможности, сюжет и описание. Главным героем чаще всего становился отважный х**. Было очень весело. По крайней мере, нам.

OVERFLY

Расскажите об игре WOLLY COMES BACK. Почему она так и не увидела свет? Может быть, у вас остались незаконченными ещё какие-то игровые проекты или нереализованные идеи?

РП: WOLLY как-то на корню растаяла, я даже и не помню об этом. Про STONE AGE опять же написал выше.

ДБ: WOLLY COMES BACK – очередная попытка передрать на БК понравившуюся на «Спектруме» игру. Графику и картинки перекачал, красивую демку рекламную сделал. Ну а потом как-то остыл к этой игре, и продолжать не стал.

Расскажите, пожалуйста, более подробно о процессе создания игры на БК. Насколько это был трудоёмкий процесс, с какими трудностями сталкивались, какие программные средства использовались и т.д. Тогда уже было разделение ролей на программиста, художника, композитора и т.д., или каждый был скорее «мастером на все руки»?

РП: Процесс трудоёмкий настолько, насколько любое программирование. Только мы писали на ассемблере (читай мнемоника машинных кодов) – это, конечно, сложнее, чем, например, Бейсик, Паскаль, Си и прочее. Вместо банальной команды **PRINT** была целая подпрограмма + шрифт (ну не прикольно же стандартным писать). Но это было обосновано ограничением ресурсов, как по памяти, так и по скорости. Сейчас никто не заморачивается ресурсами, компы супермощные, память, скорость, видео… Ассемблер остался только для микроконтроллеров, и то, думаю, ненадолго.

ДБ: Со временем на БК появились довольно удобные средства для программирования на ассемблере, с удобной отладкой и прочими фишками. Но, к сожалению, сейчас уже не вспомню название и авторов.

А так – да, в 16 килобайт запихать и код, и графику, и музыку, на мой взгляд, это было круто.

Особо разделений обязанностей поначалу не было, каждый был сам себе «корпорацией», но, конечно, если есть человек, который рисует или пишет музыку лучше тебя, почему бы не воспользоваться его помощью.

Ощущалась ли в «то время» конкуренция со стороны ZX Spectrum, PC, других платформ?

РП: В то время разве что спек.

ДБ: PC – это было что-то недоступное и жутко дорогое. «Спектрум» – да, игр больше в разы, памяти 48 КБ против 16 КБ у БК. Графика хуже. Вообще, поклонников «Спектрума» намного больше было, но это были в основном геймеры, очень мало кто развивался дальше Бейсика.

Если не секрет, скажите, разработка игр принесла вам какую-то прибыль, или же это было просто хобби?

РП: Хобби конечно, но немного (!) подзаработали. Как сказал Михаил Королёв – мы продали МК-ДОС и пошли в китайский ресторан, а мы в ответ – продали CORPSE (если не ошибаюсь) и пошли в пельменную.

ДБ: Пытались, конечно, что-то зарабатывать, и игры в компьютерные центры относили, но, как было выше сказано, процентов за продажу игры хватило один раз сходить в пельменную.

В эпоху расцвета БК вы общались с другими разработчиками игр? Например, с Бортником, Савиным, Надёжиным? Если да – то какими были ваши отношения?

РП: Так-то общались, но вот с Бортником я только по телефону пару раз говорил, с Савиным вообще нет. А с Надёжиным пересекались периодически (он вроде не писал игр, или тут я что-то упустил). Я у него, собственно, тот модуль расширения памяти и купил. Да и вообще мы рядом жили и клуб БК посещали.

ДБ: Бортник был легендой, которую мало кто видел. Савин – это вообще какой-то динозавр, когда мы начали писать на БК, он, кажется, уже закончил.

А вот г-н Надёжин и сейчас прекрасно себя чувствует и ведёт довольно популярный блог в ЖЖ: https://ammo1.livejournal.com/

Ещё я горжусь знакомством с **RDC**, тоже мощный парень был в программировании.

Как вообще в те времена общались фанаты платформы БК? Были какие-то официальные или неофициальные встречи, фестивали, конференции?

РП: Вот «Клуб любителей компьютера БК» был. Там много кого приходило. Все приносили какие-то свои файлы, сдавали дискеты, потом на них вся эта куча записывалась и раздавалась. Главный там у нас был незабвенный Пал BINарыч, и вместе с ним Стас, который и копировал все эти дискеты.

Насколько я знаю, в своё время вы выпускали электронную газету «BDS & PRS News». Если не ошибаюсь, всего вышло 13 выпусков. Расскажите об этом проекте подробнее?

РП: Было дело. Сколько выпусков не припомню, примерно так. В основном это был стёб и наш специфический юмор.

ДБ: Но был один уникальный выпуск, которым я до сих пор горжусь. BPN 5. Он весь написан на тарабарском языке, но если вчитываться, то можно проследить отголоски смысла.

Кракокоп андхиру з узмани залиссин глоду изнода блы. Изгдрштна Злодогрен Некамуц атрон. Ы сми зярли Япсорд. Гздо. Ырмедан ко мизасиролт.

Наш Девиз.

Волшебные строки.

Какие ресурсы (сайты, форумы) на тему БК вы посещаете?

РП: Сейчас никакие. Был BKDEFAULT. Пару раз заходил.

ДБ: Ещё такая штука есть, вроде даже живая до сих пор: https://bk0010.org/forum/

Существует довольно много эмуляторов БК. Какой (или какие) вы бы посоветовали нашим читателям?

РП: Я в них сам никак не разберусь. Куча валяется, а толком ни один так и не пользую.

ДБ: Эмулятор Камнева (https://bk0010.org/), говорят, хороший. Ну и под «Андроид» народ что-то мутит, с открытым кодом.

На какие годы, по вашему мнению, пришёлся пик популярности БК? Когда и почему эта популярность начала снижаться? Как с этой платформой обстоит дело сейчас – для неё по-прежнему кто-то пишет игры?

РП: Ну, наверное, вторая половина 90-х. Прогресс не стоит на месте. БК не актуален. Все писюками обзавелись давно. Не думаю, что кто-то пишет.

ДБ: БК добила постоянно снижающаяся цена и увеличивающаяся доступность IBM PC. И если раньше «писюки» были недоступными простому смертному рабочими инструментами, то со временем они заняли нишу «домашних ПК». БК же остановилась в развитии на модели 11М, которая, несмотря на неплохие на тот момент характеристики (128 КБ ОЗУ), уже «не взлетела» серьёзно.

Известны ли вам разработчики прикладного софта для БК – в частности, офисных программ, таких как редакторы текстов, таблиц и, что ещё более интересно, электронных словарей и коммуникационного софта?

РП: Ну конечно, только смысла их перечислять не вижу. Вдруг кого забуду. Все в одном котле варились.

ДБ: Текстовый редактор «Vortex!» был очень неплох. Авторы ИКЦ «ИНКОМСЕРВИС» (с) Романов Д.А. & Страхов А.Ю.

Электронный словарь только от «ПИП корп» могу посоветовать. Была такая игра «Поле чудес». И там был редактор словаря, для создания списка слов, которые будут случайно выбираться для отгадывания. Корпорация PIP провела титаническую работу и создала словарь из нескольких сотен матерных слов. :)

Существовала ли на БК своя демосцена? Если да, то, может быть, вспомните самые интересные демо, их авторов?

РП: Была. Даже как-то слёт организовали. Я, правда, не попал, но демки мне потом скинули, очень красиво. Фишка в том была, чтобы выжать это из возможных ресурсов. Авторов не помню.

А были на БК всенародно любимые игры?

РП: Всенародные не знаю. Вот у Бортника пара игрушек неплохих была – Sliris и Toorun. Потом несколько вариантов кладов... Мы одно время с приятелем любили проходить Fantastic (это Lode Runner, только для двоих игроков одновременно).

SLIRIS

Пробовали ли Вы себя в разработке игр для других платформ, например УКНЦ? Если да, то как-то влияла разница в архитектуре и наличии/различии операционных систем?

РП: Нет. УКНЦ точно нет, поиграл только немного. Проги некоторые под PC писал, небольшие.

ДБ: На закате эры БК я купил себе УКНЦ. Там было 2 прикольных игрушки. Больше ничего интересного не было. Писать под УКНЦ, видя, как умирает платформа, уже не хотелось. Операционка там на базе RT-11 была, интересная, но уже тоже никому не нужная.

Оглядываясь в прошлое, рады ли вы сейчас, что в начале 90-х были верны БК, а не пересели за IBM PC-совместимый компьютер?

РП: Замечательное время было. Даже вообще, а не только из-за БК. А PC мне бюджет не позволял. Он у меня только в начале 2000-х появился.

ДБ: Думаю, да, рад. Так досконально, как БК, PC я бы, наверное, не изучил. А может, и изучил бы. И был бы сейчас суровым бородатым системным программистом в свитере.

Скажите, пожалуйста, сохранились ли исходные коды каких-нибудь из ваших игр? Есть ли они в публичном доступе?

РП: Исходных кодов нет и быть не могло. Мы писали непосредственно в отладчике (то бишь в памяти прямо). Удобно. Не надо ничего компилировать. Запустил, посмотрел, если надо – подправил.

ДБ: Да, код любой игрушки на ассемблере можно посмотреть в отладчике. Запускаем эмулятор БК, отладчик и ковыряем.

Случались ли в вашей деятельности какие-нибудь забавные случаи, курьёзы? Может быть, что-то запомнилось?

РП: Однажды я к **Десу** спустился чего-то припаять, так он, не знаю зачем, схватил паяльник за жало. Обошлось без травм, но ржали долго. Второй случай с паяльником: приехал я как-то к товарищу материнку писюшную пропаять. Паяльник в пакет, а чтобы не проткнул, натянул на жало пробку винную. Так товарищ решил мне помочь, снял пробку вместе с жалом и обмоткой… Удалось выполнить пайку соседским паяльником. Однако приятель решил запустить машинку без кулера, и проц, вздохнув синим пламенем, помер. AMD всё же…

ДБ: Писал Corpse, был глюк, когда выходили 2 скелета одновременно, спрайты накладывались, и получалось, что они сами себя загораживают и мигают. Влом было исправлять, написал в правилах: «Бойтесь мерцающих скелетов, они отнимают в два раза больше энергии».

Чем вы занимаетесь в настоящий момент?

РП: Инженер, монтажник слаботочки… Ну и всяко разно по мелочи.

ДБ: Руководитель фирмы «1С:Франчайзинг». Так что программирование и IT-направленность остались на всю жизнь. Хотя сейчас именно кодингом практически не занимаюсь.

Петрозаводск (примерно 2010 г., слева направо: Дмитрий, Роман)

Испытываете ностальгию по старым временам?

РП: Скорее да, чем нет. Я уже написал про то, что время замечательное было. И потом, это ж наша молодость, мы ж не только БК занимались. Хотя, ностальгии как таковой нет, но времена вспоминаются с самыми приятными ощущениями.

ДБ: О да, мы были молодыми безбашенными отморозками. Всё было по кайфу, писалось легко и совсем не ради денег, а из-за куража и желания самому себе доказать, что можешь.

Что ж, нам остаётся только поблагодарить ребят за интересную беседу и пожелать им удачи и дальнейших творческих успехов.

На фото в заголовке: на концерте NAPALM DEATH (2019 год), слева направо: Дмитрий, Роман – прим. ред.

Дмитрий Барулин (Дес)
Роман Подковыров (PC)

Интервью брал:
Вячеслав Рытиков (eu6pc)

Помогали с вопросами:
Андрей Шаронов (Andrei88)
uav1606

«Баллада» о хакере

Переемены в обществе, экономике, жизни страны, случившиеся после перестройки и развала Советского Союза, если не поставили всё с ног на голову, то поменяли жизни многих людей. Практически всем пришлось перестраиваться и подстраиваться под изменившееся бытие. Так сказать, адаптироваться к новой реальности. Многие достаточно толковые люди оказались лишены возможности зарабатывать средства к существованию. По крайней мере, законной возможности. Но талант остаётся талантом в любой ситуации и порой может найти выход, даже из такой ситуации. Тем более, что найдутся люди, готовые воспользоваться умениями такого человека – не всегда в мирных целях.

Примерно такую картину рисует нам первый сезон сериала «Агент национальной безопасности», где главными героями становятся не милиционеры, не бандиты, а гебисты – нижнее звено питерского отделения ФСБ – Алексей Николаев и Андрей Краснов, которым под мудрым руководством вышестоящего начальства приходится вертеться, аки уж на сковородке. И мЫшленье новое демонстрировать, и следить, чтоб страну по винтику не растащили да не продали по сходной цене за кордон.

Противостоят же «пехоте» плаща и кинжала то национал-социалисты, экстремисты – особенно, когда нюхнут как следует – сразу противостоять и начинают, набирающая силы «русская мафия», которая как раз и пытается применить народных самородков для своих не слишком мирных целей. Среди самородков мы можем увидеть и химика, создавшего новый наркотик, не имеющий мировых аналогов, девушку-гипнотизёршу, владеющую и телекинезом до кучи, и Петю... Вот о Пете мы и поговорим подробней.

Рис. 1. Собственно, Петя. Прошу любить и жаловать

Как догадался читатель, Петя у нас – хакер. Посмотреть на него можно в серии «Петя и Вол». Как Пётр столкнулся с компьютером – история умалчивает – лет ему, где-то в 98-м, в районе 25-30, так что – может быть, повезло с информатикой – в школе, институте или дворце пионеров, где привили парню интерес к программированию, может быть, как Лёха Кот из «ДМБ 2010», Петя тоже оказался «силён в математике». Но, так или иначе, получилось, что к 97-му году многое умел, многое знал. И тут на него обратили внимание ловкие ребята, которые придумали нехитрый трюк: Петя удалённо подключается к базе данных финского банка и переводит на нужный счёт деньги; счастливый владелец счёта тут же приходит и забирает наличность. Но в предпоследний раз Пётр облажался или же финские ребята не сплоховали, в общем, к моменту начала событий фильма они были уже во всеоружии – подозрительного клиента, который, к слову, даже не представляет, какая у него красивая и круглая цифра находится на счёте, ждали. Да и сами бандюганы предпочли избавиться от человечка, который снимал деньги.

В общем, сработали грязно. Повязали ли кого-то кроме улетевшего уже в Лондон Пети, неизвестно. Но главного героя всей эпопеи

потребовали назад на родину, чему зарубежные коллеги препятствовать не стали.

Тут у товарищей из госбезопасности созрел план – не пытать, не бить, а инсценировать побег под руководством «матёрого уголовника», которому несчастный хакер и расскажет секреты – не где деньги лежат, а сдаст своих шефов. А то и вообще получится арестовать. «Матёрым уголовником» Волом назначают Алексея Николаева – самое нижнее звено.

Рис. 2. Алексей Николаев в образе Вола

Ну и, как полагается подобному фильму, не без форс-мажора, накладок и косяков. Алексею предстоит поездка в Петрозаводск – долететь до Питера помешала непогода, драка в камере с участием «страхующих» ребят из местного отделения ФСБ, «грабёж» сберкассы. Ну и, конечно же, разработанный план побега сорвался – как иначе-то. ☺ Ну и, конечно же, злые бандиты прекрасно понимают, что Пётр их сдаст – добровольно или добровольно-принудительно, потому пытаются убрать опасного соучастника. Собственно, драка в камере и была первой попыткой убийства. Второй раз попытаются убить Петю в Питере, но подстрелят Николаева...

«Спойлеришь, автор!» – скажет читатель. Но перескажи я весь фильм, всё равно интересно посмотреть – хотя бы один раз.

Впрочем, дело закончилось относительно благополучно. Петя сдал своих нанимателей и, хоть и отсидел, но хотя бы получил возможность делиться опытом в роли преподавателя компьютерных курсов.

Рис. 3. Не слишком благополучно, но всё же – и Петя лес не валит, и отечеству польза – чем не хеппи-энд?

Пока не пересмотрел эту серию года четыре назад, не думал, что она так отпечаталась в памяти. Не события, не монолог Пети о работе за 300 руб. А сам образ программиста – этакого Левши-самородка, который живёт либо грабежом банков, либо ломкой софта по заказам составителей дисков-сборников. И максимум, что толковое делает – это правит в HEX-редакторе файлы зарубежных программ, чтоб интерфейс Нортона или винды (пусть даже взломанной беты) был на русском языке. Понятно, что сие – сложная и, порой, неблагодарная работа. И с каким же энтузиазмом ваш покорный слуга вцепился в отечественные программы, когда узнал об их существовании. Когда узнал, что есть-таки проги и для DOS, и для современной оболочки, а потом и операционки – винды. Что программы-то и переводють с иностранных языков, и буковки да слова распознают с отсканированных листов, да и ворд подковывают, аки аглицкую блоху, чтоб не только слова русские распознавал, да ещё и в разных падежах, склонениях да спряжениях. Конечно, не так их сильно много, но даже версии «Русского офиса» и отдельных, входящих в него программ до сих пор не собрали, а есть ещё и другие программы – и для третьей винды, и, тем более, для девяносто пятой и девяносто восьмой... И, надеюсь, рано или поздно любой желающий может запросто скачать любую такую понравившуюся программу с Old-DOS. Хоть «Дела в порядке», хоть полную версию четвёртой «Прописи» на шести дискетах, хоть «Компас» для третьей винды.

Андрей Шаронов (Andrei88)

Информационные технологии в СССР

История техники для меня – это захватывающее приключение. Есть в ней свои герои, свои злодеи, есть материальный мир, артефакты погибших реальностей и люди, как будто занесённые из другого мира. Мира «Полудня» Стругацких и «Туманности Андромеды» Ефремова.

Рекомендую прочитать книгу Ю.В. Ревича и Б. Н. Малиновского «Информационные технологии в СССР. Создатели советской вычислительной техники».

Это попытка рассказать о победах и поражениях нашей страны.

Об ошибочных решениях государственных мужей и о людях «Понедельника, который начинался в субботу», творивших в своих лабораториях и цехах иные Миры.

На протяжении книги буквально через 5-10 страниц мы читаем о том, что было сделано впервые.

Впервые в СССР, впервые в мире. Эта книга Гимн Создателям. Они стояли на пороге нового.

Они создавали новые оригинальные концепции электронных машин и сетей.

Например, я с удивлением узнал о том, что система противоракетной обороны (ПРО), позволявшая сбивать вражеские ракеты, была в испытана 4 марта 1961 г. За 23 года до аналогичной американской системы! А первые специализированные компьютеры, способные работать в беспроводных сетях, были созданы в 1958 году! Расстояние между компьютерами могло быть до 200 км.

За десятилетие до первых ПРОВОДНЫХ сетей в США, мы могли передавать и обрабатывать базы данных со станций ПРО по всей стране и сбивать противоракетами летящие на сверхзвуке цели.

Следующим шагом должна была стать унификация компьютеров и создание единой системы государственных вычислительных центров, предложенных Анатолием Ивановичем Китовым. Они должны были решать задачи планирования и расчётов между предприятиями.

Но здесь сработали единым фронтом бюрократы в погонах и без них.

Мы были в коридоре иных Реальностей. И мы открыли не ту дверь…

Но даже там, в следующем коридоре, мы видели десятки других дверей, в которых мы могли увидеть другой мир.

Интересный факт, что при решении, копировать или не копировать технику США и искать альтернативу совместными усилиями с европейцами, ВСЕ создатели компьютерной техники СССР выступили категорически против копирования. Вместе с ними против были компьютерщики Венгрии, Чехословакии, Болгарии, Польши, Румынии. Им противостояли только ГДР и московские чиновники….

Много ещё можно писать о сделанном и несделанном, о том, как поколение покорителей космоса опережало время на десятилетия (а, возможно, и на столетия), но я всё же рекомендую купить эту книгу тем, кому это интересно. Количество 1000 экземпляров невелико.

Они СМОГЛИ. Пусть это будет нам примером и укором.

Это было «Время Первых» в создании ЭВМ, и наша беда, что мы мало знаем о них…

С.А. Лебедев. Создатель МЭСМ и БЭСМ (малой и, соответственно, большой счётной машины), первых промышленно выпускаемых компьютеров нашей Родины.

При его участии был разработан первый процессор «Эльбрус», опередивший производимые в США аналоги на 15 лет.

Система противоракетной обороны 1961 года, построенная на компьютерах «Диана-1» и «Диана-2», опередила США на 23 года.

И.С. Брук. Создатель компьютеров М-1, М-2, М-3, разработчик концепции использования ЭВМ в народном хозяйстве СССР.

Б.А. Рамеев. Один из разработчиков первой серийной ЭВМ «Стрела», главный конструктор суперкомпьютеров «Урал-1» – «Урал-16», работавших по принципу обратной совместимости программ, работы в сетях, концепции наращивания мощности, говоря сегодняшним птичьим языком – апгрейда.

В.М. Глушков. О нём я писал раньше. Настоящий компьютерный титан эпохи СССР, чьи идеи только начинают реализовываться. Напишу здесь только о самых главных.

Теория автоматов.

Анализ и синтез параллельных программ.

Советские системы автоматического проектирования. «ПРОЕКТ».

Компьютер «МИР» – первый советский ПК.

Общегосударственная автоматизированная система управления. ОГАС.

Работы по распознаванию речи и текста.

Первые работы по разработке Искусственного Интеллекта.

Основы безбумажной информатики – концепция того, что мы назвали всемирной Сетью.

И многое, многое другое. Слишком многогранен этот человек.

А.И. Китов. Пионер в разработке концепции использования ЭВМ в народном хозяйстве.

Автор первого массового учебника по программированию.

Автор внедрения АСУ в медицине.

Человек, боровшийся за ОГАС (после смерти В.М. Глушкова) до катастрофы 1991 года.

М.А. Карцев. Автор ЭВМ М-4 – первого полупроводникового компьютера СССР (1962 год).

Из всех машин, работавших в СССР, именно она оказалась наиболее высокотехнологичной и требовала минимальной настройки.

На ЭВМ М-10 была построена и отработана система ПРО СССР. Она соединяла в единую сеть компьютеры с радаров, которые засекали ракетные пуски за рубежом.

Первые работы в СССР по оптоэлектронике.

Н.П. Брусенцов. Создатель уникального компьютера «Сетунь», использовавшего троичную логику (1, 0, -1) – 1959 год. Работала в МГУ.

Ей принадлежит рекорд безотказной работы в СССР – 17 лет! Отличалась лёгкостью в программировании.

RISC-процессоры 21 века управляются 150 командами, а «Сетунь» управлялась 24!

Машина «Сетунь 70» до сих пор работает в Московском университете.

Б.Н. Малиновский. Разработчик ЭВМ «Днепр» под руководством В.М. Глушкова.

Первая машина в СССР, автоматически управляющая технологическими процессами на предприятиях народного хозяйства.

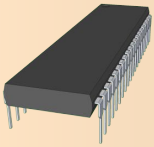
Разработчик ЭВМ 50-х годов для наведения на цель самолётов, системы «СВОЙ-ЧУЖОЙ».

Красной нитью через всю книгу идёт от личности к личности главная мысль. Эти люди были творцами. Как были творцами те, о ком мы не знаем. Кто работал рядом с ними. Теми людьми «Понедельника, начинавшегося в субботу». Ведь главная задача человека и его истинное счастье – это ТВОРЧЕСТВО.

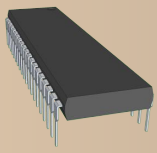
История не знает сослагательных наклонений.

Но надо помнить о победах и ошибках.

Олег Павлов



КРАТКАЯ ИСТОРИЯ МИКРОПРОЦЕССОРОВ. ЧАСТЬ ПЕРВАЯ



Развитие компьютеров, сопровождающееся миниатюризацией, привело к тому, что в 70-е годы стало возможным поместить центральную часть компьютера, или центральный процессор (арифметико-логическое устройство, устройство управления, регистровая память), внутри одной или нескольких микросхем. Так появились микропроцессоры – основа для создания микрокомпьютеров, что в дальнейшем привело к появлению персональных компьютеров.

Существуют, конечно, некоторые споры о терминах. Например, что считать микропроцессором (одну микросхему или связанный набор микросхем), микрокомпьютером, персональным компьютером. Но в целом, терминология сложилась: микропроцессор – одна или несколько микросхем, выполняющих функции центрального процессора; микрокомпьютер – компьютер, основанный на микропроцессоре; персональный компьютер – в целом, синоним микрокомпьютера, за исключением случаев, когда микрокомпьютер не предназначен для работы с ним одного пользователя, а, например, выполняет функции промышленного контроллера или сервера.

Приоритет в области создания микропроцессоров, очевидно, принадлежит американской фирме Intel, выпустившей в конце 1971 года микросхему Intel 4004 для семейства... микрокалькуляторов. Это был простой 4-разрядный микропроцессор, помещавшийся в 16-ногом корпусе DIP (в аналогичных корпусах делали, к слову, гораздо более простые микросхемы цифровой логики, например К155ИЕ7 – 4-разрядный счётчик). До применения i4004 микрокалькуляторы основывались на специализированных сложных логических микросхемах, но Intel было некогда (или лень) разрабатывать несколько таких разных микросхем, и они предложили универсальное решение, ведь микропроцессор можно использовать в разных устройствах, нужно лишь заменить его программу. Так лень в очередной раз стала двигателем прогресса :-).

Процессор i4004 в настоящее время редкость. Встретить его в России сложно, поскольку в 70...80-е годы он в СССР не поставлялся, как и иная высокотехнологичная техника (действовал запрет западной организации по экспортному контролю КОКОМ), а в 90-е, когда импортные микропроцессоры в Россию стали широко ввозить, 4004-й уже безнадёжно устарел и стал мало кому интересен, кроме тогдашних downgrade-ров. Так что, если вам попадётся микросхема в золочёном керамическом корпусе с надписью C4004 – радуйтесь: она стоит сотни долларов.

Intel 4004 был изготовлен по технологии 10 мкм или 0.01 мм (словами «технология» или «техпроцесс» обозначают расстояние между проводниками и размеры элементов на кристалле микросхемы). Для сравнения: печатные платы 7 класса точности в соответствии с ГОСТ Р 53429-2009 имеют печатные дорожки шириной 0.05 мм (таково может быть и расстояние между ними), а ошибка позиционирования дорожек не должна превышать 0.005 мм. То есть плотность монтажа на современной печатной плате лишь в 25 раз меньше, чем на кристалле первого микропроцессора! Площадь кристалла i4004 равна 12 мм^2. Значит, теоретически можно собрать из дискретных транзисторов модель (или реплику) 4004-го на печатной плате 7 класса точности площадью около 300 мм^2, например, размерами 2x1.5 см. На самом деле, плата понадобится более крупная, поскольку smd-транзисторы имеют корпуса размерами

значительно больше, чем 0.05x0.05 мм. Но если взять бескорпусные транзисторы…

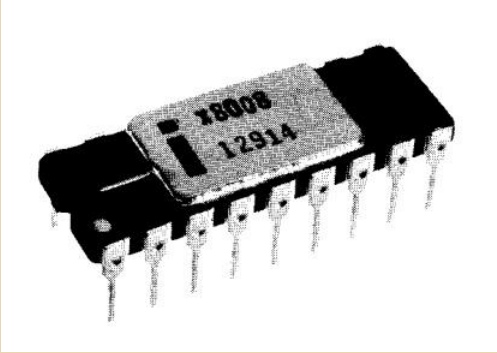
Кстати, о транзисторах. Схема Intel 4004 насчитывает «всего» 2300 этих деталей. Для сравнения: современный Intel Core i7 в МИЛЛИОН раз сложнее – примерно 2.6 млрд транзисторов. Тактовая частота i4004 – меньше 1 МГц, что примерно в 5 ТЫСЯЧ РАЗ меньше, чем у современного процессора. Производительность у i4004 примерно такая же, как у знаменитого лампового компьютера 50-х годов ENIAC.

https://ru.bmstu.wiki/Intel_4004

Да, с точки зрения компьютерщика 50-х годов 4004-й был вершиной прогресса, техникой будущего. С позиций же современного компьютерщика, избалованного многоядерными настольными «суперкомпьютерами», 4004-й имеет очень скромные возможности. Пожалуй, только в инженерном калькуляторе он сейчас и смог бы работать. Либо – мигать светодиодами в ёлочной гирлянде, подобно современному простейшему микроконтроллеру. В персональных компьютерах, кстати, i4004 так и не использовался, поскольку в начале 70-х «персоналок» ещё не существовало, а появились они чуть позже – когда возникли 8-битные микропроцессоры, имеющие бОльшие возможности.

Кстати, учёные Ральф Меркл и Роберт Фреитас создали «реплику» i4004 на базе МЭМС, то есть микроэлектромеханических наноустройств. По сути, вместо транзисторов поставили управляемые нанопереключатели (нанореле), и получили устройство, почти столь же компактное, как и оригинальный микропроцессор, но гораздо более экономичное, ведь потери энергии в МЭМС-ключе намного меньше, чем в транзисторе (ссылка).

Вслед за 4004, в 1972 году инженеры «Интел» создали 8-разрядный Intel 8008 (сейчас он тоже редок). 8008 также размещался в корпусе с маленьким количеством ножек (18), что заставляло использовать сложную логику для совместного использования одних и тех же выводов и для передачи данных, и для адресации памяти. Поэтому 8008-й большого распространения также не получил.



Но когда 8-битный процессор поместили в корпус с 40 ножками и оснастили независимыми параллельными шинами адресов и данных – тогда пошёл процесс быстрого внедрения микропроцессорной техники. Новую версию микропроцессора, появившуюся в 1974 году, назвали Intel 8080. Она уже получила широкое распространение не только в США, но и в СССР: туда обходными путями ввезли «образцы» 8080-го, под микроскопами изучили, как устроен его кристалл (провели «реверс-инжиниринг»), и года через три сделали отечественный «клон» КР580ИК80. Более поздние версии этой микросхемы назывались КР580ВМ80А.

Любопытно, что уже в наши дни энтузиасты провели «реверс-инжиниринг реверс-инжиниринга», то есть, восстановили полную схему микропроцессора КР580ВМ80А! Так что любой желающий теперь может вооружиться паяльником, большой макетной платой, 4758 транзисторами, кофе и хорошим настроением,

и спаять свою «реплику» КР580-го. Непонятно только, зачем. Впрочем, это повод, чтобы досконально изучить архитектуру микропроцессоров на сравнительно простом примере.

https://habr.com/ru/post/249613/

Конечно, в то время многие делали свои 8-битные микропроцессоры. Motorola выпустила свой процессор Motorola 6800 почти одновременно с Intel 8080. Чуть позже подтянулись японская Hitachi (HD6301), Epson (C63010), MOS Technology (6502), Zilog (Z80), STMicroelectronics и даже советский «Квазар» (вышеупомянутый КР580). Помимо собственно персональных компьютеров, 8-битные процессоры применялись в промышленных контроллерах АСУТП, контроллерах станков с ЧПУ, роботах, бортовых компьютерах, вычислительных блоках военной и ракетной техники, в автоматических измерительных приборах, в первых банкоматах, игровых автоматах, игровых приставках и много где ещё.

Возьмём, например, MOS 6502. Это очень простой и понятный микропроцессор. Он использовался в компьютерах Atari, Apple I/II, Commodore PET, игровых приставках Nintendo, и даже в советских ПК «Агат». Также 6502-м был, якобы, оснащён Терминатор T-800 :-) Возможности 6502, если судить по играм приставки «Денди», не такие уж скромные, и в плане компьютерной графики – вполне сопоставимые даже с ранними 16-разрядными персональными компьютерами типа IBM PC XT.

Благодаря своей простоте (чуть больше 4 тысяч транзисторов) группа «гиков»-энтузиастов создали свою реплику 6502 из дискретных SMD-компонентов на печатной плате размерами 300x375 мм! Это изделие было названо MOnSter 6502: https://monster6502.com

Схема MOnSter несколько отличается от схемы оригинала. Так, вместо некоторых транзисторов MOSFET были использованы резисторы, кроме того, «для красоты» были добавлены светодиоды, позволяющие визуализировать состояние регистров и работу процессора в целом (отличный способ пробудить у школьников или студентов интерес к архитектуре компьютеров!) Плата «МОнСтра» снабжена кабелем с 40-контактным разъёмом, который может вставляться в «кроватку», предназначенную для установки микросхемы настоящего MOS 6502. То есть «МОнСтр» ограниченно взаимозаменяем со своим оригиналом. За одним исключением: его максимальная тактовая частота всего 50 кГц – в 20 раз меньше, чем у микропроцессора 6502. Это вызвано большими размерами платы «МОнСтра» по сравнению с кристаллом, что обуславливает бОльшие временные задержки сигналов.

Примечательно, что у разработчиков изделия спрашивали, не хотят ли они сделать на печатной плате «реплику» современного микропроцессора. Их ответ: нет, не хотим, потому что это невозможно. Схема современного микропроцессора так сложна, что вам понадобится колоссальная плата площадью в несколько футбольных полей, чтобы разместить на ней все дискретные элементы. Это наглядно показывает, насколько чудовищно сложными стали современные компьютеры, какой огромный путь был пройден за 40 лет их развития...

MOnSter – не единственная попытка создать самодельный компьютер на дискретных

элементах. Есть ещё, например, 16-разрядный Megaprocessor, весящий более 500 кг:

http://www.megaprocessor.com/

Программно «Мегапроцессор», по-видимому, ни с чем не совместим. Его создатель Джеймс Ньюман построил его «потому, что захотел этого». Но, вообще говоря, ему просто хотелось забраться внутрь процессора и в деталях разобраться во всём, что там происходит. А с микропроцессором этого сделать невозможно.

Более приемлемый для дома вариант «самодельного процессора» собрал Пауло Константино. Собрал на белых макетных платах с гнёздами (так называемых «breadboard») общей площадью около 1 м², соединяя детали проводками с перемычками. Внешне это устройство выглядит, как клубок разноцветных перепутанных ниток. Однако, если верить его создателю, оно работает, как 8-битный микропроцессор Intel 8085!

https://habr.com/ru/post/410095/

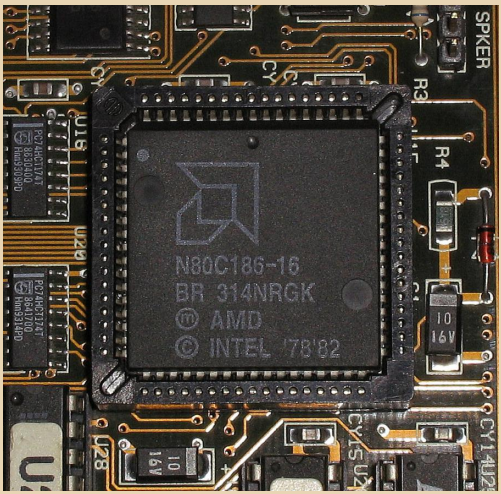
Но вернёмся к нашим старым добрым микропроцессорам 70-х годов. Всего через 4 года после создания i8080 «ударники каптруда» из Intel сделали рывок вперёд, и удвоили разрядность микропроцессора. Так появился 16-разрядный Intel 8086, которому было суждено стать первенцем в династии процессоров «x86», на котором основывалась династия «IBM PC-совместимых» компьютеров. Да, в это трудно поверить, но даже многие процессоры, выпускавшиеся в 21 веке, например, Intel Pentium IV, были программно вполне совместимы со своим «прадедушкой», появившемся в далёком 1978 году. И тот софт, который предназначался для первых IBM PC, мог безо всяких эмуляторов – хотя и не всегда – запускаться на компьютерах, выпущенных 30 годами позже.

Intel 8086 по своим возможностям намного опережал i8080, но и был намного сложнее (содержал в 5 раз больше транзисторов). Возможность непосредственной адресации 1 МБ памяти, дополнительные команды, возможность выполнять операции с числами бОльшей разрядности, более высокая тактовая частота… Кроме того, появился дополнительный математический сопроцессор Intel 8087, быстро выполняющий сложные математические операции, например вычисление синуса угла… Всё это позволяло использовать 8086-й не только для простых игрушек, но и для сложных научных и инженерных расчётов, математического моделирования, обработки и хранения баз данных… Чуть позже была создана модель i8088, отличающаяся от i8086 «урезанной» шиной данных – 8, а не 16 разрядов.

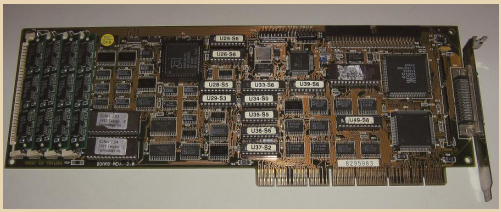
Именно на 8088-м был основан первый компьютер семейства IBM PC. Он получил огромную популярность, вытеснив основную массу прочих ПК «традиционного» вида. Причина этого – в модульности и открытости архитектуры IBM PC. В отличие от прежних 8-битных ПК, которые обычно представляли собой массивную клавиатуру с процессорной платой внутри, подключавшейся к телевизору и накопителям (магнитофону, дисководу), IBM PC основывался на системном блоке – этаком крейте, в стандартные разъёмы которого можно вставлять дополнительные платы, модернизируя компьютер. 8-битки в плане модернизации были гораздо более ограничены. У них либо не было разъёмов для доп. устройств, либо в них вставлялись только платы от того же производителя. А в IBM PC можно было вставлять платы ISA от разных производителей. При желании можно было заменить и главную – материнскую – плату, в разъёмы которой вставлялось всё прочее.

И такая гибкость в плане модернизации скоро пригодилась пользователям «PC-шек»: в 1982 году выпустили в продажу процессор нового поколения – Intel 80286. Возникает вопрос, а почему после 086 появился 286, а не 186-й процессор? На самом деле, Intel 80186 и даже Intel 80188 тоже существовали, причём появились они чуть позже 80286-го. Но в качестве

центральных процессоров ПК они практически не использовались (за редкими исключениями типа Tandy 2000). Дело в том, что на кристалле 80186-го размещались многие дополнительные устройства, которые на материнской плате 8086 или 80286-го присутствовали в виде отдельных микросхем. То есть 80186 был неким прообразом нынешних микроконтроллеров, и предназначался для этих же целей – управлять различным оборудованием.



Других отличий от 8086-го у 80186-го немного. Это – некоторые новые команды, новый 68-контактный корпус вместо 40-контактного, более высокая тактовая частота. Единственный экземпляр i80186 (производства AMD) в моей коллекции всё же нашёлся: он управляет SCSI RAID – контроллером Tecram DC820 («компьютером в компьютере») для шины EISA.

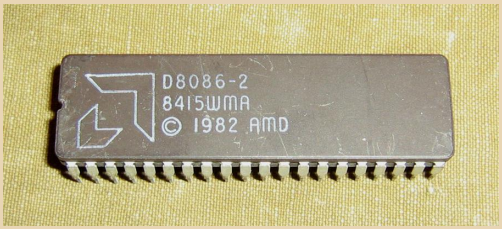


А для использования в качестве ЦП IBM PC предназначался i80286. Он также не слишком сильно отличался от i8086, в том смысле, что тоже был 16-разрядным. Однако нововведений было немало. Новый техпроцесс 1.5 мкм, новый корпус (как у 80186), тактовая частота 6 и 8 МГц вместо 4.77, несколько дополнительных команд… Но самое существенное – это возможность переходить в новый – «защищённый» режим работы, при сохранении т.н. «реального» режима, в котором 286-й представлял из себя просто более быстрый 8086-й. Защищённый режим работы стал первым шагом на пути создания многозадачных систем на базе IBM PC. Надо заметить, что в миникомпьютерах, например PDP-11 под управлением ОС UNIX, многозадачность была налажена гораздо раньше. Персональные же микрокомпьютеры долгое время рассматривались согласно парадигме: «один пользователь – одна программа». Пользователь вёл диалоговый режим, вводя в строку консоли команды и ожидая их выполнения. Именно так были устроены операционные системы для 8-битных (CP/M) и первых 16-битных (MS-DOS) ПК.

Однако на 286-м компьютере уже можно было запускать графическую оболочку (пока ещё не полноценную ОС) Windows 3.0 или 3.1, а она как раз работала в защищённом режиме 286-го процессора, позволяя держать в памяти сразу несколько программ, переключаясь между ними. В защищённом режиме возможна адресация памяти свыше 1 МБ. Поскольку у 286-го процессора не 20, а 24 адресных линии, на материнскую плату с 286-м можно установить до 16 МБ ОЗУ (в теории, поскольку многие реальные 286-е мат. платы поддерживали не более 4 МБ, а поставлялись и вовсе с 1 МБ ОЗУ). «286-й защищённый режим», однако, на практике оказался тупиковой ветвью, потому что вскоре появились 32-разрядные микропроцессоры Intel 80386, позволяющие в теории работать с 4 ГБ ОЗУ. Но об этом – в следующей части статьи…

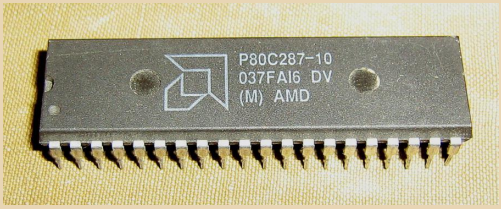
А теперь – немного о «клонах». Конечно, не только советские электронщики, сидя за «железным занавесом», копировали «интеловские» микросхемы при помощи «дралоскопа» (так остряки-инженеры называли, помимо

светокопировального стола, также микроскоп, предназначенный для изучения элементов на кристалле «трофейной» микросхемы). Этим же занимались и западные фирмы, например AMD, начиная с копирования i8080-го. Правда, они это делали по лицензии, указывая на микросхемах копирайт Intel. Впрочем, не всегда. Есть у меня и 8086-й с копирайтом только AMD.

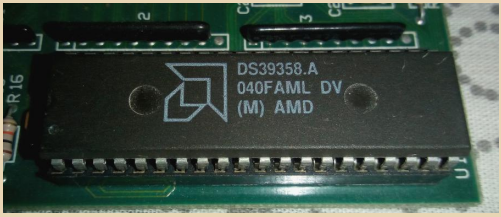


Кроме AMD, процессоры, совместимые с Intel 8088, делала японская фирма NEC. Микросхема D70108C-8 V20 (или просто V20) стоит в материнской плате «Turbo XT». Она работает на частоте 8 МГц. V20 не является строгой копией 8088. Он имеет другую внутреннюю архитектуру, что позволяет, например, быстрее выполнять команды умножения. Кроме того, у V20 есть режим эмуляции 8-битного процессора i8080.

Математические сопроцессоры также делали разные фирмы: Intel (D80287-10), AMD (P80C287-10), IIT (2C87-10)...



Представляет также интерес AMD-шная микросхема DS39358.A. Кто бы мог подумать, что это математический сопроцессор 80287! По крайней мере, установлен он именно в 286-ю системную плату...



Из аналогов 80286-го интеловского микропроцессора помимо AMD отметим Harris (CS80C 286-16) и Siemens (SAB 80286-1-N). Harris Semiconductor, США. Аналог i80286, тактовая частота 16 МГц. Существовали Harris, работавшие на частоте 25 МГц. Как заявлял изготовитель, быстродействие у такого микропроцессора в 19 раз выше, чем у 5 МГц i8086, что уже сопоставимо с

быстродействием 386-х систем. Siemens сделан в Австрии. Работает на частоте 10 МГц.

Следующая таблица обобщает технические характеристики микропроцессоров, рассмотренных в первой части нашей статьи.

И в заключение, приведём ссылки на ресурсы, посвящённые микропроцессорам, и технические описания упомянутых в статье микросхем:

http://cpu-collection.de/

http://datasheets.chipdb.org/Intel/MCS-4/datashts/intel-4004.pdf

http://datasheets.chipdb.org/Intel/MCS-8/8008/8008usersManualRev2_Nov72.pdf

http://datasheets.chipdb.org/Intel/MCS-80/intel-8080.pdf

http://datasheets.chipdb.org/Intel/x86/808x/datashts/8086/231455-005.pdf

http://datasheets.chipdb.org/Intel/x86/808x/datashts/8088/231456-006.pdf

http://datasheets.chipdb.org/Intel/x86/8018x/datashts/80186/27050008.PDF

http://archive.6502.org/datasheets/mos_6500_mpu_preliminary_may_1976.pdf

http://www.155la3.ru/4k601vm1.htm – 4.К602ВМ1, советский «клон» MOS 6502

http://datasheets.chipdb.org/Zilog/Z80/Z80_DataBook.pdf

http://www.155la3.ru/k1858.htm – советский «клон» Z80

http://datasheets.chipdb.org/NEC/V20-V30/NEC_uPD70108.pdf

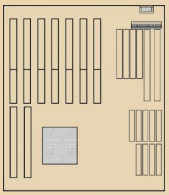
http://datasheets.chipdb.org/Harris/80c286.pdf

Михаил Бабичев (Антиквар)

Модель МП	Год начала выпуска	Разрядность АЛУ/ША/ШД	Кол-во инструкций	Техпроцесс, мкм	Кол-во транзист., тыс.	Такт. частота, МГц	Кол-во ножек	Выдел. тепла, Вт
Intel (i) 4004	1971	4/4*/4	46	10	2.3	0.74	16	1
i8008	1972	8/8**/8	48	10	3.5	0.2...0.5	18	1
i8080	1974	8/16***/8	80	6	4.5	2	40	1.5
MOS 6502	1975	8/16***/8	56	6	4.2	1...3	40	0.25...0.7
Zilog Z80	1976	8/16***/8	158	3	8.5	4...8	40, 44	1.5
i8086	1978	16/20****/16	98	3	29	5...10	40	2.5
i8088	1979	16/20****/8	98	3	29	5...8	40	2.5
i80186	1982	16/20****/16	110	3	55	10...20	68	1
i80286	1982	16/24*****/16	115	1.5	134	8...16	68	3.3

*Примечание: адресация до 32 кбит ОЗУ и 5 кбит ПЗУ (4 кБ и 0.6 кБ). Имеются 4 линии для выбора чипа ОЗУ и 4 линии данных, по которым также осуществляется адресация (не за 1 такт). Напрямую процессор может адресовать 4 кбит (0.5 кБ) ячеек памяти.
**Примечание: адресация до 128 кбит (16 кБ) ОЗУ/ПЗУ. Общая шина 8 бит используется и для данных, и для адресации (для этого требуется применение сложной внешней логики).
***Примечание: прямая адресация до 64 кБ памяти.
****Примечание: адресация до 1 МБ памяти.
*****Примечание: адресация до 16 МБ памяти.

M912 Rev.1.3 – пластиковый UMC и VLB



В 1997-м году на страницах журнала «Радио» появился цикл статей Александра Фрунзе «Модернизируем IBM-совместимый ПК». Собственно, цикл статей посвящён, как легко догадаться, апгрейду – были рассмотрены компьютеры, начиная от PC/XT и 286 (обновление путём полного перетряхивания) до 486/Pentium. Собственно, несмотря на тему обновления машины, статья содержала достаточно интересных сведений, которые заинтересовали автора в контексте возни со старыми компьютерами. К сожалению, некоторые вещи были поняты сперва не так – например, автор решил, что любую позднюю 386-ю материнскую плату можно апгрейдить до 486-ой установкой, возможно, специального процессора. Увы, оказалось, что для этого нужна особенная плата. Такие часто называли 3486-ми (например, FX-3000).

Примерно то же самое случилось с одной из участвовавших в тестах производительности плат на процессоре UMC, названном в текстах UM486SX2 80МГц. Фрунзе утверждал, что процессор впаян в плату, но на плате есть локальная шина. Увы, в статьях практически никогда не упоминались конкретные названия плат, поэтому желающему узнать о той или иной экзотической плате приходилось самостоятельно искать «примеры». Ваш покорный слуга решил, что речь идёт о какой-то плате с UMC U5S с шиной VLB. Увы, запрос «U5S VLB», помнится, ничего не дал. Я же был тогда знаком только с парой плат такого типа, где действительно был распаян U5S, но слоты расширения были только ISA-шными. Правда, такие платы содержали слот для 72-пиновых планок SIMM. Может быть, автор статей имел в виду их? В какой-то момент ваш покорный слуга уже решил, что так оно и было, и отчаялся разыскать плату с пластиковым процессором и скоростной локальной шиной.

Однако летом на «Авито» попалось объявление о продаже материнской платы с впаянным U5S. Сначала решил, что это очередная продажа материнской платы с ISA-слотами, впаянным процессором и даже без кэша (рабочая ATC-1411 продавалась на «Авито» где-то за 2 т.р.), но тут привлекли внимание два слота для SIMM 72-pin. Уже интересней – открыл объявление, и у меня отвисла челюсть. Продавалась действительно плата с впаянным UMC U5S, там было два слота для 72-пиновых планок SIMM, но на плате были распаяны два слота, похожих на VLB. Значит, не врал Фрунзе! Плата была тут же куплена, заказана, и началось ожидание. Наконец, посылка была получена, но на этом дело встало – у меня не было под рукой ISA-видеокарты, а с VLB-шной (не поверите – она-то как раз была) пускать побоялся.

Рис. 1. Собственно, виновница торжества – сейчас на ней установлены микросхемы кэша, джамперы, но тогда ничего такого не было

Так плата пролежала до конца августа, пока ваш покорный слуга не вернулся с отдыха вместе с новой находкой и не попробовал её тут же включить. Продавец писал, что плата нормально работала, но странные надписи на микросхемах чипсета и на микросхеме ПЗУ с BIOS внушали опасения. Да ещё и микросхема была короче кроватки. Но всё же подсоединил блок питания, динамик, клавиатуру. Вставил имеющуюся в наличии планку оперативной памяти. Нашёл у себя ISA-шную видеокарту. Собранную конструкцию подключил к монитору, подал питание и... ничего. Материнская плата не шевелилась и не подавала признаков жизни. Было жутко обидно и страшно – испоганил? Или же продали нерабочую – оттого и странные надписи на микросхемах чипсета, а ПЗУха в кроватке вообще может быть левой, а то и не памятью вообще. Но тут увидел, что не все перемычки установлены – например, была гребёнка, отвечающая за тактовую частоту процессора, и там не было ничего. Вот это номер! Но перемычки могли понадобиться предыдущему владельцу для чего-то ещё, и он их снял со старой материнки – почему бы и нет? Давайте поставим назад. Тут же красовалась табличка, показывающая, сколько надо установить перемычек для той или иной частоты, ну и, конечно же, куда их ставить. Полное отсутствие перемычек тоже воспринималось как заданное значение частоты – 20 МГц, но процессор на такой частоте заводиться не хотел. Что ж, самое простое было – установить частоту 25 МГц – это можно было сделать установкой одной перемычки. Находим перемычку, ставим и... слышим недовольный писк динамика. Ура? В принципе, ура – плата хоть и с какими-то проблемами, без картинки на мониторе, но ожила и недовольно пищит. Что делать дальше?

Собственно, сейчас не вспомню, что сделал сначала, что сделал потом. Будем считать, что сделал одновременно ☺ – установил перемычки для задания частоты 33 МГц – на плате распаян U5S Super 33/40 – 40 мегагерц давать побоялся ☺ – и поменял видеокарту. Всё завелось. Тогда выяснилось по выведенной на монитор строке, что передо мной плата PC Chips M912 Rev 1.3. Уже вперёд. Увы, найти самостоятельно информацию касательно локальной шины – VLB это или же какая-то хитрая (два слота вместо обычных трёх), как у помянутой выше FX-3000 – не удалось. Написал на «Полигон призраков», где объяснили, что на плате обычная VESA, то бишь VLB, что двух слотов вполне хватит, чтобы затолкать и скоростную видеокарточку, как минимум с мегабайтом памяти на борту, и мультикарту (сам привык ставить их в крайние слоты на других материнских платах), что кэш тоже можно поставить в пустые кроватки и не бояться, что там не подведены дорожки.

Что ж, тогда можно начинать. Сперва установил видеокарту и мультикарту. Система действительно запустилась и тоже показала картинку. Уже вперёд. На плате завелись нормально видеокарточки и S3, и Cirrus Logic 5429 – красота! Планка оперативной памяти оказалась 16 МБ FPM. Собственно, для такой машины более чем достаточно – планировалось установить Windows 3.11, а ей такой объём, как писали, в самый раз. На других компьютерах третья винда крутится и при восьми метрах оперативки. А вот с жёстким диском пришлось повозиться – поставил отформатированный на другом компьютере винт с установленной DOS и Windows 3.11. Данная материнка нормально его обнаруживала, но отказывалась грузиться. Примерно полмесяца повозился с ним. Так как на этой материнской плате до этого глюкнул один Cirrus Logic, побоялся, что что-то испортил и сейчас плата сможет грузиться только с ISA-шной мультикарты. Но обошлось – оказалось, что я забыл выставить в настройках режим обращения к диску – LBA. Уже сталкивался пару лет назад с таким на VLB-шных материнках, но основательно забыл, а зря. ☺ Перенастроил – диск начал грузиться.

Тут пришла очередь кэша – раз кроватки есть, то почему бы не попробовать? Наколупал немного микросхем Em51256-20P по 32 кБайт каждая. Но вот незадача – именно как устанавливать джамперы на кэш не удалось найти. Решено было действовать по аналогии с PC Chips M912 Rev. 1.7. Документация на неё находится по ссылке:

http://www.elhvb.com/webhq/models/486vlb3/m912v17.htm

Нумерация джамперов другая, но расположение похоже. Что ж, начнём. В качестве контрольной утилиты была использована программа Cache Check:

ftp://csiph.com/incoming/timc/Software/UTILS/cachechk.zip

Результаты экспериментов можно увидеть в таблице ниже.

Таблица 1. Результаты диагностики кэша в зависимости от установленных перемычек. Было установлено 8 микросхем Em51256 с тегом в оба банка.

JP9	JP16	JP17	JP18	JP19	JP20	JP21	Показания в сообщении BIOS	Показания Cache check
2-3	2-3	-	-	-	-	-	32 кБайт	Кэш отсутствует
2-3	2-3	-	-	-	-	+	64 кБайт	64 кБайт
2-3	2-3	-	-	-	+	+	128 кБайт	128 кБайт
2-3	2-3	-	-	+	+	+	256 кБайт	128 кБайт

Попытки менять положение **JP16** на 1-2 с установленным **JP21** приводили к остановке загрузки после теста оперативной памяти. Увы, манипуляции с **JP9** (там присутствует 7 пинов) ничего не дали. При некоторых других установленных джамперах BIOS определяла 256 кБайт, но либо вис HIMEM, либо утилита также находила 128 кБайт. Интересно, что на всех фотографиях этой материнской платы на **JP9** установлена только перемычка 2-3. Не знаю, виной ли этому вставленная задом наперёд микросхема кэша – сама микруха, как я понял, накрылась, но, может быть, накрылся и весь банк, либо проблемы самой материнской платы, но это единственное, что удалось с неё получить. Возможно, 512-е микросхемы, на 64 кБайт, дали бы 256 кБайт кэша, но пока что у меня только такие, какие есть.

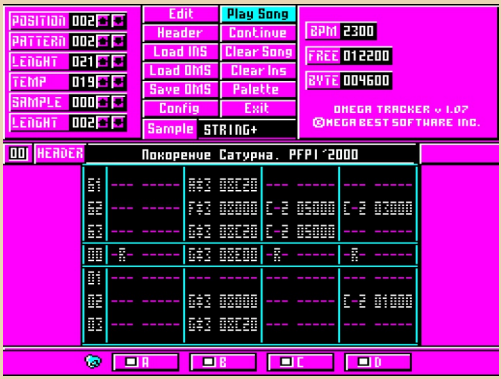
Материнская же плата, думается, будет поселена в корпус вместе с циррусовской видеокарточкой и VLB-шным мультиком и также станет очередным полигоном для downgrade-экспериментов, каким сейчас является другой компьютер на таком же процессоре, собранный в 2015-м (каком же уже далёком ☺) году.

Андрей Шаронов (Andrei88)

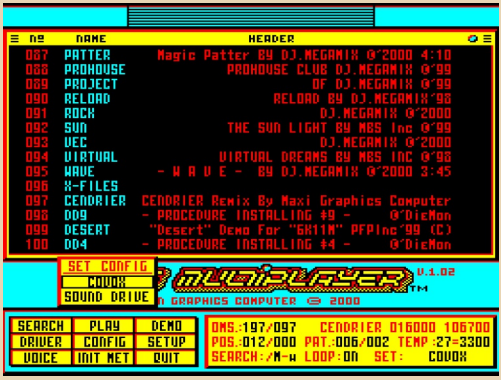
Цифровой звук на БК-0010. Ч.2

В прошлый раз я рассказал о том, с чего начался цифровой звук на БК-0010 и как в период с 1993 по 1995 год мы с **RDC** выдумывали алгоритмы проигрывания трекерной музыки. В статье были примеры кода и краткий курс ассемблера PDP-11 (именно эта система команд используется в процессоре БК). Статья наполовину состояла из занимательных историй и завершалась рассказом о музыкальном редакторе, написанном **Maxi G.C.** в 1997-ом году. Если вы пропустили первую часть статьи – обязательно прочитайте её прямо сейчас!

В 1998-ом году **Mega Best Software** (Евгений Деливрон) написал красивый и удобный Omega Tracker. Как и все программы из Самары, он работал только в операционной системе CSI-DOS. Новый трекер поддерживал до 32-х инструментов. Из Strogino Sound Tracker была заимствована таблица нот и идея сделать часть каналов упрощёнными, без изменения тона. В SST такой канал был один (он использовался для ударных), а в Omega Tracker – уже два. Это давало возможность писать более сложные партии ударных, а также размещать на упрощённых каналах сэмплы готовых аккордов. Если про Strogino Sound Tracker говорили, что у него «два с половиной» канала, то у Omega Tracker было уже «два с половиной и ещё половина». В умелых руках это большое преимущество перед SST. Omega Tracker также позволял применять к ноте одну из команд: смещение от начала сэмпла, вибрато, зацикливание.



Изначально предполагалось, что трекер будет работать с музыкальной приставкой **Sound Drive**, созданной в Казани под руководством **Lasoft** (Александр Лёвин). Но пока приставка находилась в разработке, **Mega Best** адаптировал редактор под обычный Covox. Sound Drive содержал на борту 4 независимых 8-битных ЦАПа (для этого на шине были организованы дополнительные регистры) и позволял с помощью специальной команды изменять громкость ноты на каждом из каналов. Первая версия приставки оказалась несовершенной: при изменении громкости она издавала щелчок. Пока Sound Drive проходил доработку, Omega Tracker в версии под Covox успел распространиться среди самарских музыкантов. Позже, в 1999-ом году, **FDN** (Дмитрий Фролов) написал отдельный проигрыватель OMS MultiPlayer, он предоставлял пользователю выбор: играть музыку на Covox или на Sound Drive.



Кроме того, **Mega Best** написал редактор сэмплов **Wave Master**.

Мелодий в формате **OMS** сочинили штук двести, а Sound Drive изготовили всего в двух экземплярах. В настоящее время существует проект по воссозданию этой музыкальной приставки. Сложность, однако, состоит в том, что Sound Drive приходится подключать к разъёму МПИ (магистральный параллельный интерфейс), который обычно занят контроллером дисковода или жёсткого диска. Covox же не конкурирует с другими устройствами за место на шине МПИ и занимает параллельный порт, как правило свободный.

Давайте посмотрим на код воспроизведения музыки Omega Studio. В целом, он похож на код SPCplay 1.2. Регистры **R1** и **R2** указывают на обычные сэмплы (в трекере это первые два канала), а регистры **R3** и **R4** – на сэмплы барабанов (третий и четвёртый каналы). Если в данный момент какой-то из барабанов не должен звучать, адресация в команде заменяется с косвенной автоинкрементной на обычную косвенную – например, вместо **MOVB (R3)+,R5** пишется **MOVB (R3),R5**. Это останавливает движение указателя по сэмплу.

```
       MOV #2600,R0  ; note duration
PLAY:  ADD #12345,#0 ; frequency for sample 1
       ADC R1        ; R1: pointer to sample 1
       ADD #12345,#0 ; frequency for sample 2
       ADC R2        ; R2: pointer to sample 2
       MOVB (R3)+,R5 ; R3: pointer to sample 3
       MOVB (R4)+,R6 ; R4: pointer to sample 4
       ADD R5,R6     ; mix two drum samples
       MOVB (R1),R5  ; get sample 1 data
       ADD R5,R6     ; mix sample 1
       MOVB (R2),R5  ; get sample 2 data
       ADD R5,R6     ; mix sample 2
       MOVB R6,@#177714
  SOB R0,PLAY        ; loop
```

Напомню, что **R6** – это регистр стека (он же **SP** – stack pointer), поэтому использовать его таким образом довольно рискованно. При прерывании (например, по клавише **СТОП**) адрес возврата запишется в те ячейки памяти, на которые указывает **R6**. Я бы предпочёл использовать **R6** в качестве указателя на сэмпл – тогда во время прерывания сэмпл слегка испортится, но его можно быстро восстановить с диска. Другой вариант – запретить все прерывания и опрашивать клавиатуру вручную. Но этот лишний опрос увеличит разрывы между нотами.

Процедура микширует звук с частотой дискретизации 10870 Гц на стандартной БК-0011. Нетрудно переделать её под стерео-Covox: для этого нужно вставить команду **SWAB R6** перед **MOVB (R1),R5** и команду выдачи результата в порт 177714 заменить с **MOVB** на **MOV**. Музыка замедлится примерно на 4%, а частота микширования снизится до 10417 Гц. Впрочем, насчёт стерео-Covox'а я забегаю вперёд.

CAFe'99. Впереди **Maxi G.C.**, **Manwe**, **Xanth**

В 1999-ом году ребята из **Excess team** провели в Казани демопати CAFe (Computer Art Festival). Конкурса БК-музыки почему-то не было, так что я участвовал в музыкальных конкурсах для Amiga и PC (второе и четвёртое место соответственно). Первое место в конкурсе БК demo по результатам голосования заняла работа Cat от **SK** (Стас Коровин) с музыкой под Covox. Каким-то образом простенькая Cat опередила технически выдающуюся работу Ray Dreams от **Maxi G.C.** Может быть потому, что музыка в Ray Dreams звучала через AY, к тому времени многим уже надоевший? А демо под Covox были редкостью (на ум приходят только Disco Club и Happy 1997-го года).

С 2000-го года ни на одну из российских демопати не было подано ни одной работы для БК. Организаторы CAFe переключились на платформы ZX Spectrum, Amiga и PC.

16 лет БК-сцена хранила молчание. Пока вдруг не произошла удивительная вещь. Можно даже сказать, волшебная © TRON Legacy. В январе 2016-го года на демопати DiHalt Lite команда Excess team представила демо Happy New DiHalt для БК-0011м. Чтобы сделать демо, **Lasoft** написал набор программ под Windows, облегчающих создание софта для БК. Об этой системе он рассказал в 47-ом выпуске подкаста SCENE. Через год на DiHalt Lite 2017 Excess team выставили демо «ЭлектроБулка», которое заняло первое место, обойдя работы для ZX Spectrum. А на DiHalt 2018 казанские демосценеры привезли прекрасную работу «Однажды». Об этих событиях, всколыхнувших ретросцену (и особенно поклонников БК), можно написать отдельную статью. Здесь же я опускаю подробности, потому что они не имеют отношения к цифровой музыке под Covox.

В декабре 2017-го года мне написал **Nodeus** (Илья Ширинкин) с предложением добавить в плейлист Интернет-радиостанции HypeRadio музыку с БК. Сперва я порендерил несколько мелодий из Strogino Sound Tracker с помощью эмулятора БК. Результат мне не очень понравился. Тогда я попросил **Ivanq** (Иван Мачуговский) написать на Node.js конвертер из БК-шного формата SPC в стандартный Amiga MOD. Сложность заключалась в том, что некоторые мелодии использовали команду sample offset, поэтому определить границы сэмплов в SPC-файле трудно (это ведь не исходный нотный текст, а уже скомпилированный набор адресов и частот). Полностью решить эту проблему так и не удалось. Полученные MOD-файлы я порендерил самым качественным проигрывателем трекерной музыки под Windows – XMplay. Старый трюк, который я описывал ещё в статье 1998-го года, заключается в том, чтобы разные типы сэмплов рендерить с разным типом интерполяции (например, барабаны без интерполяции, синтезаторы с линейной интерполяцией, а пианино с кубической) – это позволяет выжать максимально качественное звучание из старой трекерной музыки. Некоторые SPC-композиции не сохранились в цифровом виде, но остались на аудиокассетах. Послушав кассеты, я снова засомневался – какое звучание считать более правильным и красивым. В третий раз мне пришлось переделывать музыку (а ведь для каждой мелодии это кропотливая работа по сведению нескольких дорожек). Провозившись несколько дней, я порядком устал.

И вот как-то ночью, засыпая, я размышлял о том, в чём же проблема звучания всех этих треков. И снова (смотри прошлую статью) ответ был такой: «**барабаны**». Остальные инструменты звучали приемлемо, но именно барабаны страдали неестественностью – им не хватало высоких частот. А что если не возиться с каждой мелодией отдельно, а просто поднять высокие частоты третьему каналу (барабаны обычно звучат на нём)? Так, а что если сделать это **прямо на БК**? Стоп! А что, так можно было? Нет, минуточку, как такое возможно? Или… Блин, почему никто раньше не додумался?! Я вскочил с кровати, включил компьютер и начал писать код. На следующее утро я его отполировал.

Идея была такая: звуки на первых двух каналах микшировать как обычно, а третий канал барабанов обсчитывать вдвое чаще. Для этого звуки барабанов должны быть вдвое большей частоты и размера. Код получился таким:

```
      MOV #177714,R6 ; Covox port
      MOV #2600,R0   ; note duration
PLAY: MOVB (R1),R4   ; channel 1
      ADD #12345,#0  ; frequency for sample 1
      ADC R1         ; R1: pointer to sample 1
      ADD (R3)+,(R6) ; channel 3 delta
      MOVB (R2),R5   ; channel 2
      ADD R5,R4
      ADD #12345,#0  ; frequency for sample 2
      ADC R2         ; R1: pointer to sample 2
      ADD (R3)+,R4   ; channel 3
      MOV R4,(R6)
  SOB R0,PLAY
```

В регистре **R6** хранится адрес порта Covox. Предпоследняя команда записывает в порт сумму, накопленную в регистре **R4** (сумма всех трёх каналов). В середине программы есть ещё одна запись в Covox из **R3** (это указатель на сэмпл барабана, третий канал). Таким образом, на каждом шаге цикла данные барабана посылаются

на Covox в два раза чаще, чем данные обычных сэмплов. Хитрость в том, что нечётные слова (да, не байты) сэмпла барабана хранятся как обычно, а чётные – в виде разницы с предыдущим значением. Зачем так сделано? Нам нужно сначала послать на Covox сумму двух каналов и барабана, а затем ту же самую сумму двух каналов и следующее значение барабана. Чтобы не пересчитывать сумму заново, мы берём прошлое значение и изменяем его на столько, на сколько изменилось значение барабана (изменение может быть как положительным, так и отрицательным). Сложение происходит прямо в порту Covox'а с предыдущим его значением.

Понимание программы осложнено порядком команд. Пришлось расположить команды записи в Covox так, чтобы между ними проходило одинаковое количество тактов. Благодаря этому сэмпл барабана проигрывается равномерно (без так называемого джиттера), а значит качественно.

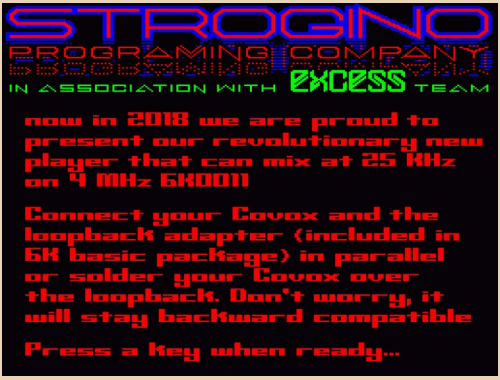
Позже замеры на реальной БК-0011М показали частоту дискретизации 12,5 кГц – ровно как у SPCplay 1.2. Только барабаны выводились вдвое чаще – аж на 25 кГц. Это уже настоящий Hi-Fi! Звучание оказалось действительно потрясающим, барабаны зазвучали как надо: басы качали, верхи звенели и шелестели. Такого звука БК ещё не знала!

Но это было потом. А во время написания программы у меня не было реальной БК. Эмулятор же вёл себя странно – выдавал не совсем то, чего я ожидал. Наверное, с самим эмулятором проблема – думал я. Как-то странно он складывал данные с портом. По счастливому стечению обстоятельств, в феврале 2018-го года мне предстояла поездка в Казань, и **Lasoft** пригласил к себе в гости. У него была настоящая БК-0011М.

В Казани мы выяснили, что эмулятор работает правильно. Стали разбираться и думать. Оказалось, что при сложении с портом процессор сначала считывает значение из порта, затем прибавляет к нему что нужно, потом записывает сумму обратно. Да только вот считывает он всегда ноль. А вовсе не прошлое записанное в него значение, как я рассчитывал. И тут **Lasoft** вспомнил, что у него есть блок нагрузок – это такая маленькая примочка из комплекта БК-0010, которая замыкает выходные биты порта на входные. Так называемый loopback adapter. Сделан он, видимо, ради одной единственной программы – теста порта. В остальном совершенно бесполезен. Был бесполезен до сегодняшнего дня. А теперь блок нагрузок – единственное наше спасение! **Lasoft** до последнего не верил, что это поможет. Но почему нет? Всё сходится: пишем в порт, читаем записанное.

Ура, всё заработало! Потрясающий звук. Мы счастливы! Единственная загвоздка: моя программа находилась в зачаточном состоянии, предстояло ещё многое дописать, а реальной БК у меня не было. Тогда **Lasoft** отправил письмо **Gid**'у (Сергей Яковлев) с просьбой добавить в эмулятор блок нагрузок. Что **Gid** и сделал буквально за один день. Это очень сильно помогло. Я вернулся в Москву и продолжил работать над программой, используя эмулятор. Время от времени посылал **Lasoft**'у программу на проверку.

Результатом стала демка In Your Space для демопати «**Мультиматограф**» в апреле 2018-го года. Выбор мелодии был сделан за меня **Adam**'ом и **Lasoft**'ом – они очень любят этот трек, написанный в 1996-м году **Real**'ом (Александром Ильиным) при моём участии (на PC в Scream Tracker 3). Кстати, ремейк этой мелодии на БК уже делал **Maxi G.C.** в 1997-м году.



Итак, нам удалось невозможное – трекерная музыка с частотой дискретизации 25 килогерц на обычной БК-0011М. Разве может быть что-то лучше?.. Ответ, конечно, «да» – ведь это демосцена. Искусство невозможного.

Теперь предстояло совместить этот алгоритм с тем, что описан в SPCinfo 95 (смотри прошлую статью). Он давал базовую частоту микширования 13,9 кГц и требовал вдвое больше памяти под сэмплы, так как следовало заранее переводить их в 16 бит. И, конечно, теперь хотелось отказаться от блока нагрузок, ведь он есть не у каждого владельца БК.

Но перед этим произошло вот что: во время моего пребывания в Казани **Adam** и **Lasoft** посветили меня в планы портирования на БК Bad Apple – популярной монохромной анимации в стиле аниме с более чем 24 миллионами просмотров на YouTube. Именно её выбрали демосценеры в качестве эталона, чтобы сравнивать возможности старых компьютеров. Существует плейлист Bad Apple on Everything с несколькими десятками роликов, крутящих анимацию Bad Apple на самых разных платформах. Каждый программист придумывал свой способ вывода анимации и звука, соразмерно возможностям выбранной платформы. Казанские демосценеры решили выводить звук через стерео-Covox (я прежде не слышал, чтобы кто-нибудь паял такой для БК), а вот способ вывода изображения вызвал дискуссию. Я предлагал упаковывать кадры и хранить только разницу между ними. Excess team предлагали гнать с винчестера несжатые кадры. Наработки на эту тему у них уже были. Меня смущала низкая частота кадров. Казанцев смущала необходимость писать сложный упаковщик видео в условиях нехватки времени перед демопати. Уже по приезду в Москву я сообразил, что монохромную анимацию можно раскрасить в стиле рекламы Apple iPod, скинул пару набросков ребятам, но они решили не рисковать и продолжили работу над более простым вариантом. В итоге Excess team со своей Bad Apple заняли первое место на демопати «Мультиматограф» в апреле 2018-го года, а мы с **Ivanq**, вооружившись их наработками для чтения с жёсткого диска, принялись делать свою версию – с упаковкой и цветом. Наша работа Good Apple заняла второе место на демопати Chaos Constructions в августе 2018-го года.

Для обеих версий я занимался звуком. Большого труда стоило найти оригинальную музыку в качественном виде. Из неё предстояло выделить вокал и обработать его для лучшей слышимости, так как после пересчёта в 8 бит и 22,7 кГц (больше БК не тянула одновременно с видео) голос в музыке становился плохо различим.

В процессе работы над Good Apple **Ivanq** написал множество полезных инструментов для программирования под БК (их описание дано в статье на Habr), а я понял, что нам нужен стерео-Covox. И понадёжней. На печатной плате, а не клубком резисторов. За этим я обратился к **Tarh**'у (Александр Васильевский). Поразмышляв, мы решили помимо звука сделать на той же плате и вход для джойстика. А лучше для двух. Фирменных, от Nintendo. Изучив спрос в Telegram-канале БК0010/11М World, мы приступили к разработке JoyVox.

Решено было поставить качественные цифро-аналоговые преобразователи, хороший предусилитель для наушников, переключатель режима стерео-моно, регулятор громкости. Основной задачей было отвязать сигнал от наводок, идущих с платы БК. Обычный Covox на резисторах сильно страдал от них. По ходу проектирования были написаны специальные

тесты для проверки качества звука, перепробованы наушники с разными сопротивлениями. Геймпады также пробовали от разных производителей, для теста я написал игру для двоих игроков.

В сентябре 2019-го года в музее Яндекса на фестивале «Демодуляция» прототип **JoyVox** прошёл боевое крещение. На мероприятии был организован турнир по игре Fist, в котором приняли участие около полусотни человек. В конце 2019-го года, после четвёртого прототипа, JoyVox отправился в производство.

Yandex Demodulation

Пока **Tarh** колдовал над схемами и платами, я разработал новый трекерный формат **ST0**, написал проигрыватель для него и одну мелодию. Когда дело дошло до второй мелодии, я понял, что набивать её в текстовом редакторе очень утомительно. И написал конвертер из формата Scream Tracker 3 (S3M). Конвертер работает прямо на БК. При написании музыки в формате S3M нужно соблюдать определённые правила. Во-первых, всего три канала. Во-вторых, сэмплы должны быть маленькими, чтобы помещаться в память БК. Обычные сэмплы нужно подготовить с частотой дискретизации 12560, а барабаны – с частотой 25120 Гц. В списке сэмплов между обычными и барабанами должно быть одно пустое место (так конвертер распознаёт два типа сэмплов). В-третьих, допустимый диапазон нот – от C2 до C5. Нельзя использовать никакие эффекты Scream Tracker 3. И всё же, несмотря на ограничения, это лучше, чем текстовый редактор. Кстати, формат ST0 – текстовый. Проигрыватель исполняет также и роль конвертера в более скоростной внутренний формат. Работает он только в MKDOS, поскольку использует прямой доступ к файлам по номерам блоков. Сконвертированные в 16 бит сэмплы занимают так много места, что затирают операционную систему в ОЗУ, поэтому в самом начале проигрыватель составляет список из номеров блоков – какой сэмпл с какого места жёсткого диска начинается. После этого операционная система уже не нужна, можно перетирать. Впрочем, для пользователя это незаметно – нажатие клавиши **СТОП** во время проигрывания приводит к выходу в систему, как обычно (точнее, к загрузке системы с нуля).

```
Strogino Sound Tracker 25KHz Covox player
Input music file name: IYS.ST0

    title: In Your Space
   author: Real and Manwe
    group: The SandS
     year: 1996
  comment: www.thesands.ru

   volume: 100
      BPM: 132
 patterns: 10
   orders: 12
     time: 51 seconds

Playing...
```

Код проигрывателя совершенно монструозный, поэтому я приведу всего один фрагмент. Сложность кода обусловлена тем, что я пытался выжать максимальную скорость из стандартной БК-0011М (даже без контроллера расширенной быстрой памяти). На каждом из трёх каналов может звучать нота, а может быть пауза. Каждая нота может принадлежать высокой октаве или низкой, а это разные алгоритмы! Поэтому возможных сочетаний пауз, высоких и низких нот 18. Столько же отдельных процедур микширования. Конвертируя исходный текст, проигрыватель определяет, для какой строки паттерна какую процедуру вызывать. Вообще, конвертер довольно сложный. Например, он отслеживает длительности инструментов и отключает канал тогда, когда сэмпл на нём заканчивается. А ещё содержит алгоритм антиклика (устранение щелчка в начале музыки).

Давайте посмотрим на процедуру, играющую низкую ноту на первом канале, высокую ноту на втором канале и барабан на третьем:

```
        MOV #177714,R5  ; Covox port
PLAY16: MOVB R4,(R5)    ; sound output every
                        ; 160 CPU cycles
        MOV (R1),R4     ; channel 1
        ADD (R2)+,R4    ; channel 2
        ADD (R3)+,R4    ; channel 3 (drums)
        ADD #12345,R6 ; frequency for sample 1
        MOVB R4,(R5)    ; sound output every 160
                        ; CPU cycles
        ADC R1          ; R1: pointer to sample 1
        ADD #12345,#0 ; frequency for sample 2
        SBC R2          ; R2: pointer to sample 2
        SWAB R4         ; seriously?!
        NOP             ; slow down before the
                        ;next sound output
 SOB R0,PLAY16
```

Регистры **R1**, **R2** и **R3** указывают на сэмплы. Они преобразованы в 16 бит, поэтому команды **MOV** и **ADD** не байтовые (в процессоре БК команды **ADDB** и не существует – вот почему пришлось отказаться от 8-битных сэмплов, об этом подробно рассказано в прошлой статье). Частота первого сэмпла накапливается не в числе (как во всех предыдущих алгоритмах), а в регистре **R6**. Это сделано для ускорения кода и выравнивания временных интервалов между двумя выводами в порт. Команда **MOVB R4,(R5)** вклинилась между **ADD** и **ADC** (во всех предыдущих алгоритмах они шли друг за другом, что логично – **ADC** прибавляет бит **C**, если при сложении командой **ADD** произошло переполнение). Поскольку **MOVB** при косвенной адресации не портит бит C, позволительно использовать эту команду перед **ADC**. Для выравнивания команд по времени мне очень пригодилась программа, написанная во время работы над Good Apple – она с высокой точностью замеряет время исполнения произвольного куска кода.

На первом канале звучит низкая нота, поэтому использована команда **ADC**. На втором канале звучит высокая нота, поэтому использована автоинкрементная адресация **(R2)+** и команда **SBC** (как в алгоритме из SPCinfo 95).

И тут мы доходим до команды **SWAB** (обмен байт местами), которая полностью ломает наш мозг. Что вообще происходит?!

Однажды, когда я рассказывал о новых алгоритмах микширования **Tarh**'у, он спросил:

– А ты не пробовал векторные операции?
– В смысле? Обработку нескольких чисел за раз?
– Да, одной командой.
– Нет, на БК такое невозможно! Это же не ARM.
– Ну вдруг как-то получится?
– На БК так не бывает.

На следующий день я осознал, что разговаривал неуважительно: как можно сразу отвергать идею, не попробовав? Да, я лучше **Tarh**'а знаю ассемблер БК, но он же крутой программист и хакер, к тому же искренне пытается мне помочь. Значит, я обязан попробовать – хотя бы для того, чтобы сказать «твоя идея не сработала». Это было бы аргументом. А отрицание с ходу – не аргумент. И я погрузился в размышления. Через несколько дней я сообразил: в демо In Your Space прошлая сумма каналов хранилась прямо в порту Covox'а. Но ведь перед записью в порт эта сумма находилась в регистре **R4**. Почему бы просто не перекинуть её в старший байт **R4** и спокойно обсчитывать новую сумму в младшем байте? А когда понадобится старая, доставать её командой **SWAB R4**. Это стало отправной точкой моих рассуждений.

Теперь обещанные векторные операции. Команды **ADD (R2)+,R4** и **ADD (R3)+,R4** одновременно прибавляют два байта из памяти к двум байтам **R4**. А ведь в байтах **R4** хранятся две разные переменные! Выходит, одна команда обрабатывает два значения за раз. Осталось понять, что и зачем прибавлять к этим двум переменным.

Напомню, что частота барабанов вдвое выше, чем у обычных сэмплов, поэтому в In Your Space на каждом шаге цикла мы обращались к обычным сэмплам один раз, а к барабанам два

раза. Теперь сделаем иначе: дважды за цикл будем извлекать и значения барабана, и значения обычных сэмплов. Поскольку нам всё равно пришлось преобразовывать 8-битные сэмплы в 16-битные, пусть в каждой паре байтов обычные сэмплы хранят одинаковые значения, а барабаны – разные значения. Будем микшировать (складывать) парами и получим сразу две суммы, которые нужно последовательно вывести в Covox.

Можно объяснить это и с другой стороны, через так называемый oversampling. Мы удвоили частоту дискретизации обычных сэмплов, просто продублировав каждое значение. Теперь все сэмплы хранятся с высокой частотой 25 кГц, и микширование ведётся на этой частоте. За один раз мы микшируем сразу два значения: текущее и следующее. И выдаём их по очереди через равные промежутки времени.

Любознательный читатель может спросить: почему бы при оверсэмплинге обычных сэмплов не применить интерполяцию? Или сразу хранить их в высоком качестве, с удвоенной частотой сэмплирования, как барабаны. Ответ таков: барабаны всегда звучат с одинаковой (максимальной) частотой, поэтому мы точно знаем, что после младшего байта нужно послать в Covox старший, а на следующем шаге цикла перейти к следующей паре байтов. С обычными сэмплами не так: они звучат с произвольной скоростью. Указатели на обычные сэмплы могут двигаться до восьми раз медленней, чем указатель на барабан. **R1** и **R2** часто остаются на месте несколько циклов подряд, указывая на один и тот же адрес, на одну и ту же пару байтов. Поэтому на каждом шаге цикла оба байта обычного сэмпла обязаны оставаться одинаковыми.

Итак, путём двойного расхода памяти и невероятных программных ухищрений, в 2018-м году мы достигли невозможного – проигрывания трекерной музыки с частотой 25 кГц на 4-мегагерцовой БК-0011М без дополнительных устройств.

Фото с CAFe 2019: **Pcat**, **Tarh**, **Vinnny**, **Megus**, **Seajeff**, **MMCM**, **Lasoft**, **XPEh**, **f0x**, **Manwe**, **EA**

В октябре 2019-го года в Казани возродилась демопати CAFe. Она прошла с большим успехом, на платформе БК было представлено 10 работ.

Что дальше? Наверное, использование музыки формата ST0 в демо. Ещё одна интересная тема – проигрыватель 4-канального формата OMS на стерео-Covox (может быть, кто-то возьмётся за это вместо меня?). Контроллер СМК-512 тоже выглядит привлекательно для музыки: 32 страницы быстрой памяти по 16 килобайт. А как насчёт игр со звуковыми эффектами под Covox?

Ссылки

Демки для БК:
https://www.pouet.net/prodlist.php?platform%5B%5D=BK-0010%2F11M

Omega Tracker с подборкой музыки:
https://www.pouet.net/prod.php?which=83926

JoyVox:
http://hardware.thesands.ru

Проигрыватель на 25 кГц:
http://thesands.ru/bk0010/st0/

Александр Мачуговский (Manwe)

Грабберы иконок для Windows 3.x

Что только не сравнивали с айсбергом, говоря, что меньшая его часть видна над поверхностью воды, а большая часть скрыта от взора в глубинах. Оказалось, что и обычный EXE-файл порой напоминает такой айсберг. Пользователь видит только один значок, когда запускает программу, но сколько их там? А ведь ещё есть и файлы библиотек, где тоже могут гнездиться маленькие симпатичные картинки. Одна только библиотека **MORICONS.DLL** из состава Windows чего стоит. В своё время с большим удовольствием разглядывал значки, которые предлагала Windows для различных ДОСовских программ.

Установить такой значок для любой программы можно средствами Windows. Насколько помню книжку Фигурнова, такая возможность существовала даже для Windows 3.x. Однако существуют программы, которые в состоянии не только дать возможность поменять картинку ярлыка, но даже выдернуть эту картинку из любого файла, сохранив её в отдельном ICO-файле. Такие программы, как, наверное, знает читатель, зовутся грабберами иконок. И вот, разглядывая на Old-DOS скриншот одной из таких программ для Win9x, автор задумался – а не было ли таких утилиток для любимой третьей винды? Сперва поиск выдал где-то три программы, однако потом автор наткнулся на страничку https://www.pcorner.com/list/WINDOWS, где относительно внимательный поиск почти сразу выдал пятёрку кандидатов на обзор (а ещё автор поживился там заставкой «Аквариум» для третьей винды ☺).

Оказалось, что можно повыбирать, посравнивать, поэтому программы предварительно были запущены на основном компьютере, чтобы выбрать наиболее интересные. Тогда выяснилось, что ряд программ является шароварными, а часть – бесплатными. Сперва я думал исключить шароварные программы из обзора, однако в конечном итоге насчитал пять программ и решил уделить внимание каждой. Итак, поехали...

Icon Thief – как лучше не делать?

Одной из первых программ, которые я скачал с сайта, стала эта (загрузил её на Old-DOS, так что, теперь можно взять по этой ссылке: http://old-dos.ru/dl.php?id=20601). Для начала все программы были запущены на основном компьютере под Windows XP – для предварительной оценки интерфейса. Так как всё это делалось несколько небрежно, программа оставила достаточно неприятное ощущение, а я сразу подумал о программе плохо – в духе: «Хотят денег, а сделано хуже бесплатной».

Программа для запуска потребовала библиотеку **VBRUN300.DLL**, но под WinXP запустилась и без неё. Но после того как автор решил скормить ей какой-нибудь файл с иконкой, был разочарован – превью отображалось криво (см. рис. 1).

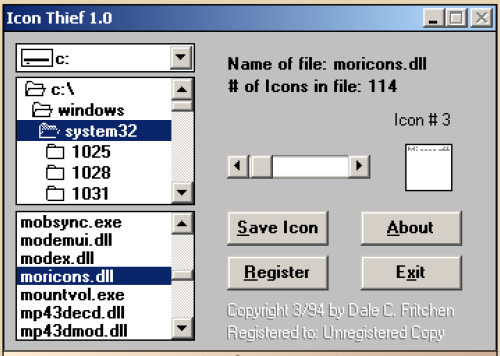


Рис. 1. Icon Thief под Windows XP – специально скормил ему библиотеку **moricons.dll** – и вот результат...

Однако на старом компьютере ситуация оказалась куда лучше, правда, программа запустилась только после того, как скопировал в её каталог **VBRUN300.DLL**. Первое, чем огорчала программа – заставкой с просьбой зарегистрироваться и ожиданием (рис. 2) – такая маленькая утилитка, а столько требует. Да ещё написано про 30-дневный срок. Без него ещё можно было бы потерпеть, но так – как-то не очень.

Рис. 2. При запуске нам сперва придётся подождать и прочесть о желании автора получить за утилиту 5 долларов и о сроке оценки программы в 30 дней

После этого нам откроется основное окно программы, которое вы уже видели на рис. 1. Под Windows 3.x оно выглядит так:

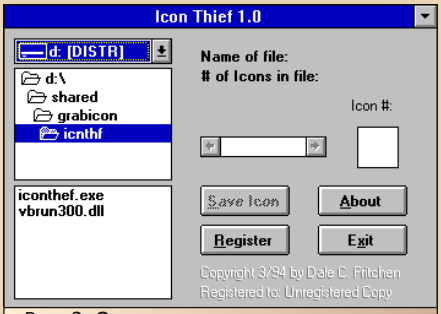


Рис. 3. Основное окно программы под Windows 3.x (т.е. на ретросистеме ☺)

Собственно, никаких особых наворотов у программы нет – как видно из рисунка, слева располагается меню выбора файла, из которого будем извлекать иконки, в квадратике справа будет выведен значок для предварительного просмотра. Прокрутка же позволит просмотреть остальные значки, если в файле их больше одного. Кнопка **Save Icon** позволит сохранить выбранную иконку.

Что ж, попробуем натравить программу на библиотеку **moricons.dll**, которая располагается в каталоге **C:\Windows**. Результат предпросмотра можно увидеть на рис. 4 – как видите, куда лучше, чем на рис. 1. ☺

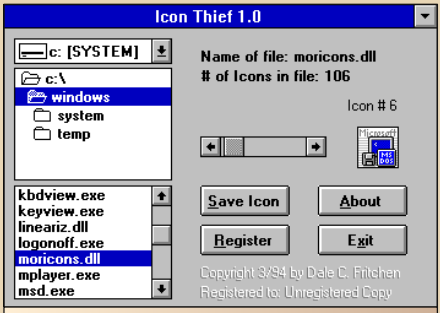


Рис. 4. Предпросмотр значка из **moricons.dll**

После извлечения выбранной иконки оказалось, что просмотреть её можно только средствами «ДИСКо Командира» – увы, ничего специально для просмотра устанавливать не стал – понадеялся, что PaintBrush сам запустится после выбора файла и нажатия на **Enter** – не запустился. Но результат в просмотрщике «ДИСКо Командира» можно увидеть на рис. 5.

Рис. 5. Извлечённая иконка в окне просмотрщика «ДИСКо Командира»

После этого, эксперимента ради, натравил программу на сам «ДИСКо Командир», а вернее на его исполняемый файл – также удалось получить нормальный предпросмотр и извлечённую иконку (рис. 6 и 7 соответственно).

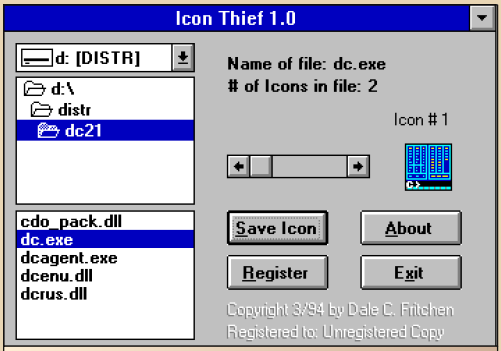


Рис. 6. Иконка «ДИСКо Командира» в предпросмотре

Рис. 7. Иконка «ДИСКо Командира» извлечена

Удалось увидеть «интересную особенность» программы – закрыть её, как обычное окно, не получилось (автор попытался применить сочетание **Alt+F4**) – программа закрывается только нажатием кнопки **Exit**.

Собственно, после запуска на старом компьютере, негативное отношение к программе притупилось. Программа как программа – небольшая, достаточно функциональная – разве что платная и выводит перед запуском заставку, но работает. Так что, из двух недостатков остался только один – да и то для кого-то он не недостаток. ☺

Icon Show – неоднозначный взгляд на проблему

Следующей в нашем обзоре идёт также условно-бесплатная программа **Icon Show** (тоже теперь доступна на Old-DOS по адресу http://old-dos.ru/dl.php?id=20602). Если к недостаткам предыдущей программы можно отнести надоедливое меню и 30-дневный срок работы (на глюки под WinXP закроем глаза ☺), а так – программа и программа – выполняет всё необходимое, то здесь – то хочется рукоплескать авторам стоя, то плюнуть на пол.

Итак, чем же отличается данная программа от всех остальных, которые будут рассмотрены в настоящей статье? В первую очередь, размером окна. Посмотрите на рис. 8. Это я ещё его специально ужал мышкой – оно было больше.

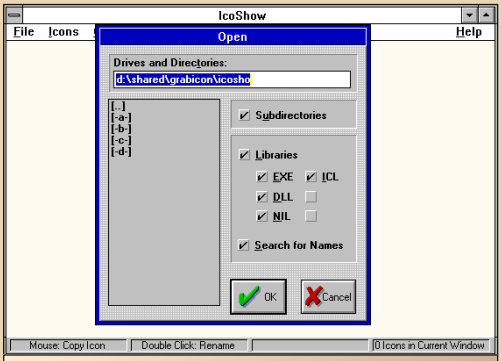


Рис. 8. Такой предстаёт перед пользователем Icon Show при запуске. Само окно ещё больше – автор его ужал немного

Следующая интересная особенность, что программа предлагает пользователю, фактически, не указывать конкретный файл, где находятся иконки, а перетрясти весь каталог, а также вложенные подкаталоги на предмет маленьких картинок. Возможно, укажи пользователь файл явно, программа откроет только его, но результат перетряхивания всех файлов в каталоге и подкаталогах просто поражает (см. рис. 9).

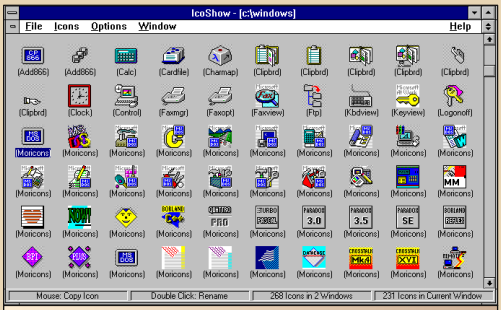


Рис. 9. Результат извлечения иконок из всех файлов в каталоге **C:\Windows** – это вам даже не **moricons.dll** трясти ☺

Каждой папке соответствует один такой экран, где будут размещены все найденные в папке иконки. Если поиск производился и в подкаталогах, их содержимое будет выведено на отдельный экран. Переключиться между ними можно через меню **Window**. На рис. 10 показан результат поиска иконок в каталоге, где автор держит дистрибутивы программ, а также ряд утилит, не требующих установки. Как видим, качество отображения прекрасное, а вывод всех иконок на один экран в чём-то гораздо удобней показа одной иконки.

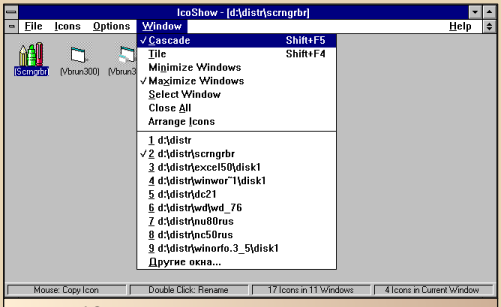


Рис. 10. Результат сканирования каталога с дистрибутивами программ

«В чём же тогда неоднозначность программы?» – спросит читатель. А неоднозначность в том, что я не смог просто сохранить ни одной иконки в отдельном файле. Щелчок мышью по любому значку приводил к вылету программы, а сохранить можно было только всё в библиотечном файле.

В принципе, собрать иконки в один файл, а потом использовать библиотеку для изменения значков – неплохая идея, но хотелось бы иметь результат в виде отдельного файла, тем более от условно-бесплатной программы. ☺

GT Icon Extract – маленький бесплатный шедевр

Теперь перейдём к бесплатным утилитам. Первой на очереди будет **GT Icon Extract**, которую можно скачать по ссылке http://old-dos.ru/dl.php?id=20603. Запустив программу, вы увидите небольшое, аккуратное и симпатичное окошко (см. рис. 11).

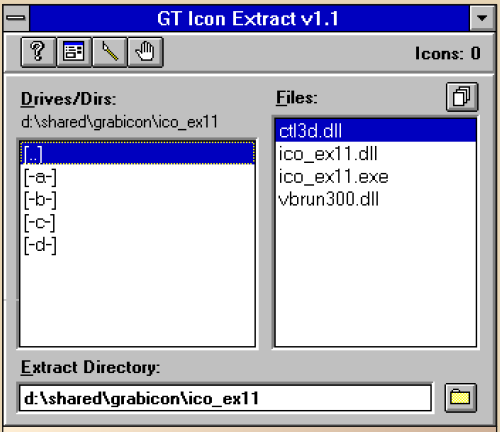


Рис. 11. Главное окно программы – маленькое и симпатичное

Интерфейс достаточно прост – слева список каталогов, справа – список содержащихся в каталогах файлов. Выбираем нужный файл, выделяем, нажимаем кнопочку  и извлекаем. Если хотим извлечь все иконки из файлов, находящихся в каталоге, перед извлечением нажимаем кнопку  – находится над списком файлов. Также над списком указывается количество иконок.

Куда будут извлечены иконки? По адресу, который указан в строке, располагающейся внизу окна. По умолчанию указан каталог, где находится программа, поэтому автор при

первом извлечении получил конкретную мешанину из кучи файлов (да, натравил снова на каталог Windows ☺)

Рис. 12. Куча иконок, и все в каталоге программы. В названии указана часть файла, из которого иконка была извлечена

Нажатием кнопки  можно вызвать диалог и указать папку для извлечения (см. рис. 13). К сожалению, не обошлось без ложки дёгтя – в диалоге нет кнопки создания новой директории, с которой программа бы стала практически идеальной.

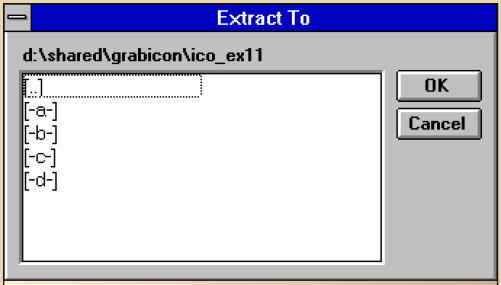


Рис. 13. Диалог выбора места для извлечения иконок

Попытка указать самостоятельно название несуществующей папки привела к сообщению об ошибке. Более того, если в строке вписан адрес несуществующего каталога, нажатие кнопки вызова диалога выбора места извлечения приведёт к вылету программы. Не самый приятный момент.

Однако даже такие неудобства не ослабляют приятные впечатления от использования программы.

Icon Extractor – триумф минимализма

О следующей программе очень трудно рассказать. Собственно, прошу любить, жаловать, скачивать (ссылка) – **Icon Extractor**. Главное окно показано на рис. 14. Всё ещё проще и минималистичней, чем у предыдущей программы.

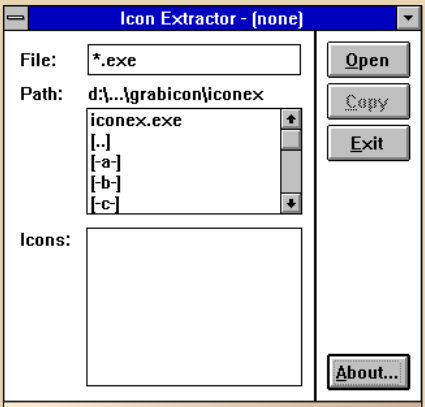


Рис. 14. Icon Extractor – главное окно – оно же и единственное ☺

В меню, располагающемся в верхней половине окна, выбираем файл (двойным щелчком или кнопкой **Open**), снизу видим содержащиеся иконки. Разве что надо выбрать в выпадающем списке расширение интересующего файла.

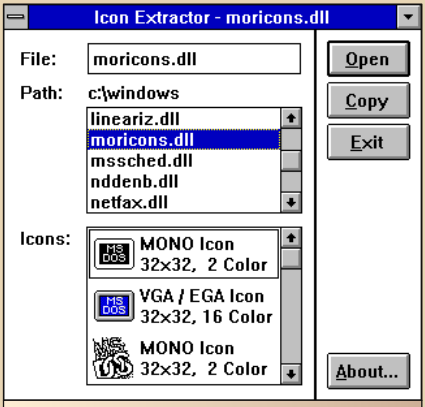


Рис. 15. Предпросмотр содержащихся иконок

Как же извлечь иконку? А вот тут хитрость и коварство маленькой программы – её надо выделить щелчком мыши, а потом скопировать в буфер обмена нажатием кнопки **Copy**. Дальше скопированный рисунок можно вставлять в любой графический редактор. Например, в Paintbrush.

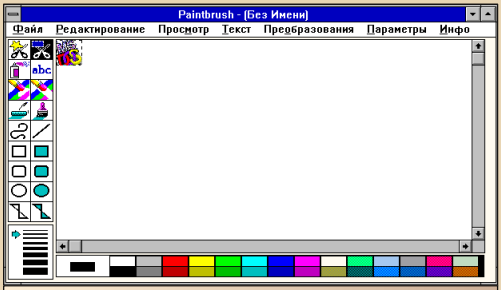


Рис. 16. Извлечённая иконка была скопирована в буфер обмена и вставлена в Paintbrush

Собственно и всё, что можно сделать с помощью этой программы. Ну, собственно, она и нужна только для этого – разве что файлы сама не создаёт.

Высасыватель иконок

Программа называется **Icon Vacuum** – ну прям иконосос какой-то... Скачать её можно по ссылке http://old-dos.ru/dl.php?id=20605. При запуске вас встретит достаточно аскетичный интерфейс (см. рис. 17), но при этом достаточно функциональный.

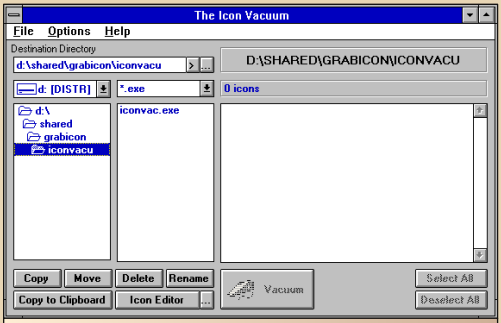


Рис. 17. Главное окно программы Icon Vacuum – аскетичность даже при обилии элементов окна

Слева находятся две панели – каталоги и файлы. Выбираем каталог, выбираем файл – правда, один. Справа отображаются содержащиеся в файле иконки (см. рис. 18).

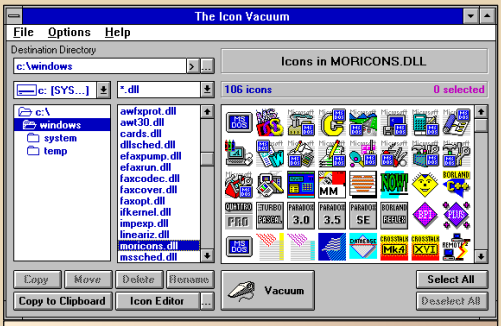


Рис. 18. Предпросмотр иконок из выбранного файла

Как и в прошлой программе, необходимо выбрать тип файла, в котором мы будем искать иконки. Это можно сделать в выпадающем списке, находящемся над списком файлов. Чуть выше располагается строка, где указывается адрес каталога извлечения. Рядом же находится кнопка, с помощью которой можно вызвать диалог выбора пути извлечения (по умолчанию папка, где находится файл с иконками) – см. рис. 19.

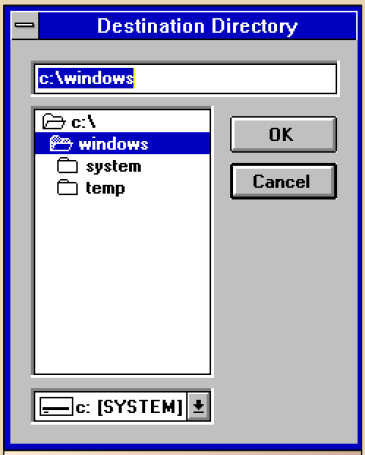


Рис. 19. Выбор каталога для извлечения иконок

Иконки можно выбрать и извлечь поодиночке – кнопка **Vacuum** – под панелькой предпросмотра, либо все сразу – если выбрать все нажатием кнопки **Select All**. При извлечении также указывается часть названия файла, откуда была извлечена иконка (см. рис. 20).

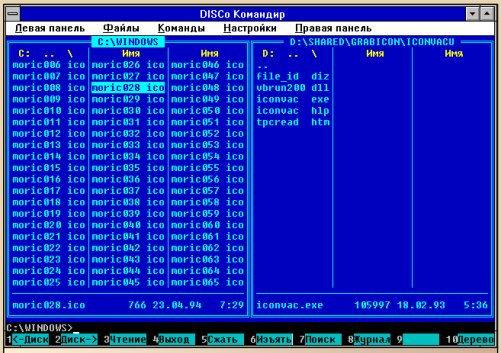


Рис. 20. Извлечённые иконки – в каталоге по умолчанию ☺

Программу автор также проверил на самом «ДИСКо Командире» – иконки файлового менеджера также были нормально отображены (см. рис. 21).

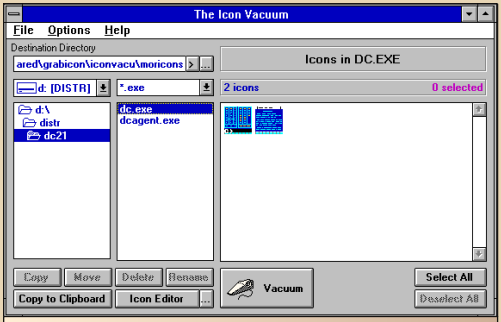


Рис. 21. Иконки «ДИСКо Командира»

Из недостатков программы можно отметить, наверное, перерисовку иконок при смене каталога или диска – для файла **moricons.dll** это заняло достаточное время. В остальном программа, наверное, наиболее удобная, полезная и толковая из всех испробованных.

Умом или сердцем?

Что ж, на этом остаётся только завершить небольшой обзор программ. Как обычно получается – планировалось одно, выходит – несколько другое. Я думал, что получится достаточно подробный обзор пары-тройки программ, однако же, простота утилиток-грабберов не оставляет большого простора для изучения. Простой и понятный интерфейс – небольшие плюхи – для бесплатной программы – почему бы и нет – тем более, что практически каждая из них первой версии. Однако иконки, в принципе, прекрасно извлекаются как условно-бесплатными программами, так и бесплатными.

В том числе по удобству интерфейса программы идут практически вровень. По большому счёту – какую выбрать – дело вкуса. Автору больше симпатичная GT Icon Extract, однако Icon Vacuum оказалась куда более функциональней, удобней и понятней. Да и не думаю, что извлекать иконки приходится пользователям каждый день, так что, можно потерпеть и строгий интерфейс. Но всё равно – мои симпатии на стороне Icon Extract. ☺

Андрей Шаронов (Andrei88)

ArtMoney – запасная колода для геймера

Компьютер у автора появился поздно, думается, это уже ни для кого не секрет (поминал это в статьях сто раз ☺). Зато у товарищей компьютеры были – машинки под управлением Windows 98 с 64-128-ю метрами памяти, 433-м селерончиком или чуть более мощным дюроном, встроенной видеокарточкой на четыре мегабайта и жёстким диском на десять гигов, разбитым на два логических. Такие машинки, собственно, и служили делу приобщения подрастающего поколения к миру вычислительной техники. За громкими словами обычно скрывалась роль «игрового автомата», на котором запускались квейки с первого по третий, Half-Life и Counter-Strike – почему-то их больше запускал товарищ, у которого был дюрон, также говорили и о серии WarCraft.

Из прикладного софта практически ничего не было – разве что офис, у одного из товарищей стоял Photoshop – ходил на курсы по работе с фотографиями, но практически на каждой машине присутствовала «виртуалка» и ArtMoney. Первая обычно использовалась, чтоб создать образ принесённого из проката диска, чтоб не спеша пройти игрушку, а не сидеть всю ночь, ибо диск отдавать уже завтра, а вот ArtMoney...

ArtMoney описывали, как взломщик игр, позволяющий добавить жизни или здоровья, патронов, денег, брони – в общем, чего душа пожелает, а игра позволит. ☺ Принцип действия виделся автору примерно таким: у программы есть некая база, которая содержит самые популярные игрушки, где содержатся сведения, где же лежат те или иные ресурсы – запускаем игру, сворачиваем и запускаем ArtMoney – если программа определила игру – можно подкручивать параметры. Если нет – на нет и суда нет – проходите честно. ☺ Отчего была такая уверенность? Просто товарищи квест «Ларри» знали только по возможности артманями подкрутить количество денег в кармане главного героя. Что за игра – не знал никто, да и не интересовался.

Особого интереса автор к этой программе никогда не питал, однако, когда захотелось собрать компьютер, который бы напомнил о начале двухтысячных (о котором писал в «Сыне своего века»), программа вспомнилась. Начал искать и несколько найденных версий залил на Old-DOS. Залил и, собственно, забыл. Ну, есть программа и есть. Тут же, при обсуждении темы номера, программа вспомнилась. Тем более, редактор журнала упомянул её как пример. Так почему бы не попробовать? Результату возни с программой и посвящена настоящая статья.

Для опытов взял ArtMoney 6.09, которую можно скачать с сайта Old-DOS:

http://old-dos.ru/dl.php?id=16476

В качестве подопытной игрушки будет выступать Warcraft 2: Tides of Darkness (взял обрезанный вариант здесь: http://old-dos.ru/dl.php?id=1213 – когда-то тоже играл обрезанным, установленным с какого-то игрового сборника). Игрушка знакомая, в которой вполне понятно, что крутить. ☺ Правда, решил не заходить в те уровни, где ресурсы уже становятся критичными, и приходилось осваивать «колонии», перевозя батраков на кораблях – остановился на начале третьего уровня.

С установкой программы проблем не возникло – используется Windows 95 OSR2 с установленным **dcom95.exe** – не знаю, заведётся ли без него. Распаковал на диск архив и запустил.

Итак, мы дошли до третьего уровня и жутко возжелали, чтоб у нас уже сейчас была куча золота (хоть и работают над этим вопросом целых два батрака ☺) (см. рис. 1).

Рис. 1. У нас уже 400 единиц золота, но мы хотим больше и быстро

Итак, останавливаем игру и «сворачиваем». На деле оказалось, что лучше нажимать не клавишу со значком Windows, а **Alt-Tab**, если что-то открыто, кроме игры. По крайней мере, у меня после вызова меню **«Пуск»** игра вылетела – связано это или нет – не знаю, ☺ но по **Alt-Tab** при следующей загрузке всё нормально работало.

Теперь запускаем саму ArtMoney. На рис. 2 показано основное окно программы.

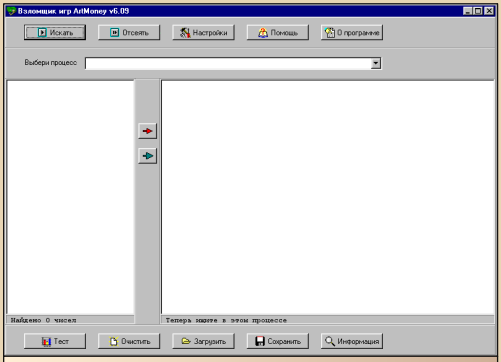


Рис. 2. Главное окно программы

В выпадающем меню **«Выбери процесс»** можно увидеть список определённых программой процессов, которые, видимо, ArtMoney потенциально считает играми. Наш WarCraft определён без проблем (см. рис. 3).

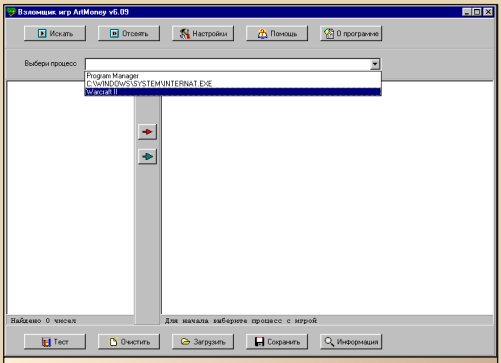


Рис. 3. Выпадающее меню со списком процессов – в частности, определён WarCraft

Выбираем нужный нам процесс. Казалось бы, сейчас вывалится диалог с параметрами «золото», «лес», «число батраков» и т.д. – ведь ArtMoney знает игру и должна знать, где что подкручивать... Но нет – области под списком пусты. Нажимаем кнопку **«Искать»**, но и тут проблема – а что искать-то? Точное значение? Автор посчитал, что под точным значением подразумевается адрес переменной, которую надо менять – увы, таких вещей он не знает, начнём перебирать остальные – например, выберем тип **«Неизвестное значение»** или **«Кодированное значение»**. Увы, ни то и ни другое результата не дали. Что ж, сдаёмся и идём на поклон вселенскому разуму – ну не может быть, чтоб такую знаменитую игрушку не ковыряли!

И да, попадается упоминание Warcraft II BNE (http://www.artmoney.ru/et_win_24.htm), но у нас обычный Warcraft 2 (думал, что его ArtMoney опознает скорее), в теме на форуме Old-Games человек делится, что пользовался артманями во втором варкрафте (вот здесь – предпоследний пост). Это вселило надежду. И, наконец, инструкция – как же применить программу – разве что варкрафт третий, а не второй. Но оказалось, что главное – принцип.

Итак, с иллюзией, что программа прекрасно знает нашу игру, как семейный врач своих

пациентов, пришлось проститься. Далее выяснилось – начал-то автор правильно – выбрал игру, запустил поиск... Только искать надо точное значение – нужно просто знать значение параметра – например, количество золота, которое сейчас в игре.

Хорошо, запускаем поиск – сейчас у нас 400 единиц золота, что мы и вписываем (см. рис. 4).

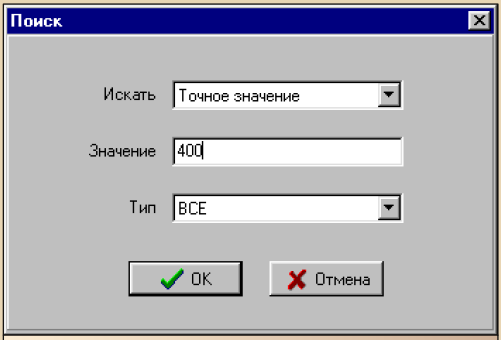


Рис. 4. Вписываем искомое значение – сейчас у нас 400 единиц золота – это число и поищем

Поиск сразу выдал огромный список и... заткнулся на полдороги. Ну, нам и этого хватит, решил автор. Теперь нужно произвести изменение параметра и вычленить изменившуюся переменную. Что ж, переходим снова в игру, и батраки у нас добывают ещё две сотни единиц золота (ну не чеканят же они монеты там – не отходя от рудника ☺). Хорошо. Теперь у нас 600 единиц, и мы жмём кнопку **«Отсеять»** и вводим новое значение. Всё бы ничего, но программа... отсеяла все...

Что ж, пробуем повторить поиск, введя уже искомое значение – 600. В этот раз поиск завершился нормально. И опять нам надо произвести изменение параметра в игре. Что ж, догоняем его до 800 единиц и повторяем отсев. В этот раз повезло – программа нашла четыре переменных. Одну двухбайтовую и три на четыре байта (у автора приведённой выше инструкции одна переменная, но и значение более хитрое – чёрта с два спутаешь). Что ж, попробуем ещё немного увеличить значение и снова поискать – догоняем до тысячи – запускаем отсев, и у нас уже три переменных.

Может быть, попробовать ещё раз? Довёл до 1100, загнал отсев, но ничего не поменялось. Что ж, пробуем тогда изменить переменные. Нажимаем зелёную стрелку **«Добавить все»** и будем редактировать по очереди и возвращать исходное значение, если не повезёт (см. рис. 5).

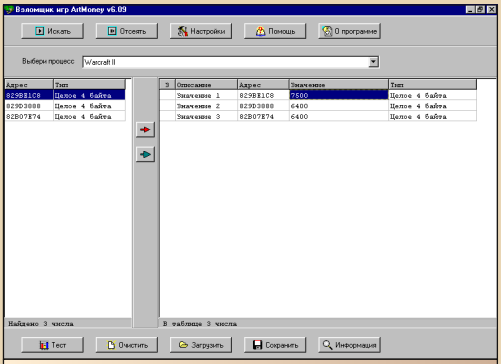


Рис. 5. Переменные готовы к редактированию. Одна изменена

Рис. 6. Результат редактирования в самой игре

Изменяем первую переменную в списке и переходим в игру... Вот повезло – количество единиц золота возросло сразу! Вернёмся назад. И тут автор увидел интересный эффект – поменялись оставшиеся две переменные.

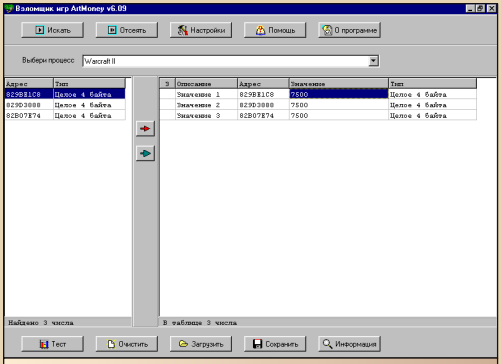


Рис. 7. После переключения в игру и возврата в ArtMoney поменялось значение остальных двух переменных

Автор проверил эффект несколько раз, поэтому исходные значения двух последних переменных на рис. 5 выше, чем 1100. ☺

Что ж, параметр удалось поменять, что не может не радовать. Видимо, так товарищи и добавляли когда-то себе в играх критичный ресурс – деньги, лес, патроны – как я понял, для ArtMoney главное, чтоб игра позволила изменить количество единиц ресурса, а уж остальное – дело техники.

Интересно, что одной из дополнительных функций ArtMoney является тест производительности компьютера (см. рис. 8).

Автор запустил программу на компьютере AMD K5 PR133 с 32 МБайт ОЗУ без кэша второго уровня (ещё не поставил ☺). Понятное дело, что Pentium MMX 166 МГц будет шустрее. ☺

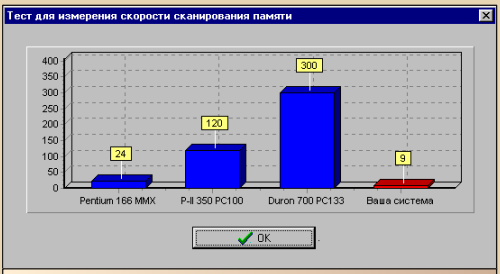


Рис. 8. Встроенный тест производительности

А что до вынесенной в названии запасной колоды, так нам когда-то рассказал человек случай, как он играл и доставал с лежащей в столе колоды тузы. Но в какой-то момент его «противник» спросил: «Так ведь в колоде больше четырёх тузов не бывает? А Вы уже шестой кидаете...» Думал о принципе игры с ArtMoney, и как-то это случай вспомнился – чем-то похоже. ☺

Андрей Шаронов (Andrei88)

Разработка программ для текстового редактора NewsMaster («Журналист»). Часть 3 из 3: конвертер клипарта

Завершая серию статей, посвящённых редактору NewsMaster (см. Downgrade N27 и N28), давайте разберём формат библиотек клипарта для возможности извлечения стандартных, а также добавления внешних изображений из уже знакомого нам по прошлой статье BMP-формата.

Замечание. Здесь, в отличие от двух предыдущих статей, бинарный код готовой программы не обязан быть максимально компактным, чтобы умещаться в статический буфер или экономить память для запуска дочернего процесса, также как и нет необходимости в низкоуровневой работе с оборудованием или операционной системой. Поэтому для удобства и упрощения программы будем использовать язык C и компилятор Borland Turbo C 2.01 (1988) для DOS, который был официально отдан правообладателем в свободный доступ в 2006 году. На всякий случай код также будет совместим и с 32-битной версией свободного компилятора GCC для возможности собрать программу на современных системах.

Для начала стоит сказать, что каждое использованное изображение клипарта сохраняется редактором NewsMaster в статью целиком. Поэтому при передаче *.NWS-файлов со статьями кому-либо, или размещения их в публичном доступе, нет необходимости также прикладывать к ним и использованные библиотеки клипарта, даже если они пользовательские и в состав NewsMaster не входят. Что удобно, экономит место и позволяет обходиться одним единственным файлом статьи.

Библиотеки клипарта, как нетрудно догадаться, лежат в каталоге **GRAPHICS**, хотя имя и путь можно изменить через параметр **GraphicsLibraryPath** конфигурационного файла **NEWS.CFG** на что угодно, даже на точку (текущий каталог) – тогда все файлы будут в корне каталога программы (что неудобно, но такие версии NewsMaster тоже встречаются на просторах Интернета). Поэтому будем опираться на NewsMaster версии 1.0 с каталогом **GRAPHICS**, в противном случае нужно искать соответствующие файлы в каталоге программы или её подкаталогах.

Легко заметить, что библиотеки клипарта состоят из трёх типов файлов (в некоторых версиях часть файлов может быть утеряна, т.к. они необязательные, но об этом позже) с расширениями: ***.SHP**, ***.SDR** и ***.SDX**. Для удобства возьмём библиотеку **NEWS**, которая практически в неизменном виде есть во всех версиях программы. Состоит она из трёх файлов (в порядке убывания размера):

NEWS.SHP – 77532 байта
NEWS.SDR – 2400 байта
NEWS.SDX – 600 байт

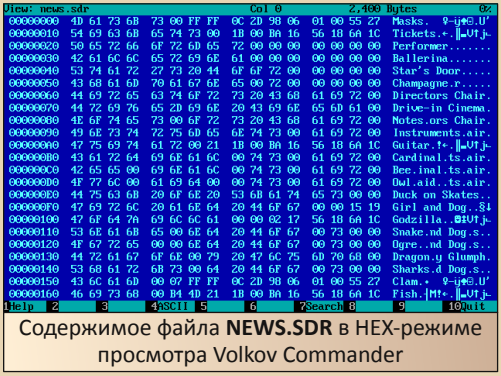
Изображения, по всей видимости, хранятся в самом большом файле – **NEWS.SHP**. Попробуем просмотреть его в любом шестнадцатеричном редакторе или HEX-режиме просмотра в большинстве файловых менеджеров (начало файла, первые 64 байта в HEX):

```
0B 34 58 00 00 3C 00 00 00 00 00 00 00 FF 80 00
FE 00 00 00 00 00 00 0F AB 00 01 FF 00 00 00 00
00 00 35 56 00 01 FF 00 00 00 00 00 00 EA AC 00
01 7F 80 00 00 00 00 03 55 58 00 02 3F 80 00 00
```

Увы, пока что, содержимое никак не радует чем-то внятным и понятным. Мы вернёмся к этому файлу позже, а пока что давайте попытаем счастья с двумя другими. Взглянем на следующий файл – **NEWS.SDR** (нулевые байты заменены на точки, а все остальные

символы из верхней половины таблицы ASCII – на знаки вопроса):

```
 M  a  s  k  s  .  ?  ?  ?  ?  ?  ?  ?  .  U  '
4D 61 73 6B 73 00 FF FF 0C 2D 98 06 01 00 55 27
 T  i  c  k  e  t  s  .  ?  .  ?  ?  V  ?  j  ?
54 69 63 6B 65 74 73 00 1B 00 BA 16 56 18 6A 1C
 P  e  r  f  o  r  m  e  r  .  .  .  .  .  .  .
50 65 72 66 6F 72 6D 65 72 00 00 00 00 00 00 00
 B  a  l  l  e  r  i  n  a  .  .  .  .  .  .  .
42 61 6C 6C 65 72 69 6E 61 00 00 00 00 00 00 00
```



Содержимое файла **NEWS.SDR** в HEX-режиме просмотра Volkov Commander

Хорошо видно, что файл состоит из строк по 16 байт с именами изображений, которые, обычно, заканчиваются на байт 000h (ASCIIZ – файл не текстовый, хоть и содержит строки), после которого могут быть «мусорные» байты, отличные от нуля. При этом нулевой байт завершает строку, только если она короче 16 символов, но ничто не запрещает ей занимать их все (тогда она не будет завершаться нулём – об этом нужно помнить). Таким образом, мы только что разобрались с предназначением *.SDR-файлов – это список имён изображений. И здесь необходимо сразу же заметить важную деталь: если загрузить библиотеку **NEWS** в программе, то первым изображением там будет отнюдь не **Masks**, а **American Eagle**. При внимательном изучении выводимого программой списка становится понятно, что он отсортирован в алфавитном порядке. Это означает, что порядок следования изображений в библиотеке не обязан совпадать со списком вывода в программе, так что при дальнейшем изучении формата опираться следует именно на содержимое *.SDR-файла. Ещё одна важная вещь, которую можно почерпнуть из этого формата и которая позже нам пригодится – это количество изображений:

2400 / 16 = 150 [1]

Размер всего файла, делённый на размер одной записи, даст 150 – это количество рисунков в данной библиотеке, в чём легко убедиться, если запустить программу и вручную подсчитать количество файлов в списке вывода.

Замечание. Этого нельзя узнать, разбирая форматы файлов, а только изучая редактор в отладчике или дизассемблере, но если запись имени начинается с байта 000h (конец ASCIIZ-строки) или 01Ah (маркер конца текстового файла), то NewsMaster считает, что файл закончился и завершает чтение. Поэтому в случае, когда в конце находятся подобные записи (встречаются в некоторых версиях программы из Интернета), полагаться на размер подобного файла уже нельзя.

Но имена нам много не дадут, поэтому перейдём к рассмотрению последнего файла – **NEWS.SDX** (значения, для наглядности, разделены на блоки):

00 00 00 00 | 41 02 00 00 | DD 03 00 00 | 1E 06 00 00
2B 08 00 00 | 04 0A 00 00 | 3A 0C 00 00 | 13 0E 00 00
28 10 00 00 | 69 12 00 00 | AA 14 00 00 | E0 16 00 00
16 19 00 00 | 4C 1B 00 00 | 8D 1D 00 00 | CE 1F 00 00

Здесь хорошо видны последовательности по 4 байта и, судя по всему, это двойные слова, но могут оказаться и парами, тройками или даже группами двойных слов. Однако теперь мы уже вооружены знанием о том, что в библиотеке 150 изображений. Снова применим деление размера файла, но теперь мы должны будем получить размер одной записи (разумеется, при условии, что они фиксированной длинны):

600 / 150 = 4

Значит, на каждую запись действительно отводится по 4 байта. Обычно в двойных словах хранят либо смещения до начала элемента, либо его размер. Подо что-то другое этот файл сложно «подогнать» по смыслу, ибо если представить, что это не двойное слово (4 байта), а, например, пара слов (два по 2 байта)

[1] В NewsMaster версии II в библиотеку добавили одно изображение, и их стало 151, именно поэтому мы рассматриваем эту библиотеку от версии 1.0.

под ширину и высоту изображения, то ни то, ни другое не может быть нулём. А в силу того, что все 150 записей используются, т.е. соответствуют каждому изображению, и размер изображения тоже не может быть пустым (4 нуля – первое значение), то можно сделать предположение, что в *.SDX-файлах действительно хранится таблица смещений на начало каждого из рисунков в **NEWS.SHP**.

Теперь пришла пора вернуться к рассмотрению файла **NEWS.SHP** – давайте взглянем на первые 16 байт по указанным в **NEWS.SDX** смещениям:

```
OFFS: 00 00 00 00
DATA: 0B 34 58 00 00 3C 00 00 00 00 00 00 00 FF 80 00

OFFS: 00 00 02 41
DATA: 0B 25 52 00 00 00 00 00 00 00 03 80 00 00 00 00

OFFS: 00 00 03 DD
DATA: 0B 34 56 00 00 00 00 00 9B B7 60 00 00 00 00 00

OFFS: 00 00 06 1E
DATA: 0A 34 4E 00 00 00 00 00 00 3E 00 00 00 00 00 06
```

В начале изображения должны храниться его размеры (потому что, как мы выяснили выше, храниться им просто-напросто больше негде), чтобы знать, сколько читать из файла и как выводить на экран. Выкинув все значения, которые хотя бы в одном столбце данных **DATA** обращаются в ноль, у нас останутся первые 3 байта. Получим такие значения в десятичном представлении:

```
11 52 88
11 37 82
11 52 86
10 52 78
```

Мы помним, что большинство рисунков в программе прямоугольные, так что ширина должна быть больше высоты. Под это отлично подходят вторая (высота) и третья (ширина) величины. А вот первое значение (11 и 10) не может быть ни шириной рисунка, ни его высотой (очевидно, что слишком мало). Также оно не зависит от второго значения – т.к. может быть 11 для 52 и 37 одновременно, но при этом и 52 может быть как для 10, так и для 11. Следовательно, значение, возможно, как-то зависит от третьей величины – ширины. Попробуем поделить ширину на него:

88 / 11 = 8
82 / 11 = 7,(45)
86 / 11 = 7,(81)
78 / 10 = 7,8

Т.е. соотношение этого числа и ширины лежит в пределах от 7 до 8. Здесь нам стоит вспомнить, что все изображения чёрно-белые, следовательно, для хранения одного пикселя хватит одного бита, а, значит, в байте их поместится 8. Но ведь в память нельзя записать, скажем, 1,5 байта, когда нужно сохранить 12 пикселей (8 + 4), поэтому округление всегда идёт в большую сторону. Итого, получаем:

88 / 8 = 11
82 / 8 = 10,25 (округляем) = 11
86 / 8 = 10,75 (округляем) = 11
78 / 8 = 9,75 (округляем) = 10

Всё совпало. Таким образом, мы, предположительно, разобрались с форматом заголовка изображения:

byte rowsize – количество байт в одной строке пикселей;

byte height – высота рисунка;

byte width – ширина рисунка.

Но рисунок может быть сжат. Самый простой способ проверить это – умножить количество байт в одной строке на высоту и прибавить 3 (размер заголовка) – мы должны будем получить точное смещение следующего рисунка. Попробуем для первого:

(rowsize * height) + 3 = (11 * 52) + 3 = 575

Смещение 575 или 023Fh, но это на два байта меньше, чем необходимое 0241h. Где же мы промахнулись? Давайте, для проверки, рассмотрим ещё парочку:

(11 * 37) + 3 = 410 = 019Ah
(11 * 52) + 3 = 575 = 023Fh

Складываем со смещениями:

0241h + 019Ah = 03DBh (+2 = 03DDh)

03DDh + 023Fh = 061Ch (+2 = 061Eh)

Т.е. у нас стабильно теряется 2 байта на каждое изображение, и прибавлять нужно не 3, а 5, тогда мы будем точно попадать по смещениям, что, кстати, заодно и подтвердило нашу гипотезу о том, что изображения хранятся не сжатыми. Но где расположены и за что отвечают пропущенные 2 байта? Здесь индукция уже не поможет – нужно набирать статистику. Для этого достаточно будет взглянуть на начало пятого изображения в **NEWS.SHP**:

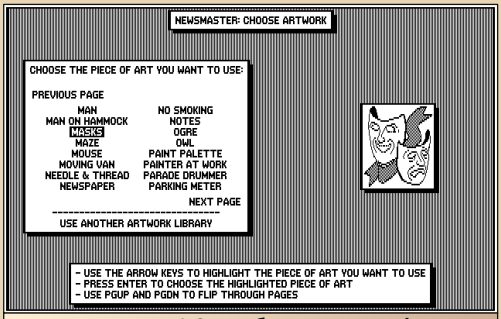
```
OFFS: 00 00 08 2B
DATA: 09 34 41 00 FF FF FF FF FF FF FF FF 80 84 51 15
```

Хорошо видно, что оно отчётливо начинается с последовательности байт 0FFh, но незатронутым остаётся четвёртый байт заголовка. Принимая во внимание, что NewsMaster работает в режиме 640x200, можно предположить, что четвёртый байт также относится к ширине рисунка, ибо 200 умещается в байт (максимальное значение байта 0FFh – 255), а 640 – нет. Эта же последовательность, кстати, подсказывает нам, где искать второй пропущенный байт – в конце изображения. Можно взять первое изображение и посмотреть на его начало и конец. В программе видно, что **Masks** имеет отступы (пустое место) в самой верхней и нижней строках изображения по горизонтали. Т.е. оно должно начинаться и заканчиваться на одинаковые байты. Из приведённого в начале заголовка мы видели, что оно начинается на нули, следовательно, и заканчиваться должно тоже на нули. Однако если мы перейдём на смещение второго рисунка и отступим на 16 (010h) байт назад, чтобы посмотреть на хвост первого, то увидим такую картину:

```
OFFS: 00 00 02 41 - 10
DATA: 07 FF 00 00 03 00 00 00 00 00 00 00 00 00 00 73
```

Последний байт 073h, а не ноль, и он здесь явно лишний.

Здесь же обратим внимание, что **Masks** имеет белый фон, а, значит, в библиотеке клипарта, как и в драйвере для печати из прошлой статьи, цвета обращены: 0 – это белый (пустое место), а 1 – чёрный (цвет напечатанной точки).



NewsMaster 1.0, изображение **Masks** из библиотеки клипарта **NEWS**

Таким образом, окончательный формат одного изображения в библиотеке клипарта выглядит так:

byte rowsize – количество байт в одной строке пикселей;

byte height – высота рисунка;

word width – ширина рисунка;

byte [rowsize * height] – графические данные рисунка;

byte – неизвестно.

Последний байт не используется (NewsMaster пропускает его при чтении, а утилита **CAPTURE.EXE**, речь о которой пойдёт ниже, оставляет там произвольное значение), поэтому и узнать, за что он отвечает, не представляется возможным. Будем, как того требует хорошая практика, заполнять его нулём и считать зарезервированным.

Разбор форматов файлов библиотек стоит дополнить упоминанием о том, что все они, кроме *.SHP, опциональны. Файлы *.SDX нужны для ускорения загрузки, чтобы программе не приходилось разбирать вручную формат *.SHP в поисках начала изображений. Файлы *.SDR нужны, чтобы ориентироваться среди изображений в алфавитном списке программы, ведь при его отсутствии имена будут

отображаться как **#001**, **#002**, **#003** и т.д. Поэтому и полагаться на существование каких-либо файлов, кроме *.SHP, не стоит. Здесь же можно добавить, что при отсутствии *.SDX NewsMaster начинает самостоятельно искать изображения внутри *.SHP в цикле с примерно таким условием (NewsMaster 1.0):

```
while (
 (read(file,head,sizeof(head))==sizeof(head))
 && (((head.width+7)/8)==head.rowsize)
){...}
```

Т.е. заголовок (4 байта – первые 3 поля из структуры выше) должен быть успешно и полностью прочитан и количество байт в строке должно совпадать с рассчитанным. Поэтому первое смещение в *.SDX-файле всегда ноль. Но, теоретически, *.SHP можно создать таким образом, чтобы условие выше не срабатывало (например, добавив в начало файла специально созданную «заглушку» в 4 байта), тогда такая библиотека не будет читаться вообще, без *.SDX, указывающего на начало изображений.

*Важно: количество записей в *.SDR- и *.SDX-файлах должно совпадать, в противном случае возможно аварийное завершение работы NewsMaster!*

Вероятно, расширения файлов были образованы от следующих сокращений:

SHP – SHaPes (рисунки);

SDX – Shapes inDeX (индекс рисунков);

SDR – Shapes DescRiption (описание рисунков).

Теперь разберёмся с граничными условиями *.SHP-формата.

Под ширину отводится слово (2 байта), т.е. теоретически, при представлении без знака, она может принимать максимальное значение в 65535 пикселей (0FFFFh), но нас ограничивает размер строки в один байт (0FFh – 255), из чего получаем: 255 * 8 = 2040. Т.е. рисунок не может превышать 2040 пикселей в ширину. Но и 2040 пикселей много – в прошлой статье мы узнали, что ширина страницы документа фиксирована и составляет 960 пикселей, так что делать изображение больше (хотя NewsMaster и будет с ним работать) смысла тоже нет – не влезет на страницу и часть останется за краем. Отсюда видно, что размер одного рисунка не может превышать 960x255 пикселей или (960 / 8) * 255 = 30600 байт. Этот размер нам будет разумно принять за верхнюю границу.

И здесь же своевременно будет задать вопрос: а какое максимальное количество изображений NewsMaster позволяет хранить в одной библиотеке? Можно попробовать прикинуть эмпирически. Максимальный размер непрерывного блока памяти, который можно выделить стандартными средствами DOS в реальном режиме, равен 65536-8 = 65528 байт (0FFFFh + 1 – 8) [2].

Из чего получим:

1) Файл **NEWS.SHP** больше 65528, следовательно, он не загружается в память целиком и изображения подгружаются оттуда по одному. Т.е. смысла опираться на размер этого файла нет.

2) Файл **NEWS.SDX** содержит смещения и грузится в память весь. Т.е. максимальный размер будет 65528 / 4 = 16382 записей. Пока что возьмём эту величину за верхнюю границу.

3) Файл **NEWS.SDR** содержит имена и, при максимальном длине имени в 16 байт, получим 65528 / 16 = 4095 записей (остаток отброшен) – верхняя граница уменьшилась.

Но при попытке создать тестовую библиотеку с таким количеством записей будет выводиться сообщение о нехватке памяти. Это связано с тем, что для работы с описаниями изображений в NewsMaster версии 1.0 отведён статический буфер всего в 5680 байт [3]. При этом 82 байта оттуда используются под нужды самой программы, итого: 5680 - 82 = 5598 байт. Но и это ещё не всё – каждое описание рисунка в буфере размещается динамически в виде структуры, состоящей из 6 байт служебных данных, строки описания изображения, завершающего нуля ASCIIZ и ещё одного байта, если размер всей структуры оказался нечётным (выравнивание).

Т.е. размер структуры у имени в 16 символов будет: 6 + 16 + 1 + 1 = 24 байта.

[2] Безусловно, можно выделять несколько блоков или использовать связанные списки из отдельных небольших элементов, но NewsMaster использует простой подход.

[3] Как и в случае с драйвером для принтера, описанным в прошлой статье, мы будем писать универсальную программу с поддержкой любой версии NewsMaster, поэтому даже если в более поздних версиях этот буфер и был увеличен, то для нас это не будет иметь значения.

А размер, скажем, для 7 символов окажется: 6 + 7 + 1 + 0 = 14 байт.

Формула для пересчёта длинны строки в размер структуры:

size = len + 8 - (len & 1)

Поэтому количество изображений, которые можно разместить в одной библиотеке, будет напрямую зависеть от суммарного размера всех структур. Тем не менее, и здесь можно посчитать верхнюю границу – при размере всех описаний в библиотеке в один символ мы получим (NewsMaster будет работать с дублирующимися описаниями изображений, но уже упомянутая утилита **CAPTURE.EXE** явно требует задавать имя, отличное от уже существующего в библиотеке):

5598 / (6 + 1 + 1 + 0) = 699 (остаток отброшен)

Таким образом, в одной библиотеке не может быть более 699 изображений.

Точно также можно подсчитать и нижнюю границу, когда каждое описание занимает целиком все 16 символов:

5598 / (6 + 16 + 1 + 1) = 233 (остаток отброшен)

Т.е. в три раза меньше, чем верхняя граница.

Дабы не смущать будущих пользователей программы сложными объяснениями, почему в одном случае они могут сохранить практически 700 изображений, а в другом – в три раза меньше, возьмём нижнюю границу в качестве предела – это сократит и упростит код конвертера, т.к. нам не нужно будет суммировать размер всех строк в переводе на структуры. Заодно опустим границу до 200 ровно – как показала практика, чисто психологически, к круглым числам вопросов возникает меньше. Если же для кого-то и 200 изображений окажется мало [4], то всегда можно будет дополнительно создать ещё новых библиотек.

Из установленного нами предела также становится видно что:

200*16=3200

200*4=800

Иными словами, файл *.SDR не может превышать 3200, а *.SDX 800 байт в размере.

Кто-то может спросить, а зачем нужны все эти подсчёты? Во-первых, это можно использовать, например, для первичной проверки входных файлов на корректность, а, во-вторых, для создания статических буферов в конвертере под все необходимые данные, что полностью исключит динамическое выделение памяти, возню с указателями, слежение за утечками и другие вещи, отсутствие которых, опять же, в итоге сократит и упростит код.

Отдельно стоит сказать про формат, создаваемый утилитой **CAPTURE.EXE**, поставляющейся вместе с NewsMaster версии II [5], в котором поддержка этого формата и была добавлена. Упомянутая утилита позволяет делать чёрно-белые снимки экрана, сохраняя их в библиотеку с именем **CAPTURE.SHP** и сопутствующими файлами **CAPTURE.SDX** и **CAPTURE.SDR**. Последние два файла ничем не отличаются от уже разобранных выше, а вот формат **CAPTURE.SHP** совершенно иной. Его полное описание будет дано ниже, потому что, во-первых, он куда более комплексный, а, во-вторых, дополнительную сложность вносит то, что утилита **CAPTURE.EXE** не заполняет некоторые поля структуры заголовка, оставляя их нулями, но вот NewsMaster использует их при вычислении параметров изображения.

Каждое изображение в **CAPTURE.SHP** имеет следующий формат:

dword zero – всегда ноль, сигнатура нового формата (подробности ниже);

word coffx – коэффициент масштабирования по ширине;

word coffy – коэффициент масштабирования по высоте;

word left – левая координата рисунка;

word top – верхняя координата рисунка;

word right – правая координата рисунка;

word bottom – нижняя координата рисунка;

word unused – не используется (всегда ноль);

dword size – размер всего рисунка в байтах;

byte [size] – графические данные рисунка.

[4] Для сравнения: среди всех версий NewsMaster максимальное количество изображений, размещённых в одной библиотеке, не превышает 151 – т.е. наш конвертер будет покрывать в том числе и размеры стандартных библиотек.

[5] Захват изображения ею можно делать и в старом формате (переключается по **Ctrl+P** – PrintMaster Capture), но почему-то только фиксированного размера в 88x52 пикселя.

Все типы слов (**word**) в данной структуре обрабатываются со знаком.

В новом формате **zero** (первое двойное слово) всегда 0, т.е. каждый рисунок начинается с четырёх нулевых байт. Это оригинальное, но простое решение позволяет легко отличить старый формат от нового, потому как в старом заголовок целиком состоит из 4-х байт, но ни одно из значений там (ширина, высота, количество байт в строке) не может быть нулём.

Параметры рисунка, записанного в файле, считаются следующим образом:

Ширина:

image_width = right - left;

Высота:

image_height = bottom - top;

Обратите внимание, что, например, в отличие от PCX и ряда других форматов, содержащих координаты изображения, здесь к разности не добавляется единица. После этого к полученным размерам изображения в файле применяются коэффициенты **coffx** и **coffy** для масштабирования перед выводом:

scaling_width = (image_width * 360) / coffx;
scaling_height = (image_height * 360) / coffy;

Важные моменты:

1. **CAPTURE.EXE** всегда заполняет **top** и **left** значениями 0, так что в **right** и **bottom**, по сути, всегда находятся ширина и высота рисунка.

2. **coffx** и **coffy** по умолчанию равны 120 и 72 (078h и 048h соответственно), но их можно изменить на произвольные через ключи командной строки, например вот так, чтобы рисунок не масштабировался, а выводился как есть:

```
CAPTURE.EXE /X360 /Y360
```

3. **scaling_width** и **scaling_height** вычисляются со знаком, так что если результат деления (целая часть) не умещается в слово со знаком, то NewsMaster будет аварийно завершать работу при загрузке такого изображения с ошибкой деления на ноль (стандартная ошибка при переполнении) – иными словами, результат деления не должен превышать 32767 (07FFFh). Отсюда видно, что:

32767 >= (image * 360) / coff
coff * 32767 >= image * 360
(coff * 32767) / 360 >= image
coff * (32767 / 360) >= image
coff * 91.019(4) >= image
coff >= image / 91.019(4)

Т.е. ширина или высота, делённая на 91 (дробная часть отброшена), не должны превышать соответствующего коэффициента:

coffx >= image_width / 91
coffy >= image_height / 91

И последнее важное отличие формата – теперь вместо размера одной строки хранится размер всего рисунка.

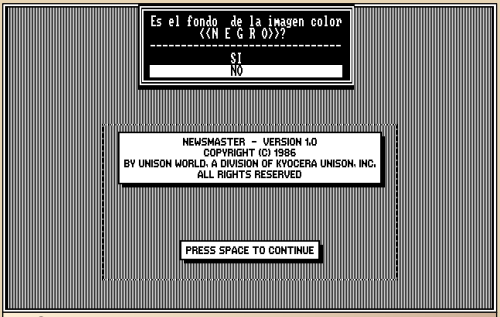
Для обратной совместимости со старыми версиями NewsMaster, наш конвертер будет поддерживать этот формат только для экспорта в BMP, но не обратно.

Очевидно, что граничные условия этого формата изменились по сравнению с предыдущим в большую сторону. Вместо эмпирических прикидок по максимальным величинам здесь будет удобнее зайти со стороны вопроса о том, кто и как этот формат создаёт – т.е. исходить из ограничений уже существующих в реальности программ, способных с ним работать.

Утилита **CAPTURE.EXE**, сохраняющая в этот формат, работает корректно только в графических видеорежимах BIOS из диапазона 00h – 11h (не работает корректно в 13h и почему-то не поддерживает работу в режиме 12h), а т.к. максимальное разрешение при этом 640x480 у режима 11h, то получим:

(640 / 8) * 480 = 38820

38820 больше, чем 30600, так что и верхняя граница буфера под размер рисунка у нас поднялась.



Сохранение рисунка при помощи утилиты **CAPTURE.EXE**

И несколько слов о том, как происходит сохранение рисунка в **CAPTURE.EXE**. После запуска этой утилиты необходимо в графическом режиме нажать **Alt+G**, стрелками курсора указать начало изображения, нажать **Enter**, указать противоположный угол и снова **Enter**. После этого нужно ответить на два вопроса и ввести описание рисунка для файла библиотеки **CAPTURE.SDR**. Вопросы же на испанском языке и звучат так:

Первый вопрос:

```
Es el fondo  de la imagen color <<N E G R O>>?
                    SI / NO
(Цвет фона изображения чёрный? Да / Нет)
```

Второй вопрос:

```
Como quiere grabar la Imagen?
(Как необходимо сохранить изображение?)
```

Первый ответ:

```
Alterado pero listo para la PRINTER.
  (Изменить для печати на принтере)
```

Второй ответ:

```
Tal cual aparece en el monitor ....
  (Оставить, как выводится на экране)
```

В первом вопросе нужно ответить отрицательно (нижний вариант – **NO**), в противном случае в сохраняемом изображении программа обратит цвета. Во втором вопросе также нужен нижний ответ (как на экране), если необходимо сохранить изображение как есть (не забываем про **/X360 /Y360**, чтобы при открытии в NewsMaster оно также не масштабировалось). Изменения для печати (первый вариант ответа) заключаются в том, что высоту рисунка (ширина всегда остаётся неизменной) специальными коэффициентами пытаются вытянуть или сжать. Массив коэффициентов статически прописан в **CAPTURE.EXE** и зависит только от текущего видеорежима. Так, например, для разрешений 320x200 и 640x200 коэффициенты будут одинаковыми – 13 и 10, для 640x350 – 74 и 100, а для 640x480 – 54 и 100:

(200 * 13) / 10 = 260

(350 * 74) / 100 = 259

(480 * 54) / 100 = 259 (остаток отброшен)

Т.е. максимальная высота искусственно подгоняется под значение 260. Пусть здесь не смущает 259, потому что для работы с дробными числами в 1988 году был необходим либо сопроцессор, либо специальная библиотека, его эмулирующая. Поэтому решение было сделано приближённо через умножение и деление целыми числами. При этом по указанным коэффициентам пересчитывается любая высота рисунка в режиме для печати. Так, например, рисунок 45x50 превратится в 45x37 в режиме 640x350 ((50 * 74) / 100 = 37).

Но важно здесь другое – хотя коэффициенты и увеличивают 640x200 до 640x260, но не делают того же с 640x480, превращая его в 640x259, что оставляет рассчитанные нами ранее величины граничных условий формата в силе (их не придётся увеличивать ещё больше).

```
nn offs vseg mul div scw sch max m s info
00 0376 0000   1   1   0  25   25 T # 40x25 16 grey
01 0376 0000   1   1   0  25   25 T # 40x25 16
02 0376 0000   1   1   0  25   25 T # 80x25 16 grey
03 0376 0000   1   1   0  25   25 T # 80x25 16
04 039D BB00  13  10 320 200 260 G + 320x200 4
05 039D BB00  13  10 320 200 260 G + 320x200 4 grey
06 03C4 B000  13  10 640 200 260 G + 640x200 2
07 03EB 0000 100 119 720 348 292 T # 80x25 2     ; 720/292 = 640/260 (~2.46)
08 FFFF ---- --- --- --- --- --- G - PCjr mode
09 FFFF ---- --- --- --- --- --- G - PCjr mode
0A FFFF ---- --- --- --- --- --- G - PCjr mode
0B FFFF ---- --- --- --- --- --- G - reserved; EGA BIOS internal
0C FFFF ---- --- --- --- --- --- G - reserved; EGA BIOS internal
0D 0412 A000  13  10 320 200 260 G + 320x200 16
0E 0439 A000  13  10 640 200 260 G + 640x200 16
0F 0460 A000  74 100 640 350 259 G + 640x350  2
10 0460 A000  74 100 640 350 259 G + 640x350 16
11 0487 A000  54 100 640 480 259 G + 640x480  2
12 FFFF ---- --- --- --- --- --- G - 640x480 16  ; why this mode unsupported?
13 04AE A000  13  20 320 200 130 G * 320x200 256 ; div=20 probably typo of 10
s = state for mode    + supported     - unsupported    # blocked    * invalid
nn = mode number (hex)    max = (sch * mul) / div    m = mode (T)ext or (G)raph
custom columns "max", "m", "s" and "info" not present inside CAPTURE.EXE file
C:\NEWS2>_
```

Список разрешений из **CAPTURE.EXE** v1.1; заблокированные из них (метка #) содержат ошибки

Разобравшись со всеми форматами и граничными условиями, мы, наконец-то, можем приступить к написанию конвертера клипарта.

Чтобы не сильно усложнять код, но показать основные приёмы работы с библиотеками, наша программа будет состоять из минимального набора функций:

– вывод списка всех изображений с порядковым номером и именем;

– извлечение любого изображения в BMP-файл по порядковому номеру;

– добавление в конец библиотеки изображения из BMP-файла.

Поэтому такие операции, как замена (обновление) какого-либо изображения, лучше делать через создание резервной копии библиотеки, к которой можно будет откатиться, при необходимости изменения последнего рисунка. А удаление придётся делать через

полный разбор библиотеки на отдельные BMP-файлы и сборку только нужных изображений назад. Это не сильно удобно для интенсивной работы, поэтому лучше создавать с нуля новую библиотеку и добавлять туда все необходимые BMP-файлы через специальный BAT-файл автоматически. Или, при желании, добавить необходимый функционал в код программы.

Все входные и выходные файлы, а также другие параметры, будут передаваться через командную строку. Чтобы и здесь упростить и сократить код, давайте договоримся о следующем простом способе вызова программы:

а) Нет параметров – на экран выводится справка по использованию.

б) Один параметр – имя библиотеки, список изображений которой будет выведен на экран.

в) Два параметра – имя библиотеки и номер изображения оттуда (с единицы), которое будет извлечено в BMP-файл, с автоматически сформированным именем **SHAPE???.BMP**, где **???** – указанный номер (добивается нулями слева).

г) Наконец, три и более параметра – имя библиотеки, куда необходимо добавить новое изображение и имя входного BMP-файла с изображением, а все остальные параметры, начиная с третьего, это текст слов, которые будут объедены через пробел в строку описания для изображения [6]. При этом только в этом режиме отсутствие исходной библиотеки не будет считаться ошибкой, а намерением создать новую с указанным именем.

В коде программы также будет применён список из следующих технических решений:

1) Уже упомянутые статические буферы максимального размера подо все структуры – никакого динамического выделения или освобождения памяти.

2) Для описания структур будут использованы стандартные типы данных **uint8_t**, **int32_t** и тому подобные, но, в силу того что заголовочного файла **stdint.h** ещё не существовало в 1988 году, то подобные вещи будут определены прямо в коде программы, как и типы **BITMAPFILEHEADER** и **BITMAPINFOHEADER**, взятые из **windows.h**.

3) Все важные места будут содержать проверки на ошибки, но только в самом минимальном их количестве, чтобы сильно не загромождать код.

4) Чтобы не разбираться, корректный ли файл *.SDX, при чтении он будет строиться через полный разбор *.SHP-файла, а при изменении – целиком перезаписан с уже построенных данных. Разбор *.SHP-файла будет завершаться при невозможности определить формат изображения, чтобы отсечь мусор в хвосте файлов библиотек (у некоторых версий NewsMaster из Интернета).

5) В файле *.SDR «мусорные» байты после 000h будут «почищены» – заполнены нулями. Также при меньшем количестве записей, чем изображений в *.SHP, недостающие будут заполнены строками «#001», «#002» и т.д. как они выводились бы в списке самого NewsMaster.

6) Экспорт изображений из библиотеки будет производиться сразу в «перевёрнутый» BMP-формат (положительная высота), т.к., в отличие от драйвера для принтера, нам изначально известна конечная высота рисунка и он полностью прочитан в память, что снимает ограничения и позволяет нам работать с изображением как угодно.

7) При этом импорт изображений BMP в библиотеку будет поддерживать оба варианта: как с отрицательной, так и с положительной высотой, а также независимо от индекса чёрного цвета в палитре (необходимость обращать цвета будет определяться по его наличию на первом месте палитры).

```
/*
 NewsMaster / PrintMaster shapes library tool
 (c) SysTools 2019
 http://systools.losthost.org/

   Both shapes format supported: original
(extract, append) and from CAPTURE.EXE tool
(extract only).
 All [u]int<8/16/32>_t types declaration taken
out from <stdint.h> header file which not
```

[6] В командной строке DOS, в отличие от Windows, ещё не обрабатывались двойные кавычки, так что и обрамлять ими строку, содержащую пробелы, чтобы она считалась единым аргументом, бесполезно.

existed yet back in 1988 and also BITMAPFILEHEADER and BITMAPINFOHEADER declarations which was taken out from the <windows.h> header file.

Note that 32bit types may need tweaking for 64bit compilers since "long" are the only 32bit type on 16bit compiler.

This tool utilizes only static buffers for all the work - no memory allocated or freed in the code below.

```
 Borland Turbo C Compiler 2.01 (DOS16)
 TCC -w -g1 -O -Z shapelib.c

 GNU C Compiler 3.2 (Win32)
 GCC -Wall -pedantic -Werror -Os -s -o shapelib
shapelib.c
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <io.h>

/* required types, DOS16 / Win32 compatible */
typedef  signed char  int8_t;
typedef unsigned char  uint8_t;
typedef  signed short  int16_t;
typedef unsigned short uint16_t;
/* note: 32 bit types may need tweaking for x64
                compilation */
typedef  signed long  int32_t;
typedef unsigned long  uint32_t;

#define BMP_TYPE_BM 0x4D42
#define SDR_NAME_SZ 16

#pragma pack(push, 1)
typedef struct {
 uint16_t bfType;
 uint32_t bfSize;
 uint16_t bfReserved1;
 uint16_t bfReserved2;
 uint32_t bfOffBits;
} BITMAPFILEHEADER;

typedef struct {
 uint32_t biSize;
 int32_t  biWidth;
 int32_t  biHeight;
 uint16_t biPlanes;
 uint16_t biBitCount;
 uint32_t biCompression;
 uint32_t biSizeImage;
 int32_t  biXPelsPerMeter;
 int32_t  biYPelsPerMeter;
 uint32_t biClrUsed;
 uint32_t biClrImportant;
} BITMAPINFOHEADER;

typedef struct {
 BITMAPFILEHEADER hdr_file;
 BITMAPINFOHEADER hdr_info;
 uint32_t      hdr_cpal[256];
} bmp_head;

typedef struct {
 uint32_t zero;
 int16_t coffx;
 int16_t coffy;
 int16_t left;
 int16_t top;
 int16_t right;
 int16_t bottom;
 int16_t unused;
 uint32_t size;
} cpt_head;

typedef struct {
 uint8_t rowsize;
 uint8_t height;
 uint16_t width;
} shp_head;

typedef struct {
 char name[SDR_NAME_SZ];
} sdr_item;

typedef struct {
 uint32_t offs;
} sdx_item;
#pragma pack(pop)
    /* set fixed limit for max images per one
                    library */
#define MAX_ITEM 200
  /* set shp max supported width and height */
#define SHP_MAX_WIDTH  960U
#define SHP_MAX_HEIGHT 255U
  /* set cpt max supported width and height */
#define CPT_MAX_WIDTH  640U
#define CPT_MAX_HEIGHT 480U
 /* max image data size in bytes 960x255 (30600
                    bytes)  */
#define MAX_DATA_SHP (((MAX_WIDTH + 7) / 8) *\
                        MAX_HEIGHT)
   /* cpt format can hold larger images and max
resolution where CAPTURE.EXE still can be used
          are 640x480 (38400 bytes) */
#define MAX_DATA_CPT (((CPT_MAX_WIDTH + 7) / 8)\
                      * CPT_MAX_HEIGHT)
```

```
/* max of two buffers */
#define MAX_DATA MAX_DATA_CPT
  /* file types index in 4-char string with
               extensions */
#define FILE_TYPE_SHP 0
#define FILE_TYPE_SDR 1
#define FILE_TYPE_SDX 2
/* library file extensions */
const static char ext_list[] =
              ".SHP\0.SDR\0.SDX";
/* buffer for max offs data */
static sdx_item sdx_list[MAX_ITEM];
/* buffer for max names data */
static sdr_item sdr_list[MAX_ITEM];
/* buffer for max image data */
static uint8_t data[MAX_DATA];
/* bitmap file headers */
static bmp_head bmp_data;
 /* global library filename for open (DOS 8.3
                 ASCIIZ) */
static char lib_name[13];
/* max items in shape library */
static int count;


/* calculate BMP image size */
static int32_t CalcBMPSize(int32_t w, int32_t
               h, uint16_t b) {
  w = ((((w * b) + 31) & ~31) >> 3);
  w *= (h < 0) ? (-h) : h;
  return(w);
}

 /* universal bitmap header initialization
                 routine */
static uint32_t InitBMPHeaders(bmp_head *bh,
 int32_t w, int32_t h, uint16_t b) {
uint32_t l;
 /* header size */
 l =
  /* file header */
  sizeof(bh->hdr_file) +
  /* bitmap header */
  sizeof(bh->hdr_info) +
  /* palette size for index color images */
  ((b <= 8) ? (sizeof(bh->hdr_cpal[0])<<b): 0);
 if (bh) {
  memset(bh, 0, sizeof(bh[0]));
  bh->hdr_info.biSize = sizeof(bh->hdr_info);
  bh->hdr_info.biWidth = w;
  bh->hdr_info.biHeight = h;
  bh->hdr_info.biPlanes = 1;
  bh->hdr_info.biBitCount = b;
  bh->hdr_info.biSizeImage =
      CalcBMPSize(w, h, b);
  bh->hdr_file.bfType = BMP_TYPE_BM;
  bh->hdr_file.bfOffBits = l;
    bh->hdr_file.bfSize = l +
     bh->hdr_info.biSizeImage;
 }
 return(l);
}

/* merge strings list to the single string with
            the spaces between */
static void StrListMerge(char *buff, int bmax,
       char *list[], int lmax) {
int i, n;
char *s;
 memset(buff, 0, bmax * sizeof(buff[0]));
 n = 0;
 for (i = 0; i < lmax; i++) {
  for (s = list[i]; *s; s++) {
   if (n == bmax) { break; }
   buff[n] = *s;
   n++;
  }
  if (n == bmax) { break; }
  buff[n] = ' ';
  n++;
 }
          /* add null at string end */
 if (n) {
  n--;
  buff[n] = 0;
 }
 /* trim space at the end if there is not enough
           memory for next word */
 if (n && (buff[n - 1] == ' ')) {
  n--;
  buff[n] = 0;
 }
}

  /* prepare all global static buffers before
                  using */
static void InitData(void) {
 count = 0;
 memset(data, 0, MAX_DATA);
 memset(&bmp_data, 0, sizeof(bmp_data));
 memset(sdx_list, 0, MAX_ITEM *
         sizeof(sdx_list[0]));
 memset(sdr_list, 0, MAX_ITEM *
         sizeof(sdr_list[0]));
}


/* add extension to the base name of the file */
static char *BuildName(char *library, int type)
                          {
int i;
    /* in case of extension - cut it (DOS 8.3
                 filename) */
 for (i = 0; i < 8; i++) {
```

```
  /* end of the string or dot from extension */
  if ((!library[i]) || (library[i] == '.')) {
     break; }
 }
                /* copy name */
 memcpy(lib_name, library, i *
        sizeof(library[0]));
         /* add required extension */
 memcpy(&lib_name[i], &ext_list[type * 5], 5 *
        sizeof(library[0]));
 /* return pointer for easy use in open() since
           it's global anyway */
 return(lib_name);
}


        /* file size helper routine */
static uint32_t FileSize(int fl) {
uint32_t p, l;
 p = tell(fl);
 lseek(fl, 0, SEEK_END);
 l = tell(fl);
 lseek(fl, p, SEEK_SET);
 return(l);
}

/* checks if current position in file points to
         the start of the shp item */
static int IsItemSHP(int fl, uint32_t sz,
                     shp_head *shp) {
uint32_t l;
  /* save current position if invalid format */
 l = tell(fl);
   /* not enough space in file for header */
 if (read(fl, shp, sizeof(shp[0])) !=
     sizeof(shp[0])) { shp = NULL; }
 /* width or height are zero or width more than
          max or invalid rowsize */
 if (shp && ((!shp->width) || (!shp->height) ||
  (shp->width > SHP_MAX_WIDTH) ||
  (((shp->width + 7) / 8) != shp->rowsize)
 )) { shp = NULL; }
 /* check that file has enough space to hold the
                whole image */
 if (shp && sz && ((l + sizeof(shp[0]) +
    (shp->rowsize * shp->height) + 1) > sz)) {
  shp = NULL;
 }
/* invalid format - rewind back file postion */
 if (!shp) { lseek(fl, 1, SEEK_SET); }
 return(shp ? 1 : 0);
}

/* checks if current position in file points to
         the start of the cpt item */
static int IsItemCPT(int fl, uint32_t sz,
                     cpt_head *cpt) {
uint32_t l;
 /* save current position if invalid format */
 l = tell(fl);
   /* not enough space in file for header */
 if (read(fl, cpt, sizeof(cpt[0])) !=
     sizeof(cpt[0])) { cpt = NULL; }
 /* must starts with 4 zero bytes to tell apart
            from the shp header */
 if (cpt && cpt->zero) { cpt = NULL; }

/* calc width and height, note that there is no
+1 here like in some other formats (i.e. width =
                x2 - x1 + 1) */
 if (cpt) {
  cpt->right -= cpt->left; /* width */
  cpt->bottom -= cpt->top; /* height */
  /* just in case */
  cpt->left = 0;
  cpt->top = 0;
 }
 if (cpt && (
  /* width and height can't be negative */
  (cpt->right <= 0) || (cpt->bottom <= 0) ||
  /* or bigger than max */
  (cpt->right > CPT_MAX_WIDTH) || (cpt->bottom
  > CPT_MAX_HEIGHT) ||
  /* coeffs can't be zero or less */
  (cpt->coffx <= 0) || (cpt->coffy <= 0) ||

     /* explanation for this weird check:
NewsMaster II recalculate image screen
dimensions by the next formula:
     width = ((right - left) * 360) / coffx
     height = ((bottom - up) * 360) / coffy
if after division result will be too big to fit
in word with sign (07FFFh) the program will
crash with the fatal error "divide by zero"
07FFFh = 32767, so let's see what's need to be
checked:
     32767 >= (dimension * 360) / coff
     coff * 32767 >= dimension * 360
     (coff * 32767) / 360 >= dimension
     coff * 91 >= dimension
     coff >= dimension / 91 */
  (cpt->coffx < (cpt->right / 91)) ||
  (cpt->coffy < (cpt->bottom / 91)) ||
  /* check the correct whole image size */
  ((((cpt->right + 7) / 8) * cpt->bottom) !=
      cpt->size)
 )) { cpt = NULL; }
 /* check that file has enough space to hold the
                whole image */
 if (cpt && sz && ((l + sizeof(cpt[0]) +
     cpt->size) > sz)) { cpt = NULL; }
/* invalid format - rewind back file postion */
 if (!cpt) { lseek(fl, 1, SEEK_SET); }
 return(cpt ? 1 : 0);
}
```

```
static int IsFileBMP(int fl, uint32_t sz,
                     bmp_head *bmp) {
int32_t h;
uint32_t l;
 /* bitmap header size */
 l = InitBMPHeaders(NULL, 0, 0, 1);
 /* not enough space in file for header */
 if (read(fl, bmp, (int) l) != l) { bmp = NULL; }
 /* negative height */
 h = bmp->hdr_info.biHeight;
 h = (h < 0) ? (-h) : h;
 /* check BMP format */
 if (bmp && (
  /* check signature */
  (bmp->hdr_file.bfType != BMP_TYPE_BM) ||
  /* disk file can't be less than size in header
           (but fine if bigger) */
  (bmp->hdr_file.bfSize > sz) ||
  /* check v3 header */
  (bmp->hdr_info.biSize !=
   sizeof(bmp->hdr_info)) ||
  /* check width and height */
  (bmp->hdr_info.biWidth <= 0) || (!h) ||
 /* and not exceed max allowed image resolution
                in shp format */
  (bmp->hdr_info.biWidth > SHP_MAX_WIDTH) ||
                      (h > SHP_MAX_HEIGHT) ||
  /* planes must be 1 */
  (bmp->hdr_info.biPlanes != 1) ||
  /* BPP must be 1 */
  (bmp->hdr_info.biBitCount != 1) ||
  /* compression unsupported */
  (bmp->hdr_info.biCompression)
 )) { bmp = NULL; }
 /* now the tricky part: since biSizeImage can
 be 0 for non-compressed images - recalc it */
 if (bmp) {
  bmp->hdr_info.biSizeImage = CalcBMPSize(
   bmp->hdr_info.biWidth,
   bmp->hdr_info.biHeight,
   bmp->hdr_info.biPlanes
  );
 }
/* last check for correct offset to the image */
 if (bmp && (
  /* in theory image can start inside header but
                it's weird */
  (bmp->hdr_file.bfOffBits < l) ||
 /* there should be enough space at image offset
           to hold image data */
  ((bmp->hdr_file.bfOffBits +
    bmp->hdr_info.biSizeImage) > sz)
 )) { bmp = NULL; }
 /* no need to rewind file ponter on error since
       testing standalone BMP file */
 return(bmp ? 1 : 0);
}
```

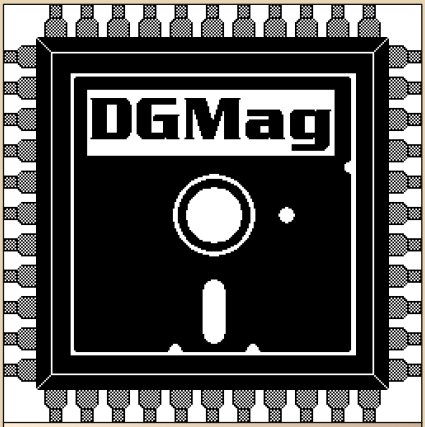
```
/* load data from library file */
static void LoadData(char *library) {
shp_head shp;
cpt_head cpt;
uint32_t sz;
int fl, i, j;
char c;
 /* open shapes library */
 fl = open(BuildName(library, FILE_TYPE_SHP),
     O_RDONLY | O_BINARY);
 if (fl != -1) {
  /* get file size */
  sz = FileSize(fl);
 while ((tell(fl) < sz) && (count < MAX_ITEM)) {
   sdx_list[count].offs = tell(fl);
/* read as SHP format first (smaller header) */
   if (IsItemSHP(fl, sz, &shp)) {
    /* skip image data */
    lseek(fl, tell(fl) + (shp.rowsize *
           shp.height) + 1, SEEK_SET);
   } else {
 /* read as CPT format next (bigger header) */
    if (IsItemCPT(fl, sz, &cpt)) {
     /* skip image data */
    lseek(fl, tell(fl) + cpt.size, SEEK_SET);
    } else {
     /* something wrong - break */
     break;
    }
   }
   count++;
  }
  /* for append */
  if (count < MAX_ITEM) {
   sdx_list[count].offs = tell(fl);
  }
  close(fl);
 }

 /* at least one image exists */
 if (count) {
  /* open descriptions file */
 fl = open(BuildName(library, FILE_TYPE_SDR),
      O_RDONLY | O_BINARY);
  if (fl != -1) {
   /* calc approximate amount items in file */
   sz = FileSize(fl) / sizeof(sdr_list[0]);
   /* can't be more than total items */
   sz = (sz > count) ? count : sz;
   /* read items descriptions */
   read(fl, sdr_list, ((int) sz) *
        sizeof(sdr_list[0]));
   close(fl);
  }
  /* fix item names */
  c = 0;
  for (i = 0; i < count; i++) {
```

```
   /* end reached if 0x00 or 0x1A as in NewsMaster
                loader code */
    if ((sdr_list[i].name[0] == 0x00) ||
      (sdr_list[i].name[0] == 0x1A)) { c = 1; }
   /* clear text items from junk bytes after zero
                tail char */
    if (!c) {
     c = 1;
     for (j = 0; j < SDR_NAME_SZ; j++) {
      sdr_list[i].name[j] *= c;
      c &= sdr_list[i].name[j] ? 1 : 0;
     }
     c = 0;
    } else {
      /* clear tail junk chars if any here from
                the sdr file */
     memset(sdr_list[i].name, 0, SDR_NAME_SZ *
            sizeof(sdr_list[0].name[0]));
       /* fill with same default names as in
        NewsMaster: #001, #002, ... */
     sprintf(sdr_list[i].name, "#%03d", i + 1);
    }
   }
  }
}

  /* output item names with thier order number
               inside library */
static void ListData(void) {
char s[SDR_NAME_SZ + 1];
int i;
 s[SDR_NAME_SZ] = 0;
 for (i = 0; i < count; i++) {
  memcpy(s, sdr_list[i].name, SDR_NAME_SZ *
         sizeof(s[0]));
  printf("%03u-%s\n", i + 1, s);
 }
}

/* extract shape by number */
static void ExtractShape(char *library, int n) {
char name[SDR_NAME_SZ + 1];
shp_head shp;
cpt_head cpt;
int fl, i;
 /* open shapes library */
 fl = open(BuildName(library, FILE_TYPE_SHP),
     O_RDONLY | O_BINARY);
 if (fl != -1) {
  lseek(fl, sdx_list[n - 1].offs, SEEK_SET);
  /* detect image format */
  if (!IsItemCPT(fl, 0, &cpt)) {
   if (IsItemSHP(fl, 0, &shp)) {
    /* unification for later usage */
    memset(&cpt, 0, sizeof(cpt));
    cpt.right = shp.width;
    cpt.bottom = shp.height;
    cpt.size = shp.rowsize * shp.height;
   } else {
    /* error */
    cpt.size = 0;
   }
  }
  if (cpt.size <= MAX_DATA) {
   /* read data */
   read(fl, data, (int) cpt.size);
  } else {
   /* error */
   cpt.size = 0;
  }
  close(fl);

 /* no error */
 if (cpt.size) {
  /* output shape name */
  memcpy(name, sdr_list[n - 1].name,
         SDR_NAME_SZ * sizeof(name[0]));
  name[SDR_NAME_SZ] = 0;
  printf("%03u|%s -> ", n, name);
  /* generate output file name */
  sprintf(name, "SHAPE%03u.BMP", n);
  printf("%s\n", name);
  /* create output file */
  fl = open(name, O_RDWR | O_BINARY | O_CREAT
      | O_TRUNC, S_IREAD | S_IWRITE);
  if (fl != -1) {
   /* invert image colors for BMP format */
   for (i = 0; i < cpt.size; i++) {
    data[i] = ~data[i];
   }
   /* init bitmap headers */
   InitBMPHeaders(&bmp_data, cpt.right,
                  cpt.bottom, 1);
   /* add white palette color */
   bmp_data.hdr_cpal[1] = 0xFFFFFFUL;
    /* actual headers size (include palette)
          will be in bfOffBits */
   write(fl, &bmp_data, (int)
         bmp_data.hdr_file.bfOffBits);
   /* row size */
   cpt.unused = cpt.size / cpt.bottom;
   /* for padding */
   cpt.zero = 0;
   /* padding size */
   n = (4 - (cpt.unused % 4)) % 4;
   /* write image data by row (bottom-top
                image) */
   while (cpt.bottom--) {
    /* write image row */
   write(fl, &data[cpt.bottom * cpt.unused],
           cpt.unused);
    /* write bmp padding bytes */
    write(fl, &cpt.zero, n);
   }
```

```
    close(fl);
    printf("\ndone\n\n");
   } else {
    printf("Error: can't create output"
           " file.\n\n");
   }
  } else {
   printf("Error: invalid shape or too big to"
          " fit in internal buffer.\n\n");
  }
 }
}

/* append new shape to the end of the library */
static void AppendShape(char *library, char
                        *image, char *name) {
uint16_t lb;
shp_head shp;
int fl, i, h;
     /* copy item name (note that offset in
        sdx_list[] was init above) */
 memcpy(sdr_list[count].name, name,
        SDR_NAME_SZ * sizeof(name[0]));
 printf("%03u|%s <- %s\n", count + 1, name,
         image);
 fl = open(image, O_RDONLY | O_BINARY);
 if (fl != -1) {
  /* check for correct BMP file */
  i = IsFileBMP(fl, FileSize(fl), &bmp_data);
  if (i) {
   /* row length (in bytes) for BMP */
   lb = CalcBMPSize(bmp_data.hdr_info.biWidth,
      1, bmp_data.hdr_info.biBitCount);
   /* fill in shape header */
   shp.rowsize=(bmp_data.hdr_info.biWidth +
                7) / 8;
   shp.width = bmp_data.hdr_info.biWidth;
   /* negative height */
   h = (int) bmp_data.hdr_info.biHeight;
   shp.height = (h < 0) ? (-h) : h;
   /* seek to the bitmap data start */
   lseek(fl, bmp_data.hdr_file.bfOffBits,
         SEEK_SET);
   /* read and convert BMP file to SHP */
   while (h) {
    /* instead of disk seeking for bottom-top
  images move pointer to correct address in
                memory */
    read(fl, &data[((h < 0) ? (shp.height + h) :
          (h - 1)) * shp.rowsize],
         shp.rowsize);
    /* skip BMP padding bytes */
    lseek(fl, lb - shp.rowsize, SEEK_CUR);
      /* negative (top-bottom) or positive
         (bottom-top) BMP file */
    h += (h < 0) ? 1 : (-1);
   }
  /* invert colors if needed */
  if (!bmp_data.hdr_cpal[0]) {
   for (i = 0; i < (shp.rowsize * shp.height);
        i++) {
    data[i] = ~data[i];
   }
  }
  /* no errors */
  i = 1;
 }
/* now fl handle can be reused (only one opened
            file at a time) */
 close(fl);


 /* no errors? */
 if (i) {
  /* open file or create if new library */
  fl = count ?
   /* at least one image exists - open existing
              library */
    open(BuildName(library, FILE_TYPE_SHP),
     O_RDWR | O_BINARY) :
   /* no images in library - create new library
              file */
    open(BuildName(library, FILE_TYPE_SHP),
    O_RDWR | O_BINARY | O_CREAT | O_TRUNC,
    S_IREAD | S_IWRITE);
  if (fl != -1) {
   /* seek to the place after last image */
   lseek(fl, sdx_list[count].offs,
         SEEK_SET);
   /* write shape header */
   write(fl, &shp, sizeof(shp));
   /* write image data */
   write(fl, data, shp.rowsize *
         shp.height);
   /* write unknown byte */
   i = 0;
   write(fl, &i, 1);
   close(fl);
  /* always overwrite updated description and
              offset files */
   count++;

   /* description file */
   fl = open(BuildName(library,
       FILE_TYPE_SDR), O_RDWR | O_BINARY |
       O_CREAT | O_TRUNC, S_IREAD | S_IWRITE);
   if (fl) {
    write(fl, sdr_list,
          sizeof(sdr_list[0]) * count);
    close(fl);
   }

   /* offsets file */
  fl = open(BuildName(library, FILE_TYPE_SDX),
```

```
        O_RDWR | O_BINARY | O_CREAT|O_TRUNC,
        S_IREAD | S_IWRITE);
    if (f1) {
     write(f1, sdx_list, sizeof
           (sdx_list[0]) * count);
     close(f1);
    }
    printf("\ndone\n\n");
   } else {
    printf("Error: can't create or open for"
           " write shape library file.\n\n");
   }
  } else {
   printf(
    "Error: not a BMP file, invalid format or"
    " larger than %ux%u pixels.\n\n",
    SHP_MAX_WIDTH, SHP_MAX_HEIGHT
   );
  }
 } else {
  printf("Error: can't open input BMP"
         " file for read.\n\n");
 }
}

int main(int argc, char *argv[]) {
char s[SDR_NAME_SZ + 1];
int n;
 printf("NewsMaster / PrintMaster shapes"
        " library tool\n\n");
 if (argc < 2) {
  printf(
   "Usage: shapelib <library> [...]\n\n"
   "List images in library TEST.SHP (1"
   " argument):\n"
   "  shapelib test\n\n"
   "Extract 4th image from library TEST.SHP"
   " (2 arguments):\n"
   "  shapelib test 4\n\n"
   "Append image to the library TEST.SHP (3 or"
   " more arguments):\n"
   "  shapelib test filename.bmp Item"
   " Description\n\n"
  );
  return(1);
 }

 /* init global structs */
 InitData();

 /* read library data */
 LoadData(argv[1]);

 /* decide what to do */
 switch (argc - 2) {
  case 0:
   if (count) {
    ListData();
   } else {
    printf("Error: can't open or invalid"
           " library file.\n\n");
   }
   break;
  case 1:
   if (count) {
    n = atoi(argv[2]);
    printf("Extract shape number \"%s\""
           " (%d)...\n", argv[2], n);
    if ((n >= 1) && (n <= count)) {
     ExtractShape(argv[1], n);
    } else {
     printf("Error: invalid shape number, must"
            " be in range: 1 <= n <= %u.\n\n",
            count);
    }
   } else {
    printf("Error: can't open or invalid"
           " library file.\n\n");
   }
   break;
  /* >= 2 */
  default:
   if (count < MAX_ITEM) {
       /* merge item name from command line
                  arguments */
    StrListMerge(s, SDR_NAME_SZ + 1, &argv[3],
                 argc - 3);
      /* append image to the end of the current
                 library */
    AppendShape(argv[1], argv[2], s);
   } else {
    printf("Error: too much shapes (>=%u) in"
           " this library, create a new"
           " one.\n\n", MAX_ITEM);
   }
   break;
 }
 return(0);
}
```

Остался последний важный, но упомянутый пока что только поверхностно момент – это соотношение сторон изображения. Чтобы понять, о чём идёт речь, давайте добавим какой-нибудь квадратный рисунок в новую библиотеку и посмотрим, как он будет отображаться в NewsMaster при использовании в документе. Для примера возьмём логотип Downgrade из шапки с сайта журнала. Его нужно будет перевести из 32 бит в 2 бита ч/б, а также уменьшить в силу ограничения по высоте в 255 пикселей.



Логотип журнала в 254x254 ч/б

Для удобства сборки библиотеки клипарта сделаем **BUILDLIB.BAT**-файл такого содержания:

```
@echo off
del shapelib.shp
shapelib.exe shapelib.shp dglogobw.bmp DGMagLogo
```

Так как наш конвертер не умеет заменять и удалять, то мы будем просто каждый раз собирать всю библиотеку заново, стирая предыдущую версию, чтобы изображения не добавлялись в конец к уже существующей.

После создания библиотеки, открываем её в NewsMaster и вставляем рисунок в документ. Нетрудно заметить, что логотип из квадрата превратился в прямоугольник, вытянутый по вертикали. Происходит это из-за того, что в силу особенности печати на матричных принтерах изображение увеличивается по высоте. Чтобы компенсировать это, весь клипарт в NewsMaster изначально сделан «сплюснутым» [7]. Помните, мы выше разбирались с таблицей коэффициентов из утилиты **CAPTURE.EXE**? Квадратным пиксель, обычно, выглядит на разрешении с соотношением сторон 4 к 3, какое было на большинстве мониторов того времени [8]. Из списка **CAPTURE.EXE** там только одно такое разрешение – 640x480 с коэффициентами 54 и 100 или 0,54 (54/100). Пересчитываем:

254 * 0,54 = 137 (остаток отброшен)

Т.е. наше изображение будет квадратным, если мы уменьшим его высоту с 254 до 137 пикселей. Однако при этом на рисунке могут исчезнуть мелкие детали. Чтобы этого не происходило, можно сделать наоборот – увеличить ширину, поменяв коэффициенты местами или поделив на 0,54:

254 / 0,54 = 470 (остаток отброшен)

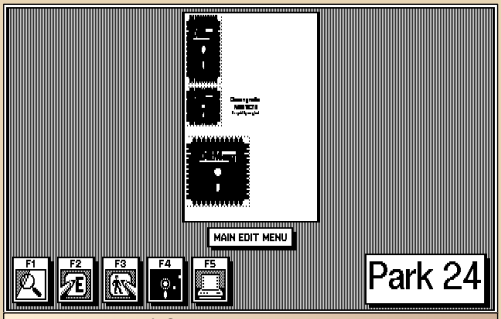
Мелкие детали при этом не исчезнут, но будут слегка искажены из-за того, что коэффициент дробный. В любом случае, при окончательном сведении рисунка его придётся вручную дорабатывать и проверять, как он отпечатывается на реальной бумаге. И виртуальный драйвер для принтера из прошлой статьи этому, увы, сильно не поможет, потому что очень простой и сохраняет изображение как есть, не соблюдая пропорции и не эмулируя многие физические особенности печати.

Попробуем в любом графическом редакторе изменить размер у изображения логотипа журнала, затем сохраним первый вариант рисунка, с уменьшенной высотой, как **DGLOGO_1.BMP**, а второй, с увеличенной шириной, как **DGLOGO_2.BMP** и немного допишем файл **BUILDLIB.BAT**:

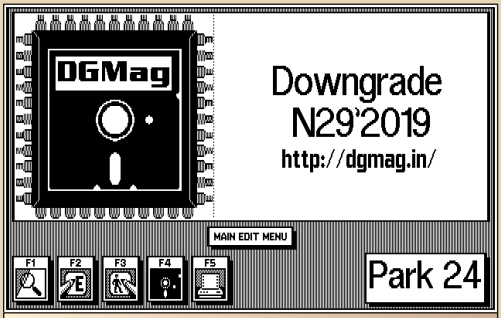
```
@echo off
del shapelib.shp
shapelib.exe shapelib.shp dglogobw.bmp DGMagLogo
shapelib.exe shapelib.shp dglogo_1.bmp DGMagTiny
shapelib.exe shapelib.shp dglogo_2.bmp DGMagWide
```

После чего создадим с его помощью обновлённую библиотеку уже с тремя рисунками, затем вставим их в NewsMaster, чтобы посмотреть на результат.

[7] Наверное, поэтому для работы программы и был выбран режим 640x200, как зрительно наиболее близкий к результату печати.
[8] Иными словами, нарисованный в графическом режиме с таким соотношением сторон четырёхугольник с прямыми углами и одинаковым количеством пикселей по каждой стороне визуально будет выглядеть именно квадратом, а не прямоугольником.



NewsMaster 1.0, все три рисунка: оригинальный, сжатый по высоте, растянутый по ширине



NewsMaster 1.0, сжатый по высоте рисунок крупным планом

Напоследок стоит заметить, что при извлечении и использовании оригинальных изображений из стандартных библиотек NewsMaster на современных мониторах и разрешениях придётся проделывать обратную операцию – т.е. вытягивать рисунки по вертикали делением высоты на 0,54 (от сокращения ширины сильно пострадает качество и без того мелких рисунков).

P.S. Если говорить об изображениях, то стоит упомянуть и файл **NEWS.ICN**, содержащий значки интерфейса программы. Он состоит из последовательно записанных друг за другом изображений следующего формата:

word height – высота рисунка;

word width – ширина рисунка;

byte [((width +7) / 8) * height] – графические данные рисунка.

Этот формат повторяется для каждого изображения до конца файла.

В NewsMaster версии II этот файл начинается с дополнительного слова (2 байта) со значением 0003Dh (61) – вероятно, количество изображений в файле (хотя их там только 60).

Несмотря на параметры перед каждым изображением, они все фиксированного размера 40x20 пикселей, при этом, в отличие от библиотек клипарта и отправляемых на печать данных, хранятся они в представлении для вывода на экран, где бит 0 – чёрный, а 1 – белый.

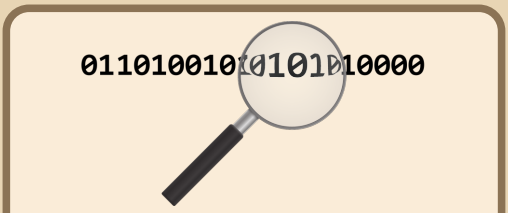
Завершая данную серию статей, необходимо подвести некоторый итог. За годы реставрации Legacy Software (устаревшее программное обеспечение) удалось вернуть к жизни, починить, оживить, а также сделать более комфортной работу у множества приложений. Нет никаких причин, чтобы отказываться от удобной, привычной и проверенной временем программы в угоду современным аналогам (если таковые вообще имеются), многие из которых, к сожалению, обладают невнятным интерфейсом, неоправданно завышенными требованиями к ресурсам или вообще были лишены необходимого функционала. Как показано в статьях на примере NewsMaster, несколько небольших утилит способны облегчить, упростить и адаптировать работу практически любой программы под современные системы и нужды. Равно как и какие-либо неудобства при работе или даже конфликты с современным оборудованием ещё не повод расставаться с программой или искать ей замену. Потому что все эти вещи решаемы и не являются проблемой.

Исходные коды из этой статьи вместе с готовыми скомпилированными программами доступны по ссылке:

http://systools.losthost.org/files/news_prg.zip

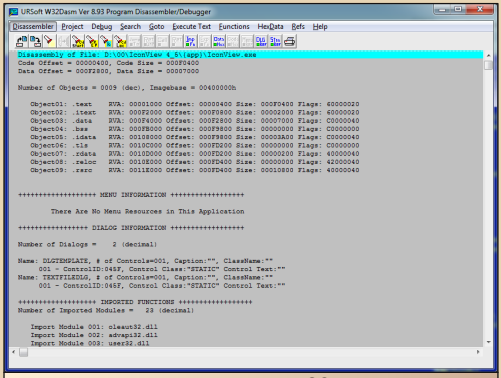
Спасибо Александру «uav1606» за подготовку статей к публикации, а также всему коллективу Downgrade за интересный и познавательный журнал.

Специально для Downgrade N29'2019
© SysTools 2019
http://systools.losthost.org/



Знакомство с исследованием программ

С чем связано исследование программ? Старожилы вспомнят древний сайт fravia.org, на котором было размещено немало статей об исследовании программ. После появления Windows 95 возникла масса разных программ, которые почему-то просили купить полный спектр функций, вложенных в программу. Удивительно, но с помощью забытой программы W32Dasm можно было найти необходимые данные для регистрации – имя и серийный номер.



Вид программы W32Dasm

С помощью этой программы было найдено немало находок. Например, мой знакомый раскопал в игре Heroes of Might And Magic встроенный cheat – если к программе дописать в командную строку **holygrail**, то получится отладочное меню, которое даст массу функций игры. С помощью программы я отучил игру Age of Wonders от проверки наличия CD, чем сделал счастливыми геймеров из старой компании.

«А как же SoftIce?» – слышен робкий голос. Да, был и такой отладчик.

Время пролетело, и изменилось многое в области теории исследования программ – появились альтернативные способы защиты и другие инструментарии. И остались старые программы или игры, которые из-за врождённой жадности просят купить игру, но не сообщают, где и как купить данные, ведь некоторые компании давно развалились.

Что нужно для того, чтобы изучить программу?

Необходимо знать ассемблер x86 и знать, как работает Windows, например, список функций Win32 API.

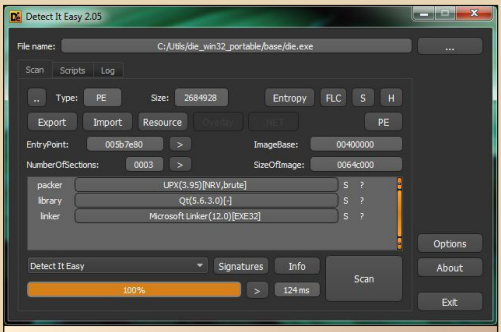
Ассемблер можно изучить самому, есть немало книг, посвящённых вопросам программирования.

Список функций API доступен как справочник **win32.hlp** – Google подскажет, где найти.

Следует заметить, что появилось немало программ, способных осложнить жизнь как взломщику, так и любопытному пользователю. Это упаковщики программ – **UPX**, **FSG** и другие. Или обфускаторы (**ASpack**, **ASprotect**, **Themida**), которые прячут ресурсы программы и меняют способ запуска так, что простым отладчиком найти код будет сложно. Поэтому может получиться так, что загрузив программу в отладчик, новичок увидит кучу мусора.

В любом случае, нужно знать, с чем придётся столкнуться. Подскажет DiE Scan:

https://www.portablefreeware.com/index.php?id=2757



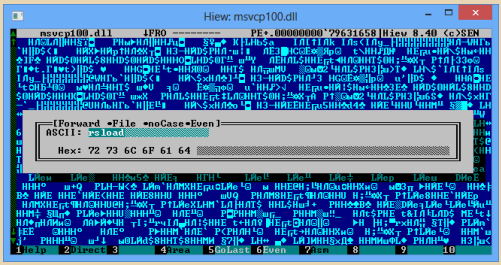
Вид окна программы

Если упаковщик не используется, то часть задачи облегчена. Если используется, то придётся поискать утилиту, которая распакует программу.

Оставлю такой материал на самостоятельное изучение. Например, UPX сам распаковывает сжатую программу (ключ **-d**).

Для изучения понадобятся две программы **OllyDbg** – это отладчик – и **IDA** (Interactive Disassembler), с помощью которого можно бегло посмотреть на исходный дизассемблированный код и найти участки исследования. Преимущество IDA в том, что программа распознаёт участки кода как функции. Например, **new** для Си.

Иногда понадобится внести в код программы изменения, которые можно сделать с помощью Hiew.

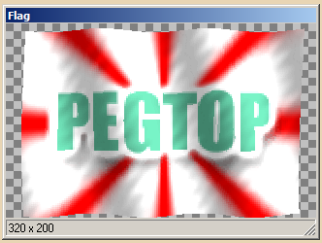


Программа платная, но найти можно в Интернетах.

Теперь примеры.

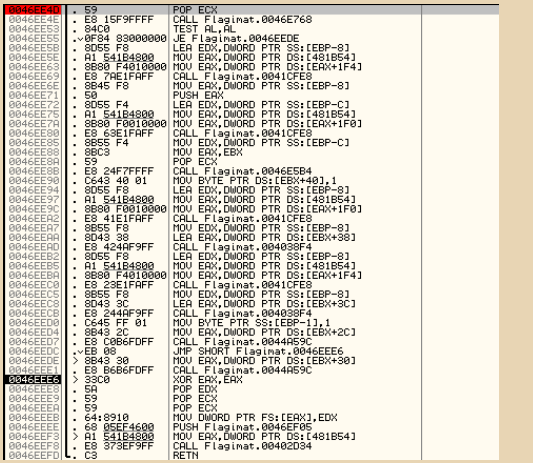
Поиск кода проверки регистрации упрощён поиском в отладчике сообщений вида «Спасибо за регистрацию» или «Код неверен».

Flagimation
(http://pegtop.de/flagimation/)



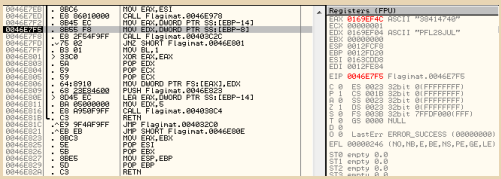
Программа создаёт .gif с изображением флага, но не позволяет сохранить картинку до регистрации.

Просмотр кода в OllyDbg подскажет адрес:



Ставлю breakpoint по адресу **46EE4D** (клавиша **F2**), в программе выбираю регистрацию, в итоге получаю сообщение о неверных данных.

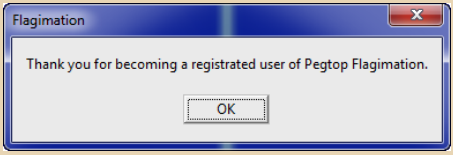
Теперь по шагам, я не стал разбираться с дизассемблером, а посмотрел в отладчике:



По адресу **46E7F5** вызывается процедура, при этом регистры **EAX** и **EDX** указывают на что-то – смотрите на окно регистров справа, OllyDbg сам определяет строки.

Прохожу участки кода, нажав **F8**, получается сообщение о неверных данных.

Ввожу то же самое имя (скрыто из понятных всем соображений) и регистрационный номер, на который указывает **EAX**:

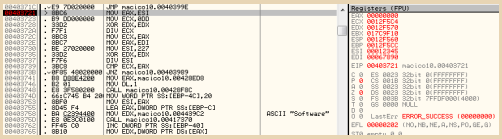


Ура! Теперь функции доступны.

MacIco, XemiCo
(ссылка на Веб-архив)

Этакий древний раритет, который позволяет конвертировать значки формата Macintosh в Windows. Программа не справляется со всеми форматами иконок, техподдержка мне ответила, что продукт больше не поддерживается. На момент написания статьи сайт пропал. Хорошо, руки развязаны. Есть и другие способы конверсии значков, но потренироваться стоило бы.

Просмотр кода и попытки вывели на адрес **403721**:



Здесь **ESI** и **EDI** – шестнадцатеричные значения, которые введены в окно регистрации. Нетрудно понять, что **n1/$DD=n2/$227**, где **n1**, **n2** – два числа.

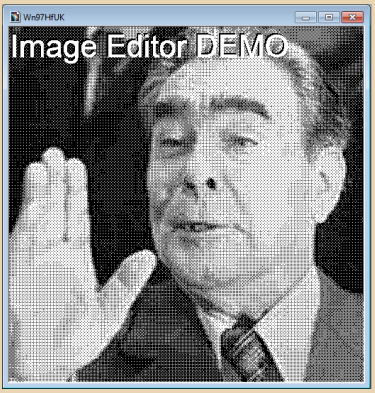
Решить просто: взять шестнадцатеричное значение, поделить нацело на **$DD** и результат умножить на **$227**, перевести в шестнадцатеричную систему.

Данные не указаны, любопытный проделает описанные действия сам (spoiler: начните с **$12345**).

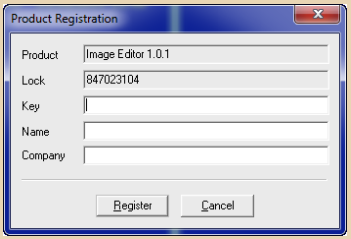
Equinox Software Image Editor 1.0

Нашёл программу 7 лет назад, потому что она использовала полезные функции при обработке изображений, даже не помню, где откопал.

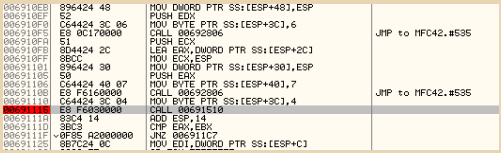
Один минус программы: надпись на изображении.



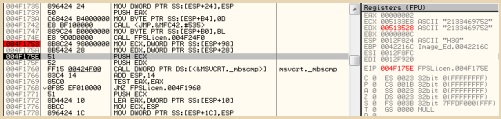
При запуске появляется диалоговое окно регистрации:



Ну хорошо, OllyDbg и гоняю по шагам: на адресе **401CF9** появляется окошко, гоняю по шагам, исследование отняло массу времени и попортило нервов:



Здесь загружается ещё одна библиотека, где проверяется регистрация:



Нетрудно догадаться, где хранятся нужные данные.

Подобных примеров найдётся немало, например, в 123renamer File and MP3 Tag Renamer 2.2 регистрационные данные сначала сохраняются в реестре, а потом по адресу **004C5C30** сравниваются введённое и нужное

значения. Но нашлись бесплатные и более полезные программы.

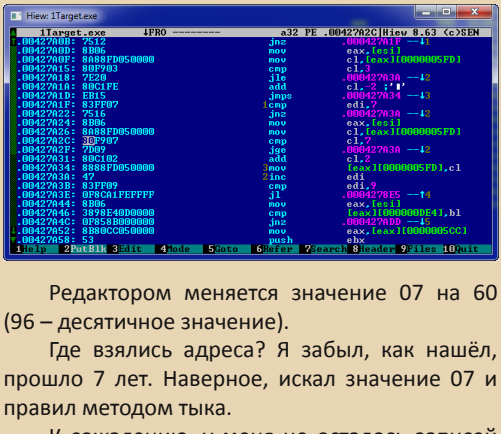
Наверное, описание покажется скучным, и процесс выглядит нелегко, но для меня подобный поиск серийных номеров – это своего рода упражнение в поиске алгоритма. Иногда приходится прибегать к другим методам. Например, в утилите **Secs** (это такой таймер, я позже написал свою версию) я сдался и поправил код проверки, чтобы процедура выдавала значение, что данные в порядке.

Правка кода коснулась и игр.

Target: Renegade/Viridis Cosmos Games

Ремейк очень популярной игры содержит два недостатка: работает только в полном экране и число жизней ограничено 7.

Открываю программу с помощью Hiew и перехожу по адресу **427A2C**, где происходит обработка счётчика:



Редактором меняется значение 07 на 60 (96 – десятичное значение).

Где взялись адреса? Я забыл, как нашёл, прошло 7 лет. Наверное, искал значение 07 и правил методом тыка.

К сожалению, у меня не осталось записей про игру Heavy Weapon Deluxe: Atomic Tank, сохранилась только модифицированная версия. Поиск шёл от сообщения **GAME OVER** и код был немного модифицирован. Подсказку я нашёл в списке Cheat Engine.

Наверное, читателю покажется лёгким способ поиска регистрационных данных, но может оказаться, что исследование не будет таким простым. Основа приложения Win32 – окно (или диалоговое окно) с отдельными стандартными элементами: **Edit**, **Button** и др. Например, чтобы получить текст, который хранится в **Edit**, нужно использовать функцию API. Например:

SendMessageA,hwnd,WM_GETTEXT, cchTextMax,lpszText

hwnd – **handle** окна **Edit** или идентификатор, с помощью которого можно обратиться к элементу;

cchTextMax – максимальное число символов, которые будут скопированы;

lpszText – адрес, где будет размещён текст.

Поиск вызова может занять массу времени, тем более, что разные компиляторы используют свой способ получения текста.

Но эта сложность вряд ли остановит любопытного пользователя, а, скорее всего, толкнёт на изучение материалов программирования.

Зная основы, любой пользователь способен написать собственную программу. 20 лет назад, когда я начал изучать программирование для Windows, я часто использовал отладчик для поиска данных по причине неточной документации – неясно описанная функция, отсутствуют константы.

Такое создание куда лучше деструктивного взлома – написание keygen/patch, в худшем случае – вируса или программы-шутки.

Поэтому, написав свою программу, Вы почувствуете, чего стоит этот труд.

«Что ты такое написал? Сам же ломал эти программы?» – скажет читатель. Я всего лишь указал примеры программ, где регистрация невозможна по причине исчезнувших фирмочек или проверка данных слишком упрощена.

На этой заметке вместо традиционного пожелания удачи в исследованиях рассказ заканчивается.

Sh

LEXICON HASN'T BEEN PROPERLY INSTALLED

«После этого повторите установку основных файлов Лексикона используя Ваш прежний регистрационный номер.
У кого пиратский CD – кумекайте сами, не сложно ;-)»
© http://www.aha.ru/~lexicon/lex/l_drv.htm

Материал статьи носит исключительно исследовательский и ознакомительный характер, при его использовании в нелегальных целях всю ответственность несут непосредственные исполнители.

Предложение разработчика «покумекать» принято, но началась эта история лет 10 назад, когда позвонил знакомый и сходу озадачил:

– Привет! У тебя «Лексикон» для DOS есть?

– Вообще, где-то был. А что случилось?

– На работе старый документ нашли, который нужно распечатать, а он в формате «Лексикона».

– Понятно. Сейчас поищу и вышлю на почту.

Однако через некоторое время раздался повторный звонок:

– Не работает твой «Лексикон».

– В смысле? А что сообщает?

– Сообщает, что не был правильно установлен.

– Странно. Копировал когда-то со списанного компьютера, где он работал, правда, не проверял после этого. Погоди, сейчас у себя запущу... да, действительно. Хорошо, посмотрю, что можно сделать и минут через 15-20 перезвоню.

Быстро найдя и заменив в отладчике условный переход на безусловный, перед выводом сообщения об ошибке, «Лексикон» был успешно запущен, а документ наконец-то распечатан. И так бы эта история и осталась всего лишь случаем из жизни, если бы в журнале «Downgrade» не собирали номер с темой «Компьютерный андеграунд». Так что появился хороший повод вернуться к программе, а заодно и разобраться, что ж там такое было-то.

Сейчас «Лексикон» можно спокойно взять с сайта Old-Dos.ru:

http://old-dos.ru/files/file_119.html

А ещё пригодится эмулятор DOSBox, который берётся отсюда:

http://www.dosbox.com/

Будем смотреть версии «Лексикона» 1.x, ибо всё, что больше (5.x, 6.x, 8.x) – это старые версии, выходившие до смены нумерации и превращения программы в полноценный коммерческий продукт, поэтому и регистрации они не требовали. Кстати, в архиве с версией 1.4 лежит регистрационный номер (далее – серийник), который также подходит и к 1.3, но вот с 1.2 Мод. 8.99 работать не будет.

Серийник состоит из двух частей, разделённых пробелом, для удобства назовём их левая и правая часть, и выглядит примерно так (с сайта Old-Dos.ru):

```
2XXXXXXXXXX 00096
```

Где вместо X – некоторые цифры. Позже подробнее его разберём, пока что запомним только правую часть (96).

Загружаем «Лексикон» 1.4, устанавливаем базовую версию, без всяких драйверов и

дополнительных файлов, вводим заботливо приложенный серийник и запускаем. Всё работает.

Т.к. 1.4 с серийником лежит в виде образов дискет в формате DDI (DiskDupe Image), то для установки можно воспользоваться слегка неживой уже программой **Extract**, выковыряв её саму предварительно с Интернет-архива Archive.org:

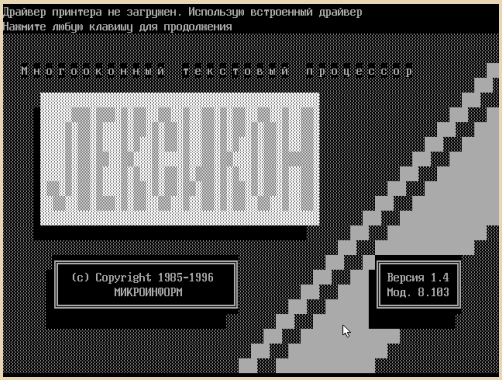
https://web.archive.org/web/20071026091939/http://ourworld.compuserve.com/homepages/gvollant/extrac21.zip

Затем нужно распаковать образы:

```
extract.exe -x lex1.ddi
extract.exe -x lex2.ddi
```

Куда-нибудь в одно место, которое потом смонтировать в DOSBox как:

```
mount a: <путь до того самого места>
```



Ладно, теперь вытаскиваем уже установленный и зарегистрированный «Лексикон» из DOSBox и запускаем в реальном DOS или под другим эмулятором.

```
ЛЕКСИКОН не был правильно установлен
LEXICON hasn't been properly installed
```

Опа, вот оно что. Программа-то привязывается к оборудованию при регистрации, чтобы особливо умные люди не могли, например, вынести её с работы домой. Ну, разве что, только вместе с компьютером, на котором «Лексикон» был установлен.

Но вернёмся к программе. Если ставить не под DOSBox (сбрасывает даты на текущую у всех файлов), а в реальном DOS или виртуальной машине, то сразу станет видно, кто ответственен за регистрацию – файл **LEXICON.EXE** изменил дату на текущую. Заглянем в файл **INSTALL.BAT**, устанавливающий программу, и слегка побродив по нему, находим такие строки:

```
:set_num
if -%3 == -r echo Копирую файлы...
if -%3 == -e echo Copying files...
lexicon -~%%In$tal01#D
if errorlevel 1 goto bad_num
```

Если сейчас запустить установленный **LEXICON.EXE** с ключом **-~%In$tall#D** (второй «%» нужен для экранирования в bat-файле), то программа сообщит, что не знает такой ключ. Это всё потому, что она уже зарегистрирована. Пока что сравним установленный файл с оригиналом:

```
C:\LEXICON>fc /b lexicon.exe a:\lexfiles\lexicon.exe

0003EDE8: 00 2B
0003EDE9: 00 EA
0003EDEA: DD 00
0003EDEB: 3E 00
0003EDEC: 65 60
0003EDED: D3 00
0003EDEE: 01 9D
0003EDEF: 00 39
0003EDF0: BB D8
0003EDF1: 7F D3
```

Видно, что поменялось всего 10 байт. Как говорил известный персонаж из сказки с опилками в голове: если байты изменились, значит,

кто-то их изменил; по-моему, так. Неплохо бы выяснить – кто. Заменяем установленный **LEXICON.EXE** оригиналом с образа установочной дискеты.

Затем берём в руки DOSBox Debugger: https://www.vogons.org/viewtopic.php?t=7323

И кидаем его к установленному DOSBox, после чего в конец секции **[autoexec]** файла **dosbox.conf** дописываем:

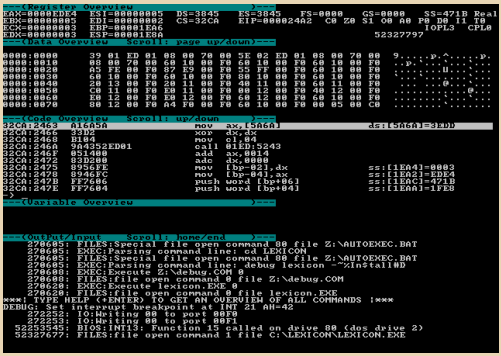
```
cd LEXICON
debug lexicon -~%%In$tall#D
```

Теперь запускаем отладчик через файл **dosbox-*-debug.exe**, и он должен сразу остановиться на точке входа. Т.к. перед записью программа будет вынуждена перейти на нужное смещение в файле, то ставим бряк на **file seek (int 21h, AH = 42h)**. Для этого во втором окне (отладчик) вводим и жмём **Enter**:

```
bpint 21 42
```

После этого запускаем программу работать дальше – **F5 (run)**.

Вводим серийник, и программа должна выпасть на установке файлового указателя. Пара регистров **CX:DX** даёт смещение в файле, куда будет совершён переход. Для установленной версии 1.4 (Мод. 8.103) это будет **0003:EDE4**. Запоминаем, а лучше записываем, чтобы не забыть. Теперь не запускаем программу на выполнение далее, а понажимаем **F10 (step over)**, пока не дойдём до команды выхода из подпрограммы **retf**. Проходим и эту команду через **F10**, теперь идём вверх (стрелка вверх) по коду примерно до **CS:2463**.



Обращаем внимание на вот это:

```
mov  ax,[5A6A]  ; ds:[5A6A]=3EDD
xor  dx,dx
mov  cl,04
call 01ED:5243
add  ax,0014
adc  dx,0000
mov  [bp-02],dx
mov  [bp-04],ax
```

Это расчёт смещения. Отладчик даже подсказывает, что по этому адресу лежит значение **3EDD**. Ничего не напоминает? См. сравнение изменений в файлах выше (3 и 4 байты). Вот откуда взялось итоговое значение **3EDE4** – это (**3EDD << 04) + 0014**. С этим разобрались, теперь нужно узнать размер структуры. Для этого ставим ещё один бряк, но уже на запись в файл **file write (int 21h, AH = 40h)**:

```
bpint 21 40
```

И выполняем дальше через **F5**.

Отладчик снова остановится, но уже на записи в файл. Тут нужно смотреть регистр **CX** – в нём хранится размер буфера, т.е. количество байт для записи. Сейчас там **000E** – это значит 14 байт. Если нажать **Alt+X (moves data view to DS:DX)** в окне отладчика, то даже можно поглазеть на буфер, который будет записан:

```
00 00 00 00 2B EA 00 00 60 00 9D 39 D8 D3
```

На этом месте отладчик DOSBox можно закрывать.

Сделаем небольшую программу для просмотра этих 14 байт, заодно представив их как 7 беззнаковых short.

```
#include <io.h>
#include <stdio.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
```

```
unsigned short list[7];
unsigned long i;
int f;
  if (argc != 3) { return(0); }
  f = open(argv[1], O_RDONLY | O_BINARY);
  if (f == -1) { return(0); }
  sscanf(argv[2], "%lx", &i);
  lseek(f, (i << 4) + 0x14, SEEK_SET);
  read(f, list, sizeof(list));
  close(f);
  printf("%08lX %s\n", i, argv[1]);
  for (f = 0; f < 7; f++) {
       printf("%2d  %04X  (%d)\n",  f  +  1,
              list[f], list[f]);
  }
  return(0);
}
```

Запускать так:

```
lxdmpreg.exe lexicon.exe 3EDD
```

Пройдёмся программой по обычной и зарегистрированной версии «Лексикона» 1.4, затем сведём результаты в общую таблицу:

Таблица 1

#	original	registered
1	0000 (0)	0000 (0)
2	0000 (0)	0000 (0)
3	0000 (0)	EA2B (-5589)
4	3EDD (16093)	0000 (0)
5	D365 (-11419)	0060 (96)
6	0001 (1)	399D (14749)
7	7FBB (32699)	D3D8 (-11304)

Тут нужно заметить, что на реальной системе (не DOSBox) первые два значения в зарегистрированной версии тоже будут отличны от нуля, видимо, «Лексикон» не может получить какие-то данные для привязки к системе под DOSBox. Нетрудно заметить, пока что, только две вещи: смещение (**#4**) обнуляется после регистрации и непонятно что (**#5**) меняется на правую часть серийника (96).

Здесь можно было бы засесть в отладчик или дизассемблер, найти код генерации серийника (он, кстати, достаточно простой) и написать генератор, но это долго и муторно, да и пользоваться такой версией неудобно, потому что при переносе на другой компьютер придётся нести не только генератор или серийник, но и чистую версию, чтобы её зарегистрировать. Можно, конечно, вернуться к тому исправленному байту перед выводом сообщения об ошибке, но он тоже по разным смещениям будет у разных версий.

Поэтому пойдём ленивым путём – скачаем все версии программы 1.x с Old-Dos.ru и поищем там эту структуру (проще всего по значению **D365**), затем пропустим через программу выше. Нужно заметить, что версия 1.0 ещё регистрации не имела, что легко определяется по отсутствию строчки **-~%In$tall#D** в файле, поэтому смотрим только версии 1.1 и выше. И также соберём все данные в одну таблицу для наглядности и удобства анализа – см. таблицу 2.

Таблица 2

ver	1.1 (Mod. 8.97)	1.2 (Mod. 8.98)	1.2 (Mod. 8.99)	1.3 (Mod.8.101)	1.4 (Mod.8.103)	1.4 (Mod.8.105)
ofs	0002E5B4	000361D4	000361D4	000357E4	0003EDE4	0003F1C4
1	0000 (0)	0000 (0)	0000 (0)	0000 (0)	0000 (0)	0000 (0)
2	0000 (0)	0000 (0)	0000 (0)	0000 (0)	0000 (0)	0000 (0)
3	0000 (0)	0000 (0)	0000 (0)	0000 (0)	0000 (0)	0000 (0)
4	0000 (0)	361C (13852)	361C (13852)	357D (13693)	3EDD (16093)	3F1B (16155)
5	D365 (-11419)	D365 (-11419)	D365 (-11419)	D365 (-11419)	D365 (-11419)	D365 (-11419)
6	6BE5 (27621)	0001 (1)	2774 (10100)	0000 (0)	0001 (1)	0001 (1)
7	A742 (-22718)	332C (13100)	7593 (30099)	7FF8 (32760)	7FBB (32699)	7FF8 (32760)

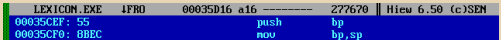
Замечаем аномалию у версии 1.1 по сравнению с остальными: во-первых, она не требует регистрации и работает на любой конфигурации, хотя структура и код там присутствуют; во-вторых, смещение на структуру (**#4**) там ноль, т.е. версия, вроде как, зарегистрирована, в-третьих, первые три значения по-прежнему нули.

Сделаем поиск числа **6BE5** (байты **E5 6B**) внутри **LEXICON.EXE** от версии 1.4 в HIEW в режиме дизассемблера. HIEW для DOS версии 6.50 можно взять на официальном сайте:

http://hiew.ru/indexr.html

Скачиваем, распаковываем, кидаем к «Лексикону» и в DOSBox запускаем:

```
hiew.exe lexicon.exe
```



Ого, да там целая проверка. Попробуемка, шутки ради, снова взять чистую 1.4, обнулить там смещение в **#4**, а в **#6** и **#7** записать **6BE5** и **A742** соответственно. Изменения должны будут получиться такие:

```
C:\LEXICON>fc /b lexicon.exe a:\lexfiles\lexicon.exe

0003EDEA: 00 DD
0003EDEB: 00 3E
0003EDEE: E5 01
0003EDEF: 6B 00
0003EDF0: 42 BB
0003EDF1: A7 7F
```

Запускаем изменённый файл...

Внезапно «Лексикону» становится безразлична как регистрация, так и оборудование, на котором он запущен. Оказывается, сами авторы предусмотрели возможность сделать «portable»-версию. Можно смастерить универсальный патч, но работать он будет только на чистой, не зарегистрированной версии из дистрибутива, где маркер **D365** и смещение до блока с регистрационными данными ещё не были изменены.

```
#include <io.h>
#include <stdio.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
unsigned short data[7];
unsigned long i;
int f;
 printf("Lexicon portable registration\n"
        "ToysLoss, 2019\n\n");
 if (argc != 2)
 {
   printf(
     "Usage: lxregist <lexicon.exe>\n\n"
     "You need a clean (NOT registered)"
     " version of \"lexicon.exe\" from"
     " distributive.\n\n"
   );
   return(0);
 }
 f = open(argv[1], O_RDWR | O_BINARY);
 if (f == -1)
 {
   printf("Error: can't open input file"
          " for read-write.\n\n");
   return(0);
 }
 printf("Searching for registration"
        " block...");
```

```
  i = 0;
  while (i < 0x100000UL) {
    lseek(f, i + 0x14, SEEK_SET);
    if (read(f, data, sizeof(data)) !=
        sizeof(data))
    {
      i = 1;
      break;
    }
    if ((data[4] == 0xD365) && (data[3] ==
         (i >> 4)))
    {
      data[3] = 0;
      data[5] = 0x6BE5;
      data[6] = 0xA742;
      lseek(f, i + 0x14, SEEK_SET);
      write(f, data, sizeof(data));
      printf("found at: %08lX\nLexicon"
             " now fully portable and"
             " registered.\n\n", i);
      i = 0;
      break;
    }
    i += 16;
  }
  close(f);
  if (i)
  {
    printf("not found\n"
           "Error: not a Lexicon"
           " executable or already"
           " registered.\n\n");
  }
  return(0);
}
```

Печально, что генератор серийника оказался не нужен, но пару слов о нём замолвить всё же можно.

Для начала, почему серийник ***00096** с Old-Dos.ru не подходит к версии 1.2 Мод. 8.99 – смотрим на общую таблицу, там **#6** и **#7** до регистрации отвечают за минимально и максимально возможные значения правой части, а 96 в интервал **[10100..30099]** не попадает, так что и программа работать с таким номером не будет. Поэтому если делать универсальный серийник, подходящий к любой версии, то лежать он должен, как видно из таблицы выше, в диапазоне **[10100..13100]**.

А ещё левая часть всегда начинается на «2», где далее идут цифры… сформированные на основе правой части через встроенный генератор случайных чисел Borland C. Правая же часть – это начальное значение (**seed**) генератора случайных чисел. Поэтому «Лексикон» и сохраняет только её, т.к. по ней всегда можно восстановить левую, а заодно узнать, чья версия утекла, если авторы, конечно, вели учёт серийников, кому они были проданы и, вообще, заморачивались подобными вопросами.

Генератор случайных серийников для совсем ленивых ниже. При обычном запуске генерирует случайный номер, работающий во всех перечисленных выше версиях, а если указать аргументом командной строки число, то сгенерирует для него левую часть для полноценного серийника.

Нужно отдать должное авторам «Лексикона», они весьма оригинально и просто подошли к созданию регистрационного номера, не изобретая велосипеды с криптосистемами шифрования, но одновременно позаботившись и о стойкости метода – подобрать левую часть практически невозможно, даже если знать, что правая – это начальное значение для генератора случайных чисел.

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

#ifdef __TURBOC__

#define BC_srand srand
#define BC_rand rand

#else
```

```
/* Borland C random routines */

/* ! 32bit int ! */

int BC_RandSeed = 1;

void BC_srand(unsigned short value) {
  BC_RandSeed = value & 0xFFFF;
}

short BC_rand(void) {
 BC_RandSeed = (0x015A4E35L * BC_RandSeed) + 1;
 return((BC_RandSeed >> 16) & 0x7FFF);
}
#endif

/* '2' + "%02d" + "%03d" + "%03d" + "%02d"
+ ' ' + "%05d" + \0 */
#define LX_KEY_LEN (1 + 2 + 3 + 3 + 2 + 1 + 5 + 1)

void LXKeyGen(char *s, short seed) {
short x;
  BC_srand((unsigned short) seed);

  x = (seed % 111) + 63;
  while (x--) { BC_rand(); }

  *s = '2'; s++;

  x = (BC_rand() *  100UL) / 32768U;
  sprintf(s, "%02d", x); s += 2;

  x = (BC_rand() * 1000UL) / 32768U;
  sprintf(s, "%03d", x); s += 3;

  x = (BC_rand() * 1000UL) / 32768U;
  sprintf(s, "%03d", x); s += 3;

  x = (BC_rand() *  100UL) / 32768U;
  sprintf(s, "%02d", x); s += 2;

  sprintf(s, " %05u", (unsigned short) seed);
}
int main(int argc, char *argv[]) {
char s[LX_KEY_LEN];
  if (argc == 2) {
    LXKeyGen(s, (short) atoi(argv[1]));
  } else {
    BC_srand((unsigned short) time(NULL));
    LXKeyGen(s, (BC_rand() % 3001) + 10100);
  }
  printf(
    "Lexicon random key generator\n"
    "ToysLoss, 2019\n\n"
    "Run program: LEXICON.EXE"
    " -~%%In$tall#D\n"
    "Enter a key: %s\n\n", s
  );
  return(0);
}
```

Материал статьи носит исключительно исследовательский и ознакомительный характер, при его использовании в нелегальных целях всю ответственность несут непосредственные исполнители.

Прим. ред.: скачать упоминаемые в статье исходники и бинарники можно здесь:

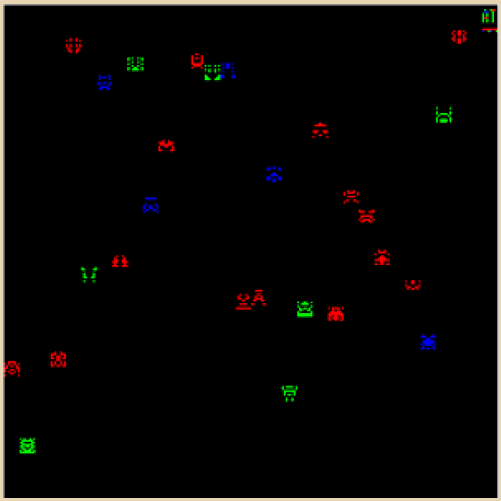
http://dgmag.in/N29/lex/lex.zip

Или здесь:

http://old-dos.ru/dl.php?id=20768

ToysLoss

Making of BKvaders



https://www.pouet.net/prod.php?which=81305

Поводом для написания программы для меня стал интерес к компьютеру БК-0010.

К тому же в 2017 на демосцене появились интересные демо:

«Однажды»: https://www.pouet.net/prod.php?which=73234

«e-Bun. ЭлектроБулка»: https://www.pouet.net/prod.php?which=68705

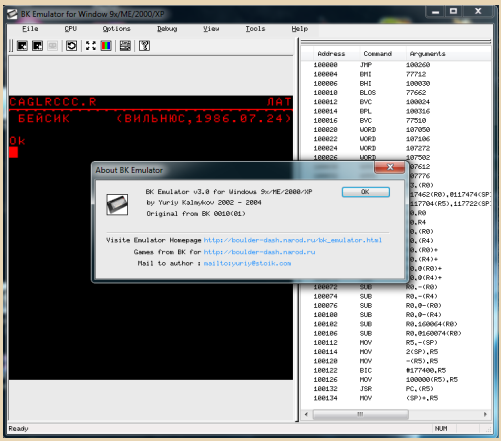
И в 2019 году было объявлено проведение CaFe demoparty, один из конкурсов относился к БК.

Поэтому я решил попробовать свои силы в программировании.

Подготовка пошла по привычной мне схеме:

1.Эмулятор

Выбор пал на BK Emulator 3.0 по той причине, что в эмулятор включён отладчик.



И ещё одно удобство: быстрый запуск файлов **.bin**.

Альтернативой может послужить другой эмулятор (http://gid.pdp-11.ru), так как он поддерживает две модели.

Найдётся эмулятор и для Android.

2. Документация

Поиск вывел на книгу Зальцмана «Микро-ЭВМ БК-0010. Архитектура и программирование на языке Ассемблера»:

http://gid.pdp-11.ru/books/Zaltsman.html

Дополнительный поиск привёл к «Программированию в машинных кодах»:

http://vak.ru/doku.php/proj/bk/bk4prog

Здесь описана полезная информация об экране и формате пикселей.

3. Ассемблер

Меня привлёк BKTurbo8: http://gid.pdp-11.ru/f/BKTurbo8.rar

От PDPy11 (https://github.com/imachug/PDPy11) я отказался, так как было сложно его настроить.

Теперь всё нужное есть под рукой, осталось изучить документацию.

Пересказывать прочитанное не имеет смысла, скажу только, что процессор совместим с PDP-11.

Есть несколько регистров: **PSW** (Processor Status Word) – состояние процессора после выполнения операций:

– бит 0 (**C**) устанавливается в 1, если при выполнении команды произошёл перенос единицы из старшего разряда результата;

– бит 1 (**V**) устанавливается в 1, если при выполнении арифметической команды (например, сложения) произошло арифметическое переполнение;

– бит 2 (**Z**) устанавливается в 1, если результат равен нулю;

– бит 3 (**N**) устанавливается в 1, если результат отрицателен.

Эти биты **PSW** используются командами условного перехода. Остальные биты **PSW** устанавливаются программистом для задания режимов работы процессора:

– бит 4 (**T**), установленный в 1, вызывает после выполнения очередной команды прерывание по вектору 14. Это используется программами-отладчиками для трассировки программы;

– бит 7 (**P**), установленный в 1, запрещает (маскирует) прерывания от внешних устройств (например, клавиатуры). Таким образом, командой установки **PSW MTPS #200** можно запретить прерывания от клавиатуры до тех пор, пока не выполнится команда **MTPS #0**.

Кроме этого, существуют 16-битные Регистры Общего Назначения (в литературе **РОН**) **R0-R7**.

R6 это стек, **R7** – Program Counter.

С остальным материалом лучше ознакомиться самостоятельно.

Нашёлся один пример (http://bk0010.narod.ru/images/scans/bk-clrscr.htm) красивой очистки экрана, я перевёл непривычные восьмеричные числа и расписал команды:

```
    MOV   #40000,R4 // R4=16384, screen
    MOV   #40,R3    // R3=32
M3: MOV   #20,R2    // R2=16
M2: MOV   R4,R0     // R0=R4
    MOV   #400,R1   // R1=256
M1: ASL   (R0)      // shift left (R0)
                    // contents
    ADD   #100,R0   // +64 - next line
    SOB   R1,M1     // loop one row
    SOB   R2,M2     // Substract One and
                    // Branch, if not equal
    INC   R4        // R4+1
    INC   R4        // R4+1
    SOB   R3,M3     // repeat 32
    RET   //
.END
```

Как работает процедура? В документации описан экран:

ОЗУ экрана (видеопамять) занимает область памяти с адреса 40000 по адрес 77777. На рисунке 22 показано соответствие адресов видеопамяти определенным участкам на экране.

Например, левому верхнему углу экрана соответствует адрес 40000 видеопамяти, а нижнему правому углу – адрес 77777.

Для того, чтобы вывести какое-либо изображение на определенный участок экрана, достаточно записать по соответствующему адресу

40000	40001	40002	40003	. . .	40076	40077
40100	40101	40102	40103	. . .	40176	40177
. . .					. . .	
77600	77601	77602	77603	. . .	77676	77677
77700	77701	77702	77703	. . .	77776	77777

Рис. 22. Соответствие адресов видеопамяти определенным участкам на экране.

видеопамяти число, определяющее вид данного изображения. Например, чтобы на участке экрана, соответствующем адресу 56036, высветить точки, расположение которых показано на рисунке 23, достаточно переслать по адресу 56036 двоичный байт 10010011 (восьмеричное число 223). На языке ассемблера такая пересылка выражается командой

```
MOVB #223,@#56036
```

красная		черная		синяя		зеленая		
■	■			■			■	<— расположение точек при цветном изображении
								<— расположение точек при черно-белом изображении
1	1	0	0	1	0	0	1	<— содержимое разрядов байта
0	1	2	3	4	5	6	7	<— номера разрядов байта

Рис. 23. Пример изображения байта на экране

Разряды байта выводятся на экран (слева направо), начиная с младшего разряда двоичного числа, что доставляет некоторые неудобства программисту.

Все сказанное выше верно только в том случае, если изображение выводится на черно-белый экран – тогда единичным битам выводимого байта соответствуют белые точки на экране, а нулевым битам – черные. Если же изображение выводится на цветной экран, то в формировании каждой "цветной" точки участвуют по 2 двоичных разряда. Красную точку дает комбинация разрядов 11, зеленую – 01, синюю – 10, и черную – 00.

Разрешение экрана 256x256 для режима RGB, 512x256 для монохромного изображения.

Один байт описывает 4 точки, получается, что на линию отведено 64 байта:

16384..16447 – одна линия,

16448..16511 – линия ниже.

Автор программы очищает область экрана с помощью сдвига **ASL (R0)** – если любопытный читатель не добрался до описания, то опкод сдвигает 16-битное значение влево по адресу, хранимому в регистре **R0**.

Цикл повторяется 256 раз по всем линиям, затем та же операция повторяется 16 раз.

Всё выглядит аккуратно и симпатично.

Сборка программы не вызвала затруднений. Делается это **.bat**-файлом:

```
set nam=bc
del %nam%.bin
BKTurbo8.exe -l -o cl %nam%.asm
BKTurbo8.exe -l -o li %nam% %nam%.obj
del %nam%.obj
del %nam%.lst
pause
```

Запуск программы делается старым индейским способом:

Набрать MO (Enter)

M (Enter)

Имя файла? (Enter)

Выбрать файл в диалоговом окне открытия файла

S (Enter)

Хорошо, пора приступить к замыслу, который называется Space Invaders.

Идея проста: выбрать случайное число 0-15 (получится 4 бита) и отразить значение зеркально.

Для 8 точек в ширину неплохо выбрать два значения 16 бит, чтобы получилось 8 точек в высоту.

Арифметика простая: 32 бита (2*16)/4 = 8 линий для invader.

Для одного invader нужно ещё два байтовых значения – координата **X** (0-63) в байтах и координата **Y** (0-255) для падения.

Первый шаг не удался, я застрял на генерации случайных чисел, процедуру подсказал **Manwe**:

```
  HM=31.
    MOV   #16384., SP

//init space invaders x,y,data1,data2

    MOV   #SIT,R3
    CLR   R5
I1:
    CALL  RND //R1-Random
    MOV   R1,R0 //R0=R1
    CALL  RND
    MOV   R1,R4 //R4=R1

//and put
    MOVB  R5,(R3)+ //X
    MOVB  R0,(R3)+ //Y
    MOV   R0,(R3)+ //data1
    MOV   R4,(R3)+ //data1
    INCB  R5
    INCB  R5
    CMPB  R5,#64.//(HM+HM+2)
    BNE   I1
```

Чтобы не путаться, достаточно запомнить **MOV source,destination**.

Для удобства я использовал координату **X** в качестве счётчика, а не как случайное число.

```
//draw sprites
DRI:
    MOV   #SIT,R3
    MOV   #HM+1,R1
D1:

//detect mask
    MOVB  2(R3),R4
    BIC   #0b1111111111111100,R4
```

Трюк был в использовании разных цветов при отрисовке, у первого слова **data1** выделяется два бита, которые отвечают за цвет.

Кстати, битовой операции **AND** нет в списке, поможет **BIC** (**Bit Clear**), которая используется по маске. Для битовой операции **OR** читайте **BIS**.

```
MOVB mask(R4),ma
MOVB mask(R4),ma+1
```

Небольшое дополнение кода: я решил рисовать на экран сразу, а потом менять цвет invaders на ходу. Такой трюк сбил с толку саму систему и появился неприятный момент – картинка мерцает.

```
MOVB   (R3)+,R0//X
MOVB   (R3)+,R4//Y

SWAB   R4 //Manwe hint:
         // Y*64+X+40000 - screen
CLRB   R4
CLC
ROR    R4
ASR    R4

BISB   R0,R4
ADD    #40000,R4
```

А здесь на рисовании вышло не пойми что – картинки рисуются до половины экрана. **Manwe** подсказал свою процедуру, и я позже разобрался, в чём причина: опкод **MOVB** не просто копирует только 8 бит, а помещает в регистр 16 бит с учётом знака, т.е. 1 даст 1, 127 – 127, а 129 даст $FF81 (запись в шестнадцатеричной системе, мне так удобнее).

Сама процедура очень элегантна: для получения адреса нужно умножить **Y** на 64, что решилось не сложением, а всего двумя сдвигами.

```
CLR   (R4)
ADD   #64.,R4

CALL  SI1
CALL  SI1
```

Теперь рисуется спрайт. Как я описал выше, случайное значение 32 бит используется для invaders высотой 8 пикселей.

```
    INCB  -5(R3)
    BNE   NXI
    MOV   R1,-(SP)
    CALL  RND
    MOV   R1,-2(R3)
    MOV   R2,-4(R3)
    MOV   (SP)+,R1
NXI:
    SOB   R1,D1
    BR    DRI//HALT
```

В конце цикла **Y**-координата увеличивается на 1, и если дошла до 256, то данные инициализируются заново, на экране не появится одинаковый «пришелец» того же цвета, а совсем другой.

```
SI1: //R4=screen
    MOV (R3)+,R0

    MOV #4,R5 //counter
M1:
    MOV R0,R2
    BIC #0b1111111111110000,R2 //clear
                               //bits
    ASL R2
    MOV SIDATA(R2),R2

    CMP R4,#100000
    BCC NXL

    BIC ma,R2
    MOV R2,(R4)

    ASR R0
    ASR R0
    ASR R0
    ASR R0
NXL:
    ADD #64.,R4
    SOB R5,M1
EXL:
    RETURN
```

Процедура рисования, которая отняла массу времени на полировку: в идеале invader должен был пропасть за нижней границей экрана, никакие ухищрения не помогали, но наконец удалось добиться желаемого эффекта.

```
mask:
    .byte 0,0b01010101,0b10101010,255.
ma: .word 0//.byte 0b10101010,0
```

Маски цвета для спрайтов. Однако вышел косяк – часть «пришельцев» рисуется чёрным цветом и их не видно. Решение: заменить 0 на 255 (это десятичная запись числа).

```
RND:    MOV   RND1,R1 // by Manwe
        MOV   RND2,R2
        ROR   R1
        ROL   R2
        SWAB  R2
        XOR   R2,R1
        MOV   R1,RND1
        MOV   R2,RND2
        RET
RND1:    .WORD 173451
RND2:    .WORD 54102
```

Процедура генерации случайных чисел.

```
SIDATA:
    .byte 0b00000000,0b00000000//0000
    .byte 0b00000011,0b11000000//0001
    .byte 0b00001100,0b00110000//0010
    .byte 0b00001111,0b11110000//0011
    .byte 0b00110000,0b00001100//0100
    .byte 0b00110011,0b11001100//0101
    .byte 0b00111100,0b00111100//0110
    .byte 0b00111111,0b11111100//0111
    .byte 0b11000000,0b00000011//1000
    .byte 0b11000011,0b11000011//1001
    .byte 0b11001100,0b00110011//1010
    .byte 0b11001111,0b11110011//1011
    .byte 0b11110000,0b00001111//1100
    .byte 0b11110011,0b11001111//1101
    .byte 0b11111100,0b00111111//1110
    .byte 0b11111111,0b11111111//1111
SIT:
```

SIT – область данных для invaders, **SIDATA** – 16 значений зеркальных бит. Из комментария к строкам видно, что бит «удваивается», так получен красный цвет. Для разнообразия применяется битовая маска.

```
    .END
```

Директива для ассемблера – конец исходного текста.

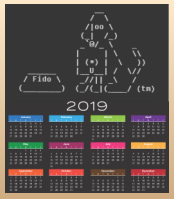
Всего на знакомство с программированием и на создание эффекта ушёл целый день. Пришельцы плавно падают, я чувствую, что немного устал. Но усталость покрывается чувством счастья, потому что поставленная цель была достигнута.

Прим. ред.: скачать бинарники и исходники к статье можно тут:

http://dgmag.in/N29/bkv/bkv.ZIP

Sh

Календарь по мотивам истории Фидонет (октябрь–декабрь)



ОКТЯБРЬ

1 октября

1985. Принята первая версия Полиси Фидонет (Policy and Procedures Guide, v.1).

1990. Принят стандарт FTS-5002.001 «Формат поинтлиста».

1998. В эхе RU.ANEKDOT Максим Товстюк устроил перепись под заголовком «Ударим переписью по безобразию в эхе !».

1999. В нодлисте появилась сеть 2:6033 (Великие Луки).

2007. Под редакцией Sean Dennis (1:1/23, Дугласвилль, штат Джорджия) начинает выходить альтернативный бюллетень Фидонет FidoGazette (выходит до сих пор). Создана одноимённая эхоконференция.

3 октября

1990. Andrew Pam выпускает программу робота-сисопа, «с которым было бы интересно поговорить» – AICHAT v.2.0.

1997. В нодлисте появилась сеть 2:5009 (Тамбов).

1997. В нодлисте появилась сеть 2:5041 (Биробиджан).

1997. В нодлисте появилась сеть 2:4656 (Харцызск, Украина).

2010. Евгений Малетин на основе FidoIP создал новый самонастраивающийся пакет программ для Фидо – «Кубик».

4 октября

2011. Павел Гульчук (2:463/68, Киев) назначен мастером DNS-зоны для региона 50. В настоящее время это домен binkp.net.

5 октября

1987. U.S. Robotics делает предложение невиданной щедрости – скидку фидошникам на модемы Courier 2400 и Microlink 2400. Обычная их цена $325, а для сисопов – $289, причём даже купившие модемы в течение последних трёх месяцев до этой акции могли вернуть себе разницу.

6 октября

1995. В нодлисте появилась сеть 2:5032 (Новгород).

1995. В нодлисте появилась сеть 2:5036 (Липецк).

1995. В нодлисте появилась сеть 2:5088 (Павлодар, Казахстан).

1998. В нодлисте появилась сеть 2:4658 (Шахтёрск, Украина).

7 октября

1985. Robert Lederman написал программу SHUFFLE для обработки файлов из файловых областей. Этот файлэхопроцессор намного превосходил аналогичные программы других авторов.

1987. Steve Bonine, 1:115/777 (д. Глен Эллин, штат Иллинойс), публикует проект правил выборов «Election Rules Version 0.10». Этот документ должен был определить порядок проведения референдума по будущему IFNA, но был положен и в основу других подобных регламентов. В частности, определялся принцип «один сисоп – один голос», указывался нодлист, по которому пройдёт голосование и т.п.

8 октября

1990. Впервые в списке фидософта, публикуемом в FidoNews, появились разделы с ПО для OS/2 и Xenix/Unix.

1995. Odinn Sorensen выпустил фидоредактор GoldEd v.2.50. Эта версия долгое время считалась «классической».

9 октября

1992. В нодлисте появилась сеть 2:5055 (Волгоград).

1998. В нодлисте появилась сеть 2:5092 (Пыть-Ях).

10 октября

1994. Принято первое эхополиси региона 50 (Россия). Оно принималось общим голосованием всех узлов региона.

1997. В нодлисте появилась сеть 2:5094 (Кызыл).

11 октября

2004. Jon Watson, 1:134/703 (Калгари, Канада) анонсирует, по-видимому, первый веб-форум с фидошными эхоконференциями. Форум был на движке Simple Machines (похож на PHPBB, PHPNUKE), с интегрированными программами Internet Rex, MBSE BBS и самописными PHP-скриптами. Неофициальное название системы – Fido On The Web (FOTW).

13 октября

2000. В нодлисте появилась сеть 2:6003 (Усинск, Республика Коми).

14 октября

2005. В нодлисте появилась сеть 2:6004 (Уссурийск).

15 октября

1993. В нодлисте появился регион 2:51 (Латвия) и сеть 2:5100 (Рига). Ранее, с 23 апреля по 29 октября 1993, латвийский регион имел нумерацию 2:479, а рижская сеть – 2:4790.

1999. В нодлисте появилась сеть 2:6001 (Рубцовск, Алтайский край).

16 октября

1987. В нодлисте появился регион 2:49. Изначально он числился за Польшей, с 8 января 1988 стал числиться за Южной Африкой, а с 28 сентября 1990 – за Эстонией.

17 октября

2015. Дмитрий Каменский, 2:5023/24 (Калуга) сообщает об открытии своей WebBBS на движке wFido https://wfido.ru

18 октября

1996. В нодлисте появилась сеть 2:5071 (Братск).

1996. В нодлисте появилась сеть 2:4600 (Севастополь, Крым).

1996. В нодлисте появилась сеть 2:4613 (Полтава, Украина).

1996. В нодлисте появилась сеть 2:4626 (Черновцы, Украина).

2012. Иван Агарков, 2:5020/848 (Москва) сделал форк от своего фидокомплекта jNode под Android. В будущем jFMailer будет использован, в частности, при разработке популярного фидокомплекта на Android – HotdogEd.

2013. Объявлено о создании англоязычной эхи HOTDOGED.

19 октября

2005. С эхобона снята «кащенитская» эхоконференция PVT.SKL.TACTICS, создав прецедент удаления с бона эхи, не соответствующей заявленной тематике.

20 октября

1991. C. Anderson выпускает документацию к первой программе для эхоконференций ArcMail. К этому времени вышла уже бета третьей версии ArcMail.

1995. В нодлисте появилась сеть 2:5006 (Новокузнецк).

1995. В нодлисте появилась сеть 2:5082 (Восточно-Казахстанская область).

1995. В нодлисте появилась сеть 2:4621 (Ровно, Украина).

21 октября

1994. Вард Дошше (Ward Dossche) становится координатором второй зоны Fidonet. На этом посту он бессменно находится до сих пор.

2005. В нодлисте появилась сеть 2:4500 для IP-only узлов из 45-го региона (Беларусь).

2005. Координатор региона 2:50 (Россия) опубликовал окончательный вариант «Положения о выдаче IP-only узлов». Согласно ему, сеть 6000 (для IP-only узлов региона 50) «находится в стадии расформирования».

22 октября

1990. Принято эхополиси первой зоны (Северная Америка) – «General Echomail Policy 1.0».

1993. В нодлисте появилась сеть 2:4641 (Запорожье, Украина).

2001. В FidoNews рассказано про группу Fidonet в проекте SETI (поиск внеземного разума). Ныне по запросу Fidonet среди команд проекта находится более десятка групп: https://setiathome.berkeley.edu/team.php .

23 октября

1986. Жалоба на первое (и, увы, не последнее) мошенничество в Фидо. Brian Walsh (1:109/640) объявил конкурс, предлагая модем на 1200 бод тому, кто первым пришлёт на дискетах 100 бесплатных программ. В конкурсе якобы победил Howard Feil, но ни модема, ни обещанного возврата своих дискет он не дождался. Brian Walsh, в свою очередь, обвинил Howard Feil в том, что тот не выполнил условия конкурса, прислав множество копий одной и той же программы под разными именами, плюс пиратские копии программ. Брайан собирался подать в суд, но чем закончилась эта история, доподлинно неизвестно.

24 октября

1988. Широкой публике анонсирован редактор MSGED, который к этому времени обзавёлся уже версией 1.87. Это был первый (и на то время единственный) Fido-редактор с открытым исходным кодом.

25 октября

2019. Единогласным голосованием четырёх зональных координаторов Ward Dossche из Бельгии избран на пост международного координатора (IC) Фидонет. Эта должность оставалась вакантной на протяжении более десяти лет, и последним, кто её до этого занимал, был тоже Ward Dossche (в 2000-2006 годах).

28 октября

1985. Впервые бюллетень Fidonews начал распространяться в запакованном виде (архиватор ARC). Было также предложено использовать архиватор и для нодлиста.

29 октября

1987. Вторая конференция сисопов Новой Англии состоялась в Массачусетсе, США.

2001. В FidoNews анонсирован проект Джейсона Скотта (Jason Scott) «BBS: The Documentary». Работа над ним началась в июле 2001 года и завершилась в 2005 году выходом восьмисерийного документального фильма (на трёх дисках) про историю и субкультуру BBS и Fido (на английском языке).

31 октября

1988. Rick Moore представлен как новый глава FTSC (Комитета по техническим стандартам Fidonet).

1997. В нодлисте появилась сеть 2:4521 (Мозырь, Беларусь).

2003. Небывалое событие. В первом туре выборов координатора сети 5020 (Москва) победил кандидат «Против всех».

НОЯБРЬ

1 ноября

1991. В нодлисте появилась сеть 2:464 (Днепропетровск, Украина).

1993. В нодлисте появилась сеть 2:462 (Львов, Украина).

1993. Hayes Microcomputer Products Inc. представила новый модем Hayes OPTIMA 288 V.FC + FAX modem с пропускной способностью до 230,400 bit/s (230.4 kbit/s). При его цене в США $579 – для сисопов сделано специальное предложение – $288.

1996. В нодлисте появилась сеть 2:5096 (Электросталь).

2011. В эхе IPV6 объявлено о первом удачном коннекте двух узлов по протоколу BinkP over IPv6. Сисопами этих узлов были Andre Grueneberg (Германия) и Benny Pedersen (Дания).

2 ноября

1990. В Fidonet создана Zone 6 (Азия). Z6C стал Honlin Lue.

1992. Randy Bush (Портленд, штат Орегон) впервые описывает систему туннелирования фидо-трафика через каналы TCP/IP.

Первоначально это было актуально для межконтинентальных гейтов.

4 ноября

2011. Andre Grueneberg (2:2411/525, Малов, Германия) сделал поддержку IPV6 для виндовой версии мейлера BinkD.

5 ноября

1990. Международный координатор Фидо Matt Whelan сообщил в FidoNews об утверждении им документа FidoNet Gateway Policy.

6 ноября

1992. В нодлисте появилась сеть 2:5002 (Барнаул).

8 ноября

2016. Игорь Горун, 2:466/4 (Николаев, Украина), разрабатывающий ifmail'овскую вебморду, запустил на этом движке экспериментальную WebBBS FTNW.

9 ноября

1984. Число узлов в нодлисте Fidonet достигло 100.

1990. В нодлисте появился Регион 2:50 (изначально – Сибирь).

Region,50,Siberia,Novosibirsk_USSR,Vladimir_Lebedev,[...]

Первоначально в его составе была одна сеть 2:5000 (Новосибирск).

Host,5000,The Court Of The Crimson King,Novosibirsk USSR,Eric Fletcher,[...]

2004. Решением координатора сети 2:5020 (Москва) Сергея Озерова отменено вошедшее в анналы истории Fido постановление Фарида Вагапова о признании чрезмерно некорректным поведением факта подписки на эху TYT.BCE.HACPEM.

11 ноября

1991. В FidoNews анонсирован новый редактор для Fido – GoldEd (автор – Odinn Sorensen). К этому времени текущей версией GoldEd'а была уже 2.31.

2005. Сняты с бона SU.KASCHENKO.LOCAL и SU.KASCHENKO.GLOBAL.

12 ноября

1993. В нодлисте появилась сеть 2:4622 (Кривой Рог, Украина).

1993. В нодлисте появилась сеть 2:4651 (Краматорск, Украина).

1999. В нодлисте появилась сеть 2:4634 (Чернигов, Украина).

13 ноября

1987. День рождения Fido в Австрии (регион 2:31).

1992. В нодлисте появилась сеть 2:5005 (Томск).

1992. В нодлисте появилась сеть 2:5025 (Воронеж).

1992. В нодлисте появилась сеть 2:4635 (Черкассы, Украина).

2008. Перевод фидополиси Алексея Виссарионова утверждён R50C как «официальный перевод Устава Фидонет на русский язык».

14 ноября

1986. В Токио состоялась встреча сисопов (FIDO/COLLIE SYSOPS' MEETING), на которой было принято решение просить отдельный регион для японского Фидо.

1987. Первая конференция сисопов Новой Англии состоялась в Массачусетсе, США.

1992. Robert Heller (Уэнделл, штат Массачусетс) публикует проект FSC A FidoNet (FTN) Domain Name Service.

16 ноября

1990. В зоне 4 (Латинская Америка) прошли «всеобщие выборы всех координаторов». Впервые посты координатора зоны, регионов и сетей Z4 (всего 9 должностей) были заняты по результатам выборов.

17 ноября

1950. Родился Вард Дошше (Ward Dossche), бессменный координатор второй зоны Fidonet с 1994 года.

1995. В нодлисте появилась сеть 2:4614 (Сумы, Украина).

18 ноября

2011. Благодаря стараниям Дмитрия Игнатова в Ярославле появился огромный

рекламный плакат Fidonet (хотя некоторые утверждают, что это был фейк).

20 ноября

1992. В нодлисте появился регион 2:45 (Беларусь) и сеть 2:450 (Минск).

1998. В нодлисте появилась сеть 2:5072 (Астана, Казахстан).

21 ноября

1991. In Fase Productions выпускает написанную Richard Faasen, 2:285/311 (Слидрехт, Нидерланды) игру для BBS Tetra – онлайн-вариант тетриса. Играть могли юзеры BBS самостоятельно, а также против сисопа.

1997. В нодлисте появилась сеть 2:4648 (Бердянск, Украина).

22 ноября

1997. Объявлены результаты голосования по кандидатурам на должность Администратора FTSC. Победил Odinn Sorensen из Дании (разработчик редактора GoldEd), набрав 80% голосов.

2003. Пётр Соболев публикует свою книгу «Эхоконференции сети FidoNet и их модерирование».

23 ноября

1985. Европейская конференция сисопов – Утрехт, Нидерланды. Проходила в рамках Hobbycomputerclub (HCC). На неё был приглашён создатель Фидо Tom Jennings, HCC оплатил ему дорожные расходы. Том был поражён массовостью Fido- и BBS-движения в Европе. Конференция освещалась на национальном телевидении, в газетах и на радио.

25 ноября

1994. В нодлисте появилась сеть 2:4624 (Хмельницкий, Украина).

1994. В нодлисте в составе региона 2:46 (Украина) появилась сеть 2:4690 (Румыния).

26 ноября

1997. Dave Carter объявил о запуске в Великобритании проекта FOTI (Fidonet On The Internet).

ДЕКАБРЬ

1 декабря

1983. Tom Jennings выпускает первую версию программы Fido.

1984. Под редакцией Тома Дженнингса вышел первый номер бюллетеня FidoNews. В самом начале Том заявил, что не хочет быть редактором: «Последнее, что мне нужно, это что-то ещё делать»: «I, Tom Jennings, am apparently the editor. I do NOT wish to be editor; the last thing I need is something else to do. See the HELP WANTED section. (Not kidding)». Помимо прочего, он тут же предложил сделать значки стикеры на бампер, чтобы фидошники могли узнавать друг друга. Кстати, чуть позже он их сделал.

1995. В нодлисте появилась сеть 2:5013 (Магнитогорск).

2 декабря

1994. В нодлисте появилась сеть 2:5075 (Тольятти).

3 декабря

1999. В нодлисте появилась сеть 2:477 для IP-only нод из R47 (Литва).

4 декабря

2004. Перезапущен сайт FleetStreet (фидо-редактор для OS/2) – http://fleetstreet.mhohner.de (доступен до сих пор).

6 декабря

1991. В нодлисте появилась сеть 2:5040 (Хабаровск).

7 декабря

1992. Стало известно о том, что Playboy подал в суд на сисопа (имя не уточняется) за размещение на BBS защищённых авторским правом GIF-изображений. Вместе с тем, Playboy предложил сисопу удалить эти файлы. Случай был урегулирован во внесудебном порядке.

1992. В зоне 1 (Северная Америка) объявлено о создании структуры SecureMail – списка узлов, которые дали явное согласие на пересылку шифрованных сообщений. На тот момент в списке присутствовало 4 региональных хаба.

1992. Компания Advanced Engineering sarl из Люксембурга анонсирует поинткомплект FrontDoor APX. Он отличался простотой установки и настройки, обширной справочной системой, полной поддержкой мыши. Релиз запланирован на 1 февраля 1993.

1998. Объявлено о создании пула адресов 1:1/3xxx для IP-узлов первой зоны Fidonet. В нодлисте использовались флаги, которые Lothar Behet (2:2406/301) предложил в FTSC.

8 декабря

1986. По аналогии с IFNA в Австралии создаётся AFNA – Australian Fido-Net Association.

9 декабря

1994. В нодлисте появилась сеть 2:4631 (Луцк, Украина).

1996. Под редакцией Hakan Andersson (2:201/601) начал выходить шведский FidoNews. Выпускался еженедельно до 2002 года включительно. http://www.fidonet.itu.se/sfnews/index.html

2018. Алексей Матюк сообщил о том, что у него на домашнем узле (2:5020/8912) теперь работают 9 диалапных линий, «и на этом можно остановиться».

10 декабря

1993. В нодлисте появилась сеть 2:5024 (Владимир).

1993. В нодлисте появилась сеть 2:5049 (Казань).

1993. В нодлисте появилась сеть 2:5052 (Йошкар-Ола).

1993. В нодлисте появилась сеть 2:5053 (Саратов).

1993. В нодлисте появилась сеть 2:5061 (Ростов-на-Дону).

11 декабря

1998. В нодлисте появился регион 2:55 и сеть 2:550, созданные специально для IP-only узлов второй зоны Fidonet.

12 декабря

1988. Анонсирован тоссер GroupMail. В отличие от системы Echomail, в которой указывается, на какие узлы отсылается конференция, в GroupMail указывалось, где можно получить эту конференцию. Для аркмейла вместо отсылки по методу file attach использовалось получение по методу file update request. В будущем технология Echomail полностью вытеснила GroupMail.

1997. В нодлисте появилась сеть 2:5016 (Коломна).

13 декабря

1991. В нодлисте появилась сеть 2:5050 (Ижевск).

15 декабря

1991. Запущена BBS «ДиалогНауки» (DialogueScience BBS). Впоследствии на её основе будет выдан узел 2:5020/69 (сисоп – Борис Чернивецкий).

1995. В нодлисте появилась сеть 2:5007 (Абакан).

16 декабря

2014. Александр Соловьёв сообщил об открытии неофициального твиттера сети 5030 (Питер) https://twitter.com/Fidonet_5030 .

17 декабря

1984. Том Дженнингс сообщает о поломке своего жёсткого диска и потере части почты. Его станция Fido #1 (на которой завязано в то время почти всё Фидо) работает на двух дискетах.

1993. В нодлисте появилась сеть 2:5021 (Тверь).

1993. В нодлисте появилась сеть 2:5083 (Алма-Ата, Казахстан).

18 декабря

1992. В нодлисте появилась сеть 2:5051 (Симбирск).

2017. Руди Тиммерманс (Rudi Timmermans, 2:292/140, Бельгия) выпускает фидософт Raptor v.2.0 под ОС Sailfish. Ранее этот поинткомплект он разрабатывал под BlackBerry OS.

20 декабря

1997. В Москве состоялся фидошный митинг против повременной оплаты телефона.

22 декабря

1995. В нодлисте появилась сеть 2:5048 (Сызрань).

2000. В нодлисте появилась сеть 2:6016 (Ступино, Московская область).

23 декабря

1994. В нодлисте появилась сеть 2:5059 (Пенза).

24 декабря

1984. В Фидо осуществлена пересылка первого межконтинентального сообщения из Джакарты, Индонезия, в Сент-Луис, США. «Tom Jennings announces in FidoNews that the first intercontinental FidoNet message was sent from Jakarta, Indonesia, to St. Louis, United States. FidoNets had already previously been sent internationally, appearing in Canada nearly immediately after the network's founding».

1984. Узлы Фидо появились в Европе (Англия) и Азии (Япония).

1993. В нодлисте появилась сеть 2:5011 (Уфа).

25 декабря

1983. Том Дженнингс начал работу над Fidonet. Точная дата неизвестна, во всех источниках упоминается, что это случилось «около Рождества 1983».

1992. В нодлисте появилась сеть 2:5047 (Магадан).

1992. В нодлисте появилась сеть 2:4632 (Александрия, Украина).

26 декабря

2011. Michiel van der Vlist, 2:280/5555 (Дриберген, Нидерланды) рассказывает о применении Unicode (UTF-8) в Фидонет. В частности, упоминает редакторы InterMail и Msged, которые могут отображать юникодные символы, а также указывает на проблему с оным в GoldEd'e.

27 декабря

1998. Вышел первый open source release редактора GoldEd pre-3.0.0-1.

2014. В Москве состоялся фест «ASWDF 2014: Зимняя сказка», на котором выступил Сергей Позитурин (2:5020/2140) с рассказом «про hotdoged, jNode и вообще».

29 декабря

2009. Вышел в свет поинткомплект FidoIP, который был вначале поинткомплектом лишь для Linux. Поле того, как FidoIP стал мультиплатформенным, он завоевал большую популярность у новопоинтов и возвращающихся в сеть.

30 декабря

1991. Сообщено о запуске двустороннего гейтования между эхоконференцией SC_HAMPK и системой любительской пакетной радиосвязи HAM Radio Packet.

31 декабря

1993. В нодлисте появилась сеть 2:5001 (Кемерово).

1993. В нодлисте появилась сеть 2:5035 (Курск).

2004. Впервые в истории Фидо вышел еженедельный пятничный нодлист с порядковым номером 366. Как заметил Michiel van der Vlist, такое событие происходит раз в 28 лет и в следующий раз случится только 31 декабря 2032 года.

2007. Ulrich Schroeter сообщил о возобновлении выпуска поинтлиста второй зоны. Первым Z2PK (сборщиком поинтлиста второй зоны – Европа) с января 2000-го был Кирилл Лебедев, в августе 2000-го его сменил Алексей Антонюк, который оставался на этой должности до 2007 года, а потом подал в отставку.

2011. Максим Сокольский выпустил FIDOSlax Linux 1.0 – компактный и быстрый русскоязычный дистрибутив Linux, который может работать на разделе жёсткого диска, накопителе USB-флеш или CD/DVD-диске. Помимо прочего, в него было интегрировано программное обеспечение для FIDO over IP.

Владимир Фёдоров, 2:50/15

Utsurun Desu

Utsurun Desu.: Kawauso Hawaii e Iku!!! («Уцурун, десу») – видеоигра в жанре платформер по мотивам манги Сэнся Ёсида.

Главный герой – человек-выдра Каваусо, который мечтает побывать на Гавайях.

По своей механике игра представляет собой классический платформер, иногда разбавленный элементами простых (и неочевидных) головоломок.

Графика представляет собой минимализм в деталях и не восхищает из-за своей простоты. Уровни короткие, боссы очень простые. Проходится игра очень быстро.

С первого взгляда – типичный мариоподобный платформер

Тогда в чём же особенность данной игры?

Начнём с титульного экрана. После запуска игры его сместило в сторону. Играл я в русскую версию от **Chief-Net**, а именно в перевод от **Guyver**. Зная профессионализм **Гайвера**, я и подумать не мог, что мне попался «сломанный ром», который я пропатчил переводом. Оказывается, всё гораздо проще, игра намеренно сдвинула экран для ощущения поломки картриджа.

Слот засорился, нужно подуть на картридж...

Из оружия у нас простой удар рукой, который помогает разобраться с медлительными врагами. Выдра неторопливо прыгает и разворачивается медленно, прокручивая анимацию разворота.

Управление стандартное. Крестовина для движения и приседания, кнопки **B** и **A** для прыжка и удара. Если зажать кнопку удара, то вверху экрана будут меняться значки, и от того, когда вы отпустите кнопку, будет зависеть, какая способность сработает. Разберём, зачем они нужны и каково их применение.

 – выдра просто танцует. Да, просто танцует. Помогает пройти второй уровень и получить небольшой бонус в кабинете офиса.

 – пламя в две противоположные стороны. Действенное оружие против боссов и особо быстрых врагов.

 – моментально убивает врагов на экране и помогает убить недосягаемых боссов.

 – выдра смотрит в пол и не двигается. Всё будет зависеть от вашей реакции: или все

враги умирают, или вы начинаете беззащитно стоять. «Сглаживает» предыдущую способность.

Разобравшись с неприветливым управлением и медлительностью выдры, я начал проходить уровень. Диссонанс от неочевидности загадок вызвал у меня ступор. Представьте ситуацию: перед вами огромная яма, в наличии у вас короткий прыжок. Представили? Ваша первая реакция? Посмотреть, есть ли платформа над шипами? Нет, её нет. Вернуться назад? Нет, не получится. Прыгнуть в яму? Отличная идея! Дело в том, что фон, который визуально расположен за игровым экраном, является платформой. Скажете, просто авторы данной игры сделали это нечаянно? Бывает и такое, некоторые дизайнеры сливают фоновые объекты с расположенными впереди, но тут всё специально. Таких «специально» в игре будет много.

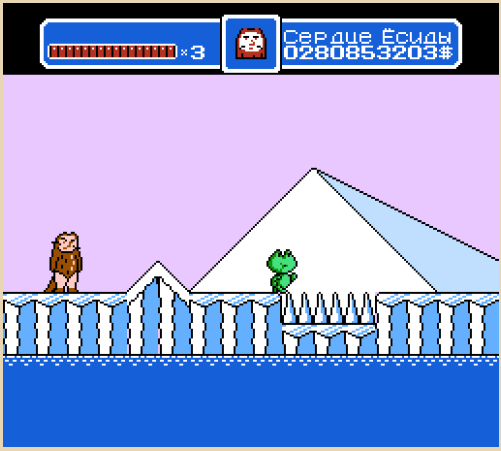
В игре абсурдно всё. Графика абсурдна своими объектами и изображением лиц персонажей, к которым уже привыкли современные ценители манги и аниме, ведь отсылки встречаются очень часто. Диалоги нетипичны и обсмеивают клише японского творчества. Абсурдными бывают и ситуации в духе «твой босс в другом месте».

Наверное, эта игра одна из немногих юмористических игр того года. Она разбита на несколько уровней, которые связаны простым сюжетом.

В игре две концовки, и от чего зависит, попадёт ли Кавауссо на Гавайи, неизвестно.

Подводя к итогу всё, что я сказал выше, могу заявить, что сама по себе игра довольно весёлая, и для проведения за ней вечера (вообще, прохождение занимает меньше часа) вполне подходит.

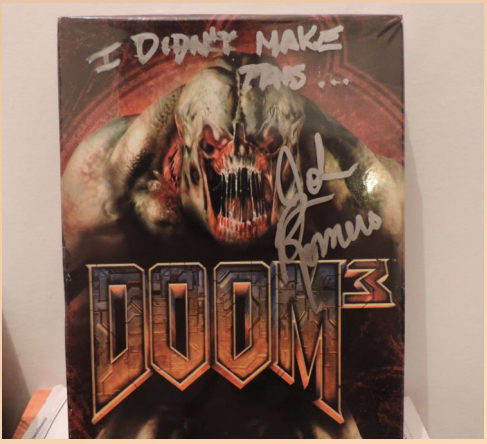
Да, она безумна, кривовата и подойдёт не всем, но негативного осадка не оставляет.



Павел Ярославцев (paha_13)

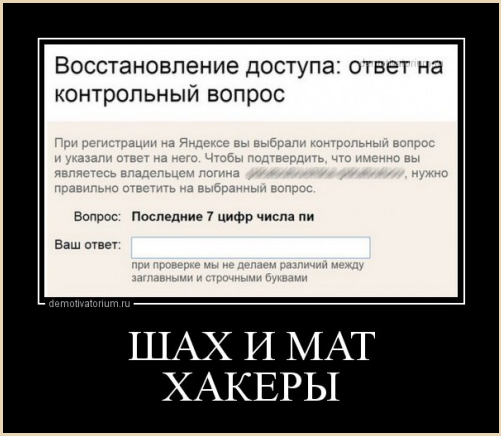
ПРОСТО РАЗНЫЙ ЮМОР

Разработчик игр Джон Ромеро подписал протянутую ему фанатом коробку с Doom 3: «Я не делал это...»

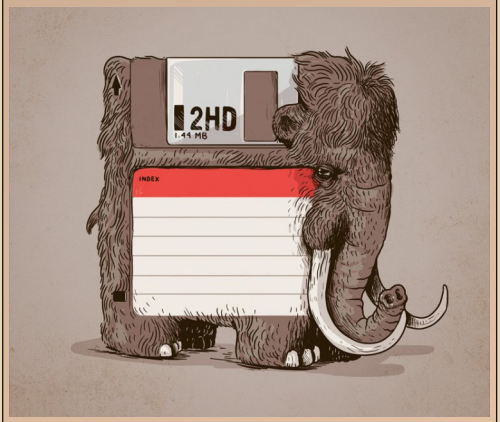


The Human Body in HTML & PHP

```
<human>
 <head>
  <hair /><br />
  <ear align="left" />
  <sight><eye align="left" />
  <eye align="right" /></sight>
  <ear align="right" /><br />
  <nose /><br />
  <form action="aliment.php"><mouth /></form><br />
  <neck height="8cm" />
 </head>
 <body>
  <tshirt style="background-color: #000;" />
   <arm align="left"><hand /></arm>
   <chestarea><?php if ($sex='female')
{echo '<tit align="left" /><tit align="right" />';}
   else {echo '<nipple align="left" />
  <nipple align="right" />';} ?></chestarea>
   <arm align="right"><hand /></arm>
<sponsor href="http://www.alvago.com.ar">Alvago Go!</sponsor>
  <br /><tummy><bellybutton /></tummy></tshirt>
  <pants size="short"><underwear>
<?php include 'private.php'; ?></underwear></pants>
  <leg align="left" />
<leg align="right" style="tattoo-image: url(img/alvago.gif);" />
  <sneaker align="left" class="nike"><foot /></sneaker>
<sneaker align="right" class="nike"><foot /></sneaker>
 </body>
</human>
```



Картины из цикла PREHISTORIC («Доисторические») от Alex Solis. С другими его работами можно ознакомиться здесь: http://cargocollective.com/oddworx/PREHISTORIC



НАД НОМЕРОМ РАБОТАЛИ

Дизайн/вёрстка/главный
редактор - uav1L0L
Редактор - Вячеслав Рытиков (euLpc)
Помощник редактора - Андрей Шаронов

Авторы:
Андрей Шаронов (Andrei88)
Михаил Бабичев (Антиквар)
SysTools
Александр Мачуговский (Manwe)
Вячеслав Рытиков (euLpc)
uav1L0L
Олег Павлов
Владимир Фёдоров
ToysLoss
Sh
Павел Ярославцев (paha_13)

Интервью:
Д.С. Барулин (Дес) и Р.С. Подковыров (PC)

Сайт журнала: http://dgmag.in

Раздел журнала на "Полигоне Призраков":
http://sannata.org/articles/dgmag/

Группа ВКонтакте: http://vk.com/dgmag

В журнале использованы рисунки с сайта
http://freepik.com/

E-mail главного редактора:
uav1L0L0 [cobaka] mail.ru